Lecture Notes in
Computer Science

Edited by G. Goos and J. Hartmanis

95

Christopher D. Marlin

Coroutines

A Programming Methodology, a Language
Design and an Implementation

Springer-Verlag
Berlin Heidelberg New York 1980

Editorial Board

W. Brauer P. Brinch Hansen D. Gries C. Moler G. Seegmüller
J. Stoer N. Wirth

Author

Christopher D. Marlin
Department of Computer Science
101 MacLean Hall
The University of Iowa
Iowa City, Iowa 52242/USA

AMS Subject Classifications (1980): 68-02, 68B05, 68F20
CR Subject Classifications (1974): 4.0, 4.12, 4.20, 4.22

ISBN 3-540-10256-6 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-10256-6 Springer-Verlag New York Heidelberg Berlin

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically those of translation, reprinting, re-use of illustrations, broadcasting, reproduction by photocopying machine or similar means, and storage in data banks. Under § 54 of the German Copyright Law where copies are made for other than private use, a fee is payable to the publisher, the amount of the fee to be determined by agreement with the publisher.

© by Springer-Verlag Berlin Heidelberg 1980
Printed in Germany

Printing and binding: Beltz Offsetdruck, Hemsbach/Bergstr.
2145/3140-543210

PREFACE

Coroutines have been known and discussed for some years, but unfortunately have acquired a reputation for leading to poorly-structured and inefficient programs. It is perhaps a consequence of this unjustified reputation that coroutines are not widely available in programming languages.

The work described in this volume began both as an investigation of methodologies for programming with coroutines and as an attempt to extend the notion of hierarchical program structure to programs involving coroutines. The results of these efforts are presented in Chapter 2.

Inadequate support for hierarchically-structured systems of coroutines in existing languages then motivated the design of a language with coroutines. Although they are not widely available in implemented programming languages, coroutines have been described and discussed extensively in the literature, with a large number of proposals for the inclusion of coroutines in programming languages being put forward. The approach to language design described in Chapter 3 was born out of a desire to draw on the experience represented by this body of coroutine-related literature. This approach involves:

- the design of semantics before that of syntax,
- the division of the design of the semantics of a language into that of three largely orthogonal aspects of the language (data structures, sequence control, and data control), and
- the use of specific abstract models to aid the design of the semantics of each of these aspects, by facilitating comparisons among previous languages and proposals, and among competing design options for the language being designed.

The result of applying this approach to the design of a language with coroutines (known as ACL) is described in Chapters 4 (semantics) and 5 (syntax). This language was designed with relatively efficient implementation as one of its goals, and Chapter 6 describes some aspects of an implementation which has been carried out.

Apart from some minor corrections and editorial changes, this volume reproduces a thesis submitted by the author to the University of

Adelaide, Adelaide, South Australia, for the degree of Doctor of Philosophy, on 16th November 1979.

I gratefully acknowledge the support and encouragement of my supervisor, Dr.C.J.Barter. My thanks are also due to Dr.J.G.Sanderson for many helpful discussions, particularly while acting temporarily as my supervisor, and to the many other people who assisted me while I was carrying out the work described here. I would also like to thank Prof. D.L.Epley of the University of Iowa for his thoughtful advice and comments.

Above all, I am grateful to my wife, Deborah, for her constant support and for her unfailing confidence in my ability to finish the task I had set myself; she is also responsible for the careful preparation of the many diagrams in this volume.

