Lecture Notes in Computer Science 4719

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

Roland Backhouse Jeremy Gibbons
Ralf Hinze Johan Jeuring (Eds.)

Datatype-Generic Programming

International Spring School, SSDGP 2006
Nottingham, UK, April 24-27, 2006
Revised Lectures

Springer

Volume Editors

Roland Backhouse
University of Nottingham
School of Computer Science
Jubilee Campus, Wollaton Road, Nottingham NG8 1BB, UK
E-mail: rcb@cs.nott.ac.uk

Jeremy Gibbons
Oxford University, Computing Laboratory
Wolfson Building, Parks Road, Oxford, OX1, 3QD, UK
E-mail: Jeremy.Gibbons@comlab.ox.ac.uk

Ralf Hinze
Universität Bonn
Institut für Informatik III
Römerstraße 164, 53117 Bonn, Germany
E-mail: ralf@informatik.uni-bonn.de

Johan Jeuring
Utrecht University
Institute of Information and Computing Science
3508 TB Utrecht, The Netherlands
E-mail: johanj@cs.uu.nl

Library of Congress Control Number: Applied for

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

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

ISSN 0302-9743

ISBN 978-3-540-76785-5 ISBN 978-3-540-76786-2 (eBook)
I 0 007 978-3-540-76786-2

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12191963 06/3180 5 4 3 2 1 0

Preface

A leitmotif in the evolution of programming paradigms has been the level and extent of parametrisation that is facilitated — the so-called *genericity* of the paradigm. The sorts of parameters that can be envisaged in a programming language range from simple values, like integers and floating-point numbers, through structured values, types and classes, to kinds (the type of types and/or classes). *Datatype-generic programming* is about parametrising programs by the structure of the data that they manipulate.

To appreciate the importance of datatype genericity, one need look no further than the internet. The internet is a massive repository of structured data, but the structure is rarely exploited. For example, compression of data can be much more effective if its structure is known, but most compression algorithms regard the input data as simply a string of bits, and take no account of its internal organisation.

Datatype-generic programming is about exploiting the structure of data when it is relevant and ignoring it when it is not. Programming languages most commonly used at the present time do not provide effective mechanisms for documenting and implementing datatype genericity. This volume is a contribution towards improving the state of the art.

The emergence of datatype genericity can be traced back to the late 1980s. A particularly influential contribution was made by the Dutch STOP (Specification and Transformation of Programs) project, led by Lambert Meertens and Doaitse Swierstra. The idea that was "in the air" at the time was the commonality in ways of reasoning about different datatypes. Reynolds' parametricity theorem, popularised by Wadler [17] as "theorems for free," and so-called "deforestation" techniques came together in the datatype-generic notions of "catamorphism," "anamorphism" and "hylomorphism," and the theorem that every hylomorphism can be expressed as the composition of a catamorphism after an anamorphism. The "theory of lists" [5] became a "theory of Fs," where F is an arbitrary datatype, and the "zip" operation on a *pair* of equal-length *lists* became a generic transformation from an F structure of same-shape G structures to a G structure of same-shape F structures [1, 11].

In response to these largely theoretical results, efforts got underway in the mid-to-late 1990s to properly reflect the developments in programming language design. The extension of functional programming to "polytypic" programming [14, 12] was begun, and, in 1998, the "generic programming" workshop was organized by Roland Backhouse and Tim Sheard at Marstrand in Sweden [4], shortly after a Dagstuhl Seminar on the same topic [13]. The advances that had been made played a prominent part in the Advanced Functional Programming summer school [16, 3, 15, 6], which was held in 1998.

Since the year 2000, the emphasis has shifted yet more towards making datatype-generic programming more practical. Research projects with this goal have been the Generic Haskell project led by Johan Jeuring at Utrecht University (see, for example, [10, 9]), the DFG-funded Generic Programming project led by Ralf Hinze at the Universtiy of Bonn, and the EPSRC-supported Datatype-Generic Programming project at the universities of Nottingham and Oxford, which sponsored the Spring School reported in this volume. (Note that although the summer school held in Oxford in August 2002 [2] was entitled "Generic Programming," the need to distinguish "datatype" generic programming from other notions of "generic" programming had become evident; the paper "Patterns in Datatype-Generic Programming" [8] is the first published occurrence of the term "datatype-generic programming.")

This volume comprises revisions of the lectures presented at the Spring School on Datatype-Generic Programming held at the University of Nottingham in April 2006. All the lectures have been subjected to thorough internal review by the editors and contributors, supported by independent external reviews.

