Executive Summary

Thomas Ball[1], Jürgen Giesl[2],
Reiner Hähnle[3], and Tobias Nipkow[4]

[1] Microsoft Research, Redmond, WA, USA
tball@microsoft.com
[2] RWTH Aachen, Germany
giesl@informatik.rwth-aachen.de
[3] Chalmers University of Technology, Gothenburg, Sweden
reiner@chalmers.se
[4] Technische Universität München, Germany
nipkow@in.tum.de

Abstract. This seminar was the ninth in the series of the Dagstuhl "Deduction" seminars held biennially since 1993. Its goal was to bring together the closely related but unnecessarily disjoint communities of researchers working in interactive and automatic program verification.

Keywords. Formal Logic, Deduction, Artificial Intelligence

1 Motivation of the Seminar

Throughout the history of modern logic, there have been two strands of research: finding natural inference systems for a given problem domain and finding automatic procedures for solving specific logical problems. In computer science, these two strands became *interactive* and *automated* deduction. Powerful systems emerged in both camps (Coq, Isabelle, etc. versus Spass, Vampire, etc.), conferences were established, and separate communities developed.

However, none of the two kinds of systems were ideal for program verification, one of the principal application areas of deduction in informatics. The interactive tools lacked the necessary automation and the automatic tools failed to cater for important aspects like arithmetic. And neither scaled well. Therefore a separate third, application-driven set of techniques and tools were developed. These are based on powerful automatic procedures for particular logical theories, ranging from propositional logic to arithmetic, and their combination, most notably in the form of SMT solvers, such as Simplify, Yices, etc. At the same time they were integrated with techniques from program analysis and automata theory, resulting in powerful tools like Microsoft's SLAM which is used for fully automatically verifying C programs with thousands of lines of code against low-level properties of code. Again, a separate scientific community evolved.

Each of the three paradigms and communities has led to impressive achievements which are briefly highlighted in the following.

1. Interactive Deduction The interactive approach has been applied successfully to complex and abstract verification tasks with a strong mathematical flavor: verification of floating point algorithms (J. Harrison in the HOL Light system at Intel), the proof of the Four Color Theorem (G. Gonthier in Coq at Microsoft), verifying the computational parts of Hales' proof of the Kepler Conjecture (T. Nipkow in the Isabelle system), etc. On the software side, the verification of a C Compiler (X. Leroy in Coq) and of an operating system kernel (the *Verisoft* project in Isabelle) are outstanding results.

2. Automated Deduction The Automated Theorem Proving (ATP) community concentrated for many years on solving mathematical problems with first-order theorem provers. A breakthrough was the automated proof of the long-standing Robbins conjecture (B. McCune with EQP) in 1996 which had been open for 60 years. Since then ATP focused more on applications from computer science such as security protocol analysis (C. Weidenbach with Spass in 1999, followed by others) or web ontology reasoning (I. Horrocks & A. Voronkov with Vampire in 2006). These case studies showed that general first-order theorem provers have reached a state of maturity where they often outperform dedicated tools in specific application areas. An important insight that strongly influenced recent ATP systems is that many practical problems are expressible in restricted forms of first-order logic. On the other hand, it is essential to be able to recognize satisfiability and to compute models. Daniel Jackson et al.'s Alloy system (2000–) was from the start conceived as an automatic first-order model analysis tool with a restricted, but syntactically sugared input language. Propositional logic had traditionally been considered as too inexpressive for modeling complex deduction problems, but following the performance leap of SAT solvers in the wake of Chaff (Malik et al., 2001), the sheer size of problems that can be handled made SAT interesting as a backend for reduction techniques. In model checking, SAT solvers became an alternative to BDDs. More recently, first-order (theory) reasoning and SAT solving are combined in SMT solvers. These are indispensable in program analysis tools (SLAM, ESC/Java, Spec#, PVS, KeY, etc., see also the next paragraph).

3. Deduction Techniques in Program Analysis Program Analysis focuses on proving properties of programs. Inference of invariants (as well as contracts of subprocedures) is a key part of program analysis. An important strand of research is how to power up SMT solvers and invariant inference from quantifier-free formulas to quantified formulas (over various theories), which are essential to describe properties of unbounded collections. Fully automated analyses rely on abstraction to ensure finite computation of fixed points. In the world of SMT solvers, McMillan has pioneered the use of interpolants to perform domain abstraction (no join operator needed) so that one can compute symbolic fixed points directly with an SMT solver. In program analysis it is important to take advantage of the rich structure of programs (procedures, types, control-flow) to make the search process more efficient. Recent work by Babic shows how to use program structure to efficiently encode verification conditions, as well as to use the incremental nature of SMT solvers to perform very efficient interprocedural analysis via goal-directed inlining. Here, it becomes clear that the very narrow interface of the SMT solver must be widened to accommodate a richer interaction between the program analysis client and the underlying decision procedures in an SMT solver. There is much progress in program analysis by combining dynamic and static approaches. In particular, one performs testing on concrete inputs with symbolic execution using SMT solvers that have the capability to go beyond what either technique can accomplish alone, as shown in recent work by Godefroid et al. Finally, there is increasing programming language support for program analysis, where a language integrates ways to talk about programs with ways to talk about formulas and abstractions as in the Saturn system by Hackett et al. This is related to the integration of verification-condition-based approaches to program verification, such as in Spec#/Boogie, with interactive theorem provers or, conversely, as in KeY, to automation of an interactive prover with SMT techniques.

2 Goals of the Seminar

There is clearly not just competition but also synergy among the three different approaches discussed in the previous section. For example, SMT solvers are successfully applied in program analysis and first-order provers are used in interactive systems. The KeY system is the result of combining an interactive approach to program verification with a high degree of automation. However, such combinations often raise questions and problems that require more interaction between the communities involved. These include

- exchange of formats for theories and proofs
- encoding of higher-order problems into first-order logic
- extension of automatic first-order provers with specific theories or abstraction techniques
- using automatic provers as servers that allow to incrementally add and delete formulas
- orchestration of interleaved automated and interactive inference
- rendering results of automated tools in human-readable form
- generation of proof certificates
- exploiting synergies between Abstract Interpretation and SMT solvers
- invariant inference, especially for quantified formulas
- exploiting program structure for efficient search
- test generation and support from SMT solvers
- programming language support for program analysis

The Dagstuhl seminar brought together the best researchers working on interactive and automatic deduction methods and tools, with a special emphasis on applications to program analysis and verification.

In total we had 52 participants, mostly from Europe, but also from USA, Israel, and Australia. A good balance between more senior and junior participants was maintained. The program consisted of 39 relatively short talks, which gave ample time for discussion, both during and after the talks as well as during the meals and in the evenings. Altogether, we perceived the seminar as a very successful one, which allowed for cross-fertilization between research on interactive and on automated deduction. Moreover, it also helped to bridge gaps between foundational research on these topics and application-driven approaches; e.g., the transfer of new theoretical results into applications, or the discovery of new research problems motivated by applications.