J. Functional Programming 4 (2): 125–126, April 1994

Type systems for object-oriented programming

Functional programming has had a broader impact on computing than the design of a few specific functional languages. One of its contributions is a deeper understanding of type systems. This special issue of *The Journal of Functional Programming* focuses on the difficult and important question of type systems for object-oriented programming.

This issue had an unusual gestation. When I heard of Kim Bruce's expository paper on a type system for object-oriented programming, I immediately invited him to submit it for publication in *JFP*. Papers by Pierce and Turner and by Abadi then came to my attention, and I solicited them as well. By this point we had a bonus-size special issue, without the usual mechanism of a call for papers. Mitchell Wand graciously provided an introduction that carefully compares the approaches of the three papers.

This issue provides an overview of the state-of-the-art, but it is certainly not the last word. Further submissions on this topic are most welcome.

An extra-size issue requires extra work. Thanks are due to the contributors and referees for the effort they have put into producing an extraordinary issue.

—PHILIP WADLER

Introduction

Object-oriented programming (OOP) has become one of the cornerstones of modern programming methodology. Yet there is little agreement on what object-oriented programming is. Different object-oriented languages typically implement different collections of facilities, and heated discussions of which facilities are necessary for true object-oriented programming flare up regularly.

In the light of these discussions, this issue of *JFP* presents three papers that study the theoretical bases of object-oriented programming. These papers illustrate the variety of choices that can be made in the design of a theory of OOP.

Kim Bruce's paper, 'A Paradigmatic Object-Oriented Programming Language: Design, Static Typing, and Semantics', seeks to model via denotational semantics as many features of conventional object-oriented languages as is possible within the functional framework. He defines a language that supports classes, objects, methods, hidden instance variables and inheritance. He presents static typing rules for the language, and then gives a model for his types using PERs. He shows the soundness of the typing rules by giving a denotational semantics, and showing that the semantics is sensible: if a phrase has static type σ, then its denotation is a value