

## Practical Foundations for Programming Languages

---

Types are the central organizing principle of the theory of programming languages. In this innovative book, Professor Robert Harper offers a fresh perspective on the fundamentals of these languages through the use of type theory. Whereas most textbooks on the subject emphasize taxonomy, Harper instead emphasizes genetics, examining the building blocks from which all programming languages are constructed.

Language features are manifestations of type structure. The syntax of a language is governed by the constructs that define its types, and its semantics is determined by the interactions among those constructs. The soundness of a language design – the absence of ill-defined programs – follows naturally.

Professor Harper's presentation is simultaneously rigorous and intuitive, relying on only elementary mathematics. The framework he outlines scales easily to a rich variety of language concepts and is directly applicable to their implementation. The result is a lucid introduction to programming theory that is both accessible and practical.

**Robert Harper** has been a member of the faculty of Computer Science at Carnegie Mellon University since 1988. His main research interest is in the application of type theory to the design and implementation of programming languages and to the development of systems for mechanization of mathematics. Professor Harper is a recipient of the Allen Newell Medal for Research Excellence and the Herbert A. Simon Award for Teaching Excellence at Carnegie Mellon, and he is a Fellow of the Association for Computing Machinery.

Cambridge University Press  
978-1-107-02957-6 - Practical Foundations for Programming Languages  
Robert Harper  
Frontmatter  
[More information](#)

---

# Practical Foundations for Programming Languages

---

Robert Harper



Cambridge University Press  
978-1-107-02957-6 - Practical Foundations for Programming Languages  
Robert Harper  
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](http://www.cambridge.org)  
Information on this title: [www.cambridge.org/9781107029576](http://www.cambridge.org/9781107029576)

© Robert Harper 2013

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 2013

Printed in the United States of America

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

*Library of Congress Cataloging in Publication Data*

Harper, Robert.  
Practical foundations for programming languages / Robert Harper.  
p. cm.

Includes bibliographical references and index.

ISBN 978-1-107-02957-6 (hardback)

1. Programming languages (Electronic computers) I. Title.

QA76.7.H377 2012

005.13—dc23 2012018404

ISBN 978-1-107-02957-6 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.

# Contents

*Preface*

*page xvii*

## Part I Judgments and Rules

1	Syntactic Objects	3
1.1	Abstract Syntax Trees	3
1.2	Abstract Binding Trees	6
1.3	Notes	10
2	Inductive Definitions	11
2.1	Judgments	11
2.2	Inference Rules	11
2.3	Derivations	13
2.4	Rule Induction	14
2.5	Iterated and Simultaneous Inductive Definitions	16
2.6	Defining Functions by Rules	17
2.7	Modes	18
2.8	Notes	19
3	Hypothetical and General Judgments	20
3.1	Hypothetical Judgments	20
3.1.1	Derivability	20
3.1.2	Admissibility	22
3.2	Hypothetical Inductive Definitions	23
3.3	General Judgments	24
3.4	Generic Inductive Definitions	26
3.5	Notes	27

## Part II Statics and Dynamics

4	Statics	31
4.1	Syntax	31
4.2	Type System	32
4.3	Structural Properties	33
4.4	Notes	35

5	Dynamics	36
5.1	Transition Systems	36
5.2	Structural Dynamics	37
5.3	Contextual Dynamics	39
5.4	Equational Dynamics	41
5.5	Notes	43
6	Type Safety	45
6.1	Preservation	45
6.2	Progress	46
6.3	Run-Time Errors	47
6.4	Notes	48
7	Evaluation Dynamics	50
7.1	Evaluation Dynamics	50
7.2	Relating Structural and Evaluation Dynamics	51
7.3	Type Safety, Revisited	52
7.4	Cost Dynamics	53
7.5	Notes	54
<b>Part III Function Types</b>		
8	Function Definitions and Values	57
8.1	First-Order Functions	57
8.2	Higher-Order Functions	59
8.3	Evaluation Dynamics and Definitional Equality	61
8.4	Dynamic Scope	62
8.5	Notes	63
9	Gödel's <b>T</b>	64
9.1	Statics	64
9.2	Dynamics	65
9.3	Definability	66
9.4	Undefinability	68
9.5	Notes	70
10	Plotkin's PCF	71
10.1	Statics	72
10.2	Dynamics	73
10.3	Definability	75
10.4	Notes	77

## Part IV Finite Data Types

11	Product Types	81
11.1	Nullary and Binary Products	81
11.2	Finite Products	83
11.3	Primitive and Mutual Recursions	84
11.4	Notes	85
12	Sum Types	86
12.1	Nullary and Binary Sums	86
12.2	Finite Sums	88
12.3	Applications of Sum Types	89
12.3.1	Void and Unit	89
12.3.2	Booleans	89
12.3.3	Enumerations	90
12.3.4	Options	91
12.4	Notes	92
13	Pattern Matching	93
13.1	A Pattern Language	94
13.2	Statics	94
13.3	Dynamics	95
13.4	Exhaustiveness and Redundancy	97
13.4.1	Match Constraints	97
13.4.2	Enforcing Exhaustiveness and Redundancy	99
13.4.3	Checking Exhaustiveness and Redundancy	100
13.5	Notes	101
14	Generic Programming	102
14.1	Introduction	102
14.2	Type Operators	103
14.3	Generic Extension	103
14.4	Notes	105

## Part V Infinite Data Types

15	Inductive and Coinductive Types	109
15.1	Motivating Examples	109
15.2	Statics	112
15.2.1	Types	112
15.2.2	Expressions	113
15.3	Dynamics	114
15.4	Notes	114

16	Recursive Types	116
16.1	Solving Type Isomorphisms	117
16.2	Recursive Data Structures	118
16.3	Self-Reference	120
16.4	The Origin of State	121
16.5	Notes	123

## Part VI Dynamic Types

17	The Untyped $\lambda$ -Calculus	127
17.1	The $\lambda$ -Calculus	127
17.2	Definability	128
17.3	Scott's Theorem	131
17.4	Untyped Means Untyped	132
17.5	Notes	133
18	Dynamic Typing	134
18.1	Dynamically Typed PCF	134
18.2	Variations and Extensions	137
18.3	Critique of Dynamic Typing	140
18.4	Notes	141
19	Hybrid Typing	142
19.1	A Hybrid Language	142
19.2	Dynamics as Static Typing	144
19.3	Optimization of Dynamic Typing	145
19.4	Static Versus Dynamic Typing	147
19.5	Notes	148

## Part VII Variable Types

20	Girard's System <b>F</b>	151
20.1	System <b>F</b>	151
20.2	Polymorphic Definability	155
20.2.1	Products and Sums	155
20.2.2	Natural Numbers	156
20.3	Parametricity Overview	157
20.4	Restricted Forms of Polymorphism	158
20.4.1	Predicative Fragment	158
20.4.2	Prenex Fragment	159
20.4.3	Rank-Restricted Fragments	160
20.5	Notes	161



21	Abstract Types	162
21.1	Existential Types	162
21.1.1	Statics	163
21.1.2	Dynamics	164
21.1.3	Safety	164
21.2	Data Abstraction Via Existentials	165
21.3	Definability of Existentials	166
21.4	Representation Independence	167
21.5	Notes	169
22	Constructors and Kinds	170
22.1	Statics	171
22.2	Higher Kinds	173
22.3	Canonizing Substitution	174
22.4	Canonization	176
22.5	Notes	178
<b>Part VIII Subtyping</b>		
23	Subtyping	181
23.1	Subsumption	181
23.2	Varieties of Subtyping	182
23.2.1	Numeric Types	182
23.2.2	Product Types	183
23.2.3	Sum Types	183
23.3	Variance	184
23.3.1	Product and Sum Types	184
23.3.2	Function Types	184
23.3.3	Quantified Types	185
23.3.4	Recursive Types	186
23.4	Safety	188
23.5	Notes	189
24	Singleton Kinds	190
24.1	Overview	191
24.2	Singletons	192
24.3	Dependent Kinds	194
24.4	Higher Singletons	197
24.5	Notes	198
<b>Part IX Classes and Methods</b>		
25	Dynamic Dispatch	201
25.1	The Dispatch Matrix	202

25.2	Class-Based Organization	203
25.3	Method-Based Organization	205
25.4	Self-Reference	206
25.5	Notes	208
26	Inheritance	209
26.1	Class and Method Extension	209
26.2	Class-Based Inheritance	210
26.3	Method-Based Inheritance	211
26.4	Notes	212

## Part X Exceptions and Continuations

27	Control Stacks	217
27.1	Machine Definition	217
27.2	Safety	219
27.3	Correctness of the Control Machine	220
	27.3.1 Completeness	221
	27.3.2 Soundness	222
27.4	Notes	223
28	Exceptions	224
28.1	Failures	224
28.2	Exceptions	226
28.3	Exception Type	227
28.4	Encapsulation of Exceptions	228
28.5	Notes	230
29	Continuations	231
29.1	Informal Overview	231
29.2	Semantics of Continuations	233
29.3	Coroutines	235
29.4	Notes	238

## Part XI Types and Propositions

30	Constructive Logic	241
30.1	Constructive Semantics	241
30.2	Constructive Logic	242
	30.2.1 Provability	243
	30.2.2 Proof Terms	244
30.3	Proof Dynamics	246
30.4	Propositions as Types	247
30.5	Notes	247

31	Classical Logic	249
31.1	Classical Logic	250
	31.1.1 Provability and Refutability	250
	31.1.2 Proofs and Refutations	252
31.2	Deriving Elimination Forms	254
31.3	Proof Dynamics	255
31.4	Law of the Excluded Middle	257
31.5	The Double-Negation Translation	258
31.6	Notes	260

## Part XII Symbols

32	Symbols	263
32.1	Symbol Declaration	264
	32.1.1 Scoped Dynamics	264
	32.1.2 Scope-Free Dynamics	265
32.2	Symbolic References	266
	32.2.1 Statics	266
	32.2.2 Dynamics	267
	32.2.3 Safety	267
32.3	Notes	268
33	Fluid Binding	269
33.1	Statics	269
33.2	Dynamics	270
33.3	Type Safety	271
33.4	Some Subtleties	272
33.5	Fluid References	273
33.6	Notes	275
34	Dynamic Classification	276
34.1	Dynamic Classes	277
	34.1.1 Statics	277
	34.1.2 Dynamics	277
	34.1.3 Safety	278
34.2	Class References	278
34.3	Definability of Dynamic Classes	279
34.4	Classifying Secrets	280
34.5	Notes	281

## Part XIII State

35	Modernized Algol	285
35.1	Basic Commands	285

35.1.1	Statics	286
35.1.2	Dynamics	287
35.1.3	Safety	288
35.2	Some Programming Idioms	289
35.3	Typed Commands and Typed Assignables	291
35.4	Notes	293
36	Assignable References	295
36.1	Capabilities	295
36.2	Scoped Assignables	296
36.3	Free Assignables	298
36.4	Safety for Free Assignables	300
36.5	Benign Effects	302
36.6	Notes	304
<b>Part XIV Laziness</b>		
37	Lazy Evaluation	307
37.1	By-Need Dynamics	308
37.2	Safety	310
37.3	Lazy Data Structures	312
37.4	Suspensions	313
37.5	Notes	315
38	Polarization	316
38.1	Positive and Negative Types	317
38.2	Focusing	318
38.3	Statics	318
38.4	Dynamics	320
38.5	Safety	321
38.6	Notes	322
<b>Part XV Parallelism</b>		
39	Nested Parallelism	325
39.1	Binary Fork–Join	325
39.2	Cost Dynamics	328
39.3	Multiple Fork–Join	331
39.4	Provably Efficient Implementations	333
39.5	Notes	336
40	Futures and Speculations	337
40.1	Futures	337
40.1.1	Statics	338

40.1.2	Sequential Dynamics	338
40.2	Speculations	338
40.2.1	Statics	339
40.2.2	Sequential Dynamics	339
40.3	Parallel Dynamics	339
40.4	Applications of Futures	342
40.5	Notes	344

## Part XVI Concurrency

41	Process Calculus	347
41.1	Actions and Events	347
41.2	Interaction	349
41.3	Replication	351
41.4	Allocating Channels	352
41.5	Communication	354
41.6	Channel Passing	357
41.7	Universality	360
41.8	Notes	361
42	Concurrent Algol	363
42.1	Concurrent Algol	363
42.2	Broadcast Communication	366
42.3	Selective Communication	368
42.4	Free Assignables as Processes	371
42.5	Notes	372
43	Distributed Algol	373
43.1	Statics	373
43.2	Dynamics	375
43.3	Safety	376
43.4	Situated Types	377
43.5	Notes	380

## Part XVII Modularity

44	Components and Linking	383
44.1	Simple Units and Linking	383
44.2	Initialization and Effects	385
44.3	Notes	386
45	Type Abstractions and Type Classes	387
45.1	Type Abstraction	388
45.2	Type Classes	389

45.3	A Module Language	392
45.4	First and Second Class	396
45.5	Notes	397
46	Hierarchy and Parameterization	399
46.1	Hierarchy	399
46.2	Parameterization	402
46.3	Extending Modules With Hierarchies and Parameterization	405
46.4	Applicative Functors	407
46.5	Notes	409
<b>Part XVIII Equational Reasoning</b>		
47	Equational Reasoning for <b>T</b>	413
47.1	Observational Equivalence	413
47.2	Logical Equivalence	417
47.3	Logical and Observational Equivalences Coincide	418
47.4	Some Laws of Equality	421
	47.4.1 General Laws	421
	47.4.2 Equality Laws	422
	47.4.3 Induction Law	422
47.5	Notes	422
48	Equational Reasoning for <b>PCF</b>	424
48.1	Observational Equivalence	424
48.2	Logical Equivalence	425
48.3	Logical and Observational Equivalences Coincide	425
48.4	Compactness	428
48.5	Conatural Numbers	431
48.6	Notes	432
49	Parametricity	433
49.1	Overview	433
49.2	Observational Equivalence	434
49.3	Logical Equivalence	435
49.4	Parametricity Properties	440
49.5	Representation Independence, Revisited	443
49.6	Notes	444
50	Process Equivalence	446
50.1	Process Calculus	446
50.2	Strong Equivalence	448
50.3	Weak Equivalence	451
50.4	Notes	452

