New Results in Software Model Checking and Analysis

Corina S. Păsăreanu
Carnegie Mellon University/NASA Ames Research Center,
Moffett Field, CA 94035, USA,
e-mail: corina.s.pasareanu@nasa.gov

The articles have been carefully reviewed and they are based on papers that were considered to be among the best at the SPIN 2009 model checking workshop [11]. The topics addressed range from probabilistic model checking and parallelization techniques for improved scalability to data race detection, symbolic analysis and model checking for security.

The first article [5], presents an effective technique for computing the probability of reaching a given set of states in a parametric Markov model. Such models can be used to reason about quantitative properties in systems where certain aspects are not fixed, but rather depend on parameters. Previous work [4] has suggested to convert the Markov chain into a finite automaton, equivalent to a regular expression. The expression can be evaluated to a closed form function representing the reachability probability. The bottleneck of the approach lies in the growth of the regular expression with the number of states. The authors propose to remedy the problem by intertwining the regular expression computation with its evaluation. This results into a practical method that has been implemented in the PARAM tool and has been demonstrated experimentally on network protocols.

The second article [2] is also concerned with probabilistic reasoning, in the context of the PRISM model checking tool [9], where the satisfaction of desired properties is quantified with some probability. The authors propose algorithms for parallel probabilistic model checking using general purpose graphic processing units. The proposed improvements target the numerical computations of the traditional sequential algorithms since these computations can be parallelized efficiently on graphic processors. The parallel algorithms have been implemented in the PRISM model checker and have been evaluated on several case studies, showing significant speed-up.

The third article [8] addresses the problem of verifying data consistency in concurrent Java programs. The
work targets data races caused by inconsistent accesses to multiple fields of an object – the so-called atomic-set serializability problem. Previous work used abstraction techniques to translate a concurrent Java program into an EML program, a modeling language based on push-down systems and a finite set of re-entrant locks, and used only a semi-decision procedure to check the program. The present article extends that work by describing a full decision procedure for verifying data consistency, i.e., atomic-set serializability, of an EML program. The procedure has been implemented and it has been applied to detect both single-location and multi-location data races in models of concurrent Java programs.

The fourth article [10] presents a generic technique for creating the basic primitives used in symbolic program analysis: forward symbolic evaluation, weakest liberal precondition, and symbolic composition. Using this technique, one can automatically generate an implementation of a (forward or backward) symbolic program execution at the cost of writing only the specification of the concrete program semantics – in the form of an interpreter for the language of interest. The technique can be used for programming languages with pointers and arithmetic operations. The technique has been implemented and it has been used to generate symbolic-analysis primitives for the x86 and PowerPC instruction sets. The symbolic analysis generated with the generic technique presented here can be used in software model checking tools such as SLAM [1] and Blast [6], as well as in other automated bug-finding tools that rely on symbolic reasoning [14, 15].

Finally, the fifth article [13] presents an application of the SPIN model checker [7] to checking signature specifications. Signatures are matching rules that are used in intrusion detection systems when searching for attack traces in the recorded audit data based on pre-defined patterns. Intrusion detection systems are one of the most important means to protect information technology systems [12]. The effectiveness of an intrusion detection system depends on the adequacy of the signatures, which are usually defined empirically. Modeling a new signature is time-consuming and error-prone; consequently the modeled signature needs to be tested carefully. In this article, the authors present an approach to automatically checking signature specifications using the SPIN model checker. The signatures are modeled in the specification language EDL (a variant of Petri-nets) and then translated into PROMELA, the input language of the SPIN model checking tool. SPIN is used to find specification errors, which are modeled using linear temporal logic.

In conclusion, the articles enclosed here describe new results in software model checking and analysis. The presented techniques are most useful at finding subtle and costly errors that can not be found with traditional testing alone. The techniques have been implemented in robust tools and therefore show good promise for adoption in industry.

References