Monographs in Computer Science

Editors

David Gries
Fred B. Schneider

Monographs in Computer Science

Abadi and Cardelli, **A Theory of Objects**

Benosman and Kang [editors], **Panoramic Vision: Sensors, Theory, and Applications**

Bhanu, Lin, Krawiec, **Evolutionary Synthesis of Pattern Recognition Systems**

Broy and Stølen, **Specification and Development of Interactive Systems: FOCUS on Streams, Interfaces, and Refinement**

Brzozowski and Seger, **Asynchronous Circuits**

Burgin, **Super-Recursive Algorithms**

Cantone, Omodeo, and Policriti, Set Theory for Computing: From Decision Procedures to Declarative Programming with Sets

Castillo, Gutiérrez, and Hadi, **Expert Systems and Probabilistic Network Models**

Downey and Fellows, **Parameterized Complexity**

Feijen and van Gasteren, **On a Method of Multiprogramming**

Herbert and Spärck Jones [editors], **Computer Systems: Theory, Technology, and Applications**

Leiss, **Language Equations**

Levin, Heydon, and Mann, **Software Configuration Management with VESTA**

McIver and Morgan [editors], **Programming Methodology**

McIver and Morgan [editors], **Abstraction, Refinement and Proof for Probabilistic Systems**

Misra, **A Discipline of Multiprogramming: Programming Theory for Distributed Applications**

Nielson [editor], **ML with Concurrency**

Paton [editor], **Active Rules in Database Systems**

Selig, **Geometrical Methods in Robotics**

Selig, **Geometric Fundamentals of Robotics, Second Edition**

Shasha and Zhu, **High Performance Discovery in Time Series: Techniques and Case Studies**

Tonella and Potrich, **Reverse Engineering of Object Oriented Code**

Iman Hafiz Poernomo
John Newsome Crossley
Martin Wirsing

Adapting Proofs-as-Programs

The Curry–Howard Protocol

Springer

Iman Hafiz Poernomo
Department of Computer Science
King's College London
Strand, London, WC2R 2LS
United Kingdom
iman@dcs.kcl.ac.uk

John Newsome Crossley
School of Computer Science and
Software Engineering
Monash University
Victoria, Australia
jnc@cs.monash.edu.au

Martin Wirsing
Institut für Informatik
Ludwig-Maximilians Universitat München
Oettingenstr. 67
D-80538 München, Germany
wirsing@informatik.uni-muenchen.de

Series Editors:
David Gries
Cornell University
Department of Computer Science
Ithaca, NY 14853
U.S.A.

Fred B. Schneider
Cornell University
Department of Computer Science
Ithaca, NY 14853
U.S.A.

Cover illustration: "Creating Seven" by Soerodipoero Paul Poernomo.

Mathematics Subject Classification (2000): (Primary): 03B15, 68N18, 68Q65 (Secondary): 03B20, 68N30

Library of Congress Cataloging-in-Publication Data
Poernomo, Iman Hafiz, 1976–
Adapting proof-as-programs : the Curry-Howard protocol / Iman Hafiz Poernomo, John Newsome Crossley, Martin Wirsing.
p. cm. — (Monographs in computer science)
Includes bibliographical references and index.
ISBN 0-387-23759-3 (hardback : acid-free paper)
1. Curry-Howard isomorphism. 2. Proof theory. 3. Logic, Symbolic and mathematical. 4. Functional programming (Computer science) 5. Lambda calculus. 6. Abstract data types (Computer science) I. Crossley, John N. II. Wirsing, M. (Martin) III. Title. IV. Series.
QA9.54.P64 2005
511.3′6—dc22 2005046411

ISBN-10: 0-387-23759-3 Printed on acid-free paper.
ISBN-13: 978-0-387-23759-6

© 2005 Springer Science+Business Media Inc.
All rights reserved. This work may not be translated or copied in whole or in part without the written permission of the publisher (Springer Science+Business Media Inc., 233 Spring Street, New York, NY 10013, U.S.A.), except for brief excerpts in connection with reviews or scholarly analysis. Use in connection with any form of information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed is forbidden.
The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

Printed in the United States of America. (TXQ/EB)

