Tarmo Uustalu (Ed.)

LNCS 4014

Mathematics of Program Construction

8th International Conference, MPC 2006
Kuressaare, Estonia, July 2006
Proceedings

Springer

Tarmo Uustalu (Ed.)

Mathematics of Program Construction

8th International Conference, MPC 2006
Kuressaare, Estonia, July 3-5, 2006
Proceedings

Springer

Volume Editor

Tarmo Uustalu
Institute of Cybernetics
Akadeemia tee 21, 12618 Tallinn, Estonia
E-mail: tarmo@cs.ioc.ee

Library of Congress Control Number: 2006927705

CR Subject Classification (1998): F.3, F.4, D.2, F.1, D.3

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

ISSN 0302-9743
ISBN-10 3-540-35631-2 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-35631-8 Springer Berlin Heidelberg New York

Springer is a part of Springer Science+Business Media

springer.com

Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11783596 06/3142 5 4 3 2 1 0

Lecture Notes in Computer Science

For information about Vols. 1–3960

please contact your bookseller or Springer

Vol. 4063: I. Gorton, G.T. Heineman, I. Crnkovic, H.W. Schmidt, J.A. Stafford, C.A. Szyperski, K. Wallnau (Eds.), Component-Based Software Engineering. XI, 394 pages. 2006.

Vol. 4060: K. Futatsugi, J.-P. Jouannaud, J. Meseguer (Eds.), Algebra, Meaning and Computation. XXXVIII, 643 pages. 2006.

Vol. 4058: L.M. Batten, R. Safavi-Naini (Eds.), Information Security and Privacy. XII, 446 pages. 2006.

Vol. 4056: P. Flocchini, L. Gąsieniec (Eds.), Structural Information and Communication Complexity. X, 357 pages. 2006.

Vol. 4055: J. Lee, J. Shim, S.-g. Lee, C. Bussler, S. Shim (Eds.), Data Engineering Issues in E-Commerce and Services. IX, 290 pages. 2006.

Vol. 4054: A. Horváth, M. Telek (Eds.), Formal Methods and Stochastic Models for Performance Evaluation. VIII, 239 pages. 2006.

Vol. 4053: M. Ikeda, K.D. Ashley, T.-W. Chan (Eds.), Intelligent Tutoring Systems. XXVI, 821 pages. 2006.

Vol. 4045: D. Barker-Plummer, R. Cox, N. Swoboda (Eds.), Diagrammatic Representation and Inference. XII, 301 pages. 2006. (Sublibrary LNAI).

Vol. 4044: P. Abrahamsson, M. Marchesi, G. Succi (Eds.), Extreme Programming and Agile Processes in Software Engineering. XII, 230 pages. 2006.

Vol. 4043: A.S. Atzeni, A. Lioy (Eds.), Public Key Infrastructure. XI, 261 pages. 2006.

Vol. 4041: S.-W. Cheng, C.K. Poon (Eds.), Algorithmic Aspects in Information and Management. XI, 395 pages. 2006.

Vol. 4040: R. Reulke, U. Eckardt, B. Flach, U. Knauer, K. Polthier (Eds.), Combinatorial Image Analysis. XII, 482 pages. 2006.

Vol. 4039: M. Morisio (Ed.), Reuse of Off-the-Shelf Components. XIII, 444 pages. 2006.

Vol. 4038: P. Ciancarini, H. Wiklicky (Eds.), Coordination Models and Languages. VIII, 299 pages. 2006.

Vol. 4037: R. Gorrieri, H. Wehrheim (Eds.), Formal Methods for Open Object-Based Distributed Systems. XVII, 474 pages. 2006.

Vol. 4036: O. H. Ibarra, Z. Dang (Eds.), Developments in Language Theory. XII, 456 pages. 2006.

Vol. 4034: J. Münch, M. Vierimaa (Eds.), Product-Focused Software Process Improvement. XVII, 474 pages. 2006.

Vol. 4033: B. Stiller, P. Reichl, B. Tuffin (Eds.), Performability Has its Price. X, 103 pages. 2006.

Vol. 4031: M. Ali, R. Dapoigny (Eds.), Innovations in Applied Artificial Intelligence. XXIII, 1353 pages. 2006. (Sublibrary LNAI).

