Book Review

Martín Abadi and Luca Cardelli
A Theory of Objects. Springer-Verlag. 1996. ISBN 0-387-94755-2. DM 64. 396pp, hardbound.

The design, implementation, and theoretical foundations of static type systems occupy an increasingly prominent place in research on programming languages. In this area, one particularly rich source of advances for more than a decade has been the search (initiated by Cardelli, among others) for type systems of sufficient power and flexibility to handle the combinations of features found in object-oriented languages such as Simula, Smalltalk, Eiffel, C++, and Java.

Research on this topic has concentrated on isolating a number of basic formal concepts, such as subtyping, parametric polymorphism, recursive types, and types for simple records, and showing how these components can be combined to model the most characteristic features of object-oriented languages: object encapsulation and identity, message passing, dynamic dispatch, interface conformance, classes, delegation, and inheritance mechanisms such as *self*, *super*, and *MyType*.

In their new book, *A Theory of Objects*, Abadi and Cardelli have organized and consolidated a vast amount of previous work in the area and given a unified basis for future developments, proposing a new mathematical framework—the *Object Calculus*—in which many threads in both theory and practice can be woven together. The Object Calculus can be thought of as a tiny programming language, analogous to Church's lambda-calculus in character. In the lambda-calculus, "everything is a function." The only basic operations are definition and application of functions, from which data structures such as numbers and lists and control structures such as iteration and recursion can be built by programming. Similarly, in the Object Calculus, every computational entity is an object, and the only basic operations are (1) object creation, (2) message passing, and (3) method override, in which a new body is specified for one of the methods of an existing object. (The use of method override as a primitive means that the Object Calculus models delegation-based languages more directly than languages with classes, which must be encoded here using objects.)

The book is divided into four main sections. It opens with a long, non-technical review of object-oriented concepts, establishing intuitions and terminology for the technical sections that follow. The second section introduces the two variants of the untyped Object Calculus—one a purely functional calculus in which updating a method of an object yields a fresh object, the other an imperative calculus where method updates happen "in place"—and develops the basic technical mechanisms of subtyping and recursive types. The third section treats second-order object calculi, using parametric polymorphism to model more advanced language features such as covariant method specialization. Finally, in the last section an object calculus with a more powerful form of polymorphism is used to model features such as "matching" and the typing of binary methods.

The first two sections of the book should be accessible to any computer scientist with a reasonably solid mathematical background. Prior familiarity with typed lambda-calculi will make the reading easier, but the presentation is both lucid and complete: it could even be used in an introductory course on type systems. In the latter half, both the material and the writing style become much more technical, though the development remains essentially self-contained; these sections will be most useful for advanced readers.

The Object Calculus is already widely used and will be a fundamental tool in future work on the foundations of object-oriented languages. This book will be both a basic reference for established researchers and an introduction for new arrivals to this exciting field.

Benjamin C. Pierce
Indiana University