

Masami Hagiya  
Philip Wadler (Eds.)

LNCS 3945

# Functional and Logic Programming

8th International Symposium, FLOPS 2006  
Fuji-Susono, Japan, April 2006  
Proceedings



Springer

TP311.1-53

F979 Masami Hagiya Philip Wadler (Eds.)

2006

# Functional and Logic Programming

8th International Symposium, FLOPS 2006  
Fuji-Susono, Japan, April 24-26, 2006  
Proceedings



Springer



E200603553

## Volume Editors

Masami Hagiya  
University of Tokyo  
and NTT Communication Science Laboratories  
Department of Computer Science  
Graduate School of Information Science and Technology  
E-mail: hagiya@is.s.u-tokyo.ac.jp

Philip Wadler  
University of Edinburgh  
Department of Informatics  
James Clerk Maxwell Building, The King's Buildings  
Mayfield Road, Edinburgh EH9 3JZ, UK  
E-mail: wadler@inf.ed.ac.uk

Library of Congress Control Number: 2006923563

CR Subject Classification (1998): D.1.6, D.1, D.3, F.3, I.2.3

LNCS Sublibrary: SL 2 – Programming and Software Engineering

ISSN 0302-9743  
ISBN-10 3-540-33438-6 Springer Berlin Heidelberg New York  
ISBN-13 978-3-540-33438-5 Springer Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

Springer is a part of Springer Science+Business Media  
springer.com

© Springer-Verlag Berlin Heidelberg 2006  
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India  
Printed on acid-free paper SPIN: 11737414 06/3142 5 4 3 2 1 0

*Commenced Publication in 1973*

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

## Editorial Board

David Hutchison

*Lancaster University, UK*

Takeo Kanade

*Carnegie Mellon University, Pittsburgh, PA, USA*

Josef Kittler

*University of Surrey, Guildford, UK*

Jon M. Kleinberg

*Cornell University, Ithaca, NY, USA*

Friedemann Mattern

*ETH Zurich, Switzerland*

John C. Mitchell

*Stanford University, CA, USA*

Moni Naor

*Weizmann Institute of Science, Rehovot, Israel*

Oscar Nierstrasz

*University of Bern, Switzerland*

C. Pandu Rangan

*Indian Institute of Technology, Madras, India*

Bernhard Steffen

*University of Dortmund, Germany*

Madhu Sudan

*Massachusetts Institute of Technology, MA, USA*

Demetri Terzopoulos

*University of California, Los Angeles, CA, USA*

Doug Tygar

*University of California, Berkeley, CA, USA*

Moshe Y. Vardi

*Rice University, Houston, TX, USA*

Gerhard Weikum

*Max-Planck Institute of Computer Science, Saarbruecken, Germany*

# Lecture Notes in Computer Science

For information about Vols. 1–3840

please contact your bookseller or Springer

Vol. 3946: T.R. Roth-Berghofer, S. Schulz, D.B. Leake (Eds.), *Modeling and Retrieval of Context*. XI, 149 pages. 2006. (Sublibrary LNAI).

Vol. 3945: M. Hagiya, P. Wadler (Eds.), *Functional and Logic Programming*. X, 295 pages. 2006.

Vol. 3939: C. Priami, L. Cardelli, S. Emmott (Eds.), *Transactions on Computational Systems Biology IV*. VII, 141 pages. 2006. (Sublibrary LNBI).

Vol. 3936: M. Lalmas, A. MacFarlane, S. Rüger, A. Tombros, T. Tsikrika, A. Yavlinsky (Eds.), *Advances in Information Retrieval*. XI, 584 pages. 2006.

Vol. 3934: J.A. Clark, R.F. Paige, F.A. C. Polack, P.J. Brooke (Eds.), *Security in Pervasive Computing*. X, 243 pages. 2006.

Vol. 3933: F. Bonchi, J.-F. Boulicaut (Eds.), *Knowledge Discovery in Inductive Databases*. VIII, 251 pages. 2006.

Vol. 3931: B. Apolloni, M. Marinaro, G. Nicosia, R. Tagliaferri (Eds.), *Neural Nets*. XIII, 370 pages. 2006.

Vol. 3930: D.S. Yeung, Z.-Q. Liu, X.-Z. Wang, H. Yan (Eds.), *Advances in Machine Learning and Cybernetics*. XXI, 1110 pages. 2006. (Sublibrary LNAI).

Vol. 3928: J. Domingo-Ferrer, J. Posegga, D. Schreckling (Eds.), *Smart Card Research and Advanced Applications*. XI, 359 pages. 2006.

Vol. 3927: J. Hespanha, A. Tiwari (Eds.), *Hybrid Systems: Computation and Control*. XII, 584 pages. 2006.

