Preface

I wrote this book to provide a description of the foundations of statically typed class-based object-oriented programming languages for those interested in learning about this area. An important goal is to explain how the different components of these languages interact, and how this results in the kind of type systems that are used in popular object-oriented languages. We will see that an understanding of the theoretical foundations of object-oriented languages can lead to the design of more expressive and flexible type systems that assist programmers in writing correct programs.

Programmers used to untyped or dynamically typed languages often complain about being straitjacketed by the restrictive type systems of object-oriented languages. In fact many existing statically typed object-oriented languages have very restrictive type systems that almost literally force programmers to use casts or other mechanisms to escape from the static type system. In this work we aim to meet the needs of a programmer who wants a more expressive type system. Thus another goal of this text is to promote richer type systems that reduce the need for bypassing the type checker.

Because of the semantic complexity of the features of object-oriented languages, particularly subtyping and inheritance, it is difficult to design a static type system that is simultaneously safe and flexible. To be sure that there are no holes in the type system we need to prove that the type system is safe (essentially that no type errors can occur at run time), but we cannot do that without a description of the meaning of programs. Thus this book contains careful formal descriptions of the syntax, type system, and semantics of several progressively more complex object-oriented programming languages. With these definitions, it is possible to prove type safety.

Object-oriented programming languages have been of great practical and theoretical interest, but most of the interesting developments in foundations have been accessible only to researchers in the area. Moreover, papers in the area have taken quite different approaches, as well as using different

notation and even different terminology from each other. As a result, it has been difficult for outsiders to learn the basic material in this area.

This book differs from other recent books in the foundations of object-oriented languages in several ways. First, the focus of attention is class-based object-oriented languages, rather than object-based or multi-method languages. Thus our study is very relevant to the most popular kind of object-oriented languages in use today.

Second, this book approaches the foundations from the point of view of a programmer or language designer wishing to understand the type systems of object-oriented languages and to see how to extend the type systems to increase the expressiveness of these languages. The semantics presented suggest extensions to the language and provide the foundations for verifying the safety of the type system.

Third, we base the foundation of object-oriented programming languages on the classical typed lambda calculus and its extensions rather than introducing new calculi to explain the fundamental constructs. Thus we can rely on classical results, only including a brief review of the lambda calculus to introduce readers to the notation.

This book is intended for several different audiences. My intention has been to make it accessible to students, especially advanced undergraduates and graduate students, to practitioners wishing to have a deeper understanding of the foundations of object-oriented programming languages, and to researchers who wish to understand developments in the foundations of object-oriented languages. It can be used as the main text for a course in the foundations of object-oriented programming languages or as a supplementary text for a course with a broader focus that includes object-oriented programming languages.

We have designed the first part of the book, comprising the first seven chapters, to be especially accessible to a wide variety of readers. These chapters provide a relatively non-technical introduction to key issues in the type systems of object-oriented programming languages. As such, this part may be especially appropriate for use in a general undergraduate or graduate course covering concepts of object-oriented programming languages or as the basis for self-study.

The next part, comprising Chapters 8 and 9, provides a relatively quick introduction to the simply typed lambda calculus and many of its extensions. The goal of this part is to have the reader understand how the lambda calculus can provide a formal description of programming language constructs. This part also introduces the formalism for writing the syntax and

type-checking rules for programming languages. For readers with a solid understanding of programming languages as provided for by Pierce's text, *Type Systems for Programming Languages* [Pie02], or Mitchell's *Foundations for Programming Languages* [Mit96], for example, these chapters will simply provide a quick review. Others will need to spend more time to understand how such a primitive language can be used as a model of important programming language concepts and to learn how to read and understand the type-checking rules. It is not necessary to understand the deep results about the lambda calculus found in more specialized texts in order to understand the use of lambda calculus in this book.

The third part of the book, comprising Chapters 10 through 14, presents the core foundational material on class-based object-oriented languages. We begin by providing a formal definition of a simple object-oriented language, $\mathcal{SOOL}$, and its type system. Chapters 11 and 12 explore understanding the semantics of $\mathcal{SOOL}$ by translating terms into a very rich extension of the typed lambda calculus. With this understanding of the language, Chapter 13 presents a proof of soundness and safety of $\mathcal{SOOL}$. This chapter is the technically most difficult of the book. The details of the proof in the first section of that chapter may be skipped on the first reading, but the statement of the soundness and safety theorems and the other material in the chapter are important as they illustrate how a careful formal definition of a language can lead to provable safety.

The language $\mathcal{SOOL}$ was kept very simple so that the proof of soundness could avoid as many complications as possible. The last chapter of this part discusses many of the more specialized concepts commonly found in object-oriented languages that were left out of $\mathcal{SOOL}$. These include references to methods from the superclass, more refined access control in classes, nil objects, and even a discussion of multiple inheritance.

The final part of this book explores extensions of the type systems of object-oriented languages suggested by our understanding of the semantics of $\mathcal{SOOL}$. The extensions include F-bounded polymorphism, a new type keyword, `MyType`, standing for the type of `self`, and a relation, matching, that is more general than subtyping. We will find that the addition of these features adds considerably to the expressiveness of object-oriented languages, yet we will prove that they do not compromise the type safety of the language. We end with the presentation of a language that incorporates `MyType`, matching, and a new form of bounded polymorphism using matching, but that no longer contains the notion of subtyping. We will see that this simpler language is still very expressive, even without subtyping.

