Dissertationen

David Faragó:
Model Checking and Model-Based Testing: Improving Their Feasibility by Lazy Techniques, Parallelization, and Other Optimizations

Promotion: Dr. rer. nat., KIT, Fakultät für Informatik
Erstgutachter: Prof. Dr. Peter H. Schmitt, Karlsruhe Institute of Technology (KIT)
Zweitgutachter: Prof. Dr. Shmuel Tyshberowicz, Academic College Tel Aviv Yaffo
Datum der Prüfung: 29.01.2016
Veröffentlichung: https://publikationen.bibliothek.kit.edu/1000059473

Kurzfassung:

Introduction Software today is ubiquitous, its complexity and correctness requirements increase. By now, quality assurance consumes over 50% of the software development costs, yet incidents due to software bugs cost tens of billions of dollar annually. One of the most promising countermeasure is using Formal Methods, which formally analyze software systems. But due to issues in feasibility, Formal Methods are adopted meagerly in industry.

The goal of the thesis is to increase the feasibility of the Formal Methods model checking and model-based testing: to improve their time and space requirements, but also their expressiveness and usability. Hereby, larger systems can be analyzed and checked for correctness, with lower effort, expertise and risk of missing a bug. This raises the return on investment of these Formal Methods, which is imperative to get adopted broadly in industry.

Contributions The first main contribution of the thesis is increasing the feasibility for model checking one of the most important liveness properties: livelock freedom. A lazy, more feasible and parallel model checking algorithm that is specialized on livelock freedom is derived: DFS_{FIFO}. Partial order reduction is adapted and strengthened.

The second main contribution is increasing the feasibility for model-based testing, especially for systems with uncontrollable nondeterminism. A lazy approach is designed, called LazyOTF. It integrates both on-the-fly and offline MBT by swapping heuristically between phases of state space traversal and test execution. The traversal can thus make use of dynamic information from previous test execution while searching for meaningful test cases. A parallel algorithm following this approach is implemented, as well as various heuristics making use of dynamic information.

All mentioned improvements are substantiated by thorough experiments (over 14 years of accumulated wall clock time):

DFS_{FIFO} was applied on four established protocols; compared to related work, DFS_{FIFO}'s space and time requirements were reduced by a factor of 3 to over 200, its on-the-flyness was over 150 times stronger. However, for parallel DFS_{FIFO} with many parallel processes, on-the-flyness was only 1.75 times stronger. Parallel DFS_{FIFO} achieves almost linear parallel speedup and can handle 4 to 5 more orders of magnitude in combination with partial order reduction. Parallel DFS_{FIFO}'s counterexamples are up to 10 times shorter.

LazyOTF was applied to a web service model using both a simulated and a real system under test. Compared to related work, its meaningfulness and CPU times were exponentially better. Distributed LazyOTF achieved (super-)linear speedup of meaningful test execution and almost linear speedup overall. However, the exemplary dynamic bound heuristics turned out to improve the trade-off between meaningfulness and CPU time only in some cases.

Conclusion The main contributions were making Formal Methods faster and more scalable:

For checking livelock freedom, DFS_{FIFO} reduces both time and space requirements by simultaneously exploring and checking the specification in one pass and without a Büchi product, by improving partial order reduction, and by parallelization with almost linear speedup. However, DFS_{FIFO} cannot be generalized to cover full LTL.

LazyOTF reduces both time and space requirements of model-based testing by traversing subgraphs of the specification for strong guidance, employing heuristics that make use of dynamic information from previous test execution phases, and by parallelization with (super-)linear speedup of meaningful test execution and almost linear speedup overall. However, applying LazyOTF for timed automata is future work.

Since both DFS_{FIFO} and LazyOTF yield shorter result traces, usability is also improved.

