Cambridge Tracts in Theoretical Computer Science 56

Rippling: Meta-Level Guidance for Mathematical Reasoning

Rippling is a radically new technique for the automation of mathematical reasoning. It is widely applicable whenever a goal is to be proved from one or more syntactically similar givens. The goal is manipulated to resemble the givens more closely, so that they can be used in its proof. The goal is annotated to indicate which subexpressions are to be moved and which are to be left undisturbed. It is the first of many new search-control techniques based on formula annotation; some additional annotated reasoning techniques are also described in the last chapter of the book.

Rippling was developed originally for inductive proofs, where the goal was the induction conclusion and the givens were the induction hypotheses. It has proved applicable to a much wider class of problems: from summing series via analysis to general equational reasoning.

The application to induction has especially important practical implications in the building of dependable IT systems. Induction is required to reason about repetition, whether this arises from loops in programs, recursive data-structures, or the behavior of electronic circuits over time. But inductive proof has resisted automation because of the especially difficult search control problems it introduces, e.g. choosing induction rules, identifying auxiliary lemmas, and generalizing conjectures. Rippling provides a number of exciting solutions to these problems. A failed rippling proof can be analyzed in terms of its expected structure to suggest a patch. These patches automate so called "eureka" steps, e.g. suggesting new lemmas, generalizations, or induction rules.

This systematic and comprehensive introduction to rippling, and to the wider subject of automated inductive theorem proof, will be welcomed by researchers and graduate students alike.

Cambridge Tracts in Theoretical Computer Science 56

Titles in the series

1. G. Chaitin *Algorithmic Information Theory*
2. L. C. Paulson *Logic and Computation*
3. M. Spivey *Understanding Z*
5. A. Ramsey *Formal Methods in Artificial Intelligence*
6. S. Vickers *Topology via Logic*
7. J. Y. Girard, Y. Lafont & P. Taylor *Proofs and Types*
8. J. Clifford *Formal Semantics & Progmatics for Natural Language Processing*
9. M. Winslett *Updating Logical Databases*
10. S. McEvoy & J. V. Tucker (eds.) *Theoretical Foundations of VLSI Design*
11. T. H. Tse *A Unifying Framework for Structured Analysis and Design Models*
12. O. Brewka *Nonmonotonic Reasoning*
14. S. G. Hoggar *Mathematics for Computer Graphics*
15. O. Dasgupta *Design Theory and Computer Science*
17. J. C. M. Baeten (ed.) *Applications of Process Algebra*
18. J. C. M. Baeten & W. D. Weijland *Process Algebra*
19. M. Manzano *Extensions of First Order Logic*
21. D. A. Wolfram *The Clausal Theory of Types*
22. V. Stoltenberg-Hansen, Lindström & E. Griffor *Mathematical Theory of Domains*
23. E. R. Olderog *Nets, Terms and Formulas*
26. P. D. Mosses *Action Semantics*
27. W. H. Hesselink *Programs, Recursion and Unbounded Choice*
28. P. Padawitz *Deductive and Declarative Programming*
29. P. Gärdenfors (ed.) *Belief Revision*
30. M. Anthony & N. Biggs *Computational Learning Theory*
31. T. F. Melham *Higher Order Logic and Hardware Verification*
32. R. L. Carpenter *The Logic of Typed Feature Structures*
33. E. G. Manes *Predicate Transformer Semantics*
34. F. Nielson & H. R. Nielson *Two Level Functional Languages*
35. L. Feijs & J. Jonkers *Formal Specification and Design*
36. S. Mauw & G. J. Veltink (eds.) *Algebraic Specification of Communication Protocols*
37. V. Stavridou *Formal Methods in Circuit Design*
38. N. Shankar *Metamathematics, Machines and Gödel's Proof*
39. J. D. Paris *The Uncertain Reasoner's Companion*
40. J. Dessel & J. Esparza *Free Choice Petri Nets*
41. J. J. Ch. Meyer & W. van der Hoek *Epistemic Logic for AI and Computer Science*
42. J. R. Hindley *Basic Simple Type Theory*
43. A. Troelstra & H. Schwichtenberg *Basic Proof Theory*
44. J. Barwise & J. Seligman *Information Flow*
45. A. Asperti & S. Guerrini *The Optimal Implementation of Functional Programming Languages*
46. R. M. Amadio & P. L. Curien *Domains and Lambda-Calculi*
47. W. I. de Roever & K. Engelhardt *Data Refinement*
48. H. Kleine Büning & F. Lettman *Propositional Logic*
49. L. Novak & A. Gibbons *Hybrid Graph Theory and Network Analysis*
51. H. Simmons *Derivation and Computation*
52. A. S. Troelstra & H. Schwictenberg *Basic Proof Theory* (Second Edition)
53. P. Blackburn, M. de Rijke & Y. Venema *Modal Logic*
54. W. I. de Roever, • de Boer, U. Hannemann, J. Hooman, K. Lakhnech, M. Poel & • Zwiers *Concurrency Verification*
55. Terese *Term Rewriting Systems*

Rippling

Meta-Level Guidance for Mathematical Reasoning

ALAN BUNDY
University of Edinburgh

DAVID BASIN
ETH Zürich

DIETER HUTTER
DFKI Saarbrücken

ANDREW IRELAND
Heriot-Watt University

CAMBRIDGE
UNIVERSITY PRESS

CAMBRIDGE UNIVERSITY PRESS
Cambridge, New York, Melbourne, Madrid, Cape Town, Singapore, São Paulo

CAMBRIDGE UNIVERSITY PRESS
The Edinburgh Building, Cambridge CB2 2RU, UK

Published in the United States of America by Cambridge University Press, New York

http://www.cambridge.org
Information on this title: www.cambridge.org/9780521834490

First published 2005

Printed in the United Kingdom at the University Press, Cambridge

Typeface Times 10/13 pt. *System* LaTeX 2_ε [TB]

A catalog record for this book is available from the British Library

Library of Congress Cataloging in Publication data

ISBN-13 978-0-521-83449-0 hardback
ISBN-10 0-521-83449-X hardback

Cambridge University Press
052183449X - Rippling: Meta-Level Guidance for Mathematical Reasoning
Alan Bundy, David Basin, Dieter Hutter and Andrew Ireland
Frontmatter
More information

To our wives, Josie, Lone, Barbara, and Maria,
for their patience during this 10 year project.

Contents

Preface

Automated theorem proving has been an active research area since the 1950s when researchers began to tackle the problem of automating human-like reasoning. Different techniques were developed early on to automate the use of deduction to show that a goal follows from givens. Deduction could be used to solve problems, play games, or to construct formal, mathematical proofs. In the 1960s and 1970s, interest in automated theorem proving grew, driven by theoretical advances like the development of resolution as well as the growing interest in program verification.

Verification, and more generally, the practical use of formal methods, has raised a number of challenges for the theorem-proving community. One of the major challenges is induction. Induction is required to reason about repetition. In programs, this arises when reasoning about loops and recursion. In hardware, this arises when reasoning about parameterized circuits built from subcomponents in a uniform way, or alternatively when reasoning about the time-dependent behavior of sequential systems.

Carrying out proofs by induction is difficult. Unlike standard proofs in first-order theories, inductive proofs often require the speculation of auxiliary lemmas. This includes both generalizing the conjecture to be proven and speculating and proving additional lemmas about recursively defined functions used in the proof. When induction is not structural induction over data types, then proof search is also complicated by the need to provide a well-founded order over which the induction is performed. As a consequence of these complications, inductive proofs are often carried out interactively rather than fully automatically.

In the late 1980s, a new theorem-proving paradigm was proposed, that of *proof planning*. In proof planning, rather than proving a conjecture by reasoning at the level of primitive inference steps in a deductive system, one could reason about and compose high-level strategies for constructing proofs.

The composite strategy could afterwards be directly mapped into sequences of primitive inferences. This technique was motivated by studying inductive proofs and was applied with considerable success to problems in this domain. Proof planning is based on the observation that most proofs follow a common pattern. In proofs by induction, if the inductive step is to be proven, then the induction conclusion (the goal to be proved) must be transformed in such a way that one can appeal to the induction hypothesis (the given). Moreover, and perhaps surprisingly, this transformation process, called *rippling*, can be formalized as a precise but general strategy.

