Instruction Selection

Gabriel Hjort Blindell

Instruction Selection

Principles, Methods, and Applications

Springer

Gabriel Hjort Blindell
Computer Systems Laboratory (CSL)
Royal Institute of Technology (KTH)
Kista
Sweden

ISBN 978-3-319-34017-3 ISBN 978-3-319-34019-7 (eBook)
DOI 10.1007/978-3-319-34019-7

Library of Congress Control Number: 2016942004

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer International Publishing AG Switzerland

För mina föräldrar
och min bror

Foreword

Compilers are ideal software projects: You have relatively stable and well-defined, partly even formal specifications for the input and the output, and lots of interesting subproblems in between.

In particular, the scanning part of compilers has been formalized using regular languages and is expressed in regular expressions, which have become not only popular for scanning programming languages, but for many other tasks as well. The parsing part has been formalized with context-free grammars in (extended) BNF and that has led to the development of parser generators that are quite popular in language implementations.

Formal specifications have been used for specifying the input language of compilers beyond context-free grammars, such as Van Wijngaarden grammars in ALGOL 68, or various formal semantics of programming languages. Among theses formalisms, attribute grammars seem to be the most popular, judging by the tool support available, but even that does not seem to be very popular: most language designers prefer the flexibility of natural language; most compiler writers prefer to specify work beyond the parser in a general-purpose programming language.

This book is about the other end of the compiler, the output part, in particular the part that has to do with the instruction set of the target processor. One might think that a similar development to frontends takes place here; but while the instruction sets of many processors are also specified in a relatively formal way, no common standard analogous to regular expressions or (extended) BNF has established itself.

One reason is that there are many different purposes for an instruction set description: assembly programming, code generation, performing assembly, simulation, automatic hardware generation, etc.

Still, for specifying instruction selection, retargetable compilers usually use machine description languages, rather than falling back to a general-purpose language, and these languages correspond to some formalism used in the instruction selector, e.g., a tree grammar in tree-parsing instruction selection. However, the qmachine description languages used differ significantly, because of differences in the instruction selection methods, and in the intermediate representations of the compilers.

One significant difference between compiler frontend and backend is that the frontend must interpret the input in only one way, while the backend can generate a variety of machine code programs for the input program, all of them correct. Therefore, the machine description is usually arranged as mapping the intermediate representation to machine instructions rather than the other way round, leading to compiler-specific machine descriptions.

Among the possible correct code sequences, the compiler will ideally select the one that is best in terms of speed, size or some other metric, so we have an optimization problem at hand, and like many compiler problems, this one is NP-complete in the general case.

Moreover, there are additional goals in instruction selection, such as short compilation time and ease of compiler development, resulting in a large variety of different instruction selection methods; some methods have been superseded, and are mainly of historical interest, but others are used because they offer a useful trade-off for various design goals. For example, Chris Fraser (who contributed to Davidson-Fraser code generation, and several tree-parser generators) once told me that he worked on tree parsing for fast compilation, but would use Davidson-Fraser if better generated code was more important.

Probably because there is no canonical instruction selection specification formalism comparable to BNF, instruction selection has been mostly neglected for a long time in many compiler textbooks. Even those textbooks that cover instruction selection have to limit themselves to just one or two techniques for space reasons.

If you want to know more, read this book! It gives a broad survey over the large body of literature on instruction selection. Also, if, after reading this book, you desire an even deeper knowledge of a particular method, this book points you to the original papers that you can then read, and it also provides background knowledge that is useful for understanding these papers.

This book is not just useful for students who are looking for knowledge beyond the compiler textbook level, but also for experts: I have published in this area, yet a lot of this material, especially the early work, was new to me when I got this book for review.

Vienna, Austria *M. Anton Ertl*
March 2016 TU Wien

Preface

Like most doctoral students, I started my studies by reviewing the existing, most prominent approaches in the field. A couple of months later I thought I had acquired a sufficient understanding of instruction selection and felt confident enough to begin developing new methods.

That confidence was short-lived.

When exploring my new ideas, I would soon face a number of problems about instruction selection that I didn't fully understand, prompting me to read more papers until I did. Empowered with new knowledge, I would resume my research, only to shortly after be confronted with yet another set of problems. After doing this for another few months, the pile of related work had grown so large that I started to wonder how many more papers the collection would need before it contained everything ever published on instruction selection. So I set out to find those missing papers. Several months later, my findings had been compiled into a 109-page technical report—which grew to 139 pages in a second revision—and although it was written primarily to be shared with others, I wrote it equally as much for myself to be used later as a manual of reference. At this point my supervisors and I believed the material to be of sufficient quality for publication, but it was simply too long to be accepted by any journal in its current form. Fortunately, Springer agreed to publish my work as a book, which is the one you are currently reading.

The ambition of this book is to (i) introduce instruction selection as an interesting problem—what it is, and why it matters—and (ii) present an exhaustive, coherent, and accessible survey on the existing methods for solving this problem. In most cases, the goal is to convey the main intuition behind a particular technique or approach. But for methods that have had a significant impact on instruction selection, the discussions are more in-depth and detailed. The prerequisites are kept to a minimum to target as wide an audience as possible: it is assumed that the reader has a basic background in computer science, is familiar with complexity theory, and has some basic skills in logic and maths. However, no expectations are made regarding prior knowledge on instruction selection, and very little on compilers in general. Hence the material presented herein should be useful to anyone interested in instruction selection, including:

- novice programmers, who have used a compiler but know nothing about its internals;
- intermediate and advanced students, who may already have taken a course on compilers; and
- expert practitioners, who have decades of experience in compiler development.

Stockholm, Sweden
January 2016

Gabriel Hjort Blindell
KTH Royal Institute of Technology

Acknowledgments

I am indebted to several persons, who, one way or another, have had a hand in shaping the material and appearance of this book.

First and foremost, I want to thank my main supervisor Christian Schulte, my colleague Roberto Castañeda Lozano, and my co-supervisor Mats Carlsson—their insightful feedback and helpful suggestions have had a significant impact in clarifying and improving much of the material.

I want to thank Ronan Nugent for taking care of the administrative tasks of publishing this book. I am especially grateful to Ronan for organizing the peer review, and I want to thank all the reviewers who agreed to participate—your comments have been invaluable. I also want to thank Karl Johansson and Jaak Randmets, who pointed out some of the errors that appeared in the earlier, report version.

I am grateful to the *Swedish Research Council* for funding this research,[1] and to the *Swedish Institute of Computer Science* for letting me use their office equipment and drink their delicious coffee.[2]

I want to thank everyone at *TEX StackExchange*[3] for helping me with all the LATEX-related problems I encountered in the course of writing this book. Without their help, the material would undoubtedly have been much less comprehensible.

Last—but certainly not least—I am indebted to Emily Fransson, who has been a dear friend since a long time and given me tremendous solace when I faced hard times during my doctoral studies. Without her support, this book might not have seen the light of day.

[1] VR grant 621-2011-6229.

[2] I am not certain which of the two played the most pivotal role in bringing this work together.

[3] `http://tex.stackexchange.com/`

Contents

List of Figures