ISSN 2186-7437

NII Shonan Meeting Report

No. 2015-5

Static Analysis Meets Runtime Verification

Cyrille Artho
Einar Broch Johnsen
Martin Leucker
Keiko Nakata

March 16–19, 2015

National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-Ku, Tokyo, Japan

Static Analysis Meets Runtime Verification

Organizers:
Cyrille Artho, AIST
Einar Broch Johnsen, University of Oslo
Martin Leucker, University of Lübeck
Keiko Nakata, FireEye Dresden

March 16–19, 2015

General background Our life is increasingly dependent on the correct functioning of software, and the role of software in complex products is expanding fast. The increase in software complexity is paralleled by the amplified risks connected to its failure. For instance, operating systems that used to trust local applications must now defend themselves against malicious applications. Safety-critical software applications with high levels of risk (such as banking and transportation) must be trustworthy, and they need reliable guarantees that they behave as intended. However, in most industrial environments, many constraints (time-to-market, cost, lack of skills, etc.) make unrealistic manual verification, validation and certification. We are in need of technologies that can be deployed within industrial constraints and that offer high levels of guarantees for complex systems.

Static analysis Static analysis examines software applications without actually executing them. It can perform sophisticated analysis, and does not require test cases or the complete code, making it very useful in industrial applications. However, due to its exhaustive nature, static analysis is difficult to scale for complex data structures without sacrificing the precision. Static analysis has been recognized as a fundamental tool for program verification, bug detection, compiler optimization, program understanding, and software maintenance. On the one hand, new theories are continuously proposed to subsume new computing models of modern software, such as distributed computing and resource-aware computing. On the other hand, techniques to design and implement static analysis tools have improved. In the last decade, we have witnessed the emergence of a wide range of static analysis tools, which are beyond the advanced prototype level.

Runtime verification Runtime verification is a computing analysis paradigm based on observing a system at runtime to check its expected behavior. Runtime verification has emerged as a practical application of formal verification and performs conventional testing in a less ad-hoc way by building monitors from formal specifications. Runtime verification scales well even when complex data structures are used. It is particularly useful, when exhaustive design time

verification is impractical or even impossible, due to, for example, the modern systems inherent complexity or the lack of availability of comprehensive models. However, the quality of runtime analysis depends on test scenarios, which makes it less robust compared to static analysis. Runtime verification-complements design time static analysis with lightweight verification techniques that check violation of intended properties online. Specifically, the field has been addressing technical challenges of generating efficient monitors from high level specification and of formally specifying recovery actions at the specification level.

Goals of the meeting The goal of this meeting is to bring together the two communities, to combine the robustness of static analysis and the flexibility of runtime verification. Recent years have seen, on one hand, the use of static analysis in the context of runtime verification to reduce the size of runtime models by pruning certain scenarios that are statically analyzed, on the other hand, the use of runtime verification in the context of static analysis to ease verification burden by deferring certain properties to be verified at runtime. These efforts involve both theoretical challenges of unifying different formalisms and engineering challenges of enabling different tools to communicate with each other. Another direction of research is to develop technologies for verifying software under open-world assumptions, where software is made from heterogeneous, possibly third-party, components, whose behavior and interactions cannot be fully controlled or predicted. Exhaustive static analysis is not possible for open-world software, but runtime verification can come to the rescue by, for instance, verifying, at runtime, assumptions about the open world that are made during the static analysis. Discussions at the meeting will help identify research needs for combining static analysis and runtime verification. We discuss challenges for the two fields, such as verification of concurrent systems. Participants will be able to discuss technical approaches that have emerged in various related research areas and assess their applicability to the challenges we face.

Meeting Schedule

- March 15th, Sunday Evening
  - Arrival
- March 16th, Monday morning
  - Opening by the local Shonan team
  - Opening by the organizers
  - Short self-introduction by the participants
  - Brief overview on static analysis: Static Program Analysis 101 – The Rough Guide (Ralf Huuck)
  - Short presentations
    - ZooBerry System: automatic generation of high-performance static analyzers and their verified validators (Kwangkeun Yi)
    - Goanna C/C++ Analysis – Quality, Security, Compliance (Ralf Huuck)
    - Formal Methods for Cyber Security at FireEye (Keiko Nakata)
- March 16th, Monday afternoon
  - Brief overview on runtime verification (Martin Leucker)
  - Short presentations
    - Runtime Enforcement of Regular Timed Properties by Suppressing and Delaying Events (Yliès Falcone)
    - Predictive Runtime Verification of Cyber-Physical Systems (Wei Dong)
    - Adaptive runtime verification (Lenore Zuck)
  - Discussion/votes on topics
  - Short presentations
    - Model-based testing (Cyrille Artho)
    - Runtime verification of distributed systems (Borzoo Bonakdarpour)
    - Guided Random Testing: When Program Analysis Meets Automatic Test Generation (Lei Ma)
  - Discussion on plenary/group discussion topics
- March 17th, Tuesday morning
  - Short presentations
    - RV for field based pervasive computing applications (Ferruccio Damiani)
    - Proof-Carrying Apps (Ina Schaefer)
    - Abstract operation contracts (Reiner Hähnle)

  - Plenary discussion: Distributed systems
  - Group discussions: dynamic updating, continuous quality assurance, symbolic execution

- March 17th, Tuesday afternoon
  - Short presentations
    * Combining statics analysis and dynamic analysis for malware and vulnerability detection (Liu Yang)
    * Test-based Model Construction (Bernhard Steffen)
    * Model repair for probabilistic controllers (Erika Abraham)
    * Checking correctness of refactorings with runtime assertions (Volker Stolz)
  - Plenary discussion: Security + privacy
  - Short presentations
    * Theorem proving for cyber-physical systems (Sarah Loos)
    * A Framework for Static and Runtime Verification (Gerardo Schneider)
    * A Tool for Static and Runtime Verification (Wolfgang Ahrendt)
    * Proving that Android's, Java's and Python's sorting algorithm is broken and showing how to fix it (Richard Bubel)
  - Group discussions: cyber-physical systems, concurrency
  - Group discussions: predictive RV, education, scalability

- March 18th, Wednesday morning
  - Short presentations
    * Synquid: Synthesizing Programs from Liquid Types (Nadia Polikarpova)
    * Large scale testing using computer clusters (Toshiaki Aoki)
    * Quantitative runtime verification (Marta Kwiatkowska)
  - Group discussions: run-time synthesis, model learning, killer applications, compositional/incremental techniques
  - Plenary discussion: combining static and dynamic tools

- March 18th, Wednesday afternoon
  - Excursion

- March 19th, Thursday morning
  - Short presentation
    * Stream Runtime Verification (Cesar Sanchez)
  - Summaries of groups discussions
  - Closing

Overview of Talks and Discussions

Model-based testing

Cyrille Artho, AIST

In model-based testing, test cases are derived from an abstract model rather than implemented directly as code. The model typically gives rise to many possible combinations of tests, and is usually derived from the specification and specified by a human. Approaches to automate model creation exist, but they are limited to simple systems, and their output is difficult to validate. This presentation outlines some challenges and opportunities in model-based testing, and how partially automated model creation may proceed in the future.

A Framework for Static and Runtime Verification/ A Tool for Static and Runtime Verification

Gerardo Schneider and Wolfgang Ahrendt (Chalmers TU)

We present the approach and tool StaRVOOrS (Static and Runtime Verification of Object-Oriented Software), which combines static and runtime verification of Java programs. It uses partial results extracted from static verification to optimize the runtime monitoring process. StaRVOOrs combines the deductive theorem prover KeY and the runtime verification tool LARVA, and uses properties written using the ppDATE specification language which combines the control-flow property language DATE used in the runtime verification tool LARVA with Hoare triples assigned to states. The formalism enables KeY to attempt verification of the Hoare triples in the specification in fully automated mode, which often results in unfinished proofs. Through an analysis of such partial proofs, path conditions for the (statically) inconclusive executions are generated, and then used to refine the Hoare triples for runtime verification. Finally, the refined Hoare triples are translated into the DATE formalism, coding them as a combination of replicated automata and operationalised pre/post-conditions. LARVA is used to instrument the resulting DATE to monitor the code. StarRVOOrS effectiveness is demonstrated by applying it to the verification scenario Mondex, an electronic purse application.

Large scale testing using computer clusters

Toshiaki Aoki, JAIST

The safety and reliability of automotive systems are becoming a big concern in our daily life. Recently, a functional safety standard which specializes in automotive systems has been proposed by the ISO. In addition, electrical throttle systems have been inspected by NHTSA and NASA due to the unintended acceleration problems of Toyota's cars. In light of such recent circumstances, we are working on practical applications of formal methods and testing to ensure the high quality of automotive operating systems. In our approach, we make a golden model with formal methods and formal verification, then test the implementation of an operating system by automatically generating test cases from

the golden model. Our way to explain the the operating system is correct is simple. That is, the golden model is correct thanks to the formal methods and formal verification, and the implementation is equal to the golden model. Hence, the implementation is correct. One may say that it is impossible to prove the equality by testing. That's true. However, it is still worthwhile to do the testing as much as possible to increase the confidence in the equality. In this approach, as a huge amount of testing has to be carried out to do that, we adopt computer cluster consisting of many PCs to conduct the testing. In this talk, we briefly show how we verify the operating system by combining the formal methods and a large scale testing using the commuter cluster.

Stream Runtime Verification

Cesar Sanchez, IMDEA Software Institute

In this short talk I will motivate stream runtime verification. Naturally, most formalisms for runtime verification are adaptations of specification languages used in static verification. These formalisms are usually logics whose semantics map a given system (or program) and a formula to a truth value, capturing whether the system satisfies the spec. In the case of trace languages, like LTL, semantics of formulas are set of traces (i.e. those traces for which the formula "evaluates to true").

Stream runtime verification is based on the observation that monitoring algorithms are agnostic to the data values being processed, which in the case of logics as specification formalisms are Boolean values. These algorithms can be easily generalized to manipulate values from richer domains, which enables to express the collection of statistics from traces (average number of re-transmissions, average waiting time for a response, etc) and to capture precisely response properties (every request has a response).

Abstract operation contracts

Reiner Hähnle, TU Darmstadt

Proof reuse in formal software verification is crucial in presence of constant evolutionary changes to the verification target. Contract-based verification makes it possible to verify large programs, because each method in a program can be verified against its contract separately. A small change to some contract, however, invalidates all proofs that rely on it, which makes reuse difficult. We introduce fully abstract contracts and class invariants which permit to completely decouple reasoning about programs from the applicability check of contracts. We implemented tool support for abstract contracts as part of the KeY verification system and empirically show the considerable reuse potential of our approach.

Runtime Enforcement of Regular Timed Properties by Suppressing and Delaying Events

Yliès Falcone, U of Grenoble 1

Runtime enforcement is a verification/validation technique aiming at cor-

recting possibly incorrect executions of a system of interest. In this talk, we consider enforcement monitoring for systems where the timing between events matter. Executions are then modeled as sequences of events composed of actions with dates (or timed words) and we consider runtime enforcement for timed specifications modeled as timed automata, in the general case of regular timed properties. The proposed enforcement mechanisms have the power of both delaying events to match timing constraints, and suppressing events when no delaying is appropriate, thus allowing the enforcement mechanisms and systems to continue executing. To ease their design and their correctness-proof, enforcement mechanisms are described at several levels: enforcement functions that specify the input-output behavior in terms of transformations of timed words, constraints that should be satisfied by such functions, enforcement monitors that describe the operational behavior of enforcement functions, and enforcement algorithms that describe the implementation of enforcement monitors. The feasibility of enforcement monitoring for timed properties is validated by prototyping the synthesis of enforcement monitors from timed automata. (Joint work with Thierry Jéron, Hervé Marcha and Srinivas Pinisett)

ZooBerry System: automatic generation of high-performance static analyzers and their verified validators

Kwangkeun Yi, Seoul National U

Having developed a collection of techniques for sound-precise-global-yet-scalable static analysis (general sparse analysis framework + selective X-sensitive framework), we thought it was high time to make the techniques automatically available to non-specialists. Also, our experience of successfully attaching a verified validator, the system will also generate verified validator with which the user can check the correctness of analysis results. The presentation covers ZooBerry's overall architecture and first results about its performance.

Static Program Analysis 101 – The Rough Guide

Ralf Huuck, NICTA

We present an overview of common static analysis techniques and jargon to serve as an introduction to people new to the field of static program analysis. In particular, we present core concepts of decidability and (sound) approximations as well as the notions of true/false positives/negatives. Moreover, we give an overview to classical techniques such as data flow analysis and abstract interpretation as well as recent advances using model checking, SMT solving and symbolic execution for program analysis. Additionally, we present an number of open topics and challenges for future work.

Goanna C/C++ Analysis – Quality, Security, Compliance

Ralf Huuck, NICTA

In this talk we present our static analyzer for C/C++ programs called Goanna. Goanna is based on model checking, abstract interpretation and SMT

solving and aims at detecting deep quality bugs and security vulnerabilities. We briefly touch on the evolution of the tool from an academic prototype to a commercial product now in use and licensed to many of the top 500 software companies.

Proving that Android's, Java's and Python's sorting algorithm is broken and showing how to fix it

Richard Bubel, TU Darmstadt

The talk was about our experiences when trying to mechanically verifying the correctness of TimSort, which is the main sorting algorithm provided by the Java standard library. During our verification attempt we discovered a bug which causes the implementation to crash. We then derived a bug-free version that does not compromise the performance and formally specified the new version and mechanically verify the absence of this bug using the verification system KeY. In the talk we also discussed the aftermath of the bug discovery.

Predictive Runtime Verification of Cyber-Physical Systems

Wei Dong, NUDT

The presentation describes the definition of predictive runtime verification, and the frameworks of how to obtain predictive words at runtime based on the model and code analysis. Some real cases of applying runtime verification in safety critical cyber-physical systems are also presented.

Model repair for probabilistic controllers

Erika Abraham, RWTH Aachen U

Whereas static and runtime analysis are frequently applied to discrete systems, there is less research done for their employment for probabilistic systems. We discuss a certain application from robotics. The task is to design a controller for a robot acting in an environment with probabilistic behaviour. After the creation of a controller, we show how the controller model can be repaired in an elegant greedy way in order to avoid unsafe states.

Theorem proving for cyber-physical systems

Sarah Loos, CMU

Formal methods and theorem proving have been used successfully in many discrete applications, such as chip design and verification, but computation is not confined to the discrete world. Increasingly, we depend on discrete software to control safety-critical components of continuous physical systems. It is vital that these hybrid (discrete and continuous) systems behave as designed, or they will cause more damage than they intend to fix. In this talk, I will present several challenges associated with verifying hybrid systems and how we use differential

dynamic logics to ensure safety for hybrid systems under a continuously infinite range of circumstances and unbounded time horizon.

I will discuss the first formal proof of safety for a distributed car control system and the first formal verification of distributed, flyable, collision avoidance protocols for aircraft. In both cases, our verification holds for an unbounded number of cars or aircraft, so the systems are protected against unexpected emergent behavior, which simulation and testing may miss.

Providing the first formal proofs of these systems required the development of several seemingly unrelated techniques. We have since identified a unifying challenge for each case study: it is difficult to compare hybrid systems, even when their behaviors are very similar. I will discuss the development of a logic with first-class support for refinement relations on hybrid systems, called differential refinement logic, which aims to address this core challenge for hybrid systems verification within a formal framework.

Runtime verification of distributed systems

Borzoo Bonakdarpour, McMaster U

Distributed systems are inherently complex and monitoring them is equally challenging. In this talk, I will present our recent work on runtime LTL verification of asynchronous distributed applications. I will also discuss our ongoing work on distributed settings where local monitors may crash.

Proof-Carrying Apps

Ina Schaefer, TU Braunschweig

In order to allow for apps to be loaded at runtime while preserving safety requirements, apps are equipped with proofs for these properties that are checked before the apps are started. If the proof cannot be verified, the app is monitored at runtime to ensure its safety requirements.

RV for field based pervasive computing applications

Ferruccio Damiani, U of Torino

I will briefly illustrate the field calculus (a recently developed computational model that provides a formal mathematical grounding for the many languages for aggregate programming) and outline the issues in devising Static Analyses and RV for field based pervasive computing applications.

Guided Random Testing: When Program Analysis Meets Automatic Test Generation

Lei Ma, U of Tokyo

A brief introduction to our automatic test generation tool GRT, which just won the first place of the Search-based Software Testing competition this year.

Synquid: Synthesizing Programs from Liquid Types

Nadia Polikarpova, MIT

Liquid Types framework has been used to verify a wide range of properties of functional programs; I will discuss how this framework could be extended to synthesize programs that are correct by construction.

Formal Methods for Cyber Security at FireEye

Keiko Nakata, FireEye Dresden

FireEye is a US-based public network security company, founded in 2004. It develops an innovative security platform, which detects malware through its malicious behaviour and uses virtualisation for behaviour inspection.

The objective of FireEye's formal methods team in Dresden is to mathematically prove security properties of FireEye's kernel products. The software is written in C++, executes concurrently, uses pointer arithmetic, embeds assembly codes, implementing system-level infrastructure such as a micro-kernel. Currently we work with the Coq proof assistant, applying programming language technologies. We are also looking at off-the-shelf program analysers and verifiers.

Combining Tools

Plenary Discussion

The plenary discussion on combine tools came to the understanding that tools in the runtime verification as well as in the static analysis space are not yet mature enough to be easily combinable. More work is needed to understand how to combine tools, and what are the right properties and the right specification language that would make a combination useful and practical. That work will be required both on a theoretical level as well as on a tool development level to mutually understand the opportunities and challenges. Finally, a number of domains were discussed that could serve as a paradigm. One domain that stood out was embedded debugging, where both static analysis on the source code and runtime verification on a target trace/logging could be used to create a combined solution that is more powerful than the sum of its parts.

Cyber-physical systems

Group Discussion

We discussed the following two problems.

- Sampling in in hybrid systems:

  The design of hybrid systems includes capturing the dynamics of the environment, typically expressed as differential equations that model the evolution over time of the input variables that the system samples. The control plant of hybrid systems consists of a finite control describing how the system actuates at each state, and transitions based on sampled values

and local state, which is later expressed as a software program. The correctness is expressed as a (temporal) formula relating the values of local variables and environment variables. A prototypical example is a thermostat. One would like to determine statically the correctness of a design, or monitor the system against the spec.

- Embedded debugging:

  This problem shows a more clear challenge in the interplay between classical program analysis and runtime verification. The idea is that when an embedded software is tested in the deployed platform, the engineers cannot use an interactive debugger to debug errors. An alternative is to produce logs to later analyze them, but the modifications in the program must be minimal (to prevent changes in the timings), the inspection of traces is tedious, and the logging maybe unfeasible because large storage are typically not available. A better alternative is to offer the debugging engineer a language to express interesting error trace descriptions (an RV language) which must then be used to modify the code for which static analysis is crucial to provide efficiency.

Teaching undergrads

Group Discussion

For general undergraduate audience, having the students really get induction would be an improvement. For undergraduates, we should give them examples that are real and that they can actually solve (e.g. the zune bug). In TU Darmstadt, an elective course called "testing, debugging, and verification" is offered. The course puts testing at one end of a spectrum of verification approaches. It has been taught at several universities, with average class size of up to 80 students.

Runtime synthesis

Group Discussion

Like in verification, in synthesis there are purely static techniques (traditional understanding of synthesis) and purely dynamic techniques (constraint programming, where no program is synthesized, but constraints are solved at runtime for given input values). Can we combine the two and get a technique that is more flexible than static synthesis, but more efficient at runtime than constraint programming? The slogan is: synthesize statically what you can, solve the rest at runtime. One idea is to draw inspiration from partial evaluation. Can we develop a solver that can be easily specialized based on static knowledge, so that is can solve constraints at run time more efficiently?

List of Participants

- Wei Dong (NUDT)
- Ralf Huuck (NICTA)
- Yliès Falcone (U of Grenoble 1)
- Richard Bubel (TU of Darmstadt)
- Reiner Hähnle (TU of Darmstadt)
- Erika Abraham (RWTH Aachen U)
- Toshiaki Aoki (JAIST)
- Lei Ma (U of Tokyo)
- Volker Stolz (Bergen U College)
- Sarah Loos (CMU)
- Lenore Zuck (U of Illinois)
- Kwangkeun Yi (Seoul National U)
- Ina Schaefer (TU Braunschweig)
- Liu Yang (Nanyang Technological U)
- Cesar Sanchez (IMDEA Software Institute)
- Wolfgang Ahrendt (Chalmers TU)
- Gerardo Schneider (U of Gothenburg)
- Bernhard Steffen (U of Dortmund)
- Marta Kwiatkowska (U of Oxford)
- Nadia Polikarpova (MIT)
- Ferruccio Damiani (U of Torino)
- Borzoo Bonakdarpour (McMaster U)
- Cyrille Artho (AIST)
- Einar Broch Johnsen (University of Oslo)
- Martin Leucker (University of Lübeck)
- Keiko Nakata (FireEye Dresden)