AVM 2012

Passau, Germany, May 21–22

— Program —

Events of AVM 2012 Mon 2012-05-21 – Tue 2012-05-22 (Berlin)

09:00 10:00 11:00 12:00 13:00 14:00 15:00 16:00 17:00 18:00 20:00

Mon 5/21	Tue 5/22
Opening 08:30 - 08:45	A Perfect Model for Bounded Verification (Keynote) 08:30 - 09:30
New Trends in Program Synthesis (Keynote) 08:45 - 09:45	Coffee Break 09:30 - 10:00
Coffee Break 09:45 - 10:15	Session Schneeberg 10:00 - 11:15
Session Zugspitze 10:15 - 11:30	Lunch 11:15 - 12:25
Lunch 11:30 - 12:50	Session Danube 12:25 - 14:30
Session Black Forrest 12:50 - 14:30	Tea Break 14:30 - 15:00
RiSE PI Meeting @ Senate Chamber 12:50 - 14:30	Session Adamello 15:00 - 16:40
Social Event - Ausflug Klettergarten 14:30 - 21:00	Closing 16:40 - 16:55
	Social Event - Ilztal Hike (optional) 17:15 - 21:00

Keynotes

New Trends in Program Synthesis (Keynote) (Mon, May 21, 08:45 – 09:45)

Thomas A. Henzinger
IST Austria, Austria

Abstract: Program synthesis has seen a revival in recent years, based on two trends. First, partial program synthesis shifts the emphasis from the problem of synthesizing programs from specifications to the problem of completing a partial program so that it satisfies a desired property. Second, quantitative program synthesis aims to find a solution that optimizes a given criterion. Put together, quantitative partial program synthesis carries the hope of replacing certain aspects of manual programming by automatic procedures. We survey some recent results in this direction.

A Perfect Model for Bounded Verification (Keynote) (Tue, May 22, 08:30 – 09:30)

Javier Esparza
TU Munich, Germany

Abstract: A class of languages C is perfect if it is closed under Boolean operations and the emptiness problem is decidable. Perfect language classes are the basis for the automata-theoretic approach to model checking: a system is correct if the language generated by the system is disjoint from the language of bad traces. Regular languages are perfect, but because the disjointness problem for CFLs is undecidable, no class containing the CFLs can be perfect. In practice, verification problems for language classes that are not perfect are often under-approximated by checking if the property holds for all behaviors of the system belonging to a fixed subset. A general way to specify a subset of behaviors is by using bounded languages (languages of the form w1* ... wk* for fixed words w1,...,wk). A class of languages C is perfect modulo bounded languages if it is closed under Boolean operations relative to every bounded language, and if the emptiness problem is decidable relative to every bounded language. We consider finding perfect classes of languages modulo bounded languages. We show that the class of languages accepted by multi-head pushdown automata are perfect modulo bounded languages, and characterize the complexities of decision problems. We show that computations of several known models of systems, such as recursive multi-threaded programs, recursive counter machines, and communicating finite-state machines can be encoded as multi-head pushdown automata, giving uniform and optimal underapproximation algorithms modulo bounded languages. [Joint work with Pierre Ganty and Rupak Majumdar.]

Session Zugspitze (Mon, May 21, 10:15 – 11:30)

Synthesizing Software Verifiers from Proof Rules

Corneliu Popeea
TU Munich, Germany

Abstract: Automatically generated tools can significantly improve programmer productivity. For example, parsers and dataflow analyzers can be automatically generated from declarative specifications in the form of grammars, which tremendously simplifies the task of implementing a compiler. In this paper, we present a method for the automatic synthesis of software verification tools. Our synthesis procedure takes as input a description of the employed proof rule, e.g., program safety checking via inductive invariants, and produces a tool that automatically discovers the auxiliary assertions required by the proof rule, e.g., inductive loop invariants and procedure summaries. We show how our method synthesizes automatic safety and liveness verifiers for programs with procedures, multi-threaded programs, and functional programs. Our experimental comparison of the resulting verifiers with existing state-of-the-art verification tools confirms the practicality of the approach.
This is joint work with Sergey Grebenshchikov, Nuno Lopes and Andrey Rybalchenko at the Technical University Munich.