9 8 7 6 5 4 3 2 1 SPIN 11303336

springeronline.com

Preface

This book finds new things to do with an old idea. The proofs-as-programs paradigm constitutes a set of approaches to developing programs from proofs in constructive logic. It has been over thirty years since the paradigm was first conceived. At that time, there was a belief that proofs-as-programs had the potential for practical application to semi-automated software development. Initial applications were mostly concerned with fine-grain, mathematical program synthesis. For various reasons, research interest in the area eventually tended toward more theoretic issues of constructive logic and type theory. However, in recent years, the situation has become more balanced, and there is increasingly active research in applying constructive techniques to industrial-scale, complex software engineering problems.

This monograph details several important advances in this direction of practical proofs-as-programs.

One of the central themes of the book is a general, abstract framework for developing new systems of program synthesis by adapting proofs-as-programs to new contexts. Framework-oriented approaches that facilitate analogous approaches to building systems for solving particular problems have been popular and successful. These methods are helpful as they provide a formal toolbox that enables a "roll-your-own" approach to developing solutions. It is hoped that our framework will have a similar impact.

The framework is demonstrated by example. We will give two novel applications of proofs-as-programs to large-scale, coarse-grain software engineering problems: contractual imperative program synthesis and structured program synthesis. These applications constitute an exemplary justification of the framework. Also, in and of themselves, these approaches to synthesis should be interesting for researchers working in the target problem domains.

The monograph serves a dual purpose of providing a state-of-the-art overview of the field and detailing tools and techniques to stimulate further research. The intended audience is graduate students in computer science or mathematics, the proofs-as-programs research community, and the wider computational logic, formal methods, and software engineering communities.

The ideas presented in this monograph originate from research conducted over the past five years by Iman Poernomo and John Crossley at Monash University in collaboration with Martin Wirsing at Ludwig–Maximilians Universität. A significant portion of the monograph is based on the PhD thesis of Iman Poernomo [Poe03b]. Some of the results were presented previously in conference papers and journal articles. Part II of the book presents ideas that resulted from work of Poernomo and Crossley [PC01], [CP01] and [JPBC03]. Part III ellaborates and extends the work done in [Poe99], [PC03] and [Poe03a]. Part IV extends work that was first presented in [CPW00] and [PCW02] by Crossley, Poernomo and Wirsing.

Several people have helped us in the preparation of this book. We are particularly grateful to John Shepherdson and Masahiko Sato for their valuable comments and suggestions for improvement. We also thank John Jeavons, Bolis Basit, Helmut Schwichtenberg, Dirk Pattinson, Stuart Allen and the Nuprl seminar group for fruitful discussions.

Iman Hafiz Poernomo
Caulfield East
May 2004

Overview

The Curry–Howard isomorphism says that intuitionistic logic can be presented as a constructive type theory in which proofs correspond to terms, formulae to types, logical rules to type inference and proof normalization to term simplification. In order to represent intuitionistic proofs, terms of the constructive type theory contain constructive information used to prove formulae. This information can be used to synthesize correct, error-free programs from proofs. Such approaches to program synthesis, based upon the Curry–Howard isomorphism, constitute the area referred to as the proofs-as-programs paradigm.

The advantage of proofs-as-programs techniques is that the task of programming a function is reduced to reasoning with domain knowledge. After more than three decades of research, proofs-as-programs constitutes a mature field with an established theory and set of best practices. State-of-the-art approaches to proofs-as-programs usually involve some form of optimization and extraction strategy, transforming intuitionistic proofs to a commonly used functional programming language that can encode a simply typed lambda calculus, such as *SML*, *Scheme* or *Haskell*.

Work has been done in providing analogous results to the Curry–Howard isomorphism and proofs-as-programs for other logical systems and programming languages. However, little work has been done in identifying a general framework that generalizes the form such analogies should take over arbitrary logical calculi and programming languages. Such a framework would serve as a guide to go about adapting proofs-as-programs to new contexts.

This book defines such a framework, which we call the Curry–Howard protocol. It requires an analogous property to the Curry–Howard isomorphism to hold between a given logic and type theory. However, generalizing state-of-the-art approaches to proofs-as-programs, the protocol requires an optimization and extraction strategy from proofs represented in the logical type theory to programs in a separate programming language. While program synthesis methods have been developed that conform to our protocol, such a framework has not been explicitly identified previously.

We then use the protocol to show how proofs-as-programs can be adapted to two different contexts.

- *Proofs-as-imperative-programs.* The Hoare logic provides a method for the simultaneous development of imperative programs and proofs of their properties. We adapt proofs-as-programs to the Hoare logic for the purpose of extending it to developing imperative programs with side-effect-free return values and views on state.
- *Structured proofs-as-programs.* Structured algebraic specifications are an approach to the compositional design of software systems based on the development of data types. There are proof systems that enable us to reason about structured specifications. We develop such a system and use proofs-as-programs–style techniques for the synthesis of programs from proofs about specifications, and the eventual refinement of specifications into structured code.

These adaptations constitute an exemplary justification for the applicability of the protocol to different contexts.

Contents

Preface v

Overview vii

Part I Prologue

1 Introduction 3
1.1 Proofs-as-programs 4
1.2 Generalizing constructive synthesis 10
1.3 Imperative programs 11
1.4 Structured specifications and programs 16
1.5 Overview 19

Part II Generalizing Proofs-as-Programs

2 Functional Program Synthesis 25
2.1 Abstract data types 27
2.2 Intuitionistic logic 30
2.3 Logical Type Theory 36
2.4 Programs in *SML* 42
2.5 Program synthesis 44
2.6 Example: Password checking system 71
2.7 Discussion 74

3 The Curry–Howard Protocol 77
3.1 From ontology to protocol 77
3.2 Formalizing the ontology 80
3.3 Using the protocol 87
3.4 Discussion 88

Part III Imperative Proofs-as-Programs

III Overview 93

4 Intuitionistic Hoare Logic 95
4.1 Signatures 97
4.2 A subset of *SML* 100
4.3 Semantics of *SML* 105
4.4 Formulae 111
4.5 Calculus 117
4.6 Comparison to Hoare logic with triples and its extensions 127
4.7 Discussion 134

5 Properties of Intuitionistic Hoare Logic 135
5.1 Model theoretic properties 136
5.2 Proof-theory of Intuitionistic Hoare Logic 148
5.3 Example: Electronic Banking System 158
5.4 Discussion 164

6 Proofs-as-Imperative-Programs 165
6.1 Realizability 168
6.2 Extraction 172
6.3 The Curry–Howard protocol for program synthesis 199
6.4 Example: electronic banking system 201
6.5 Example: synthesis of contracts 204
6.6 Discussion 212

Part IV Structured Proofs-as-Programs

7 Reasoning about Structured Specifications 217
7.1 Specifications 219
7.2 Structured specifications 229
7.3 Reasoning about *CASL* specifications 236
7.4 Soundness 247
7.5 Discussion 251

8 Proof-theoretic Properties of SSL 253
8.1 A type theory for SSL 254
8.2 Normalization and proof-term reduction 264
8.3 The Church–Rosser Property 284
8.4 Discussion 295

9 Structured Proofs-as-Programs 297
9.1 Specifying and reasoning about *SML* programs 298
9.2 Realizability 305
9.3 Extracting modular realizers 309
9.4 Extracting extended realizers 328
9.5 Example: password checking system 334
9.6 The Curry–Howard protocol for program synthesis 338
9.7 Discussion 339

10 Generic Specifications 341
10.1 Generic and instantiated specifications 342
10.2 Extensions to logical calculus 348
10.3 Extensions to the logical type theory 352
10.4 Structured proofs-as-programs revisited 358
10.5 Example: warehouse specification 362
10.6 Discussion 364

11 Structured Program Synthesis 367
11.1 Structured programs 369
11.2 Reasoning about structured programs 376
11.3 Using extraction to obtain executable refinements 380
11.4 Discussion 388

Part V Epilogue

12 Conclusions: Toward Constructive Logic as a Practical 4GL 393

Part VI Appendix

A Constructive Logic 399
A.1 Constructive logic 399
A.2 Constructive type theories 403

References 407

Index 417