Formal Techniques for Java-Like Programs

Report on the Workshop FTfJP at ECOOP 2007

John Boyland^{1,2}, Dave Clarke³, Gary Leavens⁴, Francesco Logozzo⁵,
and Arnd Poetzsch-Heffter⁶

¹ University of Wisconsin–Milwaukee, USA
`boyland@uwm.edu`

² Nanjing University, China

³ CWI, Amsterdam, Netherlands
`dave@cwi.nl`

⁴ University of Central Florida, USA
`leavens@eecs.ucf.edu`

⁵ Microsoft Research, USA
`logozzo@microsoft.com`

⁶ Universität Kaiserslautern, Germany
`poetzsch@informatik.uni-kl.de`

Abstract. Formal techniques can help analyze programs, precisely describe program behavior, and verify program properties. Newer languages such as Java and C# provide good platforms to bridge the gap between formal techniques and practical program development, because of their reasonably clear semantics and standardized libraries. Moreover, these languages are interesting targets for formal techniques, because the novel paradigm for program deployment introduced with Java, with its improved portability and mobility, opens up new possibilities for abuse and causes concern about security.

Work on formal techniques and tools for programs and work on the formal underpinnings of programming languages themselves naturally complement each other. This workshop aims to bring together people working in both these fields, on topics such as: specification techniques and interface specification languages, specification of software components and library packages, automated checking and verification of program properties, verification logics, language semantics, program analysis, type systems, security.

1 Call for Papers

The Call for Papers included the text from the abstract above and the following text:

Contributions are sought on open questions, new developments, or interesting new applications of formal techniques in the context of Java or similar languages, such as C#. Contributions should not merely present completely finished work, but also raise challenging open problems or

propose speculative new approaches. We particularly welcome contributions that simply suggest good topics for discussion at the workshop, or raise issues that you feel deserve the attention of the research community. Submissions must be in English and are limited to 10 pages using LNCS style (excluding bibliography). Papers must be submitted electronically via the workshop website: <http://cs.nju.edu.cn/boyland/ftjp>.

An informal proceedings will be made available to workshop participants. Papers will also be available from the workshop web page. There will be no formal publication of papers. We intend to invite selected papers for a special journal issue as a follow-up to the workshop, as has been done for some previous FTfJP workshops.

Indeed, after the workshop, we invited eight papers to submit updated papers for a special issue of JOT (Journal of Object Technology). Six teams of authors have responded positively thus far.

2 People

2.1 Programme Committee

- Cyrille Valentin Artho (RCIS/AIST, Japan),
- Frank S. de Boer (CWI, Netherlands),
- Fabrice Bouquet (University of Franche Comté, France),
- John Boyland (Nanjing University, China and University of Wisconsin-Milwaukee, USA) chair,
- Alex Buckley (Sun Microsystems, UK),
- Patrice Chalin (Concordia University, Canada),
- Dave Clarke (CWI, Netherlands),
- Paola Giannini (University of Piemonte Orientale, Italy),
- Marieke Huisman (INRIA Sophia Antipolis, France),
- Giovanni Lagorio (Università di Genova, Italy),
- Gary T. Leavens (Iowa State University, USA, now Central Florida University, USA),
- Francesco Logozzo (Microsoft Research, USA),
- Wojciech Mostowski (Radboud University Nijmegen, Netherlands),
- Wolfram Schulte (Microsoft Research, USA).

2.2 Organizers

- John Boyland (University of Wisconsin-Milwaukee, USA) co-chair,
- Sophia Drossopoulou (Imperial College, UK),
- Susan Eisenbach (Imperial College, UK),
- Gary T. Leavens (Central Florida University, USA),
- Peter Müller (ETH Zürich, Switzerland),
- Arnd Poetzsch-Heffter (Universität Kaiserslautern, Germany),
- Erik Poll (Radboud University Nijmegen, Netherlands), co-chair.