Refinement Framework for Monadic Programs in Isabelle/HOL

Peter Lammich
TU Munich, Germany

Abstract: When verifying algorithms, there is a trade-off between an abstract version of the algorithm, that has a nice correctness proof but is not (or not efficiently) executable, and an implementation version, that is optimized for efficiency, but has a correctness proof that tends to get obfuscated by implementation details.
Refinement is a top-down solution to this problem, first defining and proving correct the abstract algorithm, and then refining it (in possibly many steps) to the concrete version, showing that each refinement step preserves correctness.
In this talk, we will present a refinement framework in Isabelle/HOL. It is based on a refinement calculus for monadic expressions and provides tools to automate canonical tasks such as verification condition generation. The Isabelle/HOL code generator can be used to generate efficient code. It has been successfully used to verify various algorithms, among them Dijkstra's shortest path and Hopcroft's minimization algorithm.

A Formalisation of Finite Automata in Isabelle/HOL

Thomas Tuerk
TU Munich, Germany

Abstract: As part of the CAVA project (http://cava.in.tum.de), I'm developing a library for finite automata in Isabelle / HOL. The goal of the CAVA project is to verify automata based model checking algorithms and to generate verified reference implementations. For finite automata, this means that the library provides on the one hand high-level, abstract definitions that are very close to classic automata notions. On the other hand, there are low-level constructs that allow Isabelle to generate efficient code in several programming languages including OCaml, Haskell, Skala and Standard ML. Bridging the gap between the low and high-level constructs is the main challenge for developing this library.
As a case study an automaton based decision procedure for Presburger arithmetic was adapted to use this library. This adaption turned out to be straightforward and resulted in performance increasements. Another success-story is the implementation of Hopcroft's minimisation algorithm. The generated code can stand the comparison with efficient manual implementations.

Session Black Forrest (Mon, May 21, 12:50 – 14:30)

Engage: A Deployment Management System

Shahram Esmaeilsabzali
MPI-SWS, Germany

Abstract: Many modern applications are built by combining independently developed packages and services that are distributed over many machines with complex inter-dependencies. The assembly, installation, and management of such applications is hard, and usually performed either manually or by writing customized scripts. We present Engage, a system for configuring, installing, and managing complex application stacks. Engage consists of three components: a domain-specific model to describe component metadata and inter-component dependencies; a constraint-based algorithm that takes a partial installation specification and computes a full installation plan; and a runtime system that co-ordinates the deployment of the application across multiple machines and manages the deployed system. By explicitly modeling configuration metadata and inter-component dependencies, Engage enables static checking of application configurations and automated, constraint-driven, generation of installation plans across multiple machines. This reduces the tedious manual process of application configuration, installation, and management.
We have implemented Engage and we have used it to successfully host a number of applications. We describe our experiences in using Engage to manage a generic platform that hosts Django applications in the cloud or on premises. This is joint work with Jeffrey Fischer and Rupak Majumdar.

Nested Interpolants in SMTinterpol

Jochen Hoenicke
Uni Freiburg, Germany

Abstract: A recent technique in software model-checking is to generate interpolants from an infeasible program trace. An inductive sequence of interpolants can be seen as a Floyd/Hoare-style assertions of a correctness proof. However, for programs with procedures, the intermediate assertions should only contain symbols from the current procedure. Nested sequences of interpolants, which are based on the theory of nested word languages of Alur, Chaudhuri, and Madhusudan, fulfill this requirement. We describe an algorithm to compute nested sequences of interpolants from a single proof of unsatisfiability. This algorithm will be implemented in the upcoming version of SMTInterpol.

