

Cambridge University Press
978-0-521-87940-8 - Programming with Higher-Order Logic
Dale Miller and Gopalan Nadathur
Frontmatter
[More information](#)

Programming with Higher-Order Logic

Formal systems that describe computations over syntactic structures occur frequently in computer science. Logic programming provides a natural framework for encoding and animating such systems. However, these systems often embody variable binding, a notion that must be treated carefully at a computational level. This book aims to show that a programming language based on a simply typed version of higher-order logic provides an elegant and declarative means for realizing such a treatment. Three broad topics are covered in pursuit of this goal. First, a proof-theoretic framework that supports a general view of logic programming is identified. Second, an actual language called λ Prolog is developed by applying this view to a higher-order logic. Finally, a methodology for computing with specifications is exposed by showing how several computations over formal objects such as logical formulas, functional programs, λ -terms, and π -calculus expressions can be encoded in λ Prolog.

DALE MILLER is Director of Research at INRIA-Saclay and LIX, École Polytechnique where he is the Scientific Leader of the Parsifal team. He has been a professor at the University of Pennsylvania, the Pennsylvania State University, and the École Polytechnique, France. Miller is the Editor-in-Chief of the *ACM Transactions on Computational Logic* and has editorial duties on several other journals. He was awarded an ERC Advanced Grant in 2011 and is the recipient of the 2011 Test-of-Time award of the IEEE Symposium on Logic in Computer Science. He works on many topics in the general area of computational logic, including automated reasoning, logic programming, proof theory, unification theory, operational semantics, and, most recently, proof certificates.

GOPALAN NADATHUR is a Professor of Computer Science at the University of Minnesota. He has previously held faculty appointments at Duke University, University of Chicago, and Loyola University of Chicago. Nadathur's research interests span the areas of computational logic, programming languages, and logic programming. His work has been regularly funded by the National Science Foundation and has appeared in the *Journal of the Association of Computing Machinery*, *Information and Computation*, the *Journal of Automated Reasoning*, *Theoretical Computer Science*, and *Theory and Practice of Logic Programming* among other places.

Cambridge University Press
978-0-521-87940-8 - Programming with Higher-Order Logic
Dale Miller and Gopalan Nadathur
Frontmatter
[More information](#)

Cambridge University Press
978-0-521-87940-8 - Programming with Higher-Order Logic
Dale Miller and Gopalan Nadathur
Frontmatter
[More information](#)

PROGRAMMING WITH HIGHER-ORDER LOGIC

DALE MILLER
*INRIA-Saclay, Île de France &
LIX, École Polytechnique*

GOPALAN NADATHUR
University of Minnesota



CAMBRIDGE
UNIVERSITY PRESS

Cambridge University Press
978-0-521-87940-8 - Programming with Higher-Order Logic
Dale Miller and Gopalan Nadathur
Frontmatter
[More information](#)

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

Cambridge University Press
32 Avenue of the Americas, New York, NY 10013-2473, USA

www.cambridge.org
Information on this title: www.cambridge.org/9780521879408

© Dale Miller and Gopalan Nadathur 2012

This publication is in copyright. Subject to statutory exception
and to the provisions of relevant collective licensing agreements,
no reproduction of any part may take place without the written
permission of Cambridge University Press.

First published 2012

Printed in the United States of America

A catalog record for this publication is available from the British Library.

ISBN 978-0-521-87940-8 Hardback

Cambridge University Press has no responsibility for the persistence or accuracy of URLs
for external or third-party Internet Web sites referred to in this publication and does not
guarantee that any content on such Web sites is, or will remain, accurate or appropriate.

Cambridge University Press
978-0-521-87940-8 - Programming with Higher-Order Logic
Dale Miller and Gopalan Nadathur
Frontmatter
[More information](#)

To Catuscia, Nadia, and Alexis
— Dale

To the memory of my parents
— Gopalan

Cambridge University Press
978-0-521-87940-8 - Programming with Higher-Order Logic
Dale Miller and Gopalan Nadathur
Frontmatter
[More information](#)

Contents

<i>Preface</i>	<i>page xi</i>
Introduction	1
I.1 Connections between logic and computation	1
I.2 Logical primitives and programming expressivity	3
I.3 The meaning of <i>higher-order logic</i>	5
I.4 Presentation style	6
I.5 Prerequisites	7
I.6 Organization of the book	8
1 First-Order Terms and Representations of Data	10
1.1 Sorts and type constructors	10
1.2 Type expressions	12
1.3 Typed first-order terms	14
1.4 Representing symbolic objects	19
1.5 Unification of typed first-order terms	26
1.6 Bibliographic notes	32
2 First-Order Horn Clauses	34
2.1 First-order formulas	34
2.2 Logic programming and search semantics	38
2.3 Horn clauses and their computational interpretation	43
2.4 Programming with first-order Horn clauses	46
2.5 Pragmatic aspects of computing with Horn clauses	60
2.6 The relationship with logical notions	62
2.7 The meaning and use of types	66
2.8 Bibliographic notes	72

