Welcome to MVD 2016

Welcome to Ames, to Iowa State University, and to MVD 2016!

The following pages give logistical tips to help making your visit more enjoyable and rewarding.

If you need further help, ask one of us or one of the graduate students wearing a “MVD 2016 HELP STAFF” badge.

Samik Basu, Gianfranco Ciardo, Andrew Miner
Department of Computer Science
Iowa State University

Kristin Rozier
Department of Aerospace Engineering

Contents

Logistics 1
Campustown info 1
Ames and ISU info 2
Bus info 2
Parking info 2
ISU Campus Map 3
MVD 2016 Program 4
Abstracts (in presentation order) 6

Logistics

On Friday, MVD 2016 will take place in the Oak Room of the ISU Memorial Union. Lunch will be on your own; the Memorial Union has a large food court, or Campustown is just a short walk away (but be aware that the lunch break is only 70 minutes long).

On Saturday, MVD 2016 will take place in 2245 Coover Hall. A catered buffet lunch will be provided nearby, in 2200 Marston Hall.

Please note that the duration of each presentation, including questions and answers, is 60 minutes for Keynote presentations, and 20 minutes for Regular presentations.

Campustown info

The area immediately south of the ISU Campus (and of Lincoln Way) is known as “Campustown”. There, you will find many eateries, several of them moderately priced, with a variety of culinary options.

For more information, see http://www.amescampustown.com/business/
Ames and ISU info

ISU lies at the center of the City of Ames. A “CYtes” informational brochure about both ISU and Ames is included in your registration material. See also http://www.cytesofames.com/

Bus info

A “CyRide” bus schedule is included in your registration material. See also http://www.cyride.com/

Parking info

If you choose to drive your car and park on campus, we suggest that you park at the Memorial Union parking garage on Friday (since MVD sessions will be held at the Memorial Union that day). Students seeking travel reimbursement should inquire at the registration desk for a Friday parking voucher. For Saturday, you should park in Yellow/General parking spots, which is free on weekends. Be careful not to park in Red 24/7 spots, as you will be ticketed if you do so. The map below shows the parking areas at ISU:
ISU Campus Map

An interactive Iowa State map is available at [http://www.fpm.iastate.edu/maps/](http://www.fpm.iastate.edu/maps/)

Below is a map of the ISU Memorial Union:
MVD 2016 Program: Friday, Oct. 21, 2016
Oak Room ISU Memorial Union

8:00am – 8:30am  WELCOME AND BREAKFAST

8:30am – 9:30am  Keynote Speaker: Aaron Tomb
Top-to-bottom Verification of Cryptographic Algorithms

9:30am – 10:30am  Session 1
Abstraction Based Reachability Analysis for Finite Branching Stochastic Hybrid Systems
  Zhang Wenji, Prabhakar Pavithra, Natarajan Balasubramaniam
Simulation-Based Verification of Hybrid Systems involving Novel Dynamic Refinement
  Hao Ren, Ratnesh Kumar
Bounded safety analysis of parameterized linear hybrid systems
  Ratan Lal

10:30am – 10:50am  COFFEE BREAK

10:50am – 12:10pm  Session 2
An abstract concurrent memory model specification for block structured data
  David Bergvelt, Elsa Gunter, Liyi Li
VeriF-OPT: A Verification Framework for Optimizations and Program Transformations
  Elsa Gunter, William Mansky, Liyi Li, Susannah Mansky, David Bergvelt
Variable Reordering in Binary Decision Diagrams
  Chuan Jiang, Junaid Babar, Gianfranco Ciardo, Andrew S. Miner, Benjamin Smith
Program Verification by Coinduction
  Brandon Moore, Lucas Peña, Grigore Roșu

12:10pm – 1:20pm  LUNCH

1:20pm – 2:20pm  Session 3
An Implementation of Logic Programming Based on the Edinburgh Logical Framework
  Mary Southern, Gopalan Nadathur
Relational Reasoning in SMT
  Baoluo Meng, Andrew Reynolds, Cesare Tinelli
Reasoning About Specifications in Linear Logic
  Dan DaCosta, Gopalan Nadathur