Vol. 3925: A. Valmari (Ed.), *Model Checking Software*. X, 307 pages. 2006.

Vol. 3924: P. Sestoft (Ed.), *Programming Languages and Systems*. XII, 343 pages. 2006.

Vol. 3923: A. Mycroft, A. Zeller (Eds.), *Compiler Construction*. XIII, 277 pages. 2006.

Vol. 3922: L. Baresi, R. Heckel (Eds.), *Fundamental Approaches to Software Engineering*. XIII, 427 pages. 2006.

Vol. 3921: L. Aceto, A. Ingólfsdóttir (Eds.), *Foundations of Software Science and Computation Structures*. XV, 447 pages. 2006.

Vol. 3920: H. Hermanns, J. Palsberg (Eds.), *Tools and Algorithms for the Construction and Analysis of Systems*. XIV, 506 pages. 2006.

Vol. 3918: W.K. Ng, M. Kitsuregawa, J. Li, K. Chang (Eds.), *Advances in Knowledge Discovery and Data Mining*. XXIV, 879 pages. 2006. (Sublibrary LNAI).

Vol. 3917: H. Chen, F.Y. Wang, C.C. Yang, D. Zeng, M. Chau, K. Chang (Eds.), *Intelligence and Security Informatics*. XII, 186 pages. 2006.

Vol. 3916: J. Li, Q. Yang, A.-H. Tan (Eds.), *Data Mining for Biomedical Applications*. VIII, 155 pages. 2006. (Sublibrary LNBI).

Vol. 3915: R. Nayak, M.J. Zaki (Eds.), *Knowledge Discovery from XML Documents*. VIII, 105 pages. 2006.

Vol. 3914: A. Garcia, R. Choren, C. Lucena, P. Giorgini, T. Holvoet, A. Romanovsky (Eds.), *Software Engineering for Multi-Agent Systems IV*. XIV, 255 pages. 2006.

Vol. 3910: S.A. Brueckner, G.D.M. Serugendo, D. Hales, F. Zambonelli (Eds.), *Engineering Self-Organising Systems*. XII, 245 pages. 2006. (Sublibrary LNAI).

Vol. 3909: A. Apostolico, C. Guerra, S. Istrail, P. Pevzner, M. Waterman (Eds.), *Research in Computational Molecular Biology*. XVII, 612 pages. 2006. (Sublibrary LNBI).

Vol. 3907: F. Rothlauf, J. Branke, S. Cagnoni, E. Costa, C. Cotta, R. Drechsler, E. Lutton, P. Machado, J.H. Moore, J. Romero, G.D. Smith, G. Squillero, H. Takagi (Eds.), *Applications of Evolutionary Computing*. XXIV, 813 pages. 2006.

Vol. 3906: J. Gottlieb, G.R. Raidl (Eds.), *Evolutionary Computation in Combinatorial Optimization*. XI, 293 pages. 2006.

Vol. 3905: P. Collet, M. Tomassini, M. Ebner, S. Gustafson, A. Ekárt (Eds.), *Genetic Programming*. XI, 361 pages. 2006.

Vol. 3904: M. Baldoni, U. Endriss, A. Omicini, P. Torroni (Eds.), *Declarative Agent Languages and Technologies III*. XII, 245 pages. 2006. (Sublibrary LNAI).

Vol. 3903: K. Chen, R. Deng, X. Lai, J. Zhou (Eds.), *Information Security Practice and Experience*. XIV, 392 pages. 2006.

Vol. 3901: P.M. Hill (Ed.), *Logic Based Program Synthesis and Transformation*. X, 179 pages. 2006.

Vol. 3899: S. Frintrop, *VOCUS: A Visual Attention System for Object Detection and Goal-Directed Search*. XIV, 216 pages. 2006. (Sublibrary LNAI).

Vol. 3898: K. Tuyls, P.J. 't Hoen, K. Verbeeck, S. Sen (Eds.), *Learning and Adaption in Multi-Agent Systems*. X, 217 pages. 2006. (Sublibrary LNAI).

Vol. 3897: B. Preneel, S. Tavares (Eds.), *Selected Areas in Cryptography*. XI, 371 pages. 2006.

Vol. 3896: Y. Ioannidis, M.H. Scholl, J.W. Schmidt, F. Matthes, M. Hatzopoulos, K. Boehm, A. Kemper, T. Grust, C. Boehm (Eds.), *Advances in Database Technology - EDBT 2006*. XIV, 1208 pages. 2006.

Vol. 3895: O. Goldreich, A.L. Rosenberg, A.L. Selman (Eds.), *Theoretical Computer Science*. XII, 399 pages. 2006.