Iowa City, Iowa C.D.M.
July 1980

TABLE OF CONTENTS

Page

LIST OF TABLES ix

LIST OF FIGURES x

Chapter

1. INTRODUCTION 1

   1.1 Coroutines 1

   1.2 Past Applications for Coroutines 6

   1.3 Coroutines in Programming Languages 7

2. PROGRAMMING WITH COROUTINES 9

   2.1 Aspects of Program Design 9

   2.2 Program Structure 10

   2.3 A Methodology for Programming with Coroutines 11

   2.4 An Example: The Telegrams Problem 14

3. A PROGRAMMING LANGUAGE WITH COROUTINES 21

   3.1 Introduction 21

   3.2 The Design Goals 22

   3.3 Programming Language Design 24

   3.4 The Choice of Pascal as the Base Language 27

4. THE DESIGN OF THE SEMANTICS OF THE LANGUAGE 29

   4.1 The Semantics of the Data Structures Aspect 29

   4.2 The Semantics of the Sequence Control Aspect 32

      4.2.1 Introduction 32

      4.2.2 The Sequence Control Model 33

Chapter Page

4.2.3 Sequence Control in Previous Coroutine Facilities 36

. Simula 36

. Gentleman's Portable Coroutine System 45

. Coroutine PASCAL 46

. 2.PAK 50

. SL5 52

. Krieg's Cooperations of Coprocedures 53

. Pritchard's Pools of Coroutines 54

. Sajaniemi's Cogroups 55

. Alphard and CLU 56

. TELOS 57

4.2.4 Sequence Control in ACL 58

4.2.4.1 Introduction 58

4.2.4.2 Extensions to the Sequence Control Model 60

4.2.4.3 The Sequence Control Operations 65

4.3 The Semantics of the Data Control Aspect 75

4.3.1 Data Control in Programming Languages 75

4.3.1.1 Introduction 75

4.3.1.2 Data Control and Storage Management 79

4.3.1.3 Block Structure 83

4.3.1.4 Scope Rules 85

4.3.1.5 Parameters and Function Values 87

4.3.2 The Data Control Model 94

4.3.3 Data Control in Previous Programming Languages 101

4.3.3.1 Introduction 101

4.3.3.2 Pascal 101

. Local Declarations 105

. Value Parameters 106

. Variable Parameters 109

. Procedure and Function Parameters 111

. Scope Rules 113

. Returning Values from Functions 117

. A Complete Example 119

4.3.3.3 Explicit Scope Rule Schemes 122

4.3.3.4 Previous Coroutine Facilities 125

Chapter Page

4.3.4 Data Control in ACL 128

4.3.4.1 Introduction 128

4.3.4.2 Scope Rules 130

. Local Declarations 134

. RO Inheriting Declarations 135

. RW Inheriting Declarations 138

4.3.4.3 Parameters 140

. RO Reference (Seen) Parameters 142

. RW Reference (Modifiable) Parameters 145

. Value Parameters 147

4.3.4.4 Returning Values from Subprograms 150

4.3.4.5 Summary, Restrictions and Disciplines 151

5. THE SYNTAX OF THE LANGUAGE 157

5.1 Introduction 157

5.2 Declarations 157

5.2.1 Defining Declarations 158

5.2.2 Inheriting Declarations 159

5.2.3 Forward Declarations 159

5.2.4 A Difficulty of Pascal Avoided in ACL 162

5.3 Parameters 164

5.4 Statements 165

5.5 Predefined Procedures and Functions 166

6. THE IMPLEMENTATION OF THE LANGUAGE 169

6.1 Overview 169

6.2 Declarations 172

6.3 Statements 175

6.4 Storage Management 182

7. CONCLUSIONS 191

7.1 The Programming Methodology 191

Chapter Page

7.2 The Language Design 191

7.3 The Implementation 195

APPENDICES 199

Appendix A: Syntax Diagrams for ACL 199

Appendix B: Some ACL Programs 207

B.1 The Telegrams Problem 207

B.2 The Odd Word Reversal Problem 212

B.3 Hamming's Problem 218

B.4 Lynning's Solution to Grune's Problem 224

B.5 A Data Abstraction Example 226

REFERENCES 229

INDEX 242

LIST OF TABLES

Table Page

2.1 Caller-Callee Relationships in Figure 2.1 20

4.1 A Comparison of Sequence Control in Simula and 2.PAK 52

4.2 A Summary of the Sequence Control Operations of ACL 73

4.3 The Characteristics of Various Static Scope Rule Schemes 123

4.4 A Summary of the Data Control Events for ACL 152

5.1 Predefined Procedures in ACL Requiring RW Access to Actual Parameters 168

B.1 Names in Figure B.1 Corresponding to Nodes in the Discussion of Chapter 2 210