2.3 Participants

Suad Alagic, University of South Maine, alagic@cs.usm.maine.edu
Jonathan Aldrich, Carnegie Mellon University, aldrich@cs.cmu.edu
Philippe Altherr, Google, paltherr@google.com
Davide Ancona, DISI - University of Genova, davide@disi.unige.it
Eric Bodden, McGill University, eric.bodden@mail.mcgill.ca
John Boyland, University of Wisconsin-Milwaukee, boyland@uwm.edu
Alex Buckley, Sun Microsystems, alex.buckley@sun.com
Nicholas Cameron, Imperial College, London, ncameron@doc.ic.ac.uk
Patrice Chalin, Concordia University, chalin@encs.concordia.ca
Julien Charles, INRIA, julien.charles@inria.fr
Dave Clarke, CWI, Amsterdam, dave@cw.nl
Curt Clifton, Rose-Hulman Institute of Technology, clifton@rose-hulman.edu
Mario Coppo, Università di Torino, coppo@di.unito.it
Vincent Cremet, (no institution specified), vincent.cremet@gmail.com
Markus Degen, Universität Freiburg, degens@informatik.uni-freiburg.de
Mariangiola Dezani, Università di Torino, dezani@di.unito.it
Werner Dietl, ETH Zürich, dietlw@inf.ethz.ch
Sophia Drossopoulou, Imperial College, London, sd@doc.ic.ac.uk
Gilles Dubochet, Ecole Polytechnique Fédérale de Lausanne, gilles.dubochet@epfl.ch
Erik Ernst, University of Aarhus, eerst@daimi.au.dk
Manuel Fähndrich, Microsoft Research, maf@microsoft.com
Diego Garbervetsky, Universidad de Buenos Aires, diegog@dc.uba.ar
Elena Giachino, Università di Torino, giachino@di.unito.it
Paola Giannini, Università Piemonte Orientale, giannini@mf.n.unipmn.it
Philipp Haller, Ecole Polytechnique Fédérale de Lausanne, philipp.haller@epfl.ch
Clément Hurlin, INRIA, clement.hurlin@inria.fr
Atsushi Igarashi, Kyoto University, igarashi@kuis.kyoto-u.ac.jp
Einar Broch Johnsen, University of Oslo, einarj@ifi.uio.no
Christine Kehyayan, Lebanese American University, christine.kehyayan@lau.edu.lb
Eric Kerfoot, University of Oxford, eric.kerfoot@comlab.ox.ac.uk
Neelakantan Krishnaswami, Carnegie Mellon University, neelk@cs.cmu.edu
Gary T. Leavens, University of Central Florida, leavens@eecs.ucf.edu
Francesco Logozzo, Microsoft Research, logozzo@microsoft.com
Rosemary Monahan, National University of Ireland, Maynooth, rosemary.monahan@nuim.ie
Peter Müller, Microsoft Research, mueller@microsoft.com
Matthew Parkinson, University of Cambridge, matthew.parkinson@cl.cam.ac.uk
Arnd Poetzsch-Heffter, Universität Kaiserslautern, poetzsch@informatik.uni-kl.de

John Potter, University of New South Wales, Sydney,
 potter@cse.unsw.edu.au

Chieri Saito, Kyoto University, saito@kuis.kyoto-u.ac.jp

Yannis Smaragdakis, University of Oregon, yannis@cs.uoregon.edu

Daniel Wasserrab, Universität Passau, daniel.wasserrab@uni-passau.de

Stefan Wehr, Universität Freiburg, wehr@informatik.uni-freiburg.de

Tobias Wrigstad, Stockholm University, tobias@dsv.su.se

Elena Zucca, DISI - University of Genova, zucca@disi.unige.it

Johan Östlund, Stockholm University, johano@dsv.su.se

3 Summary of Contributions

There were 20 submissions. Each submission was reviewed by at least 3 programme committee members. The committee decided to accept 9 papers.

The workshop was structured as a small conference with sessions of paper presentations. Each session specialized in a particular topic. The following summaries were originally written by the session chairs.

3.1 Session 1: Types

The first session of FTfJP was devoted to Types and was chaired by Dave Clarke (CWI). The three papers represent three of the various trends in type systems work, namely, understanding concepts from existing programming languages, transferring constructs from one language (or from theory) to another, and understanding new type-theoretic constructs in terms of more traditional constructs.

Nicholas Cameron presented “Towards an Existential Types Model for Java with Wildcards”, which was joint work with Erik Ernst and Sophia Drossopoulou. Wildcards were introduced in to Java generics to soften the mismatch between subtyping and parametric polymorphism. A type system including wildcards has never been proven sound. This paper formalises wildcards using an extension of FGJ (featherweight generic Java) with existential types.

The paper proves that the calculus, called $\exists J$, is type sound, and illustrates how it models Java’s wildcards. $\exists J$ is not, however, a full model for Java’s wildcards as it does not support lower bounds for wildcards. It turns out that it cannot easily be extended to deal with lower bounds. The paper discusses how this issue can be resolved in a type sound way. The model needed to go beyond the standard existential types model due to peculiarities in Java’s Wildcards.

Philippe Altherr presented “Adding Type Constructor Parameterization to Java”, which was joint work with Vincent Cremet. This paper presented a generalization of Java’s generics to enable parametrization by type constructions, that is, functions from types to types. The extension was formalized as a calculus called $FGJ\omega$, which is proven safe and decidable. The extension is motivated by two examples, namely the definition of generic datatypes with binary methods and generalized algebraic datatypes (GADTs).

Type constructors are found in the programming language Haskell, as well as higher-order logic and other type theories. This paper follows the trend of transferring ideas from type theory and research programming languages such as Haskell into mainstream languages such as Java.