Hamed Shariat Yazdi:
Statistical Analysis and Simulation of Design Models Evolution

Promotion: Dr. rer. nat., Naturwissenschaftlich-Technische Fakultät, U. Siegen
Erstgutachter: Prof. Dr. Udo Kelter (U. Siegen)
Zweitgutachter: Prof. Dr. Lefteris Angelis (Aristotle U. Thessaloniki)
Datum der Prüfung: 21.08.2015
Veröffentlichung: http://dokumentix.ub.uni-siegen.de/opus/volltexte/2015/958/

Kurzfassung:

Tools, algorithms and methods in the context of Model-Driven Engineering (MDE) have to be assessed, evaluated and tested with regard to different aspects such as correctness, quality, scalability and efficiency. Unfortunately, appropriate test models are scarcely available and those which are accessible often lack desired properties. Therefore, one needs to resort to artificially generated test models in practice.

Many services and features of model versioning systems are motivated from the collaborative development paradigm. Testing such services does not require single models, but rather pairs of models, one being derived from the other one by applying a known sequence of edit steps. The edit operations used to modify the models should be the same as in usual development environments, e.g. adding, deleting and changing of model elements in visual model editors. Existing model generators are motivated from the testing of model transformation engines, they do not consider the true nature of evolution in which models are evolved through iterative editing steps. They provide no or very little control over the generation process and they can generate only single models rather than model histories. Moreover, the generation of stochastic and other properties of interest also are not supported in the existing approaches.

Furthermore, blindly generating models through random application of edit operations does not yield useful models, since the generated models are not (stochastically) realistic and do not reflect true properties of evolution in real software systems. Unfortunately, little is known about how models of real software systems evolve over time, what are the properties and characteristics of evolution, how one can mathematically formulate the evolution and simulate it.

To address the previous problems, we introduce a new general approach which facilitates generating (stochastically) realistic test models for model differencing tools and tools for analyzing model histories. We propose a model generator which addresses the above deficiencies and generates or modifies models by applying proper edit operations. Fine control mechanisms for the generation process are devised and the generator supports stochastic and other properties of interest in the generated models. It also can generate histories, i.e. related sequences, of models. Moreover, in our approach we provide a methodological framework for capturing, mathematically representing and simulating the evolution of real design models. The proposed framework is able to capture the evolution in terms of edit operations applied between revisions. Mathematically, the representation of evolution is based on different statistical distributions as well as different time series models. Forecasting, simulation and generation of stochastically realistic test models are discussed in detail. As an application, the framework is applied to the evolution of design models obtained from sample a set of carefully selected Java systems.

In order to study the the evolution of design models, we analyzed 9 major Java projects which have at least 100 revisions. We reverse engineered the design models from the Java source code and compared consecutive revisions of the design models. The observed changes were expressed in terms of two sets of edit operations. The first set consists of 75 low-level graph edit operations, e.g. add, delete, etc. of nodes and edges of the abstract syntax graph of the models. The second set consists of 188 high-level (user-level) edit operations which are more meaningful from a developer's point of view and are frequently found in visual model editors. A high-level operation typically comprises several low-level operations and is considered as one user action.

In our approach, we mathematically formulated the pairwise evolution, i.e. changes between each two subsequent revisions, using statistical models (distributions). In this regard, we initially considered many distributions which could be promising in modeling the frequencies of the observed low-level and high-level changes. Six distributions were very successful in modeling the changes and able to model the evolution with very good rates of success. To simulate the pairwise evolution, we studied random variate generation algorithms of our successful distributions in detail. For four of our distributions which no tailored algorithms existed, we indirectly generated their random variates.

The chronological (historical) evolution of design models was modeled using three kinds of time series models, namely ARMA, GARCH and mixed ARMA-GARCH. The comparative performance of the time series models for handling the dynamics of evolution as well as accuracies of their forecasts was deeply studied. Roughly speaking, our studies show that mixed ARMA-GARCH models are superior to other models. Moreover, we discuss the simulation aspects of our proposed time series models in detail.

The knowledge gained through statistical analysis of the evolution was then used in our test model generator in order to generate more realistic test models for model differencing, model versioning, history analysis tools, etc.