LIST OF FIGURES

Figure | Page

2.1 The Structure of the Solution to the Telegrams Problem 19

4.1 An Example of a Dynamic Hierarchy 37

4.2 The Simula Text Corresponding to the Dynamic Hierarchy of Figure 4.1 38

4.3 The Tree of Instances in an Executing Simula Program 42

4.4 Some Typical Cycles of Instances in Coroutine PASCAL Programs 48

4.5 An Example of a Master Tree 61

4.6 The Algorithm for the Computation of "live(i)" 62

4.7 The Algorithm for the Computation of "susp(i_1,i_2)" 63

4.8 The Effect of an Instance Creation Operation on the Master Tree of Figure 4.5 67

4.9 The Effect of a Generator Call Operation on the Master Tree of Figure 4.5 70

4.10 Wegner's Binding Diagram 75

4.11 A Temporal Partial Ordering on Events Concerned With a Variable 83

4.12 Two Algol 60 Fragments with the Same Data Control Structure 83

4.13 An Example of the Use of Parameters to Construct Specialized Data Control Structures 90

4.14 A Pascal Fragment Containing a Function Parameter 91

4.15 Avoiding Violations of the Principle of Disjointness Which were Due, in part, to Access via Scope Rules 93

4.16 The Pictorial Representation of Block Instances in Depictions of the Data Control Structure of Programs 96

4.17 Transmission of Access via an Intermediary Identifier 98

4.18 Allowable Transmissions of Access to a Known Identifier 99

4.19 A Pascal Fragment Illustrating Value Parameters 109

4.20 Data Control Structures Occurring during the Execution of

Figure Page

the Fragment of Figure 4.19 109

4.21 A Pascal Fragment Illustrating Variable Parameters 110

4.22 Data Control Structures Occurring during the Execution of the Fragment of Figure 4.21 111

4.23 A Pascal Fragment Illustrating Procedure and Function Parameters 112

4.24 Data Control Structures Occurring during the Execution of the Fragment of Figure 4.23 112

4.25 The Partial Ordering on the Events Comprising the Data Control Effect of Block Entry in Pascal 113

4.26 A Pascal Fragment Illustrating the Scope Rules 115

4.27 Data Control Structures Occurring during the Execution of the Fragment of Figure 4.26 116

4.28 Examples of Pascal Functions which Cannot Return a Value 117

4.29 A Pascal Fragment Containing a Function 119

4.30 Data Control Structures Occurring during the Execution of the Fragment of Figure 4.29 119

4.31 A Complete Pascal Program 120

4.32 Data Control Structures Occurring during the Execution of the Program of Figure 4.31 121

4.33 A Program Fragment Illustrating an Amomaly with Scalar Types in Pascal 136

4.34 Data Control Structures Illustrating the Effect of Local and Inheriting Declarations in ACL 139

4.35 A Simula Fragment Illustrating the Establishment of Mutual References between Instances 144

4.36 Data Control Structures Illustrating the Establishment of Mutual References between Instances in ACL 145

4.37 Data Control Structures Illustrating the Rebinding of Continuation Parameters in ACL 149

4.38 An Example of a Generator not Exhibiting Procedure-like Behaviour 154

5.1 Specifying Recursively-defined Data Types in ACL and Pascal 161

5.2 Specifying Mutually Recursive Procedures in ACL and Pascal 162

Figure Page

5.3 Two Pascal Fragments Illustrating Situations Subject to Interpretation 163

6.1 Steps in the Development of an ACL Processor from Pascal`H′ 171

6.2 The Layouts of the Various Kinds of Heap Object 177

6.3 An Example of a Heap Object and its Description List 186

B.1 A Solution to the Telegrams Problem 208

B.2 The Structure of the Program in Figure B.1 after the Initialization of its Instances 211

B.3 Barter's Solution to the Odd Word Reversal Problem 214

B.4 Another Solution to the Odd Word Reversal Problem 216

B.5 The Sequence Control Structures of the Two Solutions to the Odd Word Reversal Problem 217

B.6 Dijkstra's Solution of Hamming's Problem 219

B.7 Transforming Dijkstra's Solution of Hamming's Problem 220

B.8 Another Solution to Hamming's Problem 222

B.9 Lynning's Solution to Grune's Problem 225

B.10 The Stack Abstraction in ACL 228