Chieri Saito presented “The Essence of Lightweight Family Polymorphism”, which was joint work with Atsushi Igarashi. The paper discusses the formal calculus `.FJ` that was introduced to model lightweight family polymorphism, a programming style that supports reusable yet type-safe mutually recursive classes. This style of programming originates in Beta and has received a lot of attention in recent years.

The paper clarifies the essence of `.FJ` by providing a formal translation from `.FJ` into a variant of `FGJ` extended with a variant of F-bounded polymorphism which allows self types to appear in mutually recursive constraints on type variables. The correspondence between the two languages is achieved without losing type safety.

3.2 Session 2: Languages and Verification

The second session was chaired by Gary Leavens.

The paper “Separating Type, Behavior, and State to Achieve Very Fine-grained Reuse,” by Viviana Bono, Ferruccio Damiani, and Elena Giachino describes a language design that tries to achieve orthogonality in several mechanisms, with the aim of fostering reuse. In particular it tries to separate the mechanisms that allow reuse of: state declarations (records), types (interfaces), and behavior (traits) from each other and from the mechanism that composes these into objects (classes). In this way the language distinguishes itself from other advanced languages, such as Scala, in which traits are also types. The paper describes a calculus, Featherweight Compositional Java. This calculus combines nominal (by-name) and structural type checking, in that method parameters are nominally typed and uses of `this` are structurally typed. Its type system and a translation that flattens structures into the calculus Featherweight Trait Java with Interfaces is proved to preserve typings and to be sound.

The paper “Modular Verification of the Subject-Observer Pattern via Higher-Order Separation Logic,” by Neelakantan Krishnaswami, Lars Birkedal, and Jonathan Aldrich shows how to use higher-order separation logic to modularly verify uses of the Subject-Observer pattern. Modularity means that the subject and observer are hidden from each other, but must not interfere with each other. Furthermore, different observers may have different invariants. The approach uses quantification over predicates to hide representations. It also uses lists of higher-order predicates to maintain a list of the invariants corresponding to each observer. For reasoning about callbacks, the approach uses hypothetical separation properties. The paper formalizes the programming language and the higher-order separation logic. It gives detailed specifications for an instance of the Subject-Observer pattern, including a client program.

The paper “Automatic verification of textbook programs that use comprehensions,” by K. Rustan M. Leino and Rosemary Monahan address the problem of automatic verification of generalized quantifiers, such as `sum`, `min`, and `max`. The paper describes an approach taken in `Spec#`, a formal specification language for C#. `Spec#` has quantifiers like `sum {int k in (1:3); k*k}` which denotes $1 \times 1 + 2 \times 2 = 5$. The difficulties in automating this are: (i) how to encode such generalized quantifiers and comprehensions for an automatic, first-order theorem prover (such as `Simplify`), and (ii) what “inductive axioms” generalized quantifiers and comprehensions in a first-order prover. The approach to solving the first problem is to specify two new function symbols for each kind of quantifier. The axioms used to solve the second problem make these function symbols synonyms, but only in one direction, which avoids the problem of recursive loop triggers. Differences in such axiomatizations affect performance, sometimes dramatically, and such considerations are described in the paper.

3.3 Session 3: Analysis

This session, unlike the previous session, presented papers for the *automatic* analysis of object-oriented programs. It was chaired by Francesco Logozzo.

Manuel Fähndrich gave a talk on a new static analysis that he developed together with Diego Garbervetsky and Wolfram Schulte. The goal of the analysis is to check the presence of re-entrant calls on objects for which the invariant may not hold. The analysis is divided into two stages. First, it uses a modular point-so analysis to detect re-entrant calls. Then, it performs a simple data-flow propagation to check whether the invariant of the receiver of a re-entrant call holds or not. The analysis is supported by an implementation that was able to find that most re-entrant calls are direct calls. The two main differences of this work with respect to others presented in the FTfJP’07, but also in the IWACO’07 workshop, is (i) that the analysis is run on the full program, and (ii) it does require minimal user annotations.

Patrice Chalin presented the work that Frederic Rioux and himself have conducted on the re-definition of the (concrete) semantics of expressions appearing in JML assertions. The new semantics is based on the notion of *strong validity* which means that an expression is true iff (i) it is defined and (ii) it evaluates to `true`. Such a definition implies that when checking dynamically an assertion, it evaluates to true iff (i) it does not raise an exception (*e.g.* because a division by zero) and (ii) it evaluates to true. The implementation is built on the top of the `MultiJ` compiler. There are two main advantages of this approach: (i) it provides a more programmer-friendly semantics of assertions, and (ii) it generates more compact runtime checks.

