

Lecture Notes in Computer Science
Edited by G. Goos, J. Hartmanis, and J. van Leeuwen

2793

Springer

Berlin

Heidelberg

New York

Hong Kong

London

Milan

Paris

Tokyo

Roland Backhouse Jeremy Gibbons (Eds.)

Generic Programming

Advanced Lectures



Springer

Series Editors

Gerhard Goos, Karlsruhe University, Germany
Juris Hartmanis, Cornell University, NY, USA
Jan van Leeuwen, Utrecht University, The Netherlands

Volume Editors

Roland Backhouse
The University of Nottingham
School of Computer Science and Information Technology
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

Cataloging-in-Publication Data applied for

A catalog record for this book is available from the Library of Congress.

Bibliographic information published by Die Deutsche Bibliothek
Die Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliografie;
detailed bibliographic data is available in the Internet at <<http://dnb.ddb.de>>.

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

ISSN 0302-9743

ISBN 3-540-20194-7 Springer-Verlag 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-Verlag. Violations are liable for prosecution under the German Copyright Law.

Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH

<http://www.springer.de>

© Springer-Verlag Berlin Heidelberg 2003
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Olgun Computergrafik
Printed on acid-free paper SPIN: 10931868 06/3142 5 4 3 2 1 0

Preface

Generic programming is about making programming more effective by making it more general. This volume is about a novel form of genericity in programs, based on parameterizing programs by the structure of the data they manipulate. The material is based on lectures presented at a summer school on Generic Programming held at the University of Oxford in August 2002.

The lectures by Hinze and Jeuring introduced Generic Haskell, an extension of the Haskell programming language that allows the programmer to define a function by induction on the structure of types. The implementation of Generic Haskell provided a valuable tool for students to experiment with applications of this form of datatype genericity. The lecture material in this volume is divided into two parts. The first part (“practice and theory”) introduces Generic Haskell and the theory that underlies its design. The second part (“applications”) discusses three advanced applications of Generic Haskell in some depth.

The value of generic programming is illusory unless the nature and extent of the genericity can be described clearly and precisely. The lectures by Backhouse and Crole delve deeper into the theoretical basis for datatype genericity. Backhouse reviews the notion of parametric polymorphism (a notion well known to functional programmers) and then shows how this notion is extended to higher-order notions of parametricity. These are used to characterize what it means for a value to be stored in a datatype. Also, transformations on data structures are given precise specifications in this way. Underlying this account are certain basic notions of category theory and allegory theory. Crole presents the category theory needed for a deeper understanding of mechanisms for defining datatypes.

The final chapter, by Fiadeiro, Lopes and Wermelinger applies the mathematical “technology” of parameterization to the larger-scale architectural structure of programs. The description of a system is split into components and their interactions; architectural connectors are parameterized by components, leading to an overall system structure consisting of components and connector instances establishing the interactions between the components.

Our thanks go to all those involved in making the school a success. We are grateful to the technical support staff of the Oxford University Computing Laboratory for providing computing facilities, to Yorck Hunke, David Lacey and Silvija Seres of OUCL for assistance during the school, and to St. Anne’s College for an amenable environment for study. Thanks also go to Peter Buneman and Martin Odersky, who lectured at the school on *Semi-structured Data* and on *Object-Oriented and Functional Approaches to Compositional Programming*, respectively, but were unable to contribute to the proceedings.

June, 2003

Roland Backhouse
Jeremy Gibbons

Contributors

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/>

Roy Crole

Department of Mathematics and Computer Science,
University of Leicester, University Road, Leicester, LE1 7RH, UK
roy.crole@mcs.le.ac.uk
<http://www.mcs.le.ac.uk/~rcrole/>

José Luiz Fiadeiro

Department of Computer Science, University of Leicester, University Road,
Leicester, LE1 7RH, UK
jose@fiadeiro.org

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/>

Paul Hoogendijk

Philips Research, Prof. Holstlaan 4, 5655 AA Eindhoven, The Netherlands
Paul.Hoogendijk@philips.com

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/>

Antónia Lopes

Department of Informatics, Faculty of Sciences, University of Lisbon,
Campo Grande, 1749-016 Lisboa, Portugal
mal@di.fc.ul.pt

Michel Wermelinger

Dept. of Informatics, Faculty of Sciences and Technology, New University
of Lisbon, Quinta da Torre, 2829-516 Caparica, Portugal
mw@di.fct.unl.pt