Vol. 3894: W. Grass, B. Sick, K. Waldschmidt (Eds.), *Architecture of Computing Systems - ARCS 2006*. XII, 496 pages. 2006.

Vol. 3893: L. Atzori, D.D. Giusto, R. Leonardi, F. Pereira (Eds.), *Visual Content Processing and Representation*. X, 224 pages. 2006.

Vol. 3891: J.S. Sichman, L. Antunes (Eds.), Multi-Agent-Based Simulation VI. X, 191 pages. 2006. (Sublibrary LNAI).

Vol. 3890: S.G. Thompson, R. Ghanea-Hercock (Eds.), Defence Applications of Multi-Agent Systems. XII, 141 pages. 2006. (Sublibrary LNAI).

Vol. 3889: J. Rosca, D. Erdoğmus, J.C. Principe, S. Haykin (Eds.), Independent Component Analysis and Blind Signal Separation. XXI, 980 pages. 2006.

Vol. 3888: D. Draheim, G. Weber (Eds.), Trends in Enterprise Application Architecture. IX, 145 pages. 2006.

Vol. 3887: J.R. Correa, A. Hevia, M. Kiwi (Eds.), LATIN 2006: Theoretical Informatics. XVI, 814 pages. 2006.

Vol. 3886: E.G. Bremer, J. Hakenberg, E.-H.(S.) Han, D. Berrar, W. Dubitzky (Eds.), Knowledge Discovery in Life Science Literature. XIV, 147 pages. 2006. (Sublibrary LNBI).

Vol. 3885: V. Torra, Y. Narukawa, A. Valls, J. Domingo-Ferrer (Eds.), Modeling Decisions for Artificial Intelligence. XII, 374 pages. 2006. (Sublibrary LNAI).

Vol. 3884: B. Durand, W. Thomas (Eds.), STACS 2006. XIV, 714 pages. 2006.

Vol. 3882: M.L. Lee, K.L. Tan, V. Wuwongse (Eds.), Database Systems for Advanced Applications. XIX, 923 pages. 2006.

Vol. 3881: S. Gibet, N. Courty, J.-F. Kamp (Eds.), Gesture in Human-Computer Interaction and Simulation. XIII, 344 pages. 2006. (Sublibrary LNAI).

Vol. 3880: A. Rashid, M. Aksit (Eds.), Transactions on Aspect-Oriented Software Development I. IX, 335 pages. 2006.

Vol. 3879: T. Erlebach, G. Persinao (Eds.), Approximation and Online Algorithms. X, 349 pages. 2006.

Vol. 3878: A. Gelbukh (Ed.), Computational Linguistics and Intelligent Text Processing. XVII, 589 pages. 2006.

Vol. 3877: M. Detyniecki, J.M. Jose, A. Nürnberger, C. J. ' van Rijsbergen (Eds.), Adaptive Multimedia Retrieval: User, Context, and Feedback. XI, 279 pages. 2006.

Vol. 3876: S. Halevi, T. Rabin (Eds.), Theory of Cryptography. XI, 617 pages. 2006.

Vol. 3875: S. Ur, E. Bin, Y. Wolfsthal (Eds.), Hardware and Software, Verification and Testing. X, 265 pages. 2006.

Vol. 3874: R. Missaoui, J. Schmidt (Eds.), Formal Concept Analysis. X, 309 pages. 2006. (Sublibrary LNAI).

Vol. 3873: L. Maicher, J. Park (Eds.), Charting the Topic Maps Research and Applications Landscape. VIII, 281 pages. 2006. (Sublibrary LNAI).

Vol. 3872: H. Bunke, A. L. Spitz (Eds.), Document Analysis Systems VII. XIII, 630 pages. 2006.

Vol. 3870: S. Spaccapietra, P. Atzeni, W.W. Chu, T. Catarci, K.P. Sycara (Eds.), Journal on Data Semantics V. XIII, 237 pages. 2006.

Vol. 3869: S. Renals, S. Bengio (Eds.), Machine Learning for Multimodal Interaction. XIII, 490 pages. 2006.

Vol. 3868: K. Römer, H. Karl, F. Mattern (Eds.), Wireless Sensor Networks. XI, 342 pages. 2006.

Vol. 3866: T. Dimitrakos, F. Martinelli, P.Y.A. Ryan, S. Schneider (Eds.), Formal Aspects in Security and Trust. X, 259 pages. 2006.

Vol. 3865: W. Shen, K.-M. Chao, Z. Lin, J.-P.A. Barthès, A. James (Eds.), Computer Supported Cooperative Work in Design II. XII, 659 pages. 2006.