The Theory of Arrays with Set and Copy

Florian Merz
KIT, Germany

Abstract: The theory of arrays is widely used in program analysis, software verification, bounded model checking, symbolic execution, etc, e.g., to model memory. Nonetheless, the basic theory as introduced by McCarthy has its shortcomings for the analysis of real-world programs, since it only supports array updates at single locations. In programs, memory is often modified in larger chunks using functions such as memset or memcpy, which modify a user-specified range of locations whose size might not be known statically in advance. In this talk we will present an extension to the theory of arrays with set and copy operations which make it possible to reason about such functions. We will also discuss how we implemented different decision procedures for this extended theory (eager; using different forms of quantification; instantiation-based) in our bounded model checker LLBMC.

Simplification of Verification by Program Transformation

Alexander Schremmer
Uni Paderborn, Germany

Abstract: Automated software verification imposes a challenge on a code consumer: the verification of complex properties is time and resource consuming. Yet, the code consumer often does not have the resources to perform the verification. We therefore present a concept that shifts the verification burden to the code producer. By program transformation, we manage to narrow down the software verification effort on the consumer side by several magnitudes. Model checking techniques are used to ensure that the transformed program can be verified by a quick, single-pass data flow analysis. As a result, the code consumer can more easily verify the properties without needing to trust the code producer.
This is joint work with Daniel Wonisch and Heike Wehrheim.

Session Schneeberg (Tue, May 22, 10:00 – 11:15)

Synthesizing Robust Systems

Bettina Könighofer
TU Graz, Austria

Abstract: Specifications for reactive systems often consist of environment assumptions and system guarantees. An implementation should not only be correct, but also robust in the sense that it behaves reasonably even when assumptions (temporarily) are violated. A finite number of safety assumption violations is guaranteed to induce only a finite number of safety guarantee violations. In our presentation we show how specifications can be turned into a two-pair Streett game, and how a winning strategy corresponding to a correct and robust implementation can be computed. We implemented this approach as an extension of the requirements analysis and synthesis tool RATSY, which is able to synthesize robust systems from GR(1) specifications. Finally, we provide some experimental results.

Synthesis from Incompatible Specifications

Arjun Radhakrishna
IST, Austria

Abstract: Systems are often specified using multiple requirements on their behavior. In practice, these requirements can be contradictory. The classical approach to specification, verification, and synthesis demands more detailed specifications that resolve any contradictions in the requirements. These detailed specifications are usually large, cumbersome, and hard to maintain or modify. In contrast, quantitative frameworks allow the formalization of the intuitive idea that what is desired is an implementation that comes “closest” to satisfying the mutually incompatible requirements, according to a measure of fit that can be defined by the requirements engineer. One flexible framework for quantifying how “well” an implementation satisfies a specification is offered by simulation distances that are parameterized by an error model. We introduce this framework, study its properties, and provide an algorithmic solution for the following quantitative synthesis question: given two (or more) behavioral requirements specified by possibly incompatible finite-state machines, and an error model, find the finite-state implementation that minimizes the maximal simulation distance to the given requirements. Furthermore, we generalize the framework to handle infinite alphabet (for example, real-valued domains) and demonstrate how quantitative specifications based on simulation distances might lead to smaller and easier to modify specifications.

Parameterized Synthesis

Ayrat Khalimov
TU Graz, Austria

Abstract: Parameterized synthesis was recently proposed as a way to circumvent the bad scalability of current synthesis tools. The proposed method builds on bounded distributed synthesis with some additional constraints, and thus ultimately on the use of SMT solvers. We will present alternative encodings of the problem. By switching to a bitvector logic, using a bottom-up approach, and reducing the freedom of the SMT solver to pick solutions, we show that we can achieve a speedup of one to two orders of magnitude on a (small) set of examples.

Session Danube (Tue, May 22, 12:25 – 14:30)

Scalable Certificate Extraction for QBF

Mathias Preiner
JKU Linz, Austria

Abstract: A certificate of (un)satisfiability for a quantified Boolean formula (QBF) represents concrete assignments to the variables which act as witnesses for its truth value. Certificates are highly requested for practical applications of QBF like formal verification and model checking. We present an integrated set of tools realizing resolution-based certificate extraction for QBFs in prenex conjunctive normal form. Starting from resolution proofs produced by the solver DepQBF, we describe the workflow consisting of proof checking, certificate extraction, and certificate checking. We implemented the steps of that workflow in stand-alone tools and carried out comprehensive experiments. Our results demonstrate the practical applicability of resolution-based certificate extraction.

CEGAR for an Explicit-Value Analysis

Stefan Löwe
Uni Passau, Germany

Abstract: Abstraction is a necessity for software-verification tools in order to successfully prove properties of real-life programs. However, the question which level of abstraction is just right for any given program cannot be answered in general. Counter-example guided abstraction refinement (CEGAR) describes an on-the-fly approach, where an initially coarse abstraction is gradually refined to be just precise enough to reason about a property that is to be verified. While this technique is incorporated in several prominent tools like BLAST, SATABS, SLAM, it is only applied to domains like predicate abstraction and symbolic execution, but to date not to explicit-value analysis. We show how a simple explicit-value analysis benefits from making use of the CEGAR approach, and further contributions involve the adaptation of lazy abstraction and an ”explicit” interpolation technique, both approaches which were yet to be combined with explicit-value analysis.

Efficient Techniques for Manipulation of Non-deterministic Tree Automata

Ondrej Lengal
TU Brno, Czech Republic

Abstract: The presentation will consider several issues related to efficient use of tree automata in formal verification. First, a new efficient algorithm for inclusion checking on non-deterministic tree automata will be presented. The algorithm traverses the automaton downward, utilizing antichains and simulations to optimize its run. Results of a set of experiments show that the proposed approach often very significantly outperforms the so far common upward inclusion checking. Next, VATA, a new open source library for manipulation with non-deterministic tree automata, will be presented. The library supports two complementary encodings of tree automata: the explicit one intended for automata with small alphabets, and the semi-symbolic one suitable for automata with very large alphabets. VATA supports standard manipulation with automata and also several non-standard operations, such as checking language inclusion of a pair of tree automata or computation of the upward and downward simulation relation on the states of a tree automaton.

Symbolic Loop Bounds for WCET Analysis

Jakob Zwirchmayr
TU Vienna, Austria

Abstract: The spread of safety critical real time systems results in an increased necessity for worst-case execution time (WCET) analysis of these systems: finding the time limit within which the software system responds in all possible scenarios. Computing the WCET for programs with loops or recursion is, in general, undecidable. We present an automatic method for computing tight upper bounds on the iteration number of special classes of program loops. These upper bounds are further used in the WCET analysis of programs. The technique deploys pattern-based recurrence solving in conjunction with program flow refinement using SMT reasoning. To do so, we refine program flows using SMT reasoning and rewrite certain multi-path loops into single-path ones, possibly over-approximating the loop-bound. Single-path loops are further translated into a set of recurrence relations over program variables. Recurrences are then solved using a pattern based algorithm that computes closed form solutions for a restrcited class of recurrence equations. Finally, loop iteration bounds are derived from those solutions. We only compute closed forms for a restricted class of loops, however, in practice, these recurrences describe the behavior of a large set of program loops that are relevant to WCET analysis. Our technique is implemented in the r-TuBound tool that was successfully tried out on a number of WCET benchmarks and compared against the original TuBound tool.

Counter Attack against Byzantine Generals

Josef Widder
TU Vienna, Austria

Abstract: We consider automatic verification of threshold-based fault-tolerant distributed algorithms. These algorithms are parameterized in the size of the system, and the assumed maximal number of faulty processes. In order to automatically verify such distributed algorithms, we propose a novel technique based on counter abstraction, interval abstraction, and model checking.
We explain our method using an asynchronous broadcasting algorithm that tolerates Byzantine process faults. Although the code of the original distributed algorithm is very simple, the uncertainty imposed by the asynchrony of the processes, and the faults poses a serious difficulty for model checking algorithms.
This is joint work with Annu John, Igor Konnov, Ulrich Schmid, and Helmut Veith.