Vol. 4027: H.L. Larsen, G. Pasi, D. Ortiz-Arroyo, T. Andreasen, H. Christiansen (Eds.), Flexible Query Answering Systems. XVIII, 714 pages. 2006. (Sublibrary LNAI).

Vol. 4026: P. Gibbons, T. Abdelzaher, J. Aspnes, R. Rao (Eds.), Distributed Computing in Sensor Systems. XIV, 566 pages. 2006.

Vol. 4025: F. Eliassen, A. Montresor (Eds.), Distributed Applications and Interoperable Systems. XI, 355 pages. 2006.

Vol. 4024: S. Donatelli, P. S. Thiagarajan (Eds.), Petri Nets and Other Models of Concurrency - ICATPN 2006. XI, 441 pages. 2006.

Vol. 4021: E. André, L. Dybkjær, W. Minker, H. Neumann, M. Weber (Eds.), Perception and Interactive Technologies. XI, 217 pages. 2006. (Sublibrary LNAI).

Vol. 4020: A. Bredenfeld, A. Jacoff, I. Noda, Y. Takahashi, RoboCup 2005: Robot Soccer World Cup IX. XVII, 727 pages. 2006. (Sublibrary LNAI).

Vol. 4018: V. Wade, H. Ashman, B. Smyth (Eds.), Adaptive Hypermedia and Adaptive Web-Based Systems. XVI, 474 pages. 2006.

Vol. 4016: J.X. Yu, M. Kitsuregawa, H.V. Leong (Eds.), Advances in Web-Age Information Management. XVII, 606 pages. 2006.

Vol. 4014: T. Uustalu (Ed.), Mathematics of Program Construction. X, 455 pages. 2006.

Vol. 4013: L. Lamontagne, M. Marchand (Eds.), Advances in Artificial Intelligence. XIII, 564 pages. 2006. (Sublibrary LNAI).

Vol. 4012: T. Washio, A. Sakurai, K. Nakashima, H. Takeda, S. Tojo, M. Yokoo (Eds.), New Frontiers in Artificial Intelligence. XIII, 484 pages. 2006. (Sublibrary LNAI).

Vol. 4011: Y. Sure, J. Domingue (Eds.), The Semantic Web: Research and Applications. XIX, 726 pages. 2006.

Vol. 4010: S. Dunne, B. Stoddart (Eds.), Unifying Theories of Programming. VIII, 257 pages. 2006.

Vol. 4009: M. Lewenstein, G. Valiente (Eds.), Combinatorial Pattern Matching. XII, 414 pages. 2006.

Vol. 4007: C. Àlvarez, M. Serna (Eds.), Experimental Algorithms. XI, 329 pages. 2006.

Vol. 4006: L.M. Pinho, M. González Harbour (Eds.), Reliable Software Technologies – Ada-Europe 2006. XII, 241 pages. 2006.

Vol. 4005: G. Lugosi, H.U. Simon (Eds.), Learning Theory. XI, 656 pages. 2006. (Sublibrary LNAI).

Vol. 4004: S. Vaudenay (Ed.), Advances in Cryptology - EUROCRYPT 2006. XIV, 613 pages. 2006.

Vol. 4003: Y. Koucheryavy, J. Harju, V.B. Iversen (Eds.), Next Generation Teletraffic and Wired/Wireless Advanced Networking. XVI, 582 pages. 2006.

Vol. 4001: E. Dubois, K. Pohl (Eds.), Advanced Information Systems Engineering. XVI, 560 pages. 2006.

Vol. 3999: C. Kop, G. Fliedl, H.C. Mayr, E. Métais (Eds.), Natural Language Processing and Information Systems. XIII, 227 pages. 2006.

Vol. 3998: T. Calamoneri, I. Finocchi, G.F. Italiano (Eds.), Algorithms and Complexity. XII, 394 pages. 2006.

Vol. 3997: W. Grieskamp, C. Weise (Eds.), Formal Approaches to Software Testing. XII, 219 pages. 2006.

Vol. 3996: A. Keller, J.-P. Martin-Flatin (Eds.), Self-Managed Networks, Systems, and Services. X, 185 pages. 2006.

Vol. 3995: G. Müller (Ed.), Emerging Trends in Information and Communication Security. XX, 524 pages. 2006.

Vol. 3994: V.N. Alexandrov, G.D. van Albada, P.M.A. Sloot, J. Dongarra (Eds.), Computational Science – ICCS 2006, Part IV. XXXV, 1096 pages. 2006.