Vol. 3863: M. Kohlhase (Ed.), Mathematical Knowledge Management. XI, 405 pages. 2006. (Sublibrary LNAI).

Vol. 3862: R.H. Bordini, M. Dastani, J. Dix, A.E.F. Seghrouchni (Eds.), Programming Multi-Agent Systems. XIV, 267 pages. 2006. (Sublibrary LNAI).

Vol. 3861: J. Dix, S.J. Hegner (Eds.), Foundations of Information and Knowledge Systems. X, 331 pages. 2006.

Vol. 3860: D. Pointcheval (Ed.), Topics in Cryptology – CT-RSA 2006. XI, 365 pages. 2006.

Vol. 3858: A. Valdes, D. Zamboni (Eds.), Recent Advances in Intrusion Detection. X, 351 pages. 2006.

Vol. 3857: M.P.C. Fossorier, H. Imai, S. Lin, A. Poli (Eds.), Applied Algebra, Algebraic Algorithms and Error-Correcting Codes. XI, 350 pages. 2006.

Vol. 3855: E. A. Emerson, K.S. Namjoshi (Eds.), Verification, Model Checking, and Abstract Interpretation. XI, 443 pages. 2005.

Vol. 3854: I. Stavrakakis, M. Smirnov (Eds.), Autonomic Communication. XIII, 303 pages. 2006.

Vol. 3853: A.J. Ijspeert, T. Masuzawa, S. Kusumoto (Eds.), Biologically Inspired Approaches to Advanced Information Technology. XIV, 388 pages. 2006.

Vol. 3852: P.J. Narayanan, S.K. Nayar, H.-Y. Shum (Eds.), Computer Vision – ACCV 2006, Part II. XXXI, 977 pages. 2006.

Vol. 3851: P.J. Narayanan, S.K. Nayar, H.-Y. Shum (Eds.), Computer Vision – ACCV 2006, Part I. XXXI, 973 pages. 2006.

Vol. 3850: R. Freund, G. Păun, G. Rozenberg, A. Salomaa (Eds.), Membrane Computing. IX, 371 pages. 2006.

Vol. 3849: I. Bloch, A. Petrosino, A.G.B. Tettamanzi (Eds.), Fuzzy Logic and Applications. XIV, 438 pages. 2006. (Sublibrary LNAI).

Vol. 3848: J.-F. Boulicaut, L. De Raedt, H. Mannila (Eds.), Constraint-Based Mining and Inductive Databases. X, 401 pages. 2006. (Sublibrary LNAI).

Vol. 3847: K.P. Jantke, A. Lunzer, N. Spyrtas, Y. Tanaka (Eds.), Federation over the Web. X, 215 pages. 2006. (Sublibrary LNAI).

Vol. 3846: H. J. van den Herik, Y. Björnsson, N.S. Netanyahu (Eds.), Computers and Games. XIV, 333 pages. 2006.

Vol. 3845: J. Farré, I. Litovsky, S. Schmitz (Eds.), Implementation and Application of Automata. XIII, 360 pages. 2006.

Vol. 3844: J.-M. Bruehl (Ed.), Satellite Events at the MoDEL 2005 Conference. XIII, 360 pages. 2006.

Vol. 3843: P. Healy, N.S. Nikolov (Eds.), Graph Drawing. XVII, 536 pages. 2006.

Vol. 3842: H.T. Shen, J. Li, M. Li, J. Ni, W. Wang (Eds.), Advanced Web and Network Technologies, and Applications. XXVII, 1057 pages. 2006.

Vol. 3841: X. Zhou, J. Li, H.T. Shen, M. Kitsuregawa, Y. Zhang (Eds.), Frontiers of WWW Research and Development - APWeb 2006. XXIV, 1223 pages. 2006.

# Preface

This volume contains the proceedings of the 8th International Symposium on Functional and Logic Programming (FLOPS 2006), held in Fuji-Susono, Japan, April 24–26, 2006 at the Fuji Institute of Education and Training.

FLOPS is a forum for research on all issues concerning functional programming and logic programming. In particular it aims to stimulate the cross-fertilization as well as the integration of the two paradigms. The previous FLOPS meetings took place in Fuji-Susono (1995), Shonan (1996), Kyoto (1998), Tsukuba (1999), Tokyo (2001), Aizu (2002) and Nara (2004). The proceedings of FLOPS 1999, FLOPS 2001, FLOPS 2002 and FLOPS 2004 were published by Springer in the *Lecture Notes in Computer Science* series, as volumes 1722, 2024, 2441 and 2998, respectively.

