Trends in Functional Programming

Volume 4

Edited by

Stephen Gilmore

intellect™
Bristol, UK
Portland, OR, USA

Contents

1 Is It Time for Real-Time Functional Programming? 1
1.1 Introduction 1
1.2 What is Real-Time Programming? 2
1.2.1 The Importance of Real-Time Systems 2
1.2.2 Essential Properties of Real-Time Languages. 2
1.3 Languages for Programming Real-Time Systems 3
1.3.1 Using General Purpose Languages for Real-Time Programming 3
1.3.2 Domain-Specific Languages for Real-Time Programming 4
1.3.3 Functional Language Approaches 5
1.4 Bounding Time and Space Usage 7
1.4.1 Real-Time Dynamic Memory Management 7
1.4.2 Static Analyses for Bounding Memory Usage 7
1.4.3 Worst Case Execution Time Analysis. 8
1.4.4 Syntactically Restricted Functional Languages 9
1.5 Functional Languages for Related Problem Areas 9
1.6 The Hume Language 10
1.6.1 Real Time and Space Behaviour of FSM-Hume Programs 12
1.7 The Challenges 13
1.8 Conclusion 14

2 FSM-Hume is Finite State 19
2.1 Introduction 19
2.2 Single Box FSM-Hume Programs are Finite State 22
2.3 Multi-Box FSM-Hume Programs are Finite State 23
2.4 Example: Vehicle Simulation 25
2.4.1 Single-box FSM-Hume 26
2.5 Conclusion 28

3 Camelot and Grail: Resource-Aware Functional Programming for the JVM 29
3.1 Introduction 29
3.2 Camelot 30

3.2.1 Basic Features of Camelot 31
3.2.2 Diamonds and Resource Control 32
3.3 Grail 35
3.3.1 The Grail Type System 36
3.3.2 Compilation of Grail 36
3.4 Compiling Camelot to Grail 38
3.4.1 Representing Data 38
3.4.2 Compilation of Programs 39
3.4.3 Initial Transformations 40
3.4.4 Compilation of Expressions 41
3.5 Performance 41
3.6 Final Remarks 44

4 O'Camelot: Adding Objects to a Resource-Aware Functional Language 47
4.1 Introduction 47
4.2 Camelot 48
4.3 Extensions 49
4.4 Typing 53
4.5 Translation 55
4.6 Objects and Resource Types 57
4.7 Related Work 58
4.8 Conclusion 59

5 Static Single Information from a Functional Perspective 63
5.1 Introduction 63
5.2 Related Work 67
5.3 Static Single Information 68
5.4 Transformation 69
5.5 Optimistic versus Pessimistic 71
5.6 Converting Functional Programs Back to SSI 72
5.7 Motivation 73
5.8 Conclusions 74

6 Implementing Mobile Haskell 79
6.1 Introduction 79
6.2 Mobile Haskell 81
6.2.1 Communication Primitives 81
6.2.2 Discovering Resources 82
6.2.3 Remote Thread Creation 83
6.2.4 A Simple Example 83
6.3 Implementation Design 83
6.3.1 Introduction 83
6.3.2 Evaluating Expressions before Communication 84
6.3.3 Sharing Properties 85

6.3.4 MChannels 86
6.4 The Implementation 86
6.4.1 Packing Routines 86
6.4.2 Communicating User Defined Types 87
6.4.3 Evaluating Expressions 88
6.4.4 Implementation of MChannels 89
6.5 Initial Evaluation 90
6.6 Related Work 91
6.7 Conclusions and Future Work 92

7 Testing Scheme Programming Assignments Automatically 95
7.1 Introduction 95
7.2 WebAssign and AT(x) 97
7.3 A Sample Session 98
7.4 Structure of the AT(x) Framework 100
7.4.1 Components of the AT(x) System 100
7.4.2 Communication Interface of the Analysis Component . . . 101
7.4.3 Function and Implementation of the Interface Component 101
7.4.4 Global Security Issues 103
7.5 The Core Analysis Component 104
7.5.1 Requirements on the Analysis Components 104
7.5.2 Analysis of Scheme Programs 106
7.6 Implementation and Experiences 107
7.7 Related Work 108
7.8 Conclusions and Further Work 109

8 Testing Reactive Systems with GAST 111
8.1 Introduction 111
8.2 Overview of G∀ST 112
8.2.1 Testing and Results 113
8.2.2 Evaluating Test Results 113
8.2.3 Logical Operators in G∀ST 114
8.2.4 Automatic Generation of Test Values 114
8.3 Specifying Reactive Systems in G∀ST 115
8.3.1 Labelled Transition Systems 116
8.3.2 Example: Conference Protocol 117
8.3.3 Executing a Deterministic LTS 118
8.3.4 The Implementation Under Test 120
8.3.5 Testing the Conference Protocol 120
8.3.6 Implementations with Other Types 121
8.4 Better Test Data Generation from the LTS 121
8.5 Functional and Nondeterministic Specifications 123
8.6 Testing Nondeterministic Systems 125
8.7 Related Work 126
8.8 Conclusion 127

PREFACE

This volume is the proceedings of the Fourth International Symposium on Trends in Functional Programming held in Edinburgh, on September 11th and 12th, 2003. For the first time this year the TFP symposium was co-located with the Implementation of Functional Languages workshop.

The Trends in Functional Programming series occupies a unique place in the spectrum of functional programming events because of its highly commendable policy of encouraging new speakers, particularly PhD students, to air their work to a receptive and friendly audience. By encouraging the next generation of functional programmers in this way the workshop helps to instill the understanding that functional programming is more than just syntax, semantics and type systems and nourishes the essence of the subject itself.

This year the papers from the workshop have addressed the research problems at the forefront of practical application of functional languages as in the papers on real-time functional programming in Hume from Kevin Hammond, Greg Michaelson and Jocelyn Serot and resource-bounded functional programming in Camelot from Kenneth MacKenzie and Nicholas Wolverson.

Functional programming languages are supported by sophisticated implementations. Two papers address this aspect of functional programming research, Jeremy Singer's paper on static single information and the paper on the implementation of Mobile Haskell from André Rauber Du Bois, Phil Trinder and Hans-Wolfgang Loidl.

For all of their virtues, functional programs are not automatically error-free so the book closes with two papers on testing functional programs from Manfred Widera and from Pieter Koopman and Rinus Plasmeijer.

I would like to thank the organisers of IFL, Abyd Al Zain, André Rauber Du Bois, June Maxwell, Greg Michaelson, Jan Henry Nyström and Phil Trinder for their work in organising the workshop registrations, the excursion, delegate packs, room bookings, audio-visuals and many other aspects of the event and for allowing the TFP meeting to make use of their industriousness in making all of this run smoothly.

My thanks also go to all of the authors for preparing their papers carefully using Hans-Wolfgang Loidl's LaTeX style file and to the referees for their thorough and rapid reviewing of the papers which were submitted.

The Trends in Functional Programming workshop gratefully acknowledges the support of the British Computer Society Formal Aspects of Computer Science special interest group.

BCS FACS

Stephen Gilmore,
Edinburgh