viii	<i>Contents</i>	
3	First-Order Hereditary Harrop Formulas	75
3.1	The syntax of goals and program clauses	75
3.2	Implicational goals	77
3.3	Universally quantified goals	82
3.4	The relationship with logical notions	88
3.5	Bibliographic notes	93
4	Typed λ-Terms and Formulas	96
4.1	Syntax for λ -terms and formulas	97
4.2	The rules of λ -conversion	100
4.3	Some properties of λ -conversion	101
4.4	Unification problems as quantified equalities	105
4.5	Solving unification problems	111
4.6	Some hard unification problems	113
4.7	Bibliographic notes	116
5	Using Quantification at Higher-Order Types	118
5.1	Atomic formulas in higher-order logic programs	118
5.2	Higher-order logic programming languages	122
5.3	Examples of higher-order programming	126
5.4	Flexible atoms as goals	132
5.5	Reasoning about higher-order programs	134
5.6	Defining some of the logical constants	136
5.7	The conditional and negation-as-failure	137
5.8	Using λ -terms as functions	138
5.9	Higher-order unification is not a panacea	143
5.10	Comparison with functional programming	146
5.11	Bibliographic notes	147
6	Mechanisms for Structuring Large Programs	150
6.1	Desiderata for modular programming	150
6.2	A modules language	151
6.3	Matching signatures and modules	156
6.4	The logical interpretation of modules	160
6.5	Some programming aspects of the modules language	165
6.6	Implementation considerations	172
6.7	Bibliographic notes	173
7	Computations over λ-Terms	175
7.1	Representing objects with binding structure	175
7.2	Realizing object-level substitution	180
7.3	Mobility of binders	183
7.4	Computing with untyped λ -terms	185

Contents

ix

7.5	Computations over first-order formulas	195
7.6	Specifying object-level substitution	199
7.7	The λ -tree approach to abstract syntax	203
7.8	The L_λ subset of λ Prolog	204
7.9	Bibliographic notes	208
8	Unification of λ-Terms	211
8.1	Properties of the higher-order unification problem	212
8.2	A procedure for checking for unifiability	214
8.3	Higher-order pattern unification	220
8.4	Pragmatic aspects of higher-order unification	224
8.5	Bibliographic notes	226
9	Implementing Proof Systems	229
9.1	Deduction in propositional intuitionistic logic	229
9.2	Encoding natural deduction for intuitionistic logic	231
9.3	A theorem prover for classical logic	235
9.4	A general architecture for theorem provers	240
9.5	Bibliographic notes	245
10	Computations over Functional Programs	247
10.1	The miniFP programming language	247
10.2	Specifying evaluation for miniFP programs	250
10.3	Manipulating functional programs	255
10.4	Bibliographic notes	258
11	Encoding a Process Calculus Language	261
11.1	Representing the expressions of the π -calculus	261
11.2	Specifying one-step transitions	263
11.3	Animating π -calculus expressions	266
11.4	May- versus must-judgments	269
11.5	Mapping the λ -calculus into the π -calculus	273
11.6	Bibliographic notes	275
	Appendix: The Teyjus System	277
A.1	An overview of the Teyjus system	277
A.2	Interacting with the Teyjus system	278
A.3	Using modules within the Teyjus system	283
A.4	Special features of the Teyjus system	285
	<i>Bibliography</i>	289
	<i>Index</i>	301

Cambridge University Press
978-0-521-87940-8 - Programming with Higher-Order Logic
Dale Miller and Gopalan Nadathur
Frontmatter
[More information](#)

Preface

Formal systems in computer science frequently involve specifications of computations over syntactic structures such as λ -terms, π -calculus expressions, first-order formulas, types, and proofs. This book is concerned, in part, with using higher-order logic to express such specifications. Properties are often associated with expressions by formal systems via syntax-based inference rules. Examples of such descriptions include presentations of typing and operational semantics. Logic programming, with its orientation around rule-based specifications, provides a natural framework for encoding and animating these kinds of descriptions. Variable binding is integral to most syntactic expressions, and its presence typically translates into side conditions accompanying inference rules. While many of the concepts related to binding, such as variable renaming, substitution, and scoping, are logically well understood, their treatment at a programming level is surprisingly difficult. We show here that a programming language based on a simply typed version of higher-order logic provides an elegant approach to performing computations over structures embodying binding.