In response to the call for papers, 51 papers were submitted. Each paper was reviewed by at least three Program Committee members with the help of expert external reviewers. The Program Committee meeting was conducted electronically for a period of 2 weeks in December 2005 and January 2006. After careful and thorough discussion, the Program Committee selected 17 papers (33%) for presentation at the conference. In addition to the 17 contributed papers, the symposium included talks by two invited speakers: Guy Steele (Sun Microsystems Laboratories) and Peter Van Roy (Université Catholique de Louvain).

On behalf of the Program Committee, we would like to thank the invited speakers who agreed to give talks and contribute papers, and all those who submitted papers to FLOPS 2006. As Program Chairs, we would like to sincerely thank all the members of the FLOPS 2006 Program Committee for their excellent job, and all the external reviewers for their invaluable contribution. The support of our sponsors is gratefully acknowledged. We are indebted to the Japan Society for Software Science and Technology (JSSST), the Association of Logic Programming (ALP), and the Asian Association for Foundation of Software (AAFS). Finally we would like to thank members of the Local Arrangements Committee, in particular Yoshihiko Kakutani, for their invaluable support throughout the preparation and organization of the symposium.

February 2006

Masami Hagiya  
Philip Wadler  
Program Co-chairs  
FLOPS 2006



# Symposium Organization

## Program Chairs

Philip Wadler  
Masami Hagiya

Edinburgh, UK  
Tokyo, Japan

## Program Committee

Vincent Danos  
Jacques Garrigue  
Manuel Hermenegildo  
Gabrielle Keller  
Michael Rusinowitch  
Konstantinos Sagonas  
Ken Satoh  
Peter Selinger  
Eijiro Sumii  
Naoyuki Tamura  
Peter Thiemann  
David Warren

Paris, France  
Nagoya, Japan  
New Mexico, USA & Madrid, Spain  
UNSW, Sydney, Australia  
INRIA Lorraine, France  
Uppsala, Sweden  
NII, Tokyo, Japan  
Dalhousie, Canada  
Tohoku, Japan  
Kobe, Japan  
Freiburg, Germany  
Stony Brook, USA

## Local Arrangements Chair

Masami Hagiya

Tokyo, Japan



## Referees

Tatsuya Abe  
 Amal Ahmed  
 Kenichi Asai  
 Demis Ballis  
 Maria Garcia de la Banda  
 Bruno Blanchet  
 Daniel Cabeza  
 Venanzio Capretta  
 Olga Caprotti  
 Francois Charoy  
 Ezra Cooper  
 Markus Degen  
 Rachid Echahed  
 Carl Christian Frederiksen  
 Naoki Fukuta  
 Martin Gasbichler  
 Samir Genaim  
 Michael Hanus  
 Ralf Hinze  
 Hiroshi Hosobe  
 Haruo Hosoya  
 Zhenjiang Hu  
 Atsushi Igarashi  
 Koji Kagawa  
 Yoshihiko Kakutani  
 Dominique Larchey  
 Pedro López  
 Ugo Dal Lago  
 Toshiyuki Maeda  
 Julio Mariño  
 Yasuhiko Minamide

Jean-Yves Moyen  
 Susana Muñoz  
 Keiko Nakata  
 Jorge Navas  
 Matthias Neubauer  
 Tobias Nipkow  
 Susumu Nishimura  
 Shin-ya Nishizaki  
 Martin Odersky  
 Yoshihiro Oyama  
 Mikael Pettersson  
 David Pichardie  
 Benjamin Pierce  
 Paola Quaglia  
 Christophe Ringeissen  
 Don Sannella  
 Ganesh Sittampalam  
 Yasuyuki Tahara  
 Yoshiaki Takata  
 Yasuyuki Tsukada  
 Satoshi Tojo  
 Akihiko Tozawa  
 Rafael del Vado  
 German Vidal  
 Dimitrios Vytiniotis  
 Hironori Washizaki  
 Stefan Wehr  
 Stephanie Weirich  
 Jeremy Yallop  
 Akihiro Yamamoto  
 Mitsuharu Yamamoto

# Table of Contents

## Invited Papers

Parallel Programming and Parallel Abstractions in Fortress <i>Guy L. Steele</i> .....	1
Convergence in Language Design: A Case of Lightning Striking Four Times in the Same Place <i>Peter Van Roy</i> .....	2

## Refereed Papers

### Data Types

“Scrap Your Boilerplate” Reloaded <i>Ralf Hinze, Andres Löh, Bruno C.d.S. Oliveira</i> .....	13
Ruler: Programming Type Rules <i>Atze Dijkstra, S. Doaitse Swierstra</i> .....	30
A Framework for Extended Algebraic Data Types <i>Martin Sulzmann, Jeremy Wazny, Peter J. Stuckey</i> .....	47