2:20pm – 3:40pm  Session 4
Constraint Refined Taint Analysis for Software Vulnerability Detection
  Ruoyu Zhang, Cesare Tinelli
Mechanizing the Computationally Complete Symbolic Attacker in Coq
  Gergei Bana, Rohit Chadha, Ajay Kumar Eeralla
Towards Synthesis from Assume-Guarantee Contracts involving Infinite Theories: A Preliminary Report
  Andreas Katis, Andrew Gacek, Michael W. Whalen
Remote Attestation Protocol Synthesis and Verification
  Paul Kline

3:40pm – 4:00pm  COFFEE BREAK

4:00pm – 4:40pm  Session 5
From Design Contracts to Component Requirements Verification
  Jing (Janet) Liu, John D. Backes, Darren Cofer, Andrew Gacek
Contract-Based Verification of Complex Time-Dependent Behaviors in Avionic Systems
  Devesh Bhatt, Arunabh Chattopadhyay, Wenchao Li, David Oglesby, Sam Owre, Natarajan Shankar

4:40pm – 6:00pm  Session 6
Collective Program Analysis
  Ganesha Upadhyaya, Hridesh Rajan
Online Verification-Validation
  Jared Wright
Formal K to Isabelle Translation Framework based on K Semantics
  Liyi Li, Elsa Gunter
Dependently typed programming with lambda encodings in Cedille
  Ananda Guneratne, Chad Reynolds, Aaron Stump
8:00am – 8:30am BREAKFAST

8:30am – 9:30am Keynote Speaker: Sagar Chaki
Engineering High-Assurance Software for Distributed Adaptive Real-Time Systems

9:30am – 10:50am Session 7
Resequencing Decision Diagram Operations
Junaid Babar, Andrew S. Miner

A Semantics for Attestation Protocols using Session Types in Coq
Adam Petz

Extensions of SMTCoq to Bit-Vectors and Arrays
Burak Ekici, Chantal Keller, Alain Mebsout, Cesare Tinelli

A Mechanized Formalization of a Concurrent Core Calculus with Modular Reasoning
Swarn Priya, Mehdi Bagherzadeh, Hridesh Rajan

10:50am – 11:00am COFFEE BREAK

11:00am – 12:20pm Session 8
Verification and Test Generation for Plan Executions in Autonomy Applications
Jason Biatek, Sanjai Rayadurgam

Practical Verification? Insights from a Formal Verification Study of the Linux Kernel
Suresh C. Kothari, Payas Awadhutkar, Ahmed Tamrawi

Formal Methods for Certification: Why and How?
Lucas Wagner

Verification of a Molecular Watchdog Timer
Samuel J. Ellis

12:20pm – 1:30pm LUNCH
Top-to-bottom Verification of Cryptographic Algorithms

Aaron Tomb
Galois
Email: atomb@galois.com

Abstract. Implementations of cryptographic algorithms are especially attractive targets for verification. Several features make them fall into a sweet spot with respect to provable correctness. To start with, cryptography is a key piece of critical infrastructure, so correctness is especially important. On top of that, creation of a secure cryptographic construct requires deep and rare expertise, whereas efficient implementation requires an entirely separate set of skills, so the two tasks are often done by separate people, and both tasks could benefit from machine-assisted correctness checks. Finally, cryptographic code, while intricate and subtle, tends to consist of a relatively small amount of code that operates on a relatively small amount of data. This last feature can often make cryptographic code more amendable to verification than arbitrary software.

In this talk, I will describe some recent advancements (both at Galois and elsewhere) that have shown the top-to-bottom verification of production cryptographic code to be quite practical. At the top level, several tools exist to aid in machine-checked proofs of the security level guaranteed by common cryptographic constructions. In the middle of the stack, other tools now make it possible to prove the functional equivalence between a reference specification of an algorithm and an efficient implementation in software or hardware with a high degree of automation, as well as automating certain higher-level correctness properties. Similarly, synthesis of highly efficient, verified implementations from high-level specifications is possible, sometimes with even less effort. Finally, at the bottom of the stack, tools for automated analysis of non-functional properties, such as timing, exist and can work directly on machine code. Results on all of these levels can be tied together by considering a single, high-level description of an algorithm in a machine-readable form.

I will describe the capabilities of this collection of techniques and tools on a variety of real-world case studies, most involving verification of code that is actively used in production today.
Abstraction Based Reachability Analysis for
Finite Branching Stochastic Hybrid Systems

Zhang Wenji, Prabhakar Pavithra, Natarajan Balasubramaniam
Kansas State University

Abstract. We address the problem of computing the probability of reaching a desired set in a subclass of stochastic hybrid systems, wherein the stochasticity arises from the randomness of the initial distribution of continuous states, and the probabilistic transitions in the underlying finite state Markov chain. In particular, the continuous dynamics is deterministic for each mode and hence, there are finitely many probabilistic successors for a given state. We exploit this property and extend a partition based abstraction technique developed for finite state systems to the stochastic hybrid system setting. We prove the correctness of our algorithm by defining appropriate simulation relations that relate the stochastic hybrid system to the finite state "generalized" probabilistic transition system that we obtain as a result of abstraction. We show that the simulation relation defined provides upper and lower bounds on the probability of reachability. In particular, tighter bounds can be obtained by refining the partition. We apply our algorithm to automatically analyze a smart home application.
Simulation-Based Verification of Hybrid Systems involving Novel Dynamic Refinement

Hao Ren1 (ren@iastate.edu), Ratnesh Kumar1 (rkumar@iastate.edu)

1 Department of Electrical and Computer Engineering, Iowa State University, Ames, IA, 50014 USA

Abstract. Hybrid system verification tools based on reachability analysis suffer from fast growing error propagation introduced by overapproximation techniques used both at continuous and discrete stages. We present a simulation-based verification framework for systems modeled by hybrid automata. In this framework, a simulation is used as a reference for a class of system behaviors starting close to the start of the reference simulation. Any deviation from the reference trace is bounded at any finite time $t$. A feature of our work, that makes it distinct from prior simulation-based verification works, is the flexibility of dynamically performing repartitioning in the process. Repartitioning is applied during continuous evolution for better error propagation, where as it is applied during discrete transitions for correctly handling the effect of accumulated error that can introduce state resets and impact reachability computation. A unique feature of our approach is that it guarantees the boundedness of the error of over-approximation for convergent behaviors. We have developed a prototype verifier, HS$^3$V, implementing our algorithms and providing verification results from several benchmarks to show its effective performance.
Bounded safety analysis of parameterized linear hybrid systems

RATAN LAL
Kansas State University

Abstract. Hybrid system are a formalism for modeling systems with a combination of continuous and discrete behaviors. They are apt for modeling software systems that interact with physical entities, for instance, cruise control in automotive systems, and aircraft collision avoidance protocols. In this talk, we will consider the bounded safety analysis of a subclass of hybrid systems where the continuous dynamics is specified as a parameterized linear dynamical system. We use bounded error approximations of the executions to perform bounded safety analysis. The approximations are constructed by sampling both the time and the parameter domains, and are represented as piecewise linear functions, and thus, model checking is reduced to SMT solving. We have implemented our approach in the tool BEAVER, and our experimental results demonstrate the benefits of our approach.
An abstract concurrent memory model specification for block structured data

David Bergvelt¹, Elsa Gunter¹, and Liyi Li¹

Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana IL 61801, USA, bergvel1@illinois.edu, egunter@illinois.edu, liyili2@illinois.edu

Abstract. Formalized memory models play an important role in proving the correctness of compiler optimizations, but are often specified in a non-general way that is constrained to a particular implementation. A formalization strong enough to allow for a high degree of freedom in compiler optimizations but general enough to be applied across different implementations could solve this issue of constraint. Inspired by the work of Mansky, Garbuzov, and Zdancewic, we have created an abstract specification for concurrent memory models that supports memory operations on structured data. In addition, this specification is satisfied by models of both sequential and relaxed consistencies. The specification is defined axiomatically, by a set of axioms dictating what memory accesses a satisfying memory model “can do” at a given step in execution. We have given by transition systems two operational interpretations of the “can do” axioms, including sequential consistency and total store order.

Keywords: memory models, compiler correctness, axiomatic specification
VeriF-OPT: A Verification Framework for Optimizations and Program Transformations

Elsa Gunter¹, William Mansky²³, Liyi Li¹, Susannah Mansky¹, and David Bergvelt¹

¹ Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana IL, egunter@illinois.edu, liyili2@illinois.edu, sjohnsn2@illinois.edu, bergvel1@illinois.edu,

² Department of Computer and Information Science, University of Pennsylvania, Philadelphia PA, wmansky@seas.upenn.edu,

³ Department of Computer Science, Princeton University, Princeton, NJ

Abstract. The VeriF-OPT project is building a family of tools in the interactive theorem prover Isabelle for the specification, testing and verification of the middle end of compilers. A major component of this project includes a language Morpheus for specifying program transformations as conditional rewrites on generalized control flow graphs (GCFG) where the conditions are expressed as first order computation tree logic formulae interpreted over the GCFG. Morpheus has a formal semantics in Isabelle from which we can automatically extract to a functional programming language like OCaml or SML an executable semantics suitable for testing the behavior of transformations expressed in it.