The topics covered in this book represent an active area of research, with new papers appearing every year. There are many topics that I would have liked to have included, but could not because of a desire to keep the size of this book manageable. The best way to keep up with current research in the area is to attend or examine the proceedings of major conferences and workshops in this area. The major conferences presenting new research in the broad area of programming languages are the Principles of Programming Languages (POPL) and Programming Language Design and Implementation (PLDI) conferences. The most important conferences presenting research on object-oriented languages are the annual Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA) conference and the European Conference on Object-Oriented Programming (ECOOP). The annual Foundations of Object-Oriented Languages (FOOL) workshop provides an important, though less formal, forum for new results in the area covered by this book. Information on the FOOL workshops is available at

`http://www.cs.williams.edu/~kim/FOOL/.`

One of my favorite quotes, first encountered as a signature tag on e-mail, is the following:

> "The difference between theory and practice is greater in practice than in theory" *Author unknown*

In pursuing my own research on topics central to the issues covered in this book, I have tried to keep this quote in mind. As a result, rather than just theorizing about issues in programming language design, my students and I have implemented interpreters and compilers for languages similar to those discussed here. (For pedagogical reasons the languages described in the text are different in inessential ways from the languages we have implemented.)

The experience of implementing and using these languages has provided better insight to the strengths and limitations of the type systems discussed here. It is my hope, and indeed one of the reasons for writing this book, that the knowledge obtained by the research community in the foundations of object-oriented programming languages will eventually work its way into practical and widely used programming languages. The growing interest in the extension, GJ, of Java described in Section 4.1 provides evidence that this kind of technology transfer has already begun.

The material presented in this book is the result of the dedicated and creative work of many researchers. The *Historical Notes and References* sections at the end of each of the four parts of the book credit the contributions of

many of those doing research in this area. I have also benefitted greatly from personal and professional interactions from many researchers in this area.

Primary credit for helping me get started doing research in the semantics of programming languages goes to Albert Meyer, from whom I learned an enormous amount, both about semantics and about the process of doing research, while on my first leave from Williams College. A ten-year-long professional collaboration with Guiseppe Longo was extremely productive and enjoyable, while incidentally introducing me to the beauty of Italy and France. Peter Wegner deserves credit for introducing me to object-oriented programming languages and asking annoying questions that led to many interesting results. John Mitchell and Luca Cardelli provided key influences (and funding) during a visit to Palo Alto in the spring of 1991 that led to my work on the design and proofs of type safety of object-oriented programming languages.

A three-month visit to the Newton Institute of Mathematical Sciences in the fall of 1995 during the special program on Semantics of Computation provided a great opportunity to work with other researchers in the semantics of programming languages. The interaction with Benjamin Pierce and Luca Cardelli there led to our joint paper comparing different styles of semantics for object-oriented languages.

Similarly, early meetings of the workshops on the Foundations of Object-Oriented Languages (the FOOL workshops) resulted in many interesting discussions (and arguments), some of which led to the paper "On binary methods" [BCC+95], a paper with 8 co-authors who at times seemed to have at least 10 different opinions on how best to approach the issues involved. I have learned more through writing these papers (in spite of the difficulty of writing conclusions!) than through almost any other activity as a researcher. Teaching a graduate programming languages course while on a visiting professorship at Princeton University allowed me to begin writing this book while trying out the material on students.

Opportunities for collaboration with my computer science honors students at Williams College and my co-authors have taught me a great deal over the years. My honors students in computer science include Robert Allen, Jon Burstein, David Chelmow, John N. (Nate) Foster, Benjamin Goldberg, Gerald Kanapathy, Leaf Petersen, Dean Pomerleau, Jon Riecke, Wendy Roy, Angela Schuett, Adam Seligman, Charles Stewart, Robert van Gent, and Joseph Vanderwaart. Aside from the researchers and students mentioned above, my co-authors in programming language research papers include Roberto Amadio, Giuseppe Castagna, Jon Crabtree, Roberto DiCosmo, Allyn Dimock, Adrian

Fiech, Gary Leavens, Robert Muller, Martin Odersky, Scott Smith, and Philip Wadler.

I owe a great debt of gratitude to the National Science Foundation, most recently through the offices of Frank Anger, for their long-standing support for my research. NSF research grants supporting the research reported here include NSF CCR-9121778, CCR-9424123, CCR-9870253, and CCR-9988210. 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.

Special thanks go to those who provided comments and corrections on drafts of this manuscript. Narciso Martí-Oliet, John N. Foster, and an anonymous reviewer provided very detailed and helpful comments on a complete draft of this book. Andrew Black provided very useful and detailed comments on an early survey paper that evolved into this book. Others who provided useful comments on different portions of the book, suggested approaches, or were helpful in clearing up historical details included Martín Abadi, Luca Cardelli, Craig Chambers, Kathleen Fisher, Cheng Hu, Assaf Kfoury, John Mitchell, Benjamin Pierce, and Jack Wiledon. Thanks to my editor Bob Prior for his friendship, for his faith in this project, and for making this task less painful than it might have been. I am grateful to Christopher Manning for sharing the LaTeX macros that resulted in this book design.

I take full credit for all omissions and errors remaining in this book. Please send corrections to kim@cs.williams.edu. I will provide a web site with errata or clarifications at

```
http://www.cs.williams.edu/~kim/FOOLbook.html
```

and through MIT Press at

```
http://mitpress.mit.edu/
```

I give great thanks to my family for their love and support during the long years spent writing this book. Thanks to my colleagues in the Computer Science Department at Williams for their professional support and intellectual stimulation. Finally, thanks to my teachers whose guidance led me to begin this interesting journey. Special thanks are due to H. Jerome Keisler and the late Jon Barwise at the University of Wisconsin, the late Harry Mullikan and Paul Yale at Pomona College, and Shirley Frye and Mike Svaco at Scottsdale Arcadia High School.