

## 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

© Springer International Publishing Switzerland 2016

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.

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 these 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 machine 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  
March 2016

*M. Anton Ertl*  
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,<sup>1</sup> and to the *Swedish Institute of Computer Science* for letting me use their office equipment and drink their delicious coffee.<sup>2</sup>

I want to thank everyone at *TeX StackExchange*<sup>3</sup> 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.

---

<sup>1</sup> VR grant 621-2011-6229.

<sup>2</sup> I am not certain which of the two played the most pivotal role in bringing this work together.

<sup>3</sup> <http://tex.stackexchange.com/>

# Contents

<b>1</b>	<b>Introduction</b> .....	1
1.1	What Is Instruction Selection? .....	2
1.1.1	The Program Under Compilation .....	2
1.1.2	The Target Machine .....	3
1.1.3	Inside the Compiler .....	4
1.1.4	The Instruction Selector .....	5
1.2	Comparing Different Instruction Selection Methods .....	6
1.2.1	Introducing the Notion of Machine Instruction Characteristics .....	7
1.2.2	What Does It Mean for Instruction Selection to Be “Optimal”? .....	9
1.3	The Prehistory of Instruction Selection .....	10
<b>2</b>	<b>Macro Expansion</b> .....	13
2.1	The Principle .....	13
2.2	Naïve Macro Expansion .....	14
2.2.1	Early Applications .....	14
2.2.2	Generating the Macros from a Machine Description .....	16
2.2.3	Reducing Compilation Time with Tables .....	19
2.2.4	Falling Out of Fashion .....	21
2.3	Improving Code Quality with Peephole Optimization .....	22
2.3.1	What Is Peephole Optimization? .....	22
2.3.2	Combining Naïve Macro Expansion with Peephole Optimization .....	24
2.3.3	Running Peephole Optimization Before Instruction Selection .....	28
2.3.4	Interactive Code Generation .....	28
2.4	Summary .....	29
<b>3</b>	<b>Tree Covering</b> .....	31
3.1	The Principle .....	31
3.2	First Techniques to Use Tree-Based Pattern Matching .....	32
3.3	Using LR Parsing to Cover Trees Bottom-Up .....	36
3.3.1	The Graham-Glanville Approach .....	36

3.3.2	Extending Grammars with Semantic Handling . . . . .	43
3.3.3	Maintaining Multiple Parse Trees for Better Code Quality . .	45
3.4	Using Recursion to Cover Trees Top-Down . . . . .	46
3.4.1	First Applications . . . . .	46
3.5	A Note on Tree Rewriting vs. Tree Covering . . . . .	49
3.5.1	Handling Chain Rules in Purely Coverage-Driven Designs . .	49
3.6	Separating Pattern Matching from Pattern Selection . . . . .	50
3.6.1	Algorithms for Linear-Time, Tree-Based Pattern Matching .	51
3.6.2	Optimal Pattern Selection with Dynamic Programming . . . .	58
3.6.3	Faster Pattern Selection with Offline Cost Analysis . . . . .	63
3.6.4	Generating States Lazily . . . . .	69
3.7	Other Tree-Based Approaches . . . . .	70
3.7.1	Techniques Based on Formal Frameworks . . . . .	71
3.7.2	More Tree Rewriting-Based Methods . . . . .	72
3.7.3	Techniques Based on Genetic Algorithms . . . . .	73
3.7.4	Techniques Based on Trellis Diagrams . . . . .	73
3.8	Summary . . . . .	75
3.8.1	Restrictions That Come with Trees . . . . .	76
<b>4</b>	<b>DAG Covering . . . . .</b>	<b>77</b>
4.1	The Principle . . . . .	77
4.2	Optimal Pattern Selection on DAGs Is NP-Complete . . . . .	78
4.2.1	The Proof . . . . .	78
4.3	Straightforward, Greedy Techniques . . . . .	81
4.3.1	LLVM . . . . .	81
4.4	Techniques Based on Exhaustive Search . . . . .	82
4.4.1	Extending Means-End Analysis to DAGs . . . . .	82
4.4.2	Relying on Semantic-Preserving Transformations . . . . .	82
4.5	Extending Tree Covering Techniques to DAGs . . . . .	83
4.5.1	Undagging Program DAGs . . . . .	84
4.5.2	Extending the Dynamic Programming Approach to DAGs . .	85
4.6	Transforming Pattern Selection to an M(W)IS Problem . . . . .	88
4.6.1	Applications . . . . .	89
4.7	Transforming Pattern Selection to a Unate/Binate Covering Problem	92
4.8	Modeling Instruction Selection with IP . . . . .	94
4.8.1	Approaching Linear Solving Time with Horn Clauses . . . . .	95
4.8.2	IP-Based Designs with Multi-output Instruction Support . . .	96
4.8.3	IP-Based Designs with Disjoint-output Instruction Support .	96
4.8.4	Modeling the Pattern Matching Problem with IP . . . . .	97
4.9	Modeling Instruction Selection with CP . . . . .	98
4.9.1	Taking Advantage of Global Constraints . . . . .	99
4.10	Other DAG-Based Approaches . . . . .	101
4.10.1	More Genetic Algorithms . . . . .	101
4.10.2	Extending Trellis Diagrams to DAGs . . . . .	102
4.10.3	Hardware Modeling Techniques . . . . .	103