Accompanying Morpheus is a framework for proving the correctness of transformations built in it. This framework includes semantics for a fragment of CIL, a fragment of LLVM and a fragment of JVM. This framework also provide theory support for proving a transformed program is simulated by the original program. Work is ongoing to expand the portions of these languages supported. One direction of this expansion has been to couple Isabelle with the K specification system of Grigore Rosu. The aim of this coupling is to allow users to specify programming languages in the K environment, test the specification thoroughly there against the language’s torture tests, and then after this testing automatically translate this semantics in Isabelle where it may be used as the basis of program transformation correctness proofs.

One aspect of program semantics that is of increasing importance in the presence of parallel computing is that of the memory model assumed in the semantics. In our framework, we have specific support for various operational memory models, including partial store order, total store order and sequential consistency. However, work is ongoing to develop and more general notion of memory model based on the actions a memory "can do" and "can’t do" that generalizes all of these and the happens-before model, while still being specific enough to be able to prove program transformations correct.
Keywords: compiler correctness, programming language specification, memory models, axiomatic specification
Variable Reordering in Binary Decision
Diagrams

Chuan Jiang, Junaid Babar, Gianfranco Ciardo, Andrew S. Miner, and Benjamin Smith

Department of Computer Science, Iowa State University
Ames, IA 50011, USA
{cjiang, junaid, ciardo, asminer, bensmith}@iastate.edu

Abstract. The size of a BDD heavily depends on its variable order. Significant efforts have been made to find good variable orders, statically or dynamically. In this paper, we concentrate on a related issue, transforming a BDD from one variable order to another, as needed when an application must cope with BDDs having different variable orders. Instead of rebuilding BDDs as existing work has done, we accomplish such transformation through a sequence of adjacent variable swaps. Since there are many ways to schedule these swaps, we propose and compare several heuristics to determine good schedules.

Keywords: binary decision diagram, variable reordering, optimization
Abstract. We present a program verification approach based on coinduction, which makes it feasible to verify programs using an operational semantics. No intermediates like axiomatic semantics or verification condition generators are needed. Specifications can be written using any state predicates. We implement our approach in Coq, giving a certifying language-independent verification framework. Our sound and (relatively) complete proof system is implemented as a single module imported unchanged into language-specific proofs. Automation is reached by instantiating a generic heuristic with language-specific tactics, but manual assistance is also smoothly allowed at points the automation cannot handle. We demonstrate the power of our approach by verifying algorithms as complicated as Schorr-Waite graph marking, and its versatility by instantiating it for object languages in several styles of semantics.
An Implementation of Logic Programming Based on the Edinburgh Logical Framework

Mary Southern and Gopalan Nadathur
University of Minnesota

The Edinburgh Logical Framework (LF) is a version of the λ-calculus that embodies a rich language of dependent types that are used to classify terms. LF has become a popular means for formalizing computational systems characterized via relations between objects and where these relations are described in a rule-based fashion. In encoding such systems λ-terms represent objects, and relationships between objects are captured via types that express dependencies between terms. The overall system is formalized by describing a signature for constructing the relevant types and terms, and in this context the derivability of a relation reduces to the question of inhabitation of a type relative to the given signature. The Twelf system uses a logic programming interpretation of LF signature specifications to answer such questions. Twelf also allows parts of a type whose inhabitation is to be determined to be left unspecified, representing such parts by variables that can be instantiated.

The focus of our work is to provide Twelf-style animation of LF-style specifications by leveraging previous work on the efficient execution of specifications based on a predicate logic. We make use of an information preserving translation between LF and the logic of higher order hereditary Harrop formulas that underlies the language λProlog. One part of the translation converts dependently typed λ-terms into simply typed ones by eliding dependency information. This information is recovered in the second part using predicates over the simply typed λ-terms to capture the dependency information.

