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

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

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-corrige will be setup on site.

Aknowledgments

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