Vol. 3993: V.N. Alexandrov, G.D. van Albada, P.M.A. Sloot, J. Dongarra (Eds.), Computational Science – ICCS 2006, Part III. XXXVI, 1136 pages. 2006.

Vol. 3992: V.N. Alexandrov, G.D. van Albada, P.M.A. Sloot, J. Dongarra (Eds.), Computational Science – ICCS 2006, Part II. XXXV, 1122 pages. 2006.

Vol. 3991: V.N. Alexandrov, G.D. van Albada, P.M.A. Sloot, J. Dongarra (Eds.), Computational Science – ICCS 2006, Part I. LXXXI, 1096 pages. 2006.

Vol. 3990: J. C. Beck, B.M. Smith (Eds.), Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. X, 301 pages. 2006.

Vol. 3989: J. Zhou, M. Yung, F. Bao, Applied Cryptography and Network Security. XIV, 488 pages. 2006.

Vol. 3987: M. Hazas, J. Krumm, T. Strang (Eds.), Location- and Context-Awareness. X, 289 pages. 2006.

Vol. 3986: K. Stølen, W.H. Winsborough, F. Martinelli, F. Massacci (Eds.), Trust Management. XIV, 474 pages. 2006.

Vol. 3984: M. Gavrilova, O. Gervasi, V. Kumar, C.J. K. Tan, D. Taniar, A. Laganà, Y. Mun, H. Choo (Eds.), Computational Science and Its Applications - ICCSA 2006, Part V. XXV, 1045 pages. 2006.

Vol. 3983: M. Gavrilova, O. Gervasi, V. Kumar, C.J. K. Tan, D. Taniar, A. Laganà, Y. Mun, H. Choo (Eds.), Computational Science and Its Applications - ICCSA 2006, Part IV. XXVI, 1191 pages. 2006.

Vol. 3982: M. Gavrilova, O. Gervasi, V. Kumar, C.J. K. Tan, D. Taniar, A. Laganà, Y. Mun, H. Choo (Eds.), Computational Science and Its Applications - ICCSA 2006, Part III. XXV, 1243 pages. 2006.

Vol. 3981: M. Gavrilova, O. Gervasi, V. Kumar, C.J. K. Tan, D. Taniar, A. Laganà, Y. Mun, H. Choo (Eds.), Computational Science and Its Applications - ICCSA 2006, Part II. XXVI, 1255 pages. 2006.

Vol. 3980: M. Gavrilova, O. Gervasi, V. Kumar, C.J. K. Tan, D. Taniar, A. Laganà, Y. Mun, H. Choo (Eds.), Computational Science and Its Applications - ICCSA 2006, Part I. LXXV, 1199 pages. 2006.

Vol. 3979: T.S. Huang, N. Sebe, M.S. Lew, V. Pavlović, M. Kölsch, A. Galata, B. Kisačanin (Eds.), Computer Vision in Human-Computer Interaction. XII, 121 pages. 2006.

Vol. 3978: B. Hnich, M. Carlsson, F. Fages, F. Rossi (Eds.), Recent Advances in Constraints. VIII, 179 pages. 2006. (Sublibrary LNAI).

Vol. 3977: N. Fuhr, M. Lalmas, S. Malik, G. Kazai (Eds.), Advances in XML Information Retrieval and Evaluation. XII, 556 pages. 2006.

Vol. 3976: F. Boavida, T. Plagemann, B. Stiller, C. Westphal, E. Monteiro (Eds.), Networking 2006. Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications Systems. XXVI, 1276 pages. 2006.

Vol. 3975: S. Mehrotra, D.D. Zeng, H. Chen, B. Thuraisingham, F.-Y. Wang (Eds.), Intelligence and Security Informatics. XXII, 772 pages. 2006.

Vol. 3973: J. Wang, Z. Yi, J.M. Zurada, B.-L. Lu, H. Yin (Eds.), Advances in Neural Networks - ISNN 2006, Part III. XXIX, 1402 pages. 2006.

Vol. 3972: J. Wang, Z. Yi, J.M. Zurada, B.-L. Lu, H. Yin (Eds.), Advances in Neural Networks - ISNN 2006, Part II. XXVII, 1444 pages. 2006.