Gibbons ("Datatype-Generic Programming") opens the volume with a comprehensive review of different sorts of parametrisation mechanisms in programmming languages, including how they are implemented, leading up to the notion of datatype genericity. In common with the majority of the contributors, Gibbons chooses the functional programming language Haskell to make the notions concrete. This is because functional programming languages provide the best test-bed for experimental ideas, free from the administrative noise and clutter inherent in large-scale programming in mainstream languages. In this way, Gibbons relates the so-called design patterns introduced by Gamma, Helm, Johnson and Vlissides [7] to datatype-generic programming constructs (the different types of morphism mentioned earlier). The advantage is that the patterns are made concrete, rather than being expressed in prose by example as in a recent Publication [7].

Hinze, Jeuring and Löh ("Comparing Approaches to Generic Programming in Haskell") compare a variety of ways that datatype-generic programming techniques have been incorporated into functional programming languages, in particular (but not exclusively) Haskell. They base their comparison on a collection of standard examples: encoding and decoding values of a given datatype, comparing values for equality, and mapping a function over, "showing," and performing incremental updates on the values stored in a datatype. The comparison is based on a number of criteria, including elements like integration into a programming language and tool support.

The goal of Hinze and Löh's paper ("Generic Programming Now") is to show how datatype-generic programming can be enabled in present-day Haskell. They identify three key ingredients essential to the task: a *type reflection* mechanism, a *type representation* and a generic *view* on data. Their contribution is to show how these ingredients can be furnished using generalised algebraic datatypes.

The theme of type reflection and type representation is central to Altenkirch, McBride and Morris's contribution ("Generic Programming with Dependent

Types") . Their paper is about defining different universes of types in the *Epigram* system, an experimental programming system based on dependent types. They argue that the level of genericity is dictated by the universe that is chosen. Simpler universes allow greater levels of genericity, whilst more complex universes cause the genericity to be more restricted.

Dependent types, and the Curry-Howard isomorphism between proofs and programs, also play a central role in the Ω*mega* language introduced by Sheard ("Generic Programming in Ω*mega*"). Sheard argues for a type system that is more general than Haskell's, allowing a richer set of programming patterns, whilst still maintaining a sound balance between computations that are performed at run-time and computations performed at compile-time.

Finally, Lämmel and Meijer ("Revealing the X/O Impedance Mismatch") explore the actual problem of datatype-generic programming in the context of present-day implementations of object-oriented languages and XML data models. The X/O impedance mismatch refers to the incompatibilities between XML and object-oriented models of data. They provide a very comprehensive and up-to-date account of the issues faced by programmers, and how these issues can be resolved.

It remains for us to express our thanks to those who have contributed to the success of the School. First and foremost, we thank Fermín Reig, who was responsible for much of the preparations for the School and its day-to-day organization. Thanks also to Avril Rathbone and Pablo Nogueira for their organizational support, and to the EPSRC (under grant numbers GR/S27085/01 and GR/D502632/1) and the School of Computer Science and IT of the University of Nottingham for financial support. Finally, we would like to thank the (anonymous) external referees for their efforts towards ensuring the quality of these lecture notes.

