Benjamin C. Pierce. ***Types and Programming Languages,*** The MIT Press, Cambridge, Massachusetts, xxi + 623 pp.

Types were developed in the early part of the $20^{\text{th}}$ century in order to avoid inconsistencies in Frege's formulation of logic discovered by Russell. They therefore have a strong logical origin, confirmed by the development of Church's type theory, sometimes called higher-order logic. It contains the simply-typed $\lambda$-calculus which is at the core of most modern programming languages. Through the research of other pioneers such as Milner and Martin-Löf, type systems are now understood to be central in the design and analysis of programming languages. They provide the fundamental principles according to which languages are organized. A thorough textbook on type systems in programming languages had been long overdue and Pierce's book provided exactly that.

*Types and Programming Languages* is designed for an advanced undergraduate or graduate course and assumes some familiarity with functional programming. Since types and programming languages are by now a large subject, a stringent selection of topics is necessary. After providing some background on the $\lambda$-calculus and basic notions such as substitution and induction, the book starts with simple types. In this setting the pervasive idea of type safety is first introduced. Type safety is comprised of two properties relating typing with the operational semantics of a language: *preservation*, which guarantees that a term retains its type under evaluation, and *progress*, which guarantees that a well-typed term is either a value or can proceed to be evaluated further. Type safety is one of the recurring themes of the book and one of its most important lessons.

Following the thorough treatment of simple types, the book successively develops subtyping, recursive types, polymorphism, and type operators. Subtyping is motivated and presented as record subtyping; coercion semantics and coherence are covered as a short excursion. Recursive types are presented first informally, and then in both iso-recursive and equi-recursive forms. The treatment of co-induction in the study of recursive types is a nice addition that is difficult to find elsewhere in introductory form. The next part on polymorphism covers type inference, ML-style and impredicative polymorphism, existential types for data abstraction, and bounded quantification. Much of this should be core material in any advanced course concerned with programming languages. The final part treats type operators and higher-order polymorphism, a topic rather more advanced particularly in its application to the study of purely functional objects in the last chapter.

The pragmatics of type constructs are illustrated in interesting and novel ways. First, for most chapters there is an implementation available from a web site with supporting material. Several of these implementations are discussed in detail in the book, which can be quite helpful to students for gaining a deeper and more intuitive understanding of the fundamental concepts. Second, there are many exercises interleaved with the text whose solutions are given at the end of the book. This device is helpful, even to advanced readers, by giving them the opportunity to test their comprehension immediately. Third, the book contains several excursions called "*case studies*" where the type systems are exploited in sometimes unexpected ways to support object-oriented programming.

The book is very well-written, with clear motivations, cogent explanations, helpful examples, appropriate exercises, and thorough discussions. The material is well-structured; the incremental approach to developing a cumulative system quite successful. Besides serving as an excellent textbook, the detailed notes and suggestions for further reading make it an important reference and study resource. I have used the

book directly for two lectures and also as general supplementary reading in my junior-level undergraduate course on *Foundations of Programming Languages*. The students and I found it quite easy to use and I will likely expand its role the next time I teach the course.

The one aspect of the book that would make me hesitate to adopt it as the sole textbook for one of my own courses is the orientation of its latter parts towards a functional interpretation of object-oriented programming at the expense of other important topics. Much of the material on subtyping, bounded quantification, and type operators is motivated and explained in terms of a functional reconstruction of objects based on structural subtyping. I do not find this approach to be particularly realistic or appealing and I think it is difficult for students to relate to their own object-oriented programming experience, which almost universally stems from languages with nominal subtyping.

In an undergraduate course, I have instead covered some further topics on implementation such as environments, abstract machines, or storage management, advanced type constructs such as monads or intersection types, or other paradigms such as concurrency or logic programming. In a graduate course I might instead expand on the meta-theory, covering topics such as parametricity or normalization for System F in more detail. Admittedly, some of these topics clearly exceed the bounds of a textbook on type systems, but others, particularly monadic types, intersection types, and dependent types would have rounded out the coverage.

In summary, *Types and Programming Languages* provides an excellent, unified introduction to type systems, suitable for an advanced undergraduate or graduate course, notwithstanding minor concerns about coverage and approach. In addition, it will serve as a standard reference in the field for many years to come. Given the central role of types in modern programming language design and analysis, *Types and Programming Languages* is probably the single most important book in the area of programming languages in recent years.

Frank Pfenning

Department of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania. fp@cs.cmu.edu.

Entry for the Table of Contents: