Practical Foundations for Programming Languages

This text develops a comprehensive theory of programming languages based on type systems and structural operational semantics. Language concepts are precisely defined by their static and dynamic semantics, presenting the essential tools both intuitively and rigorously while relying on only elementary mathematics. These tools are used to analyze and prove properties of languages and provide the framework for combining and comparing language features. The broad range of concepts includes fundamental data types such as sums and products, polymorphic and abstract types, dynamic typing, dynamic dispatch, subtyping and refinement types, symbols and dynamic classification, parallelism and cost semantics, and concurrency and distribution. The methods are directly applicable to language implementation, to the development of logics for reasoning about programs, and to the formal verification language properties such as type safety.

This thoroughly revised second edition includes exercises at the end of nearly every chapter and a new chapter on type refinements.

Robert Harper is a professor in the Computer Science Department at Carnegie Mellon University. His main research interest is in the application of type theory to the design and implementation of programming languages and to the mechanization of their metatheory. Harper is a recipient of the Allen Newell Medal for Research Excellence and the Herbert A. Simon Award for Teaching Excellence, and is an Association for Computing Machinery Fellow.

Practical Foundations for Programming Languages

Second Edition

Robert Harper
Carnegie Mellon University

CAMBRIDGE
UNIVERSITY PRESS

CAMBRIDGE
UNIVERSITY PRESS

32 Avenue of the Americas, New York, NY 10013

Cambridge University Press is part of the University of Cambridge.

It furthers the University's mission by disseminating knowledge in the pursuit of education, learning, and research at the highest international levels of excellence.

www.cambridge.org
Information on this title: www.cambridge.org/9781107150300

© Robert Harper 2016

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 2016

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
Names: Harper, Robert, 1957–
Title: Practical foundations for programming languages / Robert Harper, Carnegie Mellon University.
Description: Second edition. | New York NY : Cambridge University Press, 2016. | Includes bibliographical references and index.
Identifiers: LCCN 2015045380 | ISBN 9781107150300 (alk. paper)
Subjects: LCSH: Programming languages (Electronic computers)
Classification: LCC QA76.7 .H377 2016 | DDC 005.13–dc23
LC record available at http://lccn.loc.gov/2015045380

ISBN 978-1-107-15030-0 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 to the Second Edition

Writing the second edition to a textbook incurs the same risk as building the second version of a software system. It is difficult to make substantive improvements, while avoiding the temptation to overburden and undermine the foundation on which one is building. With the hope of avoiding the second system effect, I have sought to make corrections, revisions, expansions, and deletions that improve the coherence of the development, remove some topics that distract from the main themes, add new topics that were omitted from the first edition, and include exercises for almost every chapter.

The revision removes a number of typographical errors, corrects a few material errors (especially the formulation of the parallel abstract machine and of concurrency in Algol), and improves the writing throughout. Some chapters have been deleted (general pattern matching and polarization, restricted forms of polymorphism), some have been completely rewritten (the chapter on higher kinds), some have been substantially revised (general and parametric inductive definitions, concurrent and distributed Algol), several have been reorganized (to better distinguish partial from total type theories), and a new chapter has been added (on type refinements). Titular attributions on several chapters have been removed, not to diminish credit, but to avoid confusion between the present and the original formulations of several topics. A new system of (pronounceable!) language names has been introduced throughout. The exercises generally seek to expand on the ideas in the main text, and their solutions often involve significant technical ideas that merit study. Routine exercises of the kind one might include in a homework assignment are deliberately few.

My purpose in writing this book is to establish a comprehensive framework for formulating and analyzing a broad range of ideas in programming languages. If language design and programming methodology are to advance from a trade-craft to a rigorous discipline, it is essential that we first get the definitions right. Then, and only then, can there be meaningful analysis and consolidation of ideas. My hope is that I have helped to build such a foundation.

I am grateful to Stephen Brookes, Evan Cavallo, Karl Crary, Jon Sterling, James R. Wilcox and Todd Wilson for their help in critiquing drafts of this edition and for their suggestions for modification and revision. I thank my department head, Frank Pfenning, for his support of my work on the completion of this edition. Thanks also to my editors, Ada Brunstein and Lauren Cowles, for their guidance and assistance. And thanks to Andrew Shulaev for corrections to the draft.

Neither the author nor the publisher make any warranty, express or implied, that the definitions, theorems, and proofs contained in this volume are free of error, or are consistent with any particular standard of merchantability, or that they will meet requirements for any particular application. They should not be relied on for solving a problem whose incorrect solution could result in injury to a person or loss of property. If you do use this material in such a manner, it is at your own risk. The author and publisher disclaim all liability for direct or consequential damage resulting from its use.

Pittsburgh
July 2015

Preface to the First Edition

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 Umut Acar, Jesper Louis Andersen, Carlo Angiuli, Andrew Appel, Stephanie Balzer, Eric Bergstrom, Guy E. Blelloch, Iliano Cervesato, Lin Chase, Karl Crary, Rowan Davies, Derek Dreyer, Dan Licata, Zhong Shao, Rob Simmons, and Todd Wilson for their extensive efforts in

reading and criticizing the book. I also thank the following people for their suggestions: Joseph Abrahamson, Arbob Ahmad, Zena Ariola, Eric Bergstrome, William Byrd, Alejandro Cabrera, Luis Caires, Luca Cardelli, Manuel Chakravarty, Richard C. Cobbe, James Cooper, Yi Dai, Daniel Dantas, Anupam Datta, Jake Donham, Bill Duff, Matthias Felleisen, Kathleen Fisher, Dan Friedman, Peter Gammie, Maia Ginsburg, Byron Hawkins, Kevin Hely, Kuen-Bang Hou (Favonia), Justin Hsu, Wojciech Jedynak, Cao Jing, Salil Joshi, Gabriele Keller, Scott Kilpatrick, Danielle Kramer, Dan Kreysa, Akiva Leffert, Ruy Ley-Wild, Karen Liu, Dave MacQueen, Chris Martens, Greg Morrisett, Stefan Muller, Tom Murphy, Aleksandar Nanevski, Georg Neis, David Neville, Adrian Trejo Nuñez, Cyrus Omar, Doug Perkins, Frank Pfenning, Jean Pichon, Benjamin Pierce, Andrew M. Pitts, Gordon Plotkin, David Renshaw, John Reynolds, Andreas Rossberg, Carter Schonwald, Dale Schumacher, Dana Scott, Shayak Sen, Pawel Sobocinski, Kristina Sojakova, Daniel Spoonhower, Paulo Tanimoto, Joe Tassarotti, Peter Thiemann, Bernardo Toninho, Michael Tschantz, Kami Vaniea, Carsten Varming, David Walker, Dan Wang, Jack Wileden, Sergei Winitzki, Roger Wolff, Omer Zach, Luke Zarko, and Yu Zhang. I am very 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 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 Grant Nos. 0702381 and 0716469. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Robert Harper
Pittsburgh
March 2012