J.-P. Banâtre S. B. Jones D. Le Métayer

# Prospects for Functional Programming in Software Engineering

With the cooperation of P. Fradet and A. Sinclair

Springer-Verlag
Berlin Heidelberg New York London Paris
Tokyo Hong Kong Barcelona Budapest

# CONTENTS

**Chapter 1 About programming environments** 1

1.1 Introduction . . . . . 1

1.2 The software development process . . . . . 1

1.3 Characteristics of programming environments . . . . . 3

1.3.1 What should be expected from a programming environment? . . . . . 3

1.3.2 An example of a programming environment: the CEDAR system . . . . . 4

1.4 The choice of the appropriate specification language . . . . . 7

1.4.1 The VDM specification language . . . . . 8

1.4.2 Other approaches to specification . . . . . 11

1.5 Refinement of the specification . . . . . 13

1.5.1 Program development in a conventional language . . . . . 14

1.5.2 Program development in a functional language . . . . . 15

1.6 Towards a functional programming environment . . . . . 17

1.6.1 The choice of a language . . . . . 17

1.6.2 Current research on functional programming environments . . . . . 18

1.7 Overview of the book . . . . . 19

References . . . . . 21

**Chapter 2 Aspects of functional programming** 23

2.1 Introduction . . . . . 23

2.2 The main characteristics of functional languages . . . . . 23

2.2.1 Absence of explicit sequencing . . . . . 24

2.2.2 Absence of the assignment statement . . . . . 24

2.2.3 Referential transparency . . . . . 25

2.2.4 Functions: the essence of functional programming . . . . . 25

2.3 Presentation of two widespread functional languages . . . . . 30

2.3.1 LISP . . . . . 30

2.3.2 ML . . . . . 33

2.4 Brief survey of implementation techniques. . . . . . 36

2.4.1 The SECD machine . . . . . 36

2.4.2 Graph reduction . . . . . 37

2.5 Why is functional programming relevant? . . . . . 37

2.5.1 Semantic simplicity . . . . . 38

2.5.2 Correctness aspects . . . . . 40

2.5.3 Program transformation . . . . . 42

2.6 The reference language . . . . . 42

References . . . . . 44

**Chapter 3 Program analysis by abstract interpretation** 45

3.1 The need for a method of program analysis . . . . . 45

3.2 The basic principles of abstract interpretation . . . . . 47

3.2.1 Standard semantics . . . . . 48

3.2.2 Non-standard semantics and abstract interpretation . . . . . 49

3.2.3 Application of the abstract interpretation . . . . . 54

3.3 Restricting the interpretation to finite domains . . . . . 56

References . . . . . 57

**Chapter 4 Compile-time garbage collection by sharing analysis** 59

4.1 Introduction . . . . . 59

4.2 The target for optimization . . . . . 60

4.3 Language syntax and semantics . . . . . 63

4.4 Detection of sharing in functional expressions . . . . . 65

4.5 Sharing based garbage collection . . . . . 74

4.6 Finite domains and compile-time garbage collection . . . . . 80

4.7 Comparison with previous work . . . . . 84

References . . . . . 86

**Chapter 5 Analysis of functional programs by program transformation** 87

5.1 Introduction . . . . . 87

5.2 Survey of program transformation techniques . . . . . 88

5.2.1 Program optimization . . . . . 89

5.2.2 Program synthesis . . . . . 92

5.2.3 Program compilation . . . . . 92

5.2.4 Proof of properties of programs . . . . . 93

5.3 Program analysis by program transformation . . . . . 95

5.4 Time complexity analysis in a first order language . . . . . 98

5.5 Extension of the analysis to higher order languages . . . . . 102

5.6 A measure of the potential parallelism of programs . . . . . 105

5.7 Evaluation of the space complexity of programs . . . . . 107

5.8 ACE: An automatic complexity evaluator . . . . . 111

5.8.1 Organization of the system . . . . . 111

5.8.2 Using ACE . . . . . 112

5.8.3 Some results of analysis . . . . . 113

5.8.4 Discussion . . . . . 117

5.9 Conclusion . . . . . 118

References . . . . . 119

**Chapter 6 From lambda calculus to machine code by program transformation** 121

6.1 Introduction ........ 121

6.2 Compilation of the computation rule ........ 122

6.3 Compilation of environment management ........ 126

6.4 Conclusion ........ 133

References ........ 136

**Chapter 7 On input and output in functional languages** 139

7.1 Introduction ........ 139

7.1.1 Why is there a problem? ........ 139

7.1.2 A historical perspective ........ 141

7.2 Lazy evaluation and streams for input and output ........ 142

7.2.1 Lazy evaluation, streams and processes ........ 142

7.2.2 Interactive programs ........ 147

7.3 An interactive text editor ........ 149

7.4 Repackaging streams: higher order functions and continuations ........ 151

7.4.1 A basic set of interface functions ........ 151

7.4.2 The text editor: an improved version ........ 153

7.4.3 Generalizing the interface functions ........ 155

7.5 Process networks, parallelism and non-determinism ........ 155

7.5.1 Process networks ........ 156

7.5.2 Parallelism ........ 157

7.5.3 Non-determinism ........ 158

7.6 An operating system structure ........ 160

7.6.1 The network design ........ 160

7.6.2 Low level disk control, and a simple file system ........ 165

7.7 Discussion ........ 168

References ........ 170

**Chapter 8 For imperative programmers** 173

8.1 Introduction ........ 173

8.2 Denotational semantics ........ 174

8.3 Correctness considerations ........ 178

8.3.1 The imperative case ........ 179

8.3.2 The functional case ........ 185

8.3.3 Comparison of the imperative paradigm and the functional paradigm ........ 189

8.4 Efficiency considerations ........ 191

8.5 History-sensitive programming in a purely functional language ........ 195

8.6 Related works ........ 205

8.7 Functional programming is not a mirage ........ 207

References ........ 209