Programming Linguistics

Programming Linguistics

David Gelernter and Suresh Jagannathan

The MIT Press
Cambridge, Massachusetts
London, England

This book was set in Computer Modern.
Printed and bound in the United States of America.

Library of Congress Cataloging-in-Publication Data

Gelernter, David Hillel.
Programming linguistics / David Gelernter, Suresh Jagannathan.
p. cm.
Includes bibliographical references.
ISBN 0-262-07127-4
1. Programming languages (Electronic computers)
2. Computational linguistics. I. Jagannathan, Suresh. II. Title.
QA76.7.G44 1990 90-5485
005.1—dc20 CIP

For my Jane, *eishes chayil:*

השמיעני את קולך,
כי קולך ערב, ומראיך נאוה.

By reflecting on the true nature of things, it is recognized that even when this world of multiplicity is perceived, it is only Thy non-dual Self that is apprehended, just as gold only is seen when ornaments are perceived, and clay, when pots are seen. When knowledge dawns, what happens is that this fact becomes crystal clear, just as the true nature of dream objects becomes evident on awakening, and of the worn-out rope on the dispersal of darkness. To Thee, O Krishna, my salutations.

Narayennyam: Canto 98, Verse 7

To my family: Hema, Amma, Appa and Aravind

Preface

This book is intended to serve several purposes. First, it's a text for courses in the design and evolution of programming languages. The recipe for turning the book into a course is fairly simple: students should read the book *and* all of the "basic" papers listed at the end of each chapter. Add exercises, stir and serve. To get a beefed-up graduate student version, read all of the "suggested" as well as the "basic" papers in the reading lists. None of these are extraneous or strictly for reference; they all play a significant role in the evolution of the field.

The book is designed also to serve as supplementary reading for courses in compiler construction. It's often difficult for a compilers course to leave students with a comprehensive feel for the breadth and variety of language models that compilers are called upon to support. Students sometimes emerge, besides, without a clear understanding of why anybody would want or need some of the features they've learned how to implement. We hope that this book will be one part of an effective treatment plan aimed at the *I-know-how-but-don't-ask-me-why* syndrome that so often strikes down our young people in the prime of life.

Finally, programming languages are an important topic both for computation practitioners and for researchers. This book may be of use to anyone who needs more information on a significant and intriguing area.

Beyond its specific goals as a text and an information source, the book has a more general mission. Computer science seems to be destined for a permanent identity crisis. This holds particularly for the sub-field called "Systems"—the part of computer science that deals with the design and construction of computing environments, encompassing computer architecture, operating systems, compilers, programming languages and program-building methods. The Systems area is the center of computer science, the most specifically "computer-science-ish" part of the field, and there is an ongoing struggle over its basic character.

A large number of distinguished researchers seem to feel that, if this field is ever going to find itself, the only place to look is in the mathematics department. The study of programming languages in particular is imagined in some quarters to be reducible to the study of implementation techniques on the one hand and to the formal or mathematical description of language semantics on the other. Both topics, despite the

obvious importance of implementation and the potential significance of formal semantics, beg the fundamental questions:

- What do programming languages look like, and what *should* they look like?
- Why do they offer the tools and constructs they do, what kind of programming styles do they suggest, what do they tell us about some particular vision of software structure?
- What light do they shed on the basic, unanswered question in Systems—what *is* a program anyway?

Before students worry about topics like formal semantics, they should be quite clear about what they are studying the semantics *of*. They should aim to understand the language design field in all its richness and depth, resisting the narrow outlook that is sometimes imposed in the interests of formal tractability. And they should keep in mind that programming languages are synthetic creations, and that language design is properly an *engineering* study.

We stress this last point on behalf of a larger, more important, often-neglected consideration: engineering exists at the interface between science and art. Aesthetic issues, the design judgments that are captured in a programming language and underlie some vision of the shape and form of software machinery, are just as important to this field as the theorems that define computation. The beauty and power of the best programming languages are one facet of a larger topic, having to do with the sweep and intellectual depth of engineering in general. Unfortunately, this broader topic is simply not on the menu at the typical fast food-for-thought joint that passes as a modern university (and is even less likely to be served at the trendiest bistros)—which very likely contributes to computer science's difficulties in defining itself.

The book's structure is heterogeneous. The chapters vary greatly in length and will (obviously) require varying amounts of time to get through. The three Classical Languages (Fortran, Algol 60 and Lisp) are grouped together in one long chapter, in order to accommodate some historical background and some exercises relating to all three, and in deference to the fact that together they underlie most subsequent work in the field. The Pascal chapter has a prologue dealing with Algol 68 and PL/I; the Simula 67 chapter has a postlude that deals with spec-

ification, abstract types and Ada. Through the Pascal chapter, each language is treated essentially as a whole. Thereafter we assume that enough groundwork has been laid to justify our focusing mainly on specific language features—the class in Simula, the package in Ada and so on. We follow, in a sense, the development of a single "plot" from the Classical Languages through Scheme; the plot has to do with the rise and fall of a virtual structure called the Algol Wall. (Of course there are a large number of other themes under development at the same time.) The final two language chapters, on Declarative and Parallel languages, each represent a digression of sorts after the main story has drawn to its slightly bittersweet conclusion. Two special-purpose appendices, following the Pascal and the Declarative Languages chapters, attempt to place programming language design in a broader intellectual context. Two appendices, following the second chapter, give a summary of and formal semantics for the "ideal software machine" model; they're included mainly for reference use and may be skipped without loss of continuity.

Acknowledgments. The first author thanks the Linda group at Yale, particularly Nick Carriero and Jerry Leichter. I'm grateful to Jerry especially for introducing me to the remarkable work of Billington. I have the distinct impression that, although this book deals only glancingly with the main topic of our collaboration, it could never have been written without Nick's contributions. The Computer Science Department at Yale was a stimulating place to work. It's an honor to acknowledge that extensive and wide-ranging conversations with Alan Perlis contributed a great deal to my own understanding of programming languages. Martin Schultz turned the Department into a place where our work could thrive, and I'm grateful to him for his support. The heart of the ideal software machine model, and the view of language design that underlies the book, goes back to my thesis work at Stony Brook, and to conversations there with teachers and fellow students. Of course I thank my parents, The Sibs and above all my wife—to whom the author hereby gives notice that, several years of work on this book being now finally complete, it may conceivably be possible for him to resume mowing the lawn occasionally.

The second author is grateful to the Laboratory for Computer Science, Massachusetts Institute of Technology, where the semantics and detailed

design of the ideal software machine model were developed. The proposition that this model could serve as a viable basis upon which to unify superficially diverse concepts in programming languages formed the centerpiece of my doctoral research. Although I am indebted to all my colleagues and friends at MIT for the stimulating environment they helped to create, I especially would like to thank Rishiyur Nikhil and Bert Halstead for their significant contributions to my understanding of programming languages and systems. I thank David Gelernter for his invitation to collaborate with him on the writing of this book, and the Computer Science department at Yale University for providing the superb resources that allowed us to undertake this endeavor. Most importantly, my contributions to this text were possible only because of the never-ending encouragement and support given by my family—my wife Hema, my parents, and my brother Aravind.

Programming Linguistics

Contents

List of Figures