Table of Contents

Chapter 1. Generic Haskell: Practice and Theory	1
<i>R. Hinze and J. Jeuring</i>	
1 Introduction	1
1.1 Type Systems	2
1.2 Haskell’s data Construct	3
1.3 Towards Generic Programming	7
1.4 Towards Generic Haskell	12
1.5 Stocktaking	17
1.6 Related Work	18
2 Generic Haskell—Practice	22
2.1 Mapping Functions	22
2.2 Kind-Indexed Types and Type-Indexed Values	25
2.3 Embedding-Projection Maps	26
2.4 Reductions	27
2.5 Pretty Printing	30
2.6 Running Generic Haskell	33
3 Generic Haskell—Theory	35
3.1 The Simply Typed Lambda Calculus as a Type Language	35
3.2 The Polymorphic Lambda Calculus	40
3.3 Specializing Type-Indexed Values	42
3.4 Bridging the Gap	45
4 Conclusion	52
Chapter 2. Generic Haskell: Applications	57
<i>R. Hinze and J. Jeuring</i>	
1 Introduction	57
2 Generic Dictionaries	62
2.1 Introduction	63
2.2 Signature	65
2.3 Properties	66
2.4 Type-Indexed Tries	67
2.5 Empty Tries	70
2.6 Singleton Tries	71
2.7 Look up	72
2.8 Inserting and Merging	73
2.9 Deleting	75
2.10 Related Work	76
3 XComprez: A Generic XML Compressor	77
3.1 Implementing an XML Compressor as a Generic Program	78
3.2 Analysis	83
3.3 Conclusions	84

4	The Zipper	85
4.1	The Basic Idea	85
4.2	Data Types as Fixed Points of Pattern Functors	86
4.3	Type Indices of Higher Kinds	86
4.4	Locations	87
4.5	Navigation Functions	88
5	Conclusions	93
Chapter 3. Generic Properties of Datatypes		97
<i>R. Backhouse and P. Hoogendijk</i>		
1	Introduction	97
2	Theorems for Free	98
2.1	Verifiable Genericity	102
3	Commuting Datatypes — Introduction	102
3.1	Structure Multiplication	104
3.2	Broadcasts	106
4	Allegories and Relators	109
4.1	Allegories	109
4.2	Relators	110
4.3	Composition and Relators Are Parametric	112
4.4	Division and Tabulation	113
4.5	Domains	114
5	Datatype = Relator + Membership	114
5.1	Pointwise Closure	114
5.2	Regular Relators	116
5.3	Natural Transformations	119
5.4	Membership and Fans	121
6	Commuting Datatypes — Formal Specification	125
6.1	Naturality Requirements	126
6.2	Composition	128
6.3	Half Zips and Commuting Relators	128
7	Consequences	129
7.1	Shape Preservation	129
7.2	All Regular Datatypes Commute	130
8	Conclusion	131
Chapter 4. Basic Category Theory for Models of Syntax		133
<i>R.L. Crole</i>		
1	Introduction	133
1.1	Prerequisites	133
1.2	The Aims	134
1.3	Learning Outcomes	134
2	Syntax Defined from Datatypes	134
2.1	An Example with Distinguished Variables and without Binding	135
2.2	An Example with Distinguished Variables and Binding	136
2.3	An Example with Arbitrary Variables and Binding	137
2.4	An Example without Variables but with Binding	138

3	Category Theory	139
3.1	Categories	139
3.2	Functors	143
3.3	Natural Transformations	144
3.4	Products	146
3.5	Coproducts	150
3.6	Algebras	151
3.7	The Functor $1 + (-): \mathit{Set} \rightarrow \mathit{Set}$	152
3.8	The Functor $A + (-): \mathit{Set} \rightarrow \mathit{Set}$	153
3.9	The Functor $1 + (A \times -): \mathit{Set} \rightarrow \mathit{Set}$	154
4	Models of Syntax	154
4.1	A Model of Syntax with Distinguished Variables and without Binding	157
4.2	A Model of Syntax with Distinguished Variables and with Binding	162
4.3	A Model of Syntax with Arbitrary Variables and Binding	165
4.4	A Model of Syntax without Variables but with Binding	168
4.5	Where to Now?	168
5	Appendix	170
5.1	Lists	170
5.2	Abstract Syntax Trees	170
5.3	Inductively Defined Sets	170
5.4	Rule Induction	173
5.5	Recursively Defined Functions	175
	Chapter 5. A Mathematical Semantics for Architectural Connectors	178
	<i>J.L. Fiadeiro, A. Lopes and M. Wermelinger</i>	
1	Introduction	178
2	System Configuration in CommUnity	180
2.1	Component Design	180
2.2	Configurations	184
3	Architectural Description in CommUnity	191
3.1	Architectural Connectors	191
3.2	Examples	196
4	An ADL-Independent Notion of Connector	201
4.1	Architectural Schools	201
4.2	Adding Abstraction to Architectural Connectors	207
5	Towards an Algebra of Connectors	213
5.1	Role Refinement	214
5.2	Role Encapsulation	216
5.3	Role Overlay	218
6	Concluding Remarks	219
	Author Index	223