- 4.11 Summary ..... 104
- 5 Graph Covering ..... 105**
  - 5.1 The Principle ..... 105
  - 5.2 Pattern Matching Is a Subgraph Isomorphism Problem ..... 106
    - 5.2.1 Ullmann’s Algorithm ..... 107
    - 5.2.2 The VF2 Algorithm ..... 107
    - 5.2.3 Graph Isomorphism in Quadratic Time ..... 108
  - 5.3 Graph-Based Intermediate Representations ..... 109
    - 5.3.1 Static-Single-Assignment Graphs ..... 109
    - 5.3.2 Program Dependence Graphs ..... 110
  - 5.4 Modeling Pattern Selection as a PBQ Problem ..... 112
    - 5.4.1 Extending the PBQ Approach to Pattern DAGs ..... 114
    - 5.4.2 Using Rewrite Rules Instead of Production Rules ..... 115
  - 5.5 Other Graph-Based Approaches ..... 116
    - 5.5.1 More Hardware Modeling Techniques ..... 116
    - 5.5.2 Improving Code Quality with Mutation Scheduling ..... 117
  - 5.6 Summary ..... 119
- 6 Conclusions ..... 121**
  - 6.1 Open Aspects ..... 121
  - 6.2 Future Challenges ..... 123
- A List of Techniques ..... 125**
- B Publication Timeline ..... 131**
- C Graph Definitions ..... 133**
- D Taxonomy ..... 135**
  - D.1 Common Terms ..... 135
  - D.2 Machine Instruction Characteristics ..... 136
  - D.3 Scope ..... 137
  - D.4 Principles ..... 137
- References ..... 139**
- Index ..... 169**

# List of Figures

1.1	Source code example	2
1.2	Overview of a typical compiler infrastructure	4
1.3	The factorial function, expressed in assembly for a multi-issue MIPS architecture	5
2.1	Language transfer example using SIMCMP	14
2.2	Example of IR code represented as a program tree	15
2.3	A binary addition macro in ICL	16
2.4	MIML example	17
2.5	OMML example	18
2.6	Overview of the Davidson-Fraser approach	25
2.7	An extension of the Davidson-Fraser approach	26
2.8	An instruction expressed in $\lambda$ -RTL	26
3.1	An example program expressed using Wasilew's intermediate language	33
3.2	A machine description sample for PCC	34
3.3	A straightforward, tree-based pattern matching algorithm	35
3.4	Instruction set grammar example	40
3.5	State table generated from the grammar in Fig. 3.4	40
3.6	Execution walk-through of the Graham-Glanville approach	41
3.7	An instruction set expressed as an attribute grammar	44
3.8	Examples on how chain rules can be supported	50
3.9	A state machine for string matching	52
3.10	The Hoffmann-O'Donnell tree labeling algorithm	53
3.11	Tree pattern matching using Hoffmann-O'Donnell	54
3.12	The Hoffmann-O'Donnell preprocessing algorithms	56
3.13	An example of compressing the lookup table $T_a$	57
3.14	Rule samples for TWIG, written in CGL	58
3.15	Algorithms for optimal tree-based pattern selection using dynamic programming	59
3.16	Breaking down a pattern into single-node components	62

- 3.17 Table-driven algorithm for performing optimal tree-based pattern selection and code emission ..... 64
- 3.18 BURS example ..... 65
- 3.19 An example of state explosion ..... 66
- 3.20 Trellis diagram example ..... 75
  
- 4.1 Transforming SAT to DAG covering ..... 79
- 4.2 Undagging a program DAG with a common subexpression ..... 84
- 4.3 Sharing nonterminals of tree covering on a program DAG ..... 86
- 4.4 Converting a pattern DAG into partial (tree) patterns ..... 88
- 4.5 A conflict graph example ..... 89
- 4.6 A unate covering example ..... 92
- 4.7 The CO graph of a simple processor ..... 103
  
- 5.1 Time complexities for solving pattern matching and optimal pattern selection ..... 106
- 5.2 Example of converting a regular program into SSA form ..... 110
- 5.3 SSA graph example ..... 110
- 5.4 Converting a function into a PDG ..... 111
  
- B.1 Publication timeline ..... 131
  
- C.1 Example of two simple directed graphs ..... 134