Vol. 3971: J. Wang, Z. Yi, J.M. Zurada, B.-L. Lu, H. Yin (Eds.), Advances in Neural Networks - ISNN 2006, Part I. LXVII, 1442 pages. 2006.

Vol. 3970: T. Braun, G. Carle, S. Fahmy, Y. Koucheryavy (Eds.), Wired/Wireless Internet Communications. XIV, 350 pages. 2006.

Vol. 3969: Ø. Ytrehus (Ed.), Coding and Cryptography. XI, 443 pages. 2006.

Vol. 3968: K.P. Fishkin, B. Schiele, P. Nixon, A. Quigley (Eds.), Pervasive Computing. XV, 402 pages. 2006.

Vol. 3967: D. Grigoriev, J. Harrison, E.A. Hirsch (Eds.), Computer Science – Theory and Applications. XVI, 684 pages. 2006.

Vol. 3966: Q. Wang, D. Pfahl, D.M. Raffo, P. Wernick (Eds.), Software Process Change. XIV, 356 pages. 2006.

Vol. 3965: M. Bernardo, A. Cimatti (Eds.), Formal Methods for Hardware Verification. VII, 243 pages. 2006.

Vol. 3964: M. Ü. Uyar, A.Y. Duale, M.A. Fecko (Eds.), Testing of Communicating Systems. XI, 373 pages. 2006.

Vol. 3963: O. Dikenelli, M.-P. Gleizes, A. Ricci (Eds.), Engineering Societies in the Agents World VI. XII, 303 pages. 2006. (Sublibrary LNAI).

Vol. 3962: W. IJsselsteijn, Y. de Kort, C. Midden, B. Eggen, E. van den Hoven (Eds.), Persuasive Technology. XII, 216 pages. 2006.

Preface

This volume contains the proceedings of the 8th International Conference on Mathematics of Program Construction, MPC 2006, held at Kuressaare, Estonia, July 3–5, 2006, colocated with the 11th International Conference on Algebraic Methodology and Software Technology, AMAST 2006, July 5–8, 2006.

The MPC conferences aim to promote the development of mathematical principles and techniques that are demonstrably useful and usable in the process of constructing computer programs. Topics of interest range from algorithmics to support for program construction in programming languages and systems.

The previous MPCs were held at Twente, The Netherlands (1989, LNCS 375), Oxford, UK (1992, LNCS 669), Kloster Irsee, Germany (1995, LNCS 947), Marstrand, Sweden (1998, LNCS 1422), Ponte de Lima, Portugal (2000, LNCS 1837), Dagstuhl, Germany (2002, LNCS 2386) and Stirling, UK (2004, LNCS 3125, colocated with AMAST 2004).

MPC 2006 received 45 submissions. Each submission was reviewed by four Programme Committee members or additional referees. The committee decided to accept 22 papers. In addition, the programme included three invited talks by Robin Cockett (University of Calgary, Canada), Olivier Danvy (Aarhus Universitet, Denmark) and Oege de Moor (University of Oxford, UK).

The review process and compilation of the proceedings were greatly helped by Andrei Voronkov's EasyChair system that I can only recommend to every programme chair.

MPC 2006 had one satellite workshop, the Workshop on Mathematically Structured Functional Programming, MSFP 2006, organized as a "small" workshop of the FP6 IST coordination action TYPES. This took place July 2, 2006.

Tallinn, April 2006 Tarmo Uustalu

Conference Organization

Programme Chair

Tarmo Uustalu (Institute of Cybernetics, Estonia)

Programme Committee

Roland Backhouse (University of Nottingham, UK)
Eerke Boiten (University of Kent, UK)
Venanzio Capretta (University of Ottawa, Canada)
Sharon Curtis (Oxford Brookes University, UK)
Jules Desharnais (Université de Laval, Canada)
Jeremy Gibbons (University of Oxford, UK)
Lindsay Groves (Victoria University of Wellington, New Zealand)
William Harrison (University of Missouri, USA)
Ian J. Hayes (University of Queensland, Australia)
Johan Jeuring (Universiteit Utrecht, The Netherlands)
Dexter Kozen (Cornell University, USA)
Christian Lengauer (Universität Passau, Germany)
Lambert Meertens (Kestrel Institute, USA)
Shin-Cheng Mu (Academia Sinica, Taiwan)
Bernhard Möller (Universität Augsburg, Germany)
José Nuno Oliveira (Universidade do Minho, Portugal)
Alberto Pardo (Universidad de la República, Uruguay)
Ross Paterson (City University London, UK)
Ingrid Rewitzky (University of Stellenbosch, South Africa)
Varmo Vene (University of Tartu, Estonia)