### FP Extensions

Lock Free Data Structures Using STM in Haskell <i>Anthony Discolo, Tim Harris, Simon Marlow, Simon Peyton Jones, Satnam Singh</i> .....	65
XML Type Checking Using High-Level Tree Transducer <i>Akihiko Tozawa</i> .....	81

### Type Theory

A Computational Approach to Pocklington Certificates in Type Theory <i>Benjamin Grégoire, Laurent Théry, Benjamin Werner</i> .....	97
Defining and Reasoning About Recursive Functions: A Practical Tool for the Coq Proof Assistant <i>Gilles Barthe, Julien Forest, David Pichardie, Vlad Rusu</i> .....	114

**LP Extensions**

Soutei, a Logic-Based Trust-Management System  
*Andrew Pimlott, Oleg Kiselyov* ..... 130

A Syntactic Approach to Combining Functional Notation, Lazy  
Evaluation, and Higher-Order in LP Systems  
*Amadeo Casas, Daniel Cabeza, Manuel V. Hermenegildo* ..... 146

**Analysis**

Resource Analysis by Sup-interpretation  
*Jean-Yves Marion, Romain Péchoux* ..... 163

Lazy Set-Sharing Analysis  
*Xuan Li, Andy King, Lunjin Lu* ..... 177

Size-Change Termination and Bound Analysis  
*James Avery* ..... 192

**Contracts**

Typed Contracts for Functional Programming  
*Ralf Hinze, Johan Jeuring, Andres Löb* ..... 208

Contracts as Pairs of Projections  
*Robert Bruce Findler, Matthias Blume* ..... 226

**Web and GUI**

iData for the World Wide Web – Programming Interconnected Web  
Forms  
*Rinus Plasmeijer, Peter Achten* ..... 242

Crossing State Lines: Adapting Object-Oriented Frameworks to  
Functional Reactive Languages  
*Daniel Ignatoff, Gregory H. Cooper, Shriram Krishnamurthi* ..... 259

WASH Server Pages  
*Peter Thiemann* ..... 277

**Author Index** ..... 295

# Parallel Programming and Parallel Abstractions in Fortress

Guy L. Steele

Sun Microsystems Laboratories

**Abstract.** The Programming Language Research Group at Sun Microsystems Laboratories seeks to apply lessons learned from the Java (TM) Programming Language to the next generation of programming languages. The Java language supports platform-independent parallel programming with explicit multithreading and explicit locks. As part of the DARPA program for High Productivity Computing Systems, we are developing Fortress, a language intended to support large-scale scientific computation. One of the design principles is that parallelism be encouraged everywhere (for example, it is intentionally just a little bit harder to write a sequential loop than a parallel loop). Another is to have rich mechanisms for encapsulation and abstraction; the idea is to have a fairly complicated language for library writers that enables them to write libraries that present a relatively simple set of interfaces to the application programmer. We will discuss ideas for using a rich polymorphic type system to organize multithreading and data distribution on large parallel machines. The net result is similar in some ways to data distribution facilities in other languages such as HPF and Chapel, but more open-ended, because in Fortress the facilities are defined by user-replaceable libraries rather than wired into the compiler.

# Convergence in Language Design: A Case of Lightning Striking Four Times in the Same Place

Peter Van Roy

Université catholique de Louvain,  
B-1348 Louvain-la-Neuve, Belgium  
`pvr@info.ucl.ac.be`

<http://www.info.ucl.ac.be/people/cvvanroy.html>

**Abstract.** What will a definitive programming language look like? By *definitive language* I mean a programming language that gives good solutions at its level of abstraction, allowing computer science researchers to move on and work at higher levels. Given the evolution of computer science as a field with a rising level of abstraction, it is my belief that a small set of definitive languages will eventually exist. But how can we learn something about this set, considering that many basic questions about languages have not yet been settled? In this paper, I give some tentative conclusions about one definitive language. I present four case studies of substantial research projects that tackle important problems in four quite different areas: fault-tolerant programming, secure distributed programming, network-transparent distributed programming, and teaching programming as a unified discipline. All four projects had to think about language design. In this paper, I summarize the reasons why each project designed the language it did. It turns out that all four languages have a common structure. They can be seen as layered, with the following four layers in this order: a strict functional core, then deterministic concurrency, then message-passing concurrency, and finally shared-state concurrency (usually with transactions). This confirms the importance of functional programming and message passing as important defaults; however, global mutable state is also seen as an essential ingredient.

## 1 Introduction

This paper presents a surprising example of convergence in language design.<sup>1</sup> I will present four different research projects that were undertaken to solve four very different problems. The solutions achieved by all four projects are significant contributions to each of their respective areas. The four projects are interesting to us because they all considered language design as a key factor to achieve success. The surprise is that the four projects ended up using languages that have very similar structures.

---

<sup>1</sup> This paper was written to accompany an invited talk at FLOPS 2006 and is intended to stimulate discussion.

This paper is structured as follows. Section 1.1 briefly presents each of the four projects and Section 1.2 sketches their common solution. Then Sections 2 to 5 present each of the four projects in more detail to motivate why the common solution is a good solution for it. Finally, Section 6 concludes the paper by recapitulating the common solution and making some conclusions on why it is important for functional and logic programming.

Given the similar structure of the four languages, I consider that their common structure deserves to be carefully examined. The common structure may turn out to be the heart of one possible *definitive* programming language, i.e., a programming language that gives good solutions at its level of abstraction, so that computer science researchers can move on and work at higher levels. My view is that the evolution of programming languages will follow a similar course as the evolution of parsing algorithms. In the 1970s, compiler courses were often built around a study of parsing algorithms. Today, parsing is well understood for most practical purposes and when designing a new compiler it is straightforward to pick a parsing algorithm from a set of “good enough” or “definitive” algorithms. Today’s compiler courses are built around higher level topics such as dataflow analysis, type systems, and language design. For programming languages the evolution toward a definitive set may be slower than for parsing algorithms because languages are harder to judge objectively than algorithms.

## 1.1 The Four Projects

The four projects are the following:<sup>2</sup>

- Programming highly available embedded systems for telecommunications (Section 2). This project was undertaken by Joe Armstrong and his colleagues at the Ericsson Computer Science Laboratory. This work started in 1986. The Erlang language was designed and a first efficient and stable implementation was completed in 1991. Erlang and its current environment, the OTP (Open Telecom Platform) system, are being used successfully in commercial systems by Ericsson and other companies.
- Programming secure distributed systems with multiple users and multiple security domains (Section 3). This project was undertaken over many years by different institutions. It started with Carl Hewitt’s Actor model and led via concurrent logic programming to the E language designed by Doug Barnes, Mark Miller, and their colleagues. Predecessors of E have been used to implement various multiuser virtual environments.
- Making network-transparent distributed programming practical (Section 4). This project started in 1995 with the realization that the well-factored design of the Oz language, first developed by Gert Smolka and his students in 1991 as an outgrowth of the ACCLAIM project, was a good starting point for making network transparent distribution practical. This resulted in the Mozart Programming System, whose first release was in 1999.

---

<sup>2</sup> Many people were involved in each project; because of space limitations only a few are mentioned here.

- Teaching programming as a unified discipline covering all popular programming paradigms (Section 5). This project started in 1999 with the realization by the author and Seif Haridi that Oz is well-suited to teaching programming because it covers many programming concepts, it has a simple semantics, and it has an efficient implementation. A textbook published in 2004 “reconstructs” the Oz design according to a principled approach. This book is the basis of programming courses now being taught at more than a dozen universities worldwide.

## 1.2 The Layered Language Structure

In all four research projects, the programming language has a layered structure. In its most general form, the language has four layers. This section briefly presents the four layers and mentions how they are realized in the four projects. The rest of the paper motivates the layered structure for each project in more detail. The layers are the following:

- The inner layer is a strict functional language. All four projects start with this layer.
- The second layer adds deterministic concurrency. Deterministic concurrency is sometimes called declarative or dataflow concurrency. It has the property that it cannot have race conditions. This form of concurrency is as simple to reason in as functional programming. In Oz it is realized with single-assignment variables and dataflow synchronization. Because Oz implements these variables as logic variables, this layer in Oz is also a logic language. In E it is realized by a form of concurrent programming called *event-loop concurrency*: inside a process all objects share a single thread. This means that execution inside a process is deterministic. The Erlang project skips this layer.
- The third layer adds asynchronous message passing. This leads to a simple message-passing model in which concurrent entities send messages asynchronously. All four projects have this layer. In E, this layer is used for communication between processes (deterministic concurrency is used for communication inside a single process).
- The fourth layer adds global mutable state.<sup>3</sup> Three of the four projects have global mutable state as a final layer, provided for different reasons, but always with the understanding that it is not used as often as the other layers. In the Erlang project, the mutable state is provided as a persistent database with a transactional interface. In the network transparency project, the mutable state is provided as an object store with a transactional interface and as a family of distributed protocols that is used to guarantee coherence of state across the distributed system. These protocols are expensive but they are sometimes necessary. In the teaching programming project, mutable state is used to make programs modular. The E project skips this layer.