In his talk Manuel Hermenegildo presented some ongoing work that he is doing with his students Jorge Navas and Mario Méndez-Lojo for developing a generic static analysis framework for the Java bytecode based on Abstract Interpretation. The kernel of the work is the fixpoint computation engine which is smart enough to infer non-trivial postconditions for recursive functions. Manuel illustrated the algorithm, inspired by previous work in Logic programming, and

he gave a brief demo of the tool. The main difference with other works presented during the workshop is that this tool (i) is completely automatic, (ii) it is oriented to the inference of numerical properties to be used for the absence of runtime errors, and (iii) it is adaptive, in that if information on the caller context is present, then the analysis makes use of it.

3.4 Session 4: Panel Discussion

In the last session of the workshop, a panel discussion investigated future combinations of object-oriented programming, modeling and verification. The panel consisted of four experts with different foci in the field:

- Jonathan Aldrich: Architectural aspects
- Manuel Hermenegildo: Static analysis
- Manuel Fähndrich: Verification, object-oriented programming
- Gary T. Leavens: Modeling languages, dynamic checking

The panel was moderated by Arnd Poetzsch-Heffter. In the following, we shortly summarize the initiating statements of the panelists and the discussion.

Architectural aspects. Jonathon Aldrich summarized central challenges for the verification of object-oriented programs, in particular aliased state, higher-order and event-based code, inheritance, subtyping, and concurrency. He illustrated the problems to find bugs using a graphical text editor as example: a misbehavior in a text area may result from a long event chain touching many parts of the implementation. A still open question is whether current techniques to modularity like separation logic and ownership disciplines are flexible enough to handle such scenarios. It will be important to use software architecture as a backbone to integrate code-oriented modularization like ownership structures with other techniques like permission-based reasoning and event tracking.

Static analysis. As central goals, Manuel Hermenegildo stressed the need to develop improved programming environments. They should increase programmer efficiency and better integrate verification, analysis and optimization into everyday programming. He sketched the CiaoPP approach in which static analysis, partial evaluation, theorem proving and runtime test generation works hand in hand. It will as well be important to be able to ship the derived abstractions together with the program in the sense of an “abstraction carrying code”. Languages with a higher abstraction level will not only be needed to simplify the programming task, but as well to integrate programs, abstract interpretation results and other program properties.

Programming and verification. Manuel Fähndrich opened up the discussion towards new language designs. With software systems increasing in size and a hardware trend to multi-core architectures, it will become crucial to be able to develop less buggy and highly concurrent programs in the future. It might be questioned whether the OO paradigm gives a sufficient answer to this challenge:

Current support for alias control and referential transparency is weak in OO languages; and OO concepts and constructs for concurrency are fairly low level. Functional programming provides better techniques to control mutability, and simplifies verification and concurrent programming. An interesting trend that might combine the programming paradigms are small components encapsulating data and control.

Better help for the programmer. Gary T. Leavens added two further issues to the discussion. Firstly, he reflected on how we can better help the programmer. Instead of straitjacketing ordinary programmers with difficult techniques and theories, academia should package their techniques and tools according to the programmer's needs. In particular, support should be better aligned with the design goals of the programmer. For example, if security is an issue, specific support is needed for this requirement; and such a support can crosscut different areas of technology. Gary Leaven's second issue concerned the extension and integration of new trends and developments in programming and formal methods. E.g. how can XML data handling be integrated into interface specifications, how can formal techniques be applied to aspect-oriented programming and highly dynamic programs? In addition, we still have to realize the treatment of temporal, time and space properties in the current verification and tooling landscape.

During and after these statements we had a lively discussion: How should strong, weak and no typing be used and integrated into programming and specification? Is functional programming a key to meet the challenge of correct concurrent programs? Although the emotions underlying such questions in the last century could still be felt, the bottom line of the discussion seemed to be that a well-engineered combination of the developed techniques is the challenge of today and the way to go.

4 Conclusions

Object-oriented programming in the style of Java or C# is now mainstream. Thus the topic of this workshop, applying formal techniques to these languages, is more important than ever. We are seeing large-scale practical efforts, not least JML and Spec#, being used by growing numbers of people. In this way the hard work of researchers in formal techniques is having a broad impact in programming. It is a privilege to be part of this process.

Acknowledgments

As chair, I (John Boyland) thank the providers of EasyChair which made chairing the workshop much easier than if I had to do everything myself, especially while temporarily residing in China. Thanks also to the programme committee members who put in a lot of work in a very tight reviewing schedule, and for the productive round-the-globe email discussion.

I thank Nanjing University and the State Key Laboratory for Novel Software Technology for hosting my stay in China and helping put together the technical report of contributions.

Web Resources

The homepages of the programme committee members and all the individual papers are available from the workshop web site:

`http://www.cs.uwm.edu/faculty/boyland/ftjp/index.html`