Trustworthy Compilers

Vladimir O. Safonov
St. Petersburg University

WILEY
A John Wiley & Sons, Inc., Publication
# Contents

**Preface**

**Acknowledgments**

1. Introduction  
   1.1. The Concept of a Trustworthy Compiler / 2  
   1.2. Kinds of Compilers / 4  
   1.3. Evolution of Java Compilers / 5  
   1.4. Compilation for .NET / 6  
   1.5. Phases of Compilation / 7  
   1.6. Overview of Compiler Development Principles and Technologies / 8  
   1.7. History of Compiler Development in the U.S.S.R. and in Russia / 13  
       Exercises to Chapter 1 / 15

2. Theoretical Foundations and Principles of Trustworthy Compilers 16  
   2.1. The Trustworthy Computing (TWC) Initiative / 16  
   2.2. TWC and Trustworthy Compilers / 17  
   2.3. Verified Compilers / 24  
   2.4. Spec#: Microsoft's Approach to Verifying Compilers / 26  
   2.5. Perspectives of Verified and Verifying Compilation / 28  
       Exercises to Chapter 2 / 29
3. Lexical Analysis and Its Trustworthiness Principles 31

3.1. Token Classes / 31
3.2. The Output of the Lexical Analyzer / 33
3.3. Processing White Spaces, Comments, and New Lines / 34
3.4. Theoretical Models of Lexical Analysis / 35
3.5. Lexical Errors, Error Diagnostics, and Recovery / 38
3.6. Processing Identifiers and Keywords / 38
3.7. The Architecture of a Lexical Analyzer and the Principles of Its Implementation / 42
3.8. The Lexical Analyzer Generator *Lex* / 45
3.9. Lexical Analyzer Generation in ANTLR / 48
Exercises to Chapter 3 / 51

4. Parsing and Trustworthy Methods of Syntax Error Recovery 52

4.1. Basic Concepts and Principles of Parsing / 53
4.2. Recursive Descent and Simple Lookahead Mechanism / 55
4.3. Overview of Error Recovery in Parsing: Error Recovery for Recursive Descent / 62
4.4. LR(1) and LALR(1) Parsing / 67
4.5. Error Recovery in LR Parsing / 81
4.6. The *Yacc* Parser Generator / 82
4.7. The *Bison* Parser Generator: Generalized LR Parsing / 87
4.8. The *Yacc++*, *JavaCC*, *SableCC*, *ANTLR*, and *CoCo/R* Object-Oriented Parser Generators / 89
Exercises to Chapter 4 / 95

5. Semantic Analysis and Typing: Efficient and Trustworthy Techniques 97

5.1. Basic Concepts and Principles of Semantic Analysis / 97
5.3. Definition Systems with Forward References and the Algorithm of Their One-Pass Analysis / 103
5.4. Commonly Used Semantic Attributes for Program Constructs / 107
5.5. Design Flaws of the Semantic Attribute Evaluation and Our Efficient Methods to Speed It Up / 111
5.6. Lookup—Traditional and Novel Techniques / 114
5.7. Typing and Type-Checking: Basic Concepts / 118
5.8. Representing Types at Compile Time  /  121
5.9. Efficient Method and Algorithm to Represent and Handle Types with Structural Identity  /  123
5.10. Type Identity and Type Compatibility  /  126
5.11. Type-Checking, Typing Error Diagnostics, and Recovery  /  128
5.12. Code Trustworthiness Checks During Semantic Analysis  /  131
5.13. Checks for Context Restrictions in Semantic Analysis  /  139
5.15. Postfix (Reverse Polish) Notation  /  142
5.16. PCC Trees  /  146
5.17. Triples  /  149
5.18. Summary of the Chapter  /  150
Exercises to Chapter 5  /  151

6. Trustworthy Optimizations  

6.1. Basic Concepts and Trustworthiness of Optimizations  /  152
6.2. Optimizations as Mixed Computations  /  154
6.3. Overview of the Most Common Kinds of Optimizations  /  155
6.4. Control Flow and Data Flow Dependencies  /  162
6.5. Static Single Assignment (SSA)  /  163
6.6. Data Structures Constructed and Used by the Optimizer  /  165
6.7. Optimization in Sun Studio Compilers  /  165
6.8. Optimizations of the Java Bytecode  /  167
6.10. Optimizations during JIT Compilation  /  170
Exercises to Chapter 6  /  173

7. Code Generation and Runtime Data Representation  

7.1. Target Platforms for Code Generation  /  174
7.2. Overview of Code Generation Tasks and Goals  /  175
7.3. Specifics of Code Generation for .NET  /  179
7.4. Specifics of Code Generation for SPARC Architecture  /  180
7.5. Representing Types and Addressing Variables  /  181
7.6. Representing Procedures, Functions, and Methods  /  186
7.7. Principles of SPARC Architecture  /  190
7.8. Example of Code Generation for SPARC Architecture  /  192
7.9. Generation of Debugging Information  /  195
7.10. Code Generation for Declarations (Definitions), Expressions, and Statements / 197
Exercises to Chapter 7 / 199

8. Runtime, JIT, and AOT Compilation

8.1. The Tasks of the Runtime / 200
8.2. The Relationship of the Runtime and the Operating System (OS) / 202
8.3. JIT Compilation / 203
8.4. The Architecture of FJIT—JIT Compiler for SSCLI/Rotor / 211
8.5. The Architecture of Optimizing JIT Compiler for SSCLI/Rotor / 212
8.6. AOT Compilation / 220
Exercises to Chapter 8 / 221

9. Graph Grammars and Graph Compilers

9.1. Basic Concepts of Graph Grammars and Graph Compilers / 223
9.2. Categorical Approach to Graph Transformations / 226
9.3. Reserved Graph Grammars (RGGs) / 230
9.4. Layered Graph Grammars / 232
9.5. Meta-Modeling Approach to Graph Grammars and Diameta Editor / 233
9.6. Hypergraph Approach to Graph Grammars in Diagen / 235
9.7. Graph Compiler Generation Tools / 237
Exercises to Chapter 9 / 238

10. Microsoft Phoenix, Phoenix-Targeted Tools, and Our Phoenix Projects

10.1. History of Phoenix and of Our Phoenix Projects / 240
10.2. Overview of Phoenix Architecture / 242
10.3. Phoenix-Based Tools, Passes, Phases, and Plug-Ins / 246
10.4. Phoenix Primitives: Strings and Names / 247
10.5. Phoenix Intermediate Representation (IR) / 248
10.6. Phoenix Symbol System / 253
10.7. Phoenix Type System / 257
10.9. Overview of Other Phoenix Features / 264
10.10. Example of a Phoenix-Based Plug-In / 265
10.11. Phoenix-Fete—A Compiler Front-End Development Toolkit and Environment Targeted to Phoenix / 267

10.11.1. Architectural Specifics of Phoenix-FETE / 268

10.11.2. The Input Grammar Meta-Language / 269

10.11.3. The Current Status of the Implementation / 271

Exercises to Chapter 10 / 274

Conclusions 277

References 279

Index 285