Additional Referees

José Bacelar Almeida
Ian Bayley
Yves Bertot
Marc Bezem
Ana Bove
Carlos Camarão
David Carrington
Manuel Chakravarty
Tyng-Ruey Chuang
Michael Claßen
Robert Colvin
Phil Cook
Silvia Crafa
Alcino Cunha
Ellie D'Hondt
Andreas Dolzmann
Colin Fidge
Sergei Gorlatch
Jonathan Grattage
Dan Grundy
E. C. R. Hehner
John Hughes
Peter Höfner
Benjamin Kelly

Shriram Krishnamurthi
Peeter Laud
Carlos Luna
Ralf Lämmel
Clare Martin
Larissa Meinicke
Diethard Michaelis
Till Mossakowski
Härmel Nestra
Milad Niqui
John O'Donnell
Bruno C. d. S. Oliveira
Jorge Sousa Pinto
Fermin Reig
Gunter Ritter
Eike Ritter
Ando Saabas
Lutz Schröder
Olha Shkaravska
Graeme Smith
Kim Solin
Barney Stratford
Georg Struth
Femke van Raamsdonk
Joost Visser
Da-Wei Wang
Meng Wang

Organizing Committee

Juhan Ernits, Monika Perkmann, Ando Saabas, Olha Shkaravska, Kristi Uustalu, Tarmo Uustalu (Institute of Cybernetics, Estonia).

Host Institution

Institute of Cybernetics at Tallinn University of Technology, Estonia.

Sponsors

National Centers of Excellence Programme of the Estonian Ministry of Education and Research.

Table of Contents

Invited Talks

Contributed Papers

What Is a Good Process Semantics?
(Extended Abstract)

Robin Cockett

Dept. of Computer Science, University of Calgary,
2500 University Drive NW, Calgary, Alb. T2N 1N4, Canada
robin@cpsc.ucalgary.ca

Abstract. Current mathematical tools for understanding processes predominantly support process modeling. In particular, they faithully represent all the things that can go wrong (deadlock, livelock, etc.). However, for the development of good programming abstractions in concurrent (and other) setting it is important to focus on formal systems in which things do not go wrong. So what are the formal models of processes where nothing goes wrong?

For those involved in trying to understand the mathematics of program construction the new challenge is to understand the mathematics of concurrent programs. The era of simple input/output computation has been completely superseded by an expectation of connectivity from which there is no return.

After some four decades of intense effort to provide a good calculus of processes, Robin Milner's π-calculus [5, 6] and its variants have emerged as a core paradigm. The π-calculus evolved directly from CCS and may be regarded as a response to the desire to pass information between processes beyond the mere fact of communication. To achieve this it was necessary to introduce the notion of a channel along which information could be passed and this involved solving the syntactic scope and substitution issues inherent in interaction along such channels.

A considerable portion of the theoretical effort which went into these ideas was inspired by operational considerations. In particular, the underlying paradigm for equality hinged on behavioural equivalence and the notion of bisimulation. The preoccupation with how the solution of these local technical issues lead to a coherent global notion of equality based on bisimulation seemed to an observer, such as myself, to be in tension with the desire to understand the structure of processes.

Of course, equality given through operational considerations as embodied in notions of bisimulation is a crucial sanity check: without it the production of an operational system is impossible. However, these operational considerations do not of themselves lead to a well-clothed mathematical understanding of processes. In particular, they do not directly inform us of what the manipulations of processes should be or how these manipulations should be organized. To make progress on this front it is necessary to turn to algebraic rather than operational sources for guidance.

T. Uustalu (Ed.): MPC 2006, LNCS 4014, pp. 1–3, 2006.

The λ-calculus [1] is a basis for simple input/output computations and the model of reduction in this calculus undoubtedly provided inspiration for reduction of the π-calculus. However, the λ-calculus transcended being a mere mechanism to model computation and became intimately connected into mathematics when the Curry-Howard-Lambek isomorphism was established. Terms of the typed λ-calculus correspond precisely to proofs of propositions which, in turn, form a cartesian closed category.

