Software Verification: Infinite-State Model Checking and Static Program Analysis

Dagstuhl Seminar 06081
February 19–24, 2006

Parosh Abdulla[1], Ahmed Bouajjani[2], and Markus Müller-Olm[3]

[1] Uppsala Universitet, Sweden
[2] LIAFA–Université Paris VII, France
[3] Westfälische Wilhelms-Universität Münster, Germany

Abstract. This is the executive summary of Dagstuhl Seminar 06081, "Software Verification: Infinite-State Model Checking and Static Program Analysis". The seminar was held from February 19 to February 24, 2006, at the International Conference and Research Center for Computer Science Schloss Dagstuhl, Germany.

1 Introduction

Software systems are present at the very heart of many daily-life applications, such as in communication (telephony and mobile Internet), transportation, energy, health, etc. Such systems are often *critical* in the sense that their failure can have considerable human or economical consequences. Therefore, there is a real need of rigorous and automated methods for software development which guarantee a high level of reliability. It is well-known that, to ensure reliability, development methods must include *algorithmic analysis and verification techniques* which allow (1) automatic detection of defective system behavior and (2) automatic correctness analysis of systems with respect to their specifications.

For modern software systems, many complex aspects are of crucial importance such as manipulation of data over unbounded domains (integers, reals, etc.), object-orientation, dynamic memory structures (creation of objects, pointer manipulation), dynamic control (multi-threading, procedure calls), synchronization between concurrent processes, parameterization, real-time and hybrid modeling, etc. The development of software analysis and verification methods and tools allowing to deal efficiently with such aspects constitutes a major scientific and technological challenge. Two important and quite active research communities are particularly concerned with this challenge: the community of computer-aided verification, especially the community of (infinite-state) model checking, and the community of static program analysis. The two communities are adopting different approaches: The model-checking community studies complete methods for the verification/analysis of abstract models. These abstract models may involve infinity features such as those mentioned above. On the other hand, the

Dagstuhl Seminar Proceedings 06081
Software Verification: Infinite-State Model Checking and Static Program Analysis
http://drops.dagstuhl.de/opus/volltexte/2006/797

program analysis community works with approximate analyses applied on formalisms that are closer to programming languages and specification formalisms used in practice.

While the approaches and the developed techniques are different, the two communities share a common mathematical background and their methods are based on common basic concepts and principles: transition systems and automata-based models, abstractions, fixpoint computations, reachability analysis based on symbolic representation structures of (potentially infinite) sets of configurations, etc.

2 Dagstuhl Seminar 06081

Dagstuhl Seminar 06081 "Software Verification: Infinite-State Verification and Static Program Analysis" brought together 51 researchers from these two communities in order to (1) improve and deepen the mutual understanding of the developed technologies, (2) compare these technologies and identify complementary aspects, and (3) trigger collaborations leading to new developments. The seminar was held from February 19 to February 24, 2006, at the International Conference and Research Center for Computer Science Schloss Dagstuhl, Germany. The participants came from 12 countries, mainly from Europe and the US. More specifically, 1 participant came from Austria, 2 from Belgium, 1 from The Czech Republic, 1 from Denmark, 12 from France, 12 from Germany, 3 from Israel, 1 from Italy, 1 from Russia, 2 from Switzerland, 5 from the United Kingdom, and 10 from the United States.

In 31 talks, the participants presented results of their recent research. These talks touched many issues of automatic software verification including: abstraction techniques, invariant generation, termination analysis, automata-based representation structures and applications of regular model checking, analysis of pointer and heap structures, timed and hybrid systems, multi-threaded programs, parameterized systems, probabilistic models and verification methods, etc.

In a final session on Friday morning, the participants discussed how to progress further in the field of automatic software verification and how to get the developed technology to practice. Lack of common benchmarks and notations and a tendency to evaluate techniques on academic toy examples rather than on real code (e.g., from Java libraries) were identified as obstacles to fair comparison of different analyses and broader dissemination of the results. Reasons for this are that the area is quite broad with many different aspects, and that some of the techniques are still in an experimental stage.

During the evenings and nights, the great facilities of Dagstuhl offered plenty of opportunities to enjoy other pleasures besides science, among them beer, cheese, and music (in alphabetical order). On Wednesday afternoon there was an opportunity to join an excursion to Luxembourg city.

3 Main Results and Approaches

This section summarizes briefly some of the main results and approaches which have been presented at the seminar.

