GUEST EDITORIAL

Foundations of Object-Oriented Languages

This issue contains papers selected from the fifth Workshop on the Foundations of Object-Oriented Languages (FOOL), held in San Diego, California on January 17 and 18, 1998, in conjunction with the 25th ACM Symposium on Principles of Programming Languages (POPL ’98). The workshop consisted of three invited lectures, and 12 contributed papers selected by the program committee. The three papers in this issue were selected from those presented at the workshop.

The FOOL workshops were initiated by Kim Bruce and Giuseppe Longo in 1993 with the assistance of grants from the NSF and ESPRIT to bring together researchers from around the world to discuss the theoretical foundations of object-oriented languages. Attendance at the first two workshops, held at Stanford University and CNAM in Paris, respectively, were by invitation only. Beginning with the FOOL 3 workshop, held in Brunswick, New Jersey, the workshops have been open to the public. FOOL 4, held in Paris in affiliation with POPL ’97, attracted over 100 attendees. The most recent workshop, FOOL 8, was held in London, in January of 2001.

Specific topics addressed in the workshop have included semantics, foundational calculi, type theory, and program verification of object-oriented languages. More recently special efforts have been made to include contributions on foundational issues related to concurrent and distributed object-oriented languages, and database languages with object-oriented features. More information about the FOOL workshops, including reports on previous workshops, can be found at the url http://www.cs.williams.edu/~kim/FOOL/. This is the third special issue of Theory and Practice of Object-Oriented Systems (TAPOS) with selected papers from FOOL workshops. The first, which included papers from FOOL 3, appeared as volume 4, issue 1, in 1998. The second, including papers from FOOL 4, appeared as volume 5, issue 1, of TAPOS in 1999.

The first paper in this issue, Privacy via subsumption, by Jon G. Riecke and Christopher A. Stone introduces an object calculus that supports both object extension and width structural subtyping. This has proved to be difficult to accomplish in the past because of the possibility of name clashes when a field has been “forgotten” using subsumption, and then a different field with the same name is added. These difficulties are overcome by using “dictionaries” to provide the connection between field names and the actual components referenced by those names. This also provides a mechanism to support hiding components via the use of subsumption. That is, methods and fields that are to be public can be included in an object type. The type of the full object (including hidden components) is a subtype of the type of the public view. If only the public view is presented, then the other features are hidden via the use of subsumption.

The second paper, Type Destructors, by Martin Hofmann and Benjamin C. Pierce, introduces a variant of System $F_{\omega}$ with “structural” typing rules. A well-known problem with $F_{\omega}$ is its inability to express polymorphic update functions. The authors show that this problem and similar problems can be overcome by introducing type “destructor” functions as a complement to the usual “constructor” functions. These destructor functions can be used to increase information about the elements of types that are subtypes of given types, and thus can be used to increase the expressiveness of the language.

The third paper, Objects and Classes in Algol-like Languages, by Uday Reddy, is quite different from the preceding papers. It attempts to bridge the gap between the study of Algol-like languages and object-oriented languages. To this point, the study of these subjects has been carried out by different groups of researchers, with very little communication between the groups. Reynold’s Idealized Algol has proven to be a very useful tool for the study of Algol-like languages. Reddy shows how to extend these tools to provide a deeper understanding of objects and classes in ways that differ significantly from earlier attempts to provide foundations for object-oriented languages. This approach to understanding object-oriented languages also provides formal tools for reasoning about programs.

Kim Bruce
Didier Remy,
Guest Editors