Rippling is based on the idea that the induction hypothesis (or more generally hypotheses) is syntactically similar to the induction conclusion. In particular, an image of the hypothesis is embedded in the conclusion, along with additional differences, e.g., x might be replaced by $x + 1$ in a proof by induction on x over the natural numbers. Rippling is designed to use rewrite rules to move just the differences (here "$+1$") through the induction conclusion in a way that makes progress in minimizing the difference with the induction hypothesis. In Chapter 1 we introduce and further motivate rippling.

From this initially simple idea, rippling has been extended and generalized in a wide variety of ways, while retaining the strong control on search, which ensures termination and minimizes the need for backtracking. In Chapter 2 we describe some of these extensions to rippling including the application of rippling to proving noninductive theorems.

In contrast to most other proof strategies in automated deduction, rippling imposes a strong expectation on the shape of the proof under development. As previously explained, in each proof step the induction hypothesis must be embedded in the induction conclusion and the conclusion is manipulated so that the proof progresses in reducing the differences. Proof failures usually appear as missing or mismatching rewrite rules, whose absence hinders proof progress. Alternatively, the reason for failure might also be a suboptimal choice of an induction ordering, a missing case analysis, or an over-specific formulation of the conjecture. Comparing the expectations of how a proof should proceed with the failed proof attempt, so-called *critics* reason about the possible reasons for the failure and then suggest possible solutions. In many cases this results in a patch to the proof that allows the prover to make progress. In Chapter 3 we describe how these proof critics use failure in a productive way.

Since rippling is designed to control the proof search using the restrictions mentioned above, it strongly restricts search, and even long and complex proofs can be found quickly. In Chapter 5 we present case studies exemplifying the abilities of rippling. This includes its successes as well as its failures, e.g., cases where the restrictions are too strong and thereby prohibit finding proofs. We also present examples outside of inductive theorem-proving where rippling is used as a general procedure to automate deduction.

The above-mentioned chapters introduce techniques, extensions, and case studies on using rippling in an informal way, and provide a good overview of rippling and its advantages. In contrast, in Chapters 4 and 6 we formalize rippling as well as extending it to a more general and powerful proof methodology. The casual reader may choose to skip these chapters on the first reading.

In Chapter 4 we present the formal theory underlying rippling. In the same way in which sorts were integrated into logical calculi at the end of the 1970s, rippling is based on a specialized calculus that maintains the required contextual information. The restrictions on embeddings are automatically enforced by using a specialized matching algorithm while the knowledge about differences between the hypothesis and the conclusion is automatically propagated during deduction. The explicit representation of differences inside of formulas allows for the definition of well-founded orderings on formulas that are used to guarantee the termination of the rippling process.

Rippling is a successful example of the paradigm of using domain knowledge to restrict proof search. Domain-specific information about, for example, the difference between the induction conclusion and the induction hypothesis, is represented using term annotation and manipulated by rules of a calculus. In Chapter 6 we generalize the idea of rippling in two directions. First, we generalize the kinds of contextual information that can be represented by annotation, and we generalize the calculus used to manipulate annotation. The result is a generic calculus that supports the formalization of contextual information as annotations on individual symbol occurrences, and provides a flexible way to define how these annotations are manipulated during deduction. Second, we show how the various approaches to guiding proof search can be subsumed by this generalized view of rippling. This results in a whole family of new techniques to manage deduction using annotations.

In addition to this book there is a web site on the Internet at

`http://www.rippling.org`

that provides additional examples and tools implementing rippling. We encourage our readers to experiment with these tools.

Acknowledgments

We are grateful to Rafael Accorsi, Serge Autexier, Achim Brucker, Simon Colton, Lucas Dixon, Jurgen Doser, Bill Ellis, Andy Fugard, Lilia Georgieva, Benjamin Gorry, Felix Klaedtke, Boris Köpf, Torsten Lodderstedt, Ewen Maclean, Roy McCasland, Fiona McNeil, Raul Monroy, Axel Schairer, Jan Smaus, Graham Steel, Luca Viganó and Jürgen Zimmer, who read previous versions of this book and contributed to the book by stimulating discussions and comments. An especial thanks to Ewen Maclean for help with LaTeX.

We thank our editor David Tranah for the offer to publish this book at Cambridge University Press and for his patience during its preparation.

Finally, we thank Berendina Schermers van Straalen from the Rights and Permissions Department of Kluwer Academic Publishers who kindly granted us the right to make use of former journal publications at Kluwer.