Session Adamello (Tue, May 22, 15:00 – 16:40)

Strong and Weak Controllability of Temporal Problems with Uncertainty Using SMT

Andrea Micheli
FBK, Italy

Abstract: Temporal Problems with Uncertainty are a well established formalism to model time constraints of a system interacting with an uncertain environment. This formalism is widely used in scheduling and temporal planning. Several works have addressed the definition and the solution of controllability problems, and three degrees of controllability have been proposed: weak, strong, and dynamic. However, the literature is fragmented, and specialized algorithms have been proposed for different problem classes.
We propose a comprehensive framework to handle a wide class of Temporal Problems with Uncertainty, together with general and effective solution methods for the cases of strong and weak controllability. Our approach relies on the Satisfiability Modulo Theory (SMT) framework. We address both the decision and the strategy extraction problems: extracting a strategy means finding a function from assignments to uncontrollable time points to assignments to controllable time points that fulfill all the temporal constraints.
We show a detailed experimental evaluation of the approach, over a large set of benchmarks, using various SMT solvers. We analyze the merits of the various algorithms, demonstrating the feasibility of the approach.

Library-Based Fault Injection for Formal Safety Assessment

Cristian Mattarei
FBK, Italy

Abstract: The design of safety critical systems needs a strong validation in terms of fault tolerance. This means that the behavior of such systems must be correct even under stress conditions. Such a guarantee is pivotal in areas like automotive, avionics or railways, where human lives can be harmed.
Currently, there are some methodologies that allow the verification of a safety critical system represented using a formal model. One possibility, called Fault Injection, is to induce the system in a wrong state in order to verify if the behavior in case of faults sill fulfill all the requirements.
Our approach implements Fault Injection by separately considering the behavior in the presence (faulty behavior) or absence (nominal behavior) of faults. This is done by providing libraries that describe the possible effects and dynamics that a faulty component can have.
Usually, the verification of a safety critical system is done separately on the nominal model and on the model extended with the faulty behaviors. However, this methodology does not consider the common part between the two models; we propose to exploit the results obtained in the first phase to simplify the verification of the model extended with faults.

FunFrog: Function Summarization-Based Model Checking

Grigory Fedyukovich
Uni Lugano, Switzerland

Abstract: The talk presents a tool, FunFrog, which implements a function summarization approach for software bounded model checking. It uses interpolation-based function summaries as over-approximation of function calls. In every successful verification run, FunFrog generates function summaries of the analyzed program functions. The summaries are reused to reduce the complexity of the successive verification. The interpolants symbolically represent the relevant information from the already verified portions of the code. They tend to be precise and they are easy to compute. To prevent reporting spurious errors due to too coarse over-approximation, the tool incorporates a counterexample-guided refinement loop. FunFrog is implemented within the CProver framework and employs OpenSMT for symbolic reasoning and intepolation.

Accelerating Interpolants

Hossein Hojjat
EPFL, Switzerland

Abstract: We present Counterexample-Guided Accelerated Abstraction Refinement (CEGAAR), a new algorithm for verifying infinite-state transition systems. CEGAAR combines interpolation-based predicate discovery in counterexample-guided predicate abstraction with acceleration technique for computing the transitive closure of loops. CEGAAR applies acceleration to dynamically discovered looping patterns in the unfolding of the transition system, and combines overapproximation with underapproximation. It constructs inductive invariants that rule out an infinite family of spurious counterexamples, alleviating the problem of divergence in predicate abstraction without losing its adaptive nature. We give a comparison on dynamic acceleration and static acceleration on a set of benchmarks. [Joint work with Viktor Kuncak, Radu Iosif, Philipp Ruemmer, and Filip Konecny.]