Formal Techniques for Java Programs

Formal techniques can help analyze programs, precisely describe program behavior, and verify program properties. Applying such techniques to object-oriented technology is especially interesting because:

- the OO-paradigm forms the basis for the software component industry with their need for certification techniques.
- it is widely used for distributed and network programming.
- the potential for reuse in OO-programming carries over to reusing specifications and proofs.

Such formal techniques are sound, only if based on a formalization of the language itself.

Java is a good platform to bridge the gap between formal techniques and practical program development. It plays an important role in these areas and is on the way to becoming a *de facto standard* because of its reasonably clear semantics and its standardized library.

However, Java contains novel language features, which are not fully understood yet. More importantly, Java supports a novel paradigm for program deployment, and improves interactivity, portability and manageability. This paradigm opens new possibilities for abuse and causes concern about security.

The ECOOP 2000 workshop on *Formal Techniques for Java Programs* was held in Sophia Antipolis, France. It was a follow-up for last year's ECOOP workshop on the same topic [1] and the Formal Underpinnings of the Java Paradigm workshop held at OOPSLA '98 [2].

Proceedings containing all the papers are available as a technical report of the Computer Science Department of the FernUniversität Hagen [3]. This special issue contains extended and refereed versions of four papers, selected from the best papers presented at the workshop.

Eva Rose and Kristoffer Rose suggest in their paper "Java Access Protection through Typing" the integration of a dedicated read-only field access into the Java type system. The advantage is that "getter" methods can be eliminated. This allows for 1) static look-up as opposed to dynamic dispatch, 2) avoidance of the space for the code of the getter method, 3) avoidance of denial of service attacks on field access, 4) discovery of access protection violating through Java's bytecode verifier. They describe this through an extension of the bytecode verifier.

Bytecode verification is a technique to check whether a piece of bytecode is type sound. Resource-bounded implementations of the Java Virtual Machine on smart cards do not provide bytecode verification. They do not allow dynamic loading at all or use cryptographic techniques to rely on off-card verification. To enable on-card verification, Eva and Kristoffer Rose in [2] developed a sparse *annotation* of bytecode with type information to drastically simplify type reconstruction. This way, on-card verification essentially becomes a linear type check. This technique is called lightweight bytecode verification.

Gerwin Klein and Tobias Nipkow in their paper "Verified Lightweight Bytecode Verification" presented a formal proof that lightweight bytecode verification is sound and complete. Soundness means that the lightweight verifier should only accept an annotated pieces of code if the fullweight verifier accepts the stripped version. Completeness means that the lightweight verifier should accept every piece of code that is annotated with the types reconstructed by the fullweight verifier if the fullweight verifier accepts the code.

Alessandro Coglio and Allen Goldberg describe in "Type safety in the JVM; some problems in Java 2 SDK 1.2 and proposed solution" a problem caused by unprecise treatment of type identity. For some operations the bytecode verifier loads the wrong class because it does not always take into account the association between loaders and classes. This problem can be fixed by using as full names both the class name and the loader name which loaded it. As a more general approach, Coglio and Goldberg proposed a solution that more clearly separates bytecode verification and class loading. The idea is to generate refined loading constraints during bytecode verification.

David von Oheimb's "Hoare Logic for Java in Isabelle/HOL" involves the language Java$^{\ell ight}$, which is a nearly full subset of sequential Java. Earlier a deep embedding of Java$^{\ell ight}$ into the higher order logic of the theorem prover Isabelle has been developed, together with an operational semantics. Von Oheimb described a Hoare logic for Java$^{\ell ight}$, which he has proven (in Isabelle) to be both sound and complete (w.r.t. the operational semantics). The logic covers challenging features like exception handling, static initialization of classes and dynamic binding of methods. The soundness and completeness proofs rely on the type safety property (dynamic types of expressions are subtypes of their static, declared types), which was already proven for Java$^{\ell ight}$.

We would like to thank all the reviewers for their careful reading and helpful comments for the papers. We are impressed that they are responsible for a noticable improvement of quality. The papers were refereed by Gilad Bracha, Martin Berger, John Boyland, Drew Dean, Sophia Drossopoulou, Susan Eisenbach, Steve Freund, Arnd Poetzsch-Heffter, Atsushi Igarashi, Bart Jacobs, Thomas Jensen, Gary Leavens, Rustan Leino, Markus Lumpe, Jens Palsberg, Lawrence Paulson, Geoffrey Smith, Scott Smith, Raymie Stata, and Hayo Thielecke.

Finally we would like to thank the rest of the workshop organizers Sophia Drossopoulou (Imperial College, Great Britain), Bart Jacobs (University of Nijmegen, The Netherlands), Peter Müller (FernUniversität Hagen, Germany), and Arnd Poetzsch-Heffter (FernUniversität Hagen, Germany) for their work on both the workshop and allowing us to 'borrow' some of their words for this introduction to the special journal section.

Susan Eisenbach
Imperial College
Gary T. Leavens
Iowa State University

References

[1] B. Jacobs, G. Leavens, P. Müller, and A. Poetzsch-Heffter, *Formal Techniques for Java Programs*, in A. Moreira and D. De Meyer, *eds.*, ECOOP'99 Workshop Reader, vol 1743 of LNCS, Springer-Verlag, 1999.

[2] S. Eisenbach, *ed*, *Formal Underpinnings of Java*, 1998, Workshop report, http://www.doc.ic.ac.uk/ sue/oopsla/cfp.html.

[3] http://www.informatik.fernuni-hagen.de/import/pi5/workshops/ecoop2000_papers.html.