THE AUTOMATIC GENERATION OF TEST PROGRAMS FOR NORMAL AND OPTIMISING FORTRAN COMPILERS

Contents

List of Figures viii

List of Tables ix

1 Introduction 1

1.1 The need for program testing . . . 1
1.2 Practical Software Testing . . . 3
1.3 The testing of a Compiler . . . 4
1.3.1 Design of Test Suites . . . 4
1.3.2 Automatic Generation of Test Cases . . . 5
1.3.3 The Correctness of Compiler Optimisation . . . 7

2 Techniques for the automatic generation of compiler test programs 9

2.1 The phases of a compiler . . . 9
2.2 Generation of compiler test programs . . . 10
2.3 Syntax Testing . . . 11
2.4 Compiler-Time Testing . . . 12
2.5 Code generation testing . . . 15

3 Testing the correctness of the code generation performed by a compiler 21

3.1 Choice of language . . . 21
3.2 Definition of the test programs . . . 22
3.3 Scope of the tests . . . 23
3.3.1 Features of the language . . . 23
3.3.2 Compiler optimisation . . . 25
3.4 Description of the attribute grammar . . . 28
3.4.1 Introduction . . . 28
3.4.2 Controlling the semantics . . . 32
3.4.3 The use of guards . . . 33
3.4.4 Repetition factor . . . 35

4 The generation of test programs for a Fortran compiler **37**
4.1 Testing strategy . . . 37
4.1.1 Ensuring successful compilation . . . 39
4.1.2 Avoiding run-time errors . . . 43
4.1.3 Design of the self-checking code . . . 44
4.2 Generation of normal programs . . . 55
4.2.1 Expressions . . . 55
4.2.2 Assignment statements . . . 64
4.2.3 Subprograms . . . 67
4.2.4 DO-loops . . . 70
4.2.5 Flow control constructions . . . 72
4.2.6 Other statements and features . . . 77

5 Generation of test programs for a Fortran optimising compiler **78**
5.1 Introduction . . . 78
5.2 Positioning of self-checking code . . . 79
5.3 Generation of features suitable for optimisation . . . 81
5.3.1 Designing test program allowing common subexpression elimination . . . 81
5.3.2 Generating test programs involving common subexpressions . 85
5.3.3 Copy propagation . . . 91
5.3.4 Constant folding . . . 93
5.3.5 Code motion . . . 96
5.3.6 Induction variable elimination and reduction in strength . . . 97
5.3.7 Dead code elimination . . . 103
5.3.8 Algebraic optimisations . . . 105

6 Design of the attribute grammar semantics for the generator **108**
6.1 Introduction . . . 108
6.2 Description of the data structures . . . 109
6.3 User-supplied information . . . 110
6.4 Identifier table and value arrays . . . 111
6.4.1 Identifier table . . . 111
6.4.2 Value arrays . . . 114
6.5 Data structures for the subprograms . . . 120
6.5.1 Subprogram table . . . 121
6.5.2 Parameter list . . . 123
6.5.3 Common variable table . . . 123
6.6 Data structures for the optimisation features . . . 123
6.7 The use of the data structures by the semantic routines . . . 126
6.7.1 The generation of identifier . . . 126
6.7.2 The generation of subprograms . . . 129
6.7.3 The generation of common subexpressions . . . 132

7 **Testing strategy** 134
7.1 The user interface to the generator 134
7.1.1 User-supplied declarations 134
7.1.2 Guiding the contexts of the test programs 137
7.1.3 Generating sequences of test programs 139
7.2 A description of a complete generated test program 142
7.3 The design of a comprehensive test suite for testing the whole of a compiler 145
7.3.1 Overall scheme 145
7.3.2 The levels of testing 147
7.3.3 The number of test programs used 150
7.4 Tailoring to a specific compiler 151

8 **Results obtained from the testing of Fortran compilers** 153
8.1 The testing of specific compilers 153
8.1.1 Introduction 153
8.1.2 Testing at the simple level 158
8.1.3 Testing at the user level 162
8.1.4 Testing at the Limit level 166
8.1.5 Modification of the test programs 169
8.1.6 Overall performance of the compilers 170
8.2 Experience with the system 172
8.3 Example of a test program which contains all the major program features for possible optimisations 175
8.3.1 Common subexpression elimination 178
8.3.2 Copy propagation 181
8.3.3 Constant folding 182
8.3.4 Algebraic optimisations 183
8.3.5 Loop optimisations 183
8.3.6 Dead code elimination 186
8.4 Performance of the system 188
8.4.1 Grammar pre-processor 188
8.4.2 The generator 188
8.5 Portability of the system 189

9 **Conclusions and extensions** 191
9.1 The achievements of the project 191
9.2 Overview of the effectiveness of the generator 191
9.3 The advantages and disadvantages of the automatic generation of test programs 193
9.3.1 Language features covered 193
9.3.2 Optimisation features covered 196
9.3.3 Practical use of the generator 199

9.4 Extensions and enhancements to the generator 200
9.5 Other uses of the generator 201
9.5.1 Testing other Fortran compilers on other operating systems. . 201
9.5.2 Testing other compilers 201
9.5.3 Testing other software 202
9.5.4 During compiler development 202

Appendix **203**

A Syntax of the attribute grammar **203**

B Job Control File **205**

C User-supplied declarations **210**

D User guide for compiler testing using the generator **219**

E Description of the generator **223**

F The attribute grammar for Fortran 77 used in the testing **226**

Bibliography **250**