Lambek's contribution to this was the categorical end, but it was also really much broader: for it was categorical proof theory itself [4]. He understood that the cut-elimination process is the operational semantics of composition. Furthermore he realized that there is a correspondence between proof theories and categorical doctrines. While one of Lambek's motivation was to use the reduction processes from proof theory to throw light on categorical coherence issues, his observation opened up a connection through which ideas could flow in both directions. Examples of categorical doctrines occur throughout mathematics and they can (and have) been used as a rich source from which to develop a deeper understanding of the corresponding proof theories.

So what is the categorical proof theory of processes? I will argue that it is, in fact, an old and thorny friend: multiplicative additive linear logic. This is a thorn friend as the coherence issues of this logic are still the subject of active research [7]. Indeed, at this time, it is not clear that the definitive view of even these most basic issue has yet emerged. Equality of proofs, however, is known to be decidable [3]and one way to show this is to use a term logic reminiscent of the π-calculus. These ideas go right back to Bellin and Scott's early work [2].

Recalled the proof theoretic systems for typed λ-calculi are powerful enough to secure good termination properties. However, these formal properties are bought at a cost to expressiveness and consequently programmability. It is still open, for example, whether the loss of expressiveness due to the imposed type discipline can be successfully arranged in a manner to satisfy a significant programming community.

To make the proof theory for concurrent processes usable as a language in which reasonable concurrent problems can be programmed it is necessary to add datatypes and value passing. Datatypes, in the process world, correspond to protocols. The resulting type systems for the proof theory of linear logic do actually secure all the good properties one wants: progressiveness, deadlock freedom, and livelock freedom.

Unfortunately I do not claim to know (yet) how to turn this into something which approaches a practical programming language! This is still seems a distant goal. However, the motivation for formally based languages to support concurrent computation, when compared to that for simple input/output computations, is much greater. This simply because so much more can go wrong. Furthermore, the paradigms for expressing concurrent computation are still relatively crude and this means there is much to be gained, even for todays programs, from studying the mathematical structure of these formal systems.

References

1. Barendregt, H. P.: The Lambda Calculus: Its Syntax and Semantics. Revised edn. Vol. 103 of Studies in Logic and the Foundations of Mathematics. North-Holland (1984)
2. Bellin, G., Scott, P. J.: On the pi-calculus and linear logic. Theor. Comput. Sci. **135**(1) (1994) 11–65
3. Cockett, J. R. B., Pastro, C.: A language for multiplicative-additive linear logic. In Proc. of 10th Conf. on Category Theory and Computer Science, CTCS 2004. Vol. 122 of Electron. Notes in Theor. Comput. Sci. Elsevier (2005) 23–65.
4. Lambek, J.: Deductive systems and categories II. Proc. of Conf. on Category Theory, Homology Theory and Their Applications, Vol. 1. Vol. 87 of Lect. Notes in Math. Springer-Verlag (1969) 76–122
5. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes I. Inform. and Comput. **100**(1) (1992) 1–40
6. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes II. Inform. and Comput. **100**(1) (1992) 41–77
7. van Glabbeek, R. J., Hughes, D. J. D.: Proof nets for unit-free multiplicative-additive linear logic. ACM Trans. on Comput. Logic **6**(4) (2005) 784–842

Refunctionalization at Work

Olivier Danvy

BRICS,
Department of Computer Science, University of Aarhus,
IT-parken, Aabogade 34, DK-8200 Aarhus N, Denmark
danvy@brics.dk

Abstract. First-order programs are desired in a variety of settings and for a variety of reasons. Their coming into existence in first-order form may be unplanned or it could be the deliberate result of a form of "firstification" such as closure conversion, (super)combinator conversion, or defunctionalization. In the latter case, they are higher-order programs in disguise, just as iterative programs with accumulators are often recursive programs in disguise.

This talk is about Reynolds's defunctionalization [1, 2]. Over the last few years, we have observed that a number of existing first-order programs turn out to be in the range of defunctionalization, and therefore they directly correspond to higher-order programs, even though they were designed independently of any higher-order representation. Not all first-order programs, however, are in defunctionalized form.

The goal of this talk is to refine our earlier characterization of what it means to be in defunctionalized form [3], and to investigate how one can tease a first-order program into defunctionalized form. On the way, we present a variety of independently known programs that are in (or can be teased into) defunctionalized form, and we exhibit their functional counterpart—a process we refer to as 'refunctionalization' since it is a left inverse of defunctionalization.