June 2007

Roland Backhouse
Jeremy Gibbons
Ralf Hinze
Johan Jeuring

References

1. Backhouse, R.C., Doornbos, H., Hoogendijk, P.: A class of commuting relators (September 1992), Available via World-Wide Web at `http://www.cs.nott.ac.uk/~rcb/MPC/papers`
2. Backhouse, R., Gibbons, J. (eds.): Generic Programming. LNCS, vol. 2793. Springer, Heidelberg (2003)
3. Backhouse, R., Jansson, P., Jeuring, J., Meertens, L.: Generic programming. An introduction. In: Swierstra, et al., pp. 28–115, `Braga98`
4. Backhouse, R., Sheard, T. (eds.): Workshop on Generic Programming, Informal proceedings (1998), available at `http://www.win.tue.nl/cs/wp/papers.html`
5. Bird, R.S.: An introduction to the theory of lists. In: Broy, M. (ed.) Logic of Programming and Calculi of Discrete Design. NATO ASI Series, vol. F36, Springer, Heidelberg (1987)
6. de Moor, O., Sittampalam, G.: Generic program transformation. In: Swierstra, et al., pp. 116–149, `Braga98`
7. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns – Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading (1995)
8. Gibbons, J.: Patterns in datatype-generic programming. In: Striegnitz, J., Davis, K. (eds.) Multiparadigm Programming, John von Neumann Institute for Computing (NIC), First International Workshop on Declarative Programming in the Context of Object-Oriented Languages (DPCOOL), vol. 27 (2003)
9. Hinze, R., Jeuring, J.: Generic Haskell: Applications. In: Backhouse and Gibbons, pp. 57–96, `gp03`
10. Hinze, R., Jeuring, J.: Generic Haskell: Practice and theory. In: Backhouse and Gibbons, pp. 1–56, `gp03`
11. Hoogendijk, P., Backhouse, R.: When do datatypes commute? In: Moggi, E., Rosolini, G. (eds.) CTCS 1997. LNCS, vol. 1290, pp. 242–260. Springer, Heidelberg (1997)
12. Jansson, P., Jeuring, J.: PolyP - a polytypic programming language extension. In: POPL '97: The 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 470–482. ACM Press, New York (1997)
13. Jazayeri, M., Musser, D.R., Loos, R.G.K. (eds.): Generic Programming. LNCS, vol. 1766. Springer, Heidelberg (2000), at `http://www.cs.rpi.edu/~musser/gp/dagstuhl/`
14. Jeuring, J., Jansson, P.: Polytypic programming. In: Launchbury, J., Sheard, T., Meijer, E. (eds.) Advanced Functional Programming. LNCS, vol. 1129, pp. 68–114. Springer, Heidelberg (1996)
15. Sheard, T.: Using MetaML: a staged programming language. In: Swierstra, et al., pp. 207–239, `Braga98`
16. Swierstra, S.D., Oliveira, J.N. (eds.): AFP 1998. LNCS, vol. 1608. Springer, Heidelberg (1999)
17. Wadler, P.: Theorems for free. In: 4'th Symposium on Functional Programming Languages and Computer Architecture, pp. 347–359. ACM, London (1989)

Contributors

Thorsten Altenkirch
School of Computer Science and Information Technology,
University of Nottingham, Nottingham, NG8 1BB, UK
txa@cs.nott.ac.uk
http://www.cs.nott.ac.uk/~txa/.

Roland Backhouse
School of Computer Science and Information Technology,
University of Nottingham, Nottingham, NG8 1BB, UK
rcb@cs.nott.ac.uk
http://www.cs.nott.ac.uk/~rcb/.

Jeremy Gibbons
Computing Laboratory, University of Oxford, Oxford, OX1 3QD, UK
jeremy.gibbons@comlab.ox.ac.uk
http://www.comlab.ox.ac.uk/jeremy.gibbons/.

Ralf Hinze
Institut für Informatik III, Universität Bonn, Römerstraße 164,
53117 Bonn, Germany
ralf@informatik.uni-bonn.de
http://www.informatik.uni-bonn.de/~ralf/.

Johan Jeuring
Institute of Information and Computing Sciences, Utrecht University,
P.O.Box 80.089, 3508 TB Utrecht, The Netherlands
and
Open University, Heerlen, The Netherlands
johanj@cs.uu.nl
http://www.cs.uu.nl/~johanj/

Ralf Lämmel
Data Programmability Team, Microsoft Corporation, Redmond WA, USA
ralf.lammel@microsoft.com
http://homepages.cwi.nl/~ralf/.

Andres Löh
Institut für Informatik III, Universität Bonn, Römerstraße 164,
53117 Bonn, Germany.
loeh@informatik.uni-bonn.de
http://www.informatik.uni-bonn.de/~loeh/.

Conor McBride
School of Computer Science and Information Technology,
University of Nottingham, Nottingham, NG8 1BB, UK
ctm@cs.nott.ac.uk
http://www.cs.nott.ac.uk/~ctm/.

Erik Meijer
SQL Server, Microsoft Corporation, Redmond WA, USA
emeijer@microsoft.com
http://research.microsoft.com/~emeijer/.

Peter Morris
School of Computer Science and Information Technology,
University of Nottingham, Nottingham, NG8 1BB, UK
pwm@cs.nott.ac.uk
http://www.cs.nott.ac.uk/~pwm/.

Tim Sheard
Computer Science Department, Maseeh College of Engineering and
Computer Science, Portland State University, Portland OR, USA
sheard@cs.pdx.edu
http://web.cecs.pdx.edu/~sheard/

Table of Contents