

Texts in Computer Science

Editors

David Gries

Fred B. Schneider

For other titles published in this series, go to
<http://www.springer.com/series/3191>

Stefano Crespi Reghizzi

Formal Languages and Compilation

Stefano Crespi Reghizzi
Politecnico di Milano
Italy
stefano.crespireghizzi@polimi.it

Series Editors:

David Gries
Department of Computer Science
415 Boyd Graduate Studies Research Center
The University of Georgia
Athens, GA 30602-7404, USA

Fred B. Schneider
Department of Computer Science
Upson Hall
Cornell University
Ithaca, NY 14853-7501, USA

ISBN 978-1-84882-049-4 e-ISBN 978-1-84882-050-0
DOI 10.1007/978-1-84882-050-0

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

Library of Congress Control Number: 2008942386

©Springer-Verlag London Limited 2009

Apart from any fair dealing for the purposes of research or private study, or criticism or review, as permitted under the Copyright, Designs and Patents Act 1988, this publication may only be reproduced, stored or transmitted, in any form or by any means, with the prior permission in writing of the publishers, or in the case of reprographic reproduction in accordance with the terms of licences issued by the Copyright Licensing Agency. Enquiries concerning reproduction outside those terms should be sent to the publishers.

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

The publisher makes no representation, express or implied, with regard to the accuracy of the information contained in this book and cannot accept any legal responsibility or liability for any errors or omissions that may be made.

Printed on acid-free paper

Springer Science+Business Media
springer.com

Preface

State of books on compilers

The book collects and condenses the experience of years of teaching compiler courses and doing research on formal language theory, 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 over half a century, and its relevance for theory and practice is as important as in the early days.

This state of affairs of a topic, which is central to computer science and is based on consolidated principles, might lead us to believe that the accompanying textbooks are by now consolidated, much as the classical books on mathematics. In fact this is rather not true: there exist fine books on the mathematical aspects of language and automata theory, but the best books on translators are sort of encyclopaedias of algorithms, design methods, and practical know-how used in compiler design. Indeed a compiler is a microcosm, featuring a variety of aspects ranging from algorithmic wisdom to CPU and memory exploitation. As a consequence the textbooks have grown in size, and compete with respect to their coverage of the last developments on programming languages, processor architectures and clever mappings from the former to the latter.

A basic textbook on compilation

To put things into order, in my opinion it is better to separate such complex topic into two parts, basic and advanced, which correspond with good approximation to the two subsystems of a compiler: the user language-specific front-end, and the machine language-specific back-end. The basic part is the subject of this book; it covers the principles and algorithms widely used for defining the syntax of languages and implementing simple translators. It does not include: the specific know-how needed for various classes of program-

ming languages (imperative, functional, object oriented, etc.), the computer architecture-related aspects, and the optimization methods used to improve the machine code produced by the compiler.

In other textbooks the bias towards technological aspects, related to software and hardware architectures, has reduced the attention to the fundamental concepts of language specification and translation. This perhaps explains why such books do not exploit the improvements and simplifications made possible by decades of extensive use of syntax-directed methods, and still keep the irritating variants and repetitions, to be found in the historical papers which introduced the theory of translation. Moving from these premises, I decided to present in a simple minimalist way the essential principles and methods used in designing syntax-directed translators. Just a few examples: the coverage of the algorithms for processing regular expressions and finite automata is rather complete and condensed. The systematic discussion of ambiguous forms is intended to avoid pitfalls when designing grammars. The standard presentation of parsing algorithms has been improved, by unifying the concepts and notations used in different approaches, thus extending methods coverage with a reduced definitional apparatus. The concepts of syntactic translation are effectively linked to regular expressions, grammars and abstract automata, and pave the way to attribute grammars and syntax-directed translation. The book is not restricted to syntax. The sections on translation, semantic functions (attribute grammars), and static program analysis by data flow equations provide a more comprehensive understanding of the compilation process.

Presentation

The text is illustrated by many small yet realistic and paradigmatic examples, to ease the understanding of the theory and the transfer to application. Many diagrams and figures enlighten the presentation. This book has been written by an engineer for future engineers and compiler or language designers: the choice of the theoretical properties is always driven by their utility and the conceptual economy they allow. Theoretical models of automata, transducers and formal grammars are extensively used, whenever practical motivation warrants. Formal properties are intuitively justified and illustrated by examples; proofs are outlined whenever possible, and reference is given to publications. Algorithms are described in a pseudo-code to avoid the disturbing details of a programming language, yet they are straightforward to convert to executable procedures. Links to further readings and published references are provided as footnotes.

Intended audience

The main material can be taught in about 50 class hours to computer science or engineering students of the third year (graduate or upper division undergraduate), but of course cuts and selective specialization are possible. Actually the material is largely self-contained and is also suitable to self-learning. The first three chapters can be also used for introducing students (especially engineering ones) to the foundations of formal languages and automata, but other topics of theoretical computer science (such as computability and complexity) are not covered.

This book should be welcome by those willing to teach or to learn the essential concepts of syntax-directed compilation, without the need to rely on software tools and implementations. I believe that learning by doing is not always the best approach, and that early and excessive commitment to practical work may sometimes hinder the acquisition of the conceptual foundations. In the case of formal languages and data-flow analysis, the elegance and simplicity of the underlying theory allow 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, most relevant algorithms are simple enough to be practiced by paper and pencil. Of course, students should be encouraged to enroll in a parallel hands-on laboratory for experimenting syntax-directed methods and tools (like *flex* and *bison*) on realistic cases.

Supplementary Web materials

Course slides and numerous problems with solutions (prepared by L. Breveglieri for the English language class) are available from the author Web site, hosted by Politecnico di Milano, <http://www.dei.polimi.it/>. Error indications and comments from readers are appreciated, and an errata-corrigé will be set-up on site.

Acknowledgments

I remember and thank: Antonio Grasselli, who first fascinated me with a subject combining linguistic, mathematical and technological aspects; David Martin and Michel Melkanoff, my masters of “compilation” at UCLA; my colleagues Angelo Morzenti, Licia Sbattella, and especially Luca Breveglieri, for their critical revision; Alessandra Cherubini, Matteo Pradella, and Pierluigi San Pietro, research companions on formal languages and automata theory; my PhD students, former and present ones, and in particular, Giampaolo Agosta, Martino Sykora, and Simone Campanoni; ST Microelectronics, and especially Marco Cornero and Erven Rohou, for calling my attention to com-

pilation technology for advanced microprocessors counteracting my theoretical drift.

Stefano Crespi Reghizzi
Milan, September 2008

Contents

1	Introduction	1
1.1	Intended Scope and Audience	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	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	23
2.3.4	Closure Properties of <i>REG</i> Family	24
2.4	Linguistic Abstraction	25
2.4.1	Abstract and Concrete Lists	26
2.5	Context-Free Generative Grammars	30
2.5.1	Limits of Regular Languages	30
2.5.2	Introduction to Context-Free Grammars	31
2.5.3	Conventional Grammar Representations	33
2.5.4	Derivation and Language Generation	35
2.5.5	Erroneous Grammars and Useless Rules	37
2.5.6	Recursion and Language Infinity	39
2.5.7	Syntax Trees and Canonical Derivations	40
2.5.8	Parenthesis Languages	44

2.5.9	Regular Composition of Context-Free Languages	46
2.5.10	Ambiguity	47
2.5.11	Catalogue of Ambiguous Forms and Remedies	49
2.5.12	Weak and Structural Equivalence	57
2.5.13	Grammar Transformations and Normal Forms	59
2.6	Grammars of Regular Languages	67
2.6.1	From Regular Expressions to Context-Free Grammars	67
2.6.2	Linear Grammars	69
2.6.3	Linear Language Equations	71
2.7	Comparison of Regular and Context-Free Languages	73
2.7.1	Limits of Context-Free Languages	76
2.7.2	Closure Properties of <i>REG</i> and <i>CF</i>	78
2.7.3	Alphabetic Transformations	80
2.7.4	Grammars with Regular Expressions	83
2.8	More General Grammars and Language Families	87
2.8.1	Chomsky Classification	87
3	Finite Automata as Regular Language Recognizers	93
3.1	Introduction	93
3.2	Recognition Algorithms and Automata	94
3.2.1	A General Automaton	95
3.3	Introduction to Finite Automata	98
3.4	Deterministic Finite Automata	100
3.4.1	Error State and Total Automata	101
3.4.2	Clean Automata	101
3.4.3	Minimal Automata	103
3.4.4	From Automata to Grammars	106
3.5	Nondeterministic Automata	108
3.5.1	Motivation of Nondeterminism	108
3.5.2	Nondeterministic Recognizers	110
3.5.3	Automata with Spontaneous Moves	112
3.5.4	Correspondence between Automata and Grammars	114
3.5.5	Ambiguity of Automata	115
3.5.6	Left-Linear Grammars and Automata	116
3.6	Directly from Automata to Regular Expressions: BMC Method	117
3.7	Elimination of Nondeterminism	119
3.7.1	Construction of Accessible Subsets	121
3.8	From Regular Expression to Recognizer	124
3.8.1	Thompson Structural Method	124
3.8.2	Algorithm of Glushkov, McNaughton and Yamada	126
3.8.3	Deterministic Recognizer by Berry and Sethi Algorithm	134
3.9	Regular Expressions with Complement and Intersection	137
3.9.1	Product of Automata	138
3.10	Summary of Relations between Regular Languages, Grammars, and Automata	143