References

1. Reynolds, J. C.: Definitional interpreters for higher-order programming languages. In: Proc. of 25th ACM Nat. Conf. ACM Press (1972) 717–740 // Reprinted in Higher-Order and Symb. Comput. **11**(4) (1998) 363–397
2. Reynolds, J. C.: Definitional interpreters revisited. Higher-Order and Symb. Comput. **11**(4) (1998) 355–361
3. Danvy, O., Nielsen, L. R.: Defunctionalization at work. In Proc. of 3rd Int. ACM SIGPLAN Conf. on Principles and Practice of Declarative Programming PPDP'01. ACM Press (2001) 162–174

T. Uustalu (Ed.): MPC 2006, LNCS 4014, p. 4, 2006.

Aspects and Data Refinement
(Extended Abstract)

Pavel Avgustinov[1], Eric Bodden[2], Elnar Hajiyev[1], Oege de Moor[1], Neil Ongkingco[1], Damien Sereni[1], Ganesh Sittampalam[1], and Julian Tibble[1]

[1] Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford OX1 3QD, United Kingdom
[2] School of Computer Science, McGill University, Montréal, Québec H3A 2A7, Canada

Abstract. We give an introduction to aspect-oriented programming from the viewpoint of data refinement. Some data refinements are conveniently expressed via aspects. Unlike traditional programming language features for data refinement, aspects conceptually transform run-time events, not compile-time programs.

1 Introduction

Data refinement is a powerful tool in program construction: we start with an existing module, adding some new variables related to the existing ones via a *coupling invariant*, and possibly adding new operations as well. Next we refine each of the existing operations so that the coupling invariant is maintained. Finally, if any existing variables have become redundant, they are removed [1].

The idea is pervasive, and it is no surprise, therefore, that numerous researchers have attempted to capture it in a set of programming language features. An early example of this trend can be found in the work of Bob Paige, who advocated the use of a program transformation system to achieve the desired effect [2]. The idea was again raised by David Gries and Dennis Volpano in their design of the *transform* in the Polya programming language [3]. Very recently, Annie Liu and her coworkers [4] breathed new life into this line of work by updating it to the context of object-oriented programming.

All these systems are very powerful, and they are complete in that all data refinements can be expressed, at least in principle. In another community, a set of programming language features has been proposed that is less powerful, but still suitable for direct expression of simple data refinements. These features are collectively known under the name of 'aspects' [5].

In this talk, we shall examine some examples of data refinement expressed as aspects. Conceptually aspects transform run-time computations, unlike the above systems, which are all based on the idea of compile-time transformation. For efficiency, aspect compilers do as much transformation as possible at compile-time [6], but that is an implementation technique, not the semantics. We argue that to write reusable data refinements, which are independent of the syntactic details of the program being refined, the run-time view offered by aspects is preferable.

T. Uustalu (Ed.): MPC 2006, LNCS 4014, pp. 5–9, 2006.

2 Data Refinement

Consider an interface in Java for bags (multisets) of integers; an example of such an interface is shown in Figure 1. It includes an operation that returns an iterator over the elements of a bag; the order of such an iteration is not further specified.

```
interface Bag {
    void add(int i);
    void remove(int i);
    java.util.Iterator iterator ();
}
```

Fig. 1. *Bag* interface in *Java*

Now suppose we wish to augment this interface, and all classes that implement it, with an operation that returns the average of the bag of integers. A naive implementation would be to re-calculate the average each time, but that requires time proportional to the size of the bag.

To achieve a contant-time implementation of *average*, we introduce two new variables via data refinement, namely *sum* and *size*. The coupling invariant is that *sum* holds the sum of the abstract bag, and *size* the number of elements.

```
public aspect Average {
    private int Bag.sum;
    private int Bag.size;
    public float Bag.average() {
        return (size == 0 ? ((float)sum) / ((float)size) : 0);
    }
    after(Bag b,int i) returning() :
        execution(void Bag.add(int)) &&
        this(b) &&
        args(i)
    {
        b.sum += i;
        b.size += 1;
    }
    after(Bag b,int i) returning() :
        execution(void Bag.remove(int)) &&
        this(b) &&
        args(i)
    {
        b.sum -= i;
        b.size -= 1;
    }
}
```

Fig. 2. Aspect for data refinement