---

**Part XIX Appendix**

<i>Appendix: Finite Sets and Finite Functions</i>	455
<i>Bibliography</i>	457
<i>Index</i>	465

Cambridge University Press  
978-1-107-02957-6 - Practical Foundations for Programming Languages  
Robert Harper  
Frontmatter  
[More information](#)

---



## Preface

Types are the central organizing principle of the theory of programming languages. Language features are manifestations of type structure. The syntax of a language is governed by the constructs that define its types, and its semantics is determined by the interactions among those constructs. The soundness of a language design – the absence of ill-defined programs – follows naturally.

The purpose of this book is to explain this remark. A variety of programming language features are analyzed in the unifying framework of type theory. A language feature is defined by its *statics*, the rules governing the use of the feature in a program, and its *dynamics*, the rules defining how programs using this feature are to be executed. The concept of *safety* emerges as the coherence of the statics and the dynamics of a language.

In this way we establish a foundation for the study of programming languages. But why these particular methods? The main justification is provided by the book itself. The methods we use are both *precise* and *intuitive*, providing a uniform framework for explaining programming language concepts. Importantly, these methods *scale* to a wide range of programming language concepts, supporting rigorous analysis of their properties. Although it would require another book in itself to justify this assertion, these methods are also *practical* in that they are *directly applicable* to implementation and *uniquely effective* as a basis for mechanized reasoning. No other framework offers as much.

Being a consolidation and distillation of decades of research, this book does not provide an exhaustive account of the history of the ideas that inform it. Suffice it to say that much of the development is not original, but rather is largely a reformulation of what has gone before. The notes at the end of each chapter signpost the major developments but are not intended as a complete guide to the literature. For further information and alternative perspectives, the reader is referred to such excellent sources as Constable (1986, 1998), Girard (1989), Martin-Löf (1984), Mitchell (1996), Pierce (2002, 2004), and Reynolds (1998).

The book is divided into parts that are, in the main, independent of one another. Parts I and II, however, provide the foundation for the rest of the book and must therefore be considered prior to all other parts. On first reading it may be best to skim Part I and begin in earnest with Part II, returning to Part I for clarification of the logical framework in which the rest of the book is cast.

Numerous people have read and commented on earlier editions of this book and have suggested corrections and improvements to it. I am particularly grateful to Andrew Appel, Iliano Cervesato, Lin Chase, Derek Dreyer, Zhong Shao, and Todd Wilson for their extensive efforts in reading and criticizing the book. I also thank the following people for their suggestions: Arbob Ahmad, Zena Ariola, Eric Bergstrom, Guy Blelloch, William Byrd,

Luis Caires, Luca Cardelli, Manuel Chakravarty, Richard C. Cobbe, Karl Crary, Yi Dai, Daniel Dantas, Anupam Datta, Jake Donham, Favonia, Matthias Felleisen, Kathleen Fisher, Dan Friedman, Peter Gammie, Maia Ginsburg, Byron Hawkins, Kevin Hely, Justin Hsu, Cao Jing, Salil Joshi, Gabriele Keller, Scott Kilpatrick, Danielle Kramer, Akiva Leffert, Ruy Ley-Wild, Dan Licata, Karen Liu, Dave MacQueen, Chris Martens, Greg Morrisett, Tom Murphy, Aleksandar Nanevski, Georg Neis, David Neville, Doug Perkins, Frank Pfenning, Jean Pichon, Benjamin Pierce, Andrew M. Pitts, Gordon Plotkin, David Renshaw, John Reynolds, Carter Schonwald, Dale Schumacher, Dana Scott, Robert Simmons, Pawel Sobocinski, Daniel Spoonhower, Paulo Tanimoto, Peter Thiemann, Bernardo Toninho, Michael Tschantz, Kami Vaniea, Carsten Varming, David Walker, Dan Wang, Jack Wileden, Roger Wolff, Omer Zach, Luke Zarko, and Yu Zhang. I am grateful to the students of 15–312 and 15–814 at Carnegie Mellon, who have provided the impetus for the preparation of this book and who have endured the many revisions to it over the last ten years.

I thank the Max Planck Institute for Software Systems in Germany for its hospitality and support. I also thank Espresso a Mano in Pittsburgh, CB2 Cafe in Cambridge, and Thonet Cafe in Saarbrücken for providing a steady supply of coffee and a conducive atmosphere for writing.

This material is, in part, based on work supported by the National Science Foundation under grants 0702381 and 0716469. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation.

Robert Harper  
Pittsburgh  
March 2012