4	Pushdown Automata and Top-down Parsing	147
4.1	Introduction	147
4.1.1	Pushdown Automaton	148
4.1.2	From Grammar to Pushdown Automaton	149
4.1.3	Definition of Pushdown Automaton	152
4.2	One Family for Context-Free Languages and Pushdown Automata	157
4.2.1	Intersection of Regular and Context-Free Languages	160
4.2.2	Deterministic Pushdown Automata and Languages	160
4.3	Syntax Analysis	170
4.3.1	Top-Down and Bottom-Up Analysis	170
4.3.2	Grammar as Network of Finite Automata	172
4.3.3	Nondeterministic Recognition Algorithm	176
4.4	Top-Down Deterministic Syntax Analysis	178
4.4.1	Condition for $LL(1)$ Parsing	179
4.4.2	How to Obtain $LL(1)$ Grammars	192
4.4.3	Increasing Look-ahead	196
5	Bottom-Up and General Parsing	203
5.1	Introduction	203
5.2	Bottom-Up Deterministic Syntax Analysis	203
5.2.1	$LR(0)$ Method	204
5.2.2	$LR(0)$ Grammars	206
5.2.3	Shift-Reduce Parser	211
5.2.4	Syntax Analysis with $LR(k)$ Look-Ahead	214
5.2.5	$LR(1)$ Parsing Algorithm	226
5.2.6	Properties of $LR(k)$ Language and Grammar Families	226
5.2.7	How to Obtain $LR(1)$ Grammars	229
5.2.8	$LR(1)$ Parsing with Extended Context-Free Grammars	232
5.2.9	Comparison of Deterministic Families REG , $LL(k)$, and $LR(k)$	241
5.3	A General Parsing Algorithm	242
5.3.1	Introductory Example	243
5.3.2	Earley Algorithm	247
5.3.3	Computational Complexity	252
5.3.4	Handling of Empty Rules	254
5.3.5	Further Developments	256
5.4	How to Choose a Parser	256
6	Translation Semantics and Static Analysis	259
6.1	Introduction	259
6.1.1	Chapter Outline	260
6.2	Translation Relation and Function	262
6.3	Transliteration	264
6.4	Regular Translations	264

6.4.1	Two-Input Automaton	266
6.4.2	Translation Functions and Finite Transducers	270
6.5	Purely Syntactic Translation	275
6.5.1	Infix and Polish Notations	277
6.5.2	Ambiguity of Source Grammar and Translation	281
6.5.3	Translation Grammars and Pushdown Transducers	282
6.5.4	Syntax Analysis with Online Translation	287
6.5.5	Top-Down Deterministic Translation	287
6.5.6	Bottom-Up Deterministic Translation	290
6.5.7	Comparisons	295
6.5.8	Closure Properties of Translations	296
6.6	Semantic Translations	297
6.6.1	Attribute Grammars	299
6.6.2	Left and Right Attributes	301
6.6.3	Definition of Attribute Grammar	305
6.6.4	Dependence Graph and Attribute Evaluation	307
6.6.5	One Sweep Semantic Evaluation	311
6.6.6	Other Evaluation Methods	315
6.6.7	Combined Syntax and Semantic Analysis	316
6.6.8	Typical Applications of Attribute Grammars	324
6.7	Static Program Analysis	334
6.7.1	A Program as an Automaton	334
6.7.2	Liveness Intervals of Variables	338
6.7.3	Reaching Definitions	345
	References	353
	Index	357