The agenda just described has a prerequisite: We must be able to make sense of a higher-order logic as a programming language. This is a nontrivial task that defines a second theme that permeates this book. Usual developments of logic programming are oriented around formulas in clausal form with resolution as the sole inference rule. Sometimes a semantics-based presentation is also used, expanding typically into the idea of minimal (Herbrand) models. Neither of these approaches is suitable in a higher-order setting: Model theory is not a well-developed tool here, and substitutions for predicate variables that can appear in a higher-order logic can take formulas in a restricted form, such as the conjunctive-normal clausal form, into new formulas that no longer adhere to this form. Faced with this situation, we have turned in our work to the sequent calculus of Gentzen. We have found this to be a versatile and flexible tool for understanding and analyzing the metatheory and computational properties of

logics. Using it, we have been able to identify logic programming languages as ones that support a particular goal-directed approach to proof search. This viewpoint allows us to extend naturally the Horn clause logic that underlies languages such as Prolog to richer first-order logics that offer support at the programming level to scoping mechanisms. The same approach generalizes to higher-order logic and also to contexts that we do not explicitly treat here, such as linear logic and the dependently typed λ -calculus. Indeed, understanding proof search through the perspective of the sequent calculus seems to be an essential part of grasping the significance of logic as a tool for computing. We accordingly expose this line of thinking as we develop a higher-order logic for programming.

Gaining facility with new ideas in programming usually requires concrete experimentation with them. Many of the ideas that we expose here have an actual realization in the language λ Prolog. Programs written in λ Prolog often will be used to illuminate discussions of logic and theoretical principles. These programs can be run using the Teyjus implementation of λ Prolog that we provide an introduction to in the Appendix. The Teyjus system can be freely downloaded, and the distribution material accompanying it contains many programs, including the ones discussed in this book, that illustrate the special capabilities of λ Prolog. We anticipate that a reader of this book eventually will have enough expertise to develop his or her own programs in a number of application areas where binding is an important part of syntactic structure.

This book, then, covers three broad topics: a proof search-based view of computation, a higher-order logic-based approach to programming, and a particular language that realizes these ideas. We hope to leave the reader in the end with an appreciation of how higher-order logic may be used to specify computations and with the ability to use a logic programming language based on such a logic to build actual systems. We believe that this kind of background is becoming increasingly useful as demands of the programming process get more sophisticated. One pertinent application area is that where logic and deduction function as “gatekeepers” that ensure the security and integrity of lower-level processes; a specific example of this kind appears within the proof-carrying-code framework that has been proposed as a vehicle for ensuring, for instance, the safety of mobile code. Another application area is the mechanization of the metatheory of logics and languages that is the focus, for example, of the recently posed POPLmark challenge. This book develops a fruitful approach to specifying computations over logical expressions and program phrases, all of which are central to such metatheoretic manipulations. The λ Prolog language also provides a means for prototyping and implementing such specifications. While we do not discuss this issue significantly in this book, these λ Prolog specifications should further facilitate rich and interesting new approaches to

reasoning about metatheoretic properties of the logics and languages encoded in them.

The material we present here owes a lot to collaborations with colleagues. The foundational ideas relating to logic programming were developed in the late 1980s in interactions with Frank Pfenning and Andre Scedrov. We also received valuable input from Natarajan Shankar during this phase. A Prolog-based implementation of λ Prolog followed soon after. The understanding we now have of the capabilities of this language derives significantly from the experiments conducted with this system by Amy Felty, Elsa Gunter, John Hannan, Fernando Pereira, Remo Pereschi, and Frank Pfenning, among many other researchers who we surely err in not mentioning explicitly. The language subsequently has received implementations in Common Lisp by Conal Elliott and Frank Pfenning; in C by Pascal Brisset and Olivier Ridoux; and in Standard ML by Conal Elliott, Amy Felty, Dale Miller, Frank Pfenning, and Philip Wickline. Gopalan Nadathur has led a long-term project focused on compiling this language in which Andrew Gacek, Steven Holte, Bharat Jayaraman, Keehang Kwon, Dustin Mitchell, Xiaochu Qi, and Zachary Snow have participated. This work has resulted in two different versions of the Teyjus system. We have received comments and helpful suggestions on this book from Jim Blandy, Iliano Cervesato, Giorgio Delzanno, Joern Dinkla, Zhiping Duan, Daniel Friedman, Andrew Gacek, Clément Houtmann, H. Krishnapriyan, Gary Leavens, Chuck Liang, Jim Lipton, Tong Mei, Catuscia Palamidessi, Olivier Ridoux, Jenny Simon, and Yuting Wang. INRIA has provided support during the writing of this book by facilitating a sabbatical visit by Gopalan Nadathur and through its “Equipes Associées” Slimmer. The National Science Foundation has funded the development of the ideas we present through grants at various points, most recently through the Grants CCF-0429572 and NSF/CCF-0917140. Any opinions, findings, and conclusions or recommendations expressed in this book are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Palaiseau, France
Minneapolis, MN, USA
October 2011

Dale Miller
Gopalan Nadathur

Cambridge University Press
978-0-521-87940-8 - Programming with Higher-Order Logic
Dale Miller and Gopalan Nadathur
Frontmatter
[More information](#)