Past work has shown this approach capable of faithfully capturing the behavior of Twelf when queries have no instantiatable variables in them. We will describe an extension of these results to the situation where types contain parts that must also be determined during computation. This extension requires two new issues to be addressed:

1. The computational process must determine values for the missing parts of types. In the translation based approach, they are determined via unification over the simply typed λ-terms obtained by ignoring constraints encoded in dependencies. We must ensure that the required constraints on variable instantiations are enforced in the translation, albeit in a different way.

2. Once a suitable instantiation has been determined for the translated form, it is necessary to extract an instantiation that “works” for the query in the LF form. We must articulate a systematic process for “inverting” the translation of dependently typed λ-terms to simply typed ones. Given the lossy nature of the translation, it is necessary to ensure both that we are dealing with terms that are actually invertible and that there is a meaningful way to pick one of the many possible target dependently typed λ-terms.
In this talk we will present our work that addresses these two issues. We also describe the use of our results in an implementation of Twelf that is based on the translation of LF programs and queries into ones in $\lambda$Prolog and the subsequent execution of queries using the Teyjus implementation of $\lambda$Prolog.
Relational Reasoning in SMT

Baoluo Meng, Andrew Reynolds, and Cesare Tinelli
Department of Computer Science
The University of Iowa, Iowa City, IA 52242

Abstract. Many computational problems require reasoning about relational structures such as system high-level design, architectural configurations of network systems, ontology, and verification of programs with linked data structure. Relational logic is an appealing choice for reasoning about such problems. Alloy is a declarative language for modeling structurally-rich systems based on relational logic with built-in transitive closure and cardinality. The analysis of Alloy models is performed automatically by the Alloy Analyzer - a SAT-based finite model finder. However, the Analyzer is limited in two major aspects: i) it can only disprove properties of input models, not prove them ii) it has limited support for numerical reasoning. Satisfiability Modulo Theories (SMT) solvers are efficient automated tools that can check the satisfiability of complex constraints over several domains, including arithmetic and the theory of finite sets. In this talk, I will present a novel approach for addressing the limitations of the Alloy Analyzer by leveraging the power of SMT solvers. The approach is based on extending the set subsolver of the SMT solver CVC4 to a subsolver for the theory of finite relations, and reducing Alloy problems to SMT constraints over a combination of that theory with linear integer arithmetic.
Reasoning About Specifications in Linear Logic

Dan DaCosta and Gopalan Nadathur

Computer Science Department,
University of Minnesota

This research aims to develop a framework for reasoning about specifications in linear logic. The formalization of many computational systems, ranging from hardware to (imperative) programming languages to logic, requires an accounting of resource use. Linear logic provides a means for modelling the use of resources at a logical level, thereby raising the possibility of simplifying reasoning about this aspect by moving it to a level at which it can benefit from meta-theoretic support. The objective of our work is to demonstrate the benefits of such an approach by first building the capability for reasoning about such specifications in a mechanized way and then by exhibiting its use in relevant applications.

Our work is oriented around a particular presentation of linear logic called Forum that was described by Dale Miller. Forum provides a treatment of the complete collection of logical symbols in linear logic. A particular virtue of this system is that it does so while providing a goal directed treatment of derivability. One benefit of this treatment is that it allows rule-based specifications in object languages to be formalized in a way that accords closely with their intuitive understanding. Reasoning about such formalizations boils down eventually to considering what is and is not derivable in the underlying logic. A second benefit to using Forum shows up herein: the goal-directed nature of derivations can be leveraged in arguments about derivability.

Underlying our work is the two-level logic approach to reasoning about specifications. In this approach, we encode Forum in a second reasoning logic and then reason about specifications in Forum through this encoding. This approach actually underlies the Abella theorem prover that currently uses an intuitionistic logic for specifications. Replacing this specification logic with Forum raises special challenges. A particular challenge arises from the fact that in Forum we must consider the division of resources between different subderivations. In inductive arguments about derivability, the particular extent of resources is unknown, thereby posing a difficulty for an exhaustive consideration of all possibilities. Our observation, however, is that there are regularities in the form of the resources and of the goals that need to be derived and that these regularities can be exploited in a complete characterization of the division of resources in the reasoning process. We present a way to structure the reasoning framework towards exploiting the regularities that are relevant to particular contexts. We use this idea to build a version of the Abella system that supports the possibility of reasoning about linear logic specifications. We demonstrate the usefulness of the resulting system through specific applications.
Constraint Refined Taint Analysis for Software Vulnerability Detection

Ruoyu Zhang, Cesare Tinelli
The University of Iowa

Abstract. With the rapid growth of software industry, the demand of detecting zero-day software vulnerabilities has been increasing. In order to find the flaws in the system, static taint analyzers mark untrusted data as tainted and track them for unsecured behaviors. The approach is efficient and has detected numerous software vulnerabilities. However, one major drawback is that the most of static taint analyzers suffer from high rates of false positives. The main reason for the inaccuracy is that the traditional taint analysis is path-insensitive. To address this problem, we propose a new mechanism to improve the traditional static taint analysis by adding logic constraint into the taint propagation procedure. Moreover, a formal system is presented to capture the semantics of conditional taint propagation. TAR (Taint Analysis Refinement) is a prototype system implementing the idea of conditional tainting. It could achieve a lower false positive rate and produce path-sensitive analysis results without losing much of the efficiency.
Mechanizing the Computationally Complete Symbolic Attacker in Coq

Gergei Bana¹, Rohit Chadha², and Ajay Kumar Eeralla²

¹ INRIA Paris-Rocquencourt
² University of Missouri, Columbia, MO

Abstract. In recent years, a new approach has been developed for verifying security protocols with the aim of combining the benefits of symbolic attackers and the benefits of unconditional soundness: the technique of the computationally complete symbolic attacker of Bana and Comon (BC) [1]. The BC technique has the advantage that it is syntactically simple and standard computational security notions can be translated to it in a straightforward matter. In this work, we further develop the BC framework for indistinguishability properties first introduced in [1] by formulating axioms and mechanizing security proofs in Coq, an interactive theorem-prover. In Coq, we proved the real-or-random secrecy of the Diffie-Hellman (DH) protocol as well as authentication property of a simplified Station-to-Station (STS) protocol. Our results establish that the BC framework can be effectively used for verification of not only equivalence properties, but trace properties of protocols as well.

References

Towards Synthesis from Assume-Guarantee Contracts involving Infinite Theories: A Preliminary Report

Andreas Katis\(^2\), Andrew Gacek\(^1\), Michael W. Whalen\(^2\)

\(^1\) Rockwell Collins Advanced Technology Center  
400 Collins Rd. NE, Cedar Rapids, IA, 52498, USA  
{andrew.gacek,john.backes,darren.cofer}@rockwellcollins.com

\(^2\) Department of Computer Science and Engineering,  
University of Minnesota, 200 Union Street, Minneapolis, MN 55455, USA  
katis001@umn.edu, whalen@cs.umn.edu

Abstract. In previous work, we have introduced a contract-based realizability checking algorithm for assume-guarantee contracts involving infinite theories, such as linear integer/real arithmetic and uninterpreted functions over infinite domains. This algorithm can determine whether or not it is possible to construct a realization (i.e., an implementation) of an assume-guarantee contract. The algorithm is similar to \(k\)-induction model checking, but involves the use of quantifiers to determine implementability.

While our work on realizability is inherently useful for virtual integration in determining whether it is possible for suppliers to build software that meets a contract, it also provides the foundations to solving the more challenging problem of component synthesis. In this paper, we provide an initial synthesis algorithm for assume-guarantee contracts involving infinite theories. To do so, we take advantage of our realizability checking procedure and a skolemization solver for \(\forall \exists\)-formulas, called AE-VAL. We show that it is possible to immediately adapt our existing algorithm towards synthesis by using this solver, using a demonstration example. We then discuss challenges towards creating a more robust synthesis algorithm.
Remote Attestation Protocol Synthesis and Verification

Paul Kline
The Information and Telecommunication Technology Center,
University of Kansas

The process of remote attestation can be tricky. Once we solve the problem of exactly what it is we would like to know, we must still
1. Respect our own privacy policy
2. Respond to counter-attestation request
3. Avoid "Measurement Deadlock" situations

In addition to these things, want to ensure that under all circumstances, both sides of the remote attestation process “line up,” i.e. the appraiser receives when the attester sends and vice versa. Using the theorem prover Coq we explore how to represent an imperative protocol language incorporating send and receive statements and how to automatically generate such a respectful protocol. We explore representing execution of this protocol as a relation and the properties we can verify about the process.
From Design Contracts to Component Requirements Verification

Jing (Janet) Liu, John D. Backes, Darren Cofer, and Andrew Gacek

Advanced Technology Center, Rockwell Collins
{jing.liu,john.backes,darren.cofer,andrew.gacek}@rockwellcollins.com

Abstract. During the development and verification of complex airborne systems, a variety of languages and development environments are used for different levels of the system hierarchy. As a result, there may be manual steps to translate requirements between these different environments. This paper presents a tool-supported export technique that translates high-level requirements from the software architecture modeling environment into observers of requirements that can be used for verification in the software component environment. This allows efficient verification that the component designs comply with their high-level requirements. It also provides an automated tool chain supporting formal verification from system requirements down to low-level software requirements that is consistent with certification guidance for avionics systems. The effectiveness of the technique has been evaluated and demonstrated on a medical infusion pump and an aircraft wheel braking system.

Keywords: design contracts, specification model, design model, AGREE, Simulink, requirements-based verification, certification
Contract-Based Verification of Complex Time-Dependent Behaviors in Avionic Systems

Devesh Bhatt\textsuperscript{1}, Arunabh Chattopadhyay\textsuperscript{1}, Wenchao Li\textsuperscript{2},
David Oglesby\textsuperscript{1}, Sam Owre\textsuperscript{2}, Natarajan Shankar\textsuperscript{2}

\textsuperscript{1}Honeywell Aerospace Labs \quad \textsuperscript{2}SRI International

Abstract. Avionic systems involve complex time-dependent behaviors across interacting components. This presentation demonstrates a contract-based approach for formally verifying these behaviors in a compositional manner. Included in the approach is a technique to abstract time-dependent behavior for time bounds that are prohibitively expensive to process with a model checker. Another unique feature of our contract-based tool is the support of architectural specification for multi-rate platforms. Preliminary results on applying this approach to the verification of an aircraft cabin pressure control system are promising.
Collective Program Analysis

Ganesha Upadhyaya and Hridesh Rajan

Iowa State University
{ganeshau,hridesh}@iastate.edu

Open source software repositories such as SourceForge, GitHub contain a large corpus of source code, its evolution history, and related artifacts available under various software projects. Recently there has been significant interest in leveraging information contained within this openly available corpora to help software construction and maintenance, e.g. for defect prediction, programming patterns, bug fix suggestions, specification inference, etc. Compared to traditional program analyses that focuses on a single project, the approaches that leverage this open source corpora perform analyses that cuts across projects (collective program analysis). An example of such an analysis is mining API preconditions.

We target the problem of accelerating collective program analysis, and clustering programs that behave similar for the given analysis. We consider running analysis on programs as interaction between two processes, an analysis process and a program process. Their interaction captures the parts of the program or the information that is relevant to the analysis. The interaction pattern depicts the information being queried by the analysis. Consider a classic data-flow analysis, “uninitialized variables” that checks for uses of uninitialized variables. The parts of the program that are relevant for this analysis are the variable definitions that initializes the variables and variable uses. The analysis is looking for the patterns where there are variable uses prior to their definitions.

For accelerating the collective program analysis, we limit the analysis to relevant program parts. A key challenge is determining the information relevant to an analysis by analyzing the arbitrary analysis code. For clustering programs, we use the interaction pattern of the analysis and the program. We believe many practical program analysis including data-flow analysis, path-sensitive analysis, model checking, etc can benefit from our approach. Our preliminary evaluation indicates that our technique can cut down the overall analysis time by up to one-fourth and clustering programs based on analysis-program interaction pattern can yield larger program clusters when compared to structural graph clustering.

There has been efforts to reduce the analysis time of a program by detecting and eliminating infeasible paths before performing the analysis. Pruning the paths and retaining only the analysis related nodes has proven to solve the scalability issues with path-sensitive analysis. Reusing analysis results to accelerate interprocedural analysis by computing partial or complete procedure summaries is also studied. Recently, there also has been efforts to accelerate programs that share large modules of code. However, to best to our knowledge there is no technique that can benefit analysis across programs and cluster programs specific to analysis.
Online Verification-Validation

Jared Wright
University of Colorado-Boulder

Abstract. Today’s programmers face a false choice between creating software that is extensible and software that is verifiable. Dynamic languages permit software that is richly extensible (via dynamic code loading, dynamic object extension, and various forms of reflection), allowing programmers to enrich extensible languages (e.g., via common JavaScript libraries). However, such extensions cannot be statically verified, leading to a proliferation of programming errors.

This talk discusses a proposed system of online verification-validation (OVV), a language and VM design suited to verification in this context. The design enables a “phaseless” approach to program analysis, interposing abstract interpretation with concrete execution, allowing an analyzer to use dynamic information from concrete executions to provide universal guarantees about future execution. This talk presents a conceptual overview of OVV via a few typical motivating examples, a current (simplified) implementation of the proposed system, and directions for future research.
Formal $\mathbb{K}$ to Isabelle Translation Framework
based on $\mathbb{K}$ Semantics

Liyi Li and Elsa Gunter \{liyili, egunter\}@illinois.edu
Department of
Computer Science, University of
Illinois at Urbana-Champaign

Abstract. $\mathbb{K}$ [1] is a formal executable framework to define programming languages and type systems based on a new invention of Rewriting Logic. It provides formal analysis tools to analyze the syntax and semantics of a language. The way that $\mathbb{K}$ views programming language semantics allows programmers to easily and quickly define programming constructs and investigate the relations and behaviors between the programming languages and different external program environments. Isabelle [2] is a generic proof engine which allows mathematical formulas to be built in a formal language and provides tools to prove those formulas in a logical calculus.

In this paper, we investigate a way to connect $\mathbb{K}$ and Isabelle by building a translation framework to translate programming languages in $\mathbb{K}$ to theories defined in Isabelle because it can not only allow programmers to define their programming languages easily in $\mathbb{K}$ but also have the ability to reason about their languages in Isabelle. In order to show a well-established translation, we also define the $\mathbb{K}$ semantics in the paper. By defining the $\mathbb{K}$ semantics, we prove that every language translated by the $\mathbb{K}$ to Isabelle translation is sound and complete with respect to the language defined based on $\mathbb{K}$ semantics in Isabelle.

References

Dependently typed programming with lambda encodings in Cedille

Ananda Guneratne, Chad Reynolds, and Aaron Stump

Computer Science, The University of Iowa, Iowa City, Iowa, USA
{ananda-guneratne, chad-reynolds, aaron-stump}@uiowa.edu

Abstract. This talk presents Cedille, a dependent type theory based on lambda encodings. Cedille is an extension of the Calculus of Constructions with new type features enabling induction and large eliminations (computing a type by recursion on a term) for lambda encodings, which are not available for lambda-encoded data in related type theories. Cedille is presented through a number of examples, including both programs and proofs.
Engineering High-Assurance Software for Distributed Adaptive Real-Time Systems

Sagar Chaki
Software Engineering Institute
Carnegie Mellon University
Email: chaki@sei.cmu.edu

Abstract. Distributed Adaptive Real-Time (DART) systems are cyber-physical systems with physically separated nodes that communicate and coordinate to achieve their goals, and self-adapt to their environment to improve likelihood of success. DART systems promise to revolutionize several domains that impact our daily lives in critical ways, such as robotics, transportation, energy, and healthcare. However, to fully realize this potential, the software controlling DART systems must be engineered to have high-assurance, and certified to operate safely and effectively. Achieving this goal is challenging (and infeasible with current testing-based approaches) due to complexity resulting from concurrency and coordination, environment uncertainty, and unpredictable system evolution caused by (self-)adaptation.
In this talk, we present a sound engineering approach based on domain specific languages with precise semantics, rigorous analysis, and design constraints, which leads to assured behavior of DART systems. The approach consists of three interacting stages:

1. Design: First, the DART system and the properties to be assured are specified using two languages: (i) Architecture Analysis and Design Language (AADL) is used to specify the system design nodes, processes, threads and their timing attributes such as frequency, execution time, and criticality; and (ii) DART Modeling and Programming Language (DMPL) is used to specify the logical behavior of threads, and communication and synchronization between them. Both languages have precise syntax and semantics and work synergistically to specify the system in sufficient detail to enable rigorous analysis. The languages also support specification of two types of target properties: (i) critical properties that must always hold (e.g., collision avoidance); and (ii) probabilistic properties must satisfy a likelihood threshold (e.g., degree of area coverage).

2. Analyze: Next, the DART system is analyzed to verify the target properties. Three analyses are currently used: (i) Zero- Slack Rate Monotonic (ZSRM) schedulability is used to ensure that all threads are schedulable under normal conditions and that high critical threads never miss their deadlines even under overload conditions; (ii) Software Model Checking is used