---

<sup>3</sup> By *global*, I mean that the mutable state has a scope that is as large as necessary, not that it necessarily covers the whole program.



This layered structure has an influence on program design. In all four projects, the starting point is the functional inner layer, complemented by the message-passing layer which is just as important. In three of the four projects, the final layer (global mutable state) is less used than the others, but it provides a critical functionality that cannot be eliminated.

Note that the network-transparent distribution project and the teaching programming project were undertaken by many of the same people and started with the same programming language. Both projects were undertaken because we had reasons to believe Oz would be an adequate starting point. Each project had to adapt the Oz language to get a good solution. In the final analysis, both projects give good reasons why their solutions are appropriate, as explained in Sections 4 and 5.

## 2 Fault-Tolerant Programming

The Erlang programming language and system is designed for building high availability telecommunications systems. Erlang was designed at the Ericsson Computer Science Laboratory [5,4]. Erlang is designed explicitly to support programs that tolerate both software and hardware faults. Note that software faults are unavoidable: studies have shown that even with extensive testing, software still has bugs. Any system with high availability must therefore have a way to tolerate faults due to software bugs. Erlang has been used to build commercial systems of very high availability [8]. The most successful of these systems is the AXD 301 ATM switch, which contains around 1 million lines of Erlang, a similar amount of C/C++ code, and a small amount of Java [29].

An Erlang program consists of a (possibly very large) number of processes. An Erlang process is a lightweight entity with its own memory space. A process is programmed with a strict functional language. Each process has a unique identity, which is a constant that can be stored in data structures and in messages. Processes communicate by sending asynchronous messages to other processes. A process receives messages in its mailbox, and it can extract messages from the mailbox with pattern matching. Note that a process can do dynamic code change by receiving a new function in a message and installing it as the new process definition. We conclude that this structure gives the Erlang language two layers: a functional layer for programming processes, and a message-passing layer for allowing them to communicate.

To support fault tolerance, two processes can be linked together. When one process fails, for example because of a software error, then the other fails as well. Each process has a supervisor bit. If a process is set to supervisor mode, then it does not fail when a linked process fails, but it receives a message generated by the run-time system. This allows the application to recover from the failure. Erlang is well-suited to implement software fault tolerance because of process isolation and process linking.

Erlang also has a database called Mnesia. The database stores consistent snapshots of critical program data. When processes fail, their supervisors can use the database to recover and continue execution. The database provides a

transactional interface to shared data. The database is an essential part of Erlang programs. It can therefore be considered as a third layer of the Erlang language. This third layer, mutable state with a transactional interface, implements a form of shared-state concurrency [26].

Because Erlang processes do not share data, they can be implemented over a distributed system without any changes in the program. This makes distributed programming in Erlang straightforward. Using process linking and supervisors, Erlang programs can also recover from hardware failures, i.e., partial failures of the distributed system.

### 3 Secure Distributed Programming

The E programming language and system is designed for building secure distributed systems [21, 19]. The E language consists of objects (functions that share an encapsulated state) hosted in secure processes called *vats* that communicate through a secure message-passing protocol based on encryption. Within the language, security is provided by implementing all language references (including object references) as capabilities. A *capability* is an unforgeable reference that combines two properties that cannot be separated: it designates a language entity and it provides permission to perform a well-defined set of operations on the entity. The only way to perform an operation is to have a capability for that operation.

Capabilities are passed between language entities according to well-defined rules. The primary rule is that the only way to get a capability is that an entity to which you already have a capability passes you the capability (“connectivity begets connectivity”). A system based on capabilities can support the Principle of Least Authority (POLA): give each entity just enough authority to carry out its work. In systems based on POLA the destructive abilities of malicious programs such as viruses largely go away. Unfortunately, current programming languages and operating systems only have weak support for POLA. This is why projects such as E and KeyKOS (see below) are so important [25].

Inside a vat, there is a single thread of execution and all objects take turns executing in this thread. Objects send other objects asynchronous messages that are queued for execution. Objects execute a method when they receive a message. This is a form of deterministic concurrency that is called *event-loop concurrency*. Single threading within a vat is done to ensure that concurrency introduces no security problems due to the nondeterminism of interleaving execution. Event-loop concurrency works well for secure programs; a model based on shared-state concurrency is much harder to program with [20]. Between two or more vats, execution is done according to a general message-passing model.

In a system such as E that is based on capabilities, there is no *ambient authority*, i.e., a program does not have the ability to perform an operation just because it is executing in a certain context. This is very different from most other systems. For example, in Unix a program has all the authority of the user that executes it. The lack of ambient authority does not mean that E necessarily does not have global mutable state. For example, there could be a capability