Abstraction techniques. A lot of effort is devoted to automated data abstraction methods, following the idea to combine predicate abstraction with counter-example guided abstraction refinement (CEGAR). One of the main issues in this context is to provide powerful and scalable techniques for automatic detection of abstractions which are sufficiently accurate for the given property. Recent developments on this topic are based on using the notion of interpolants. Another important issue is to adapt these techniques to programs with complex control features such as procedure calls and multi-threading. Recent results propose the extension of the CEGAR framework to such programs using abstract model-checking techniques for communicating pushdown systems.

Abstraction techniques can also be used in order to enforce termination of symbolic reachability analysis. Instead of checking a property on an abstract model (which is the approach generally adopted in the model-checking community), it is possible to introduce abstraction in the analysis by considering approximate successor computations (which is usually the approach followed in the abstract interpretation community). Recent work tries to define such abstract analysis algorithms which are complete for significant classes of models (i.e., they can decide for these models whether some reachability property holds or not). Interesting results have been obtained concerning complete forward reachability analysis algorithms for the class of well-structured systems (such as Petri nets and lossy channel systems).

Automata-based techniques. Finite-state (word/tree) automata can represent potentially infinite (regular) sets of (encodings of) system configurations. These representations can be used in the computation of reachability sets (or approximations of these sets) for the given system (when each operation of the system is modeled as a transformation on the word/tree encodings of the configurations). Techniques such as meta-transition based (or transitive closure based) acceleration, widening, or (finite range) abstraction are used in order to ensure the termination of the analysis. This approach (more and more known under the name of "regular model checking") has been adopted for dealing with various classes of systems such as counter systems (using binary encodings of integers), pushdown systems, FIFO channel systems, parameterized networks of processes, and more recently, systems with dynamic linked structures (such as lists, doubly linked lists, trees, etc).

Pointer and heap structure analysis. Reasoning about programs with pointers and dynamic management of the memory is one of the most challenging issues in software verification. A lot of effort is devoted to finding powerful and scalable methods and techniques dealing with significant classes of such programs.

Several of these works concentrate on the case of programs with lists (with possibility of sharing and cyclicality). Among these works there are approaches based on (1) logics such as fragments of separation logic or the first-order theory of Boolean algebra of sets, (2) word abstract regular model checking, (3) translations to counter automata (where counters allow to reason about the lengths of the lists), (4) instrumentation of programs, etc.

Other works provide approaches and frameworks for dealing with more general classes of programs based on (1) logics such as separation logic or fragments of first-order logic on graphs with reachability predicates, (2) tree abstract regular model checking, (3) graph rewriting, etc.

Few other works propose techniques allowing to reason on both the shapes of the dynamic linked structures and on the data they carry (i.e., the values in the data fields in each object of the structure).

Termination analysis. Several groups are developing approaches for automatic verification of program termination. One of the approaches is based on checking the existence of a decreasing ranking function. Recent developments concern the reduction of this problem to an arithmetical decision problem. Another interesting approach is based on checking the existence of a finite union of well-founded relations covering the transitive closure of the transition relation of the program. Recent work concerning this approach proposes systematic techniques for discovering such well-founded relations iteratively using a counter-example guided principle.

More specific approaches have been developed for dealing with programs with lists. By including information on the length of the lists in the program models, it is possible to reduce the termination problem of such programs to the termination analysis of programs with counters. The latter problem can be solved using the techniques mentioned earlier.

Probabilistic models. Recent research directions consider the verification problem of probabilistic (infinite-state) models of programs. Impressive new decidability results have been obtained recently concerning probabilistic pushdown systems and probabilistic lossy channel systems. Also decidability of equivalence and refinement checking of probabilistic programs have been studied.

Parametric verification. Parametric verification intends to verify systems comprising a network of an arbitrary number of identical or similar components running concurrently. Typical examples of such systems are mutual exclusion, cache coherence, and broadcast protocols. Recent work in this area is concerned with inferring invariants of such networks automatically. Another technique uses abstraction: it views the network from the perspective of one component and abstracts the other components by a combination of predicate and counter abstraction.

Timed systems. In the area of timed systems, recent decidability results about metric temporal logics and dense-time Petri nets were presented at the seminar. Although the details are quite different, a recurring idea is to reduce the problem to a problem about an untimed or discrete model for which decidability is well-understood, e.g., using the theory of well-structured systems based on the notion of well-quasi orderings.

Acknowledgments

As the organizers we wish to thank the staff of the Dagstuhl office in Saarbrücken and at Schloss Dagstuhl itself for their great support in organizing and running the seminar. We also would like to thank all the participants of the seminar for their talks and their expertise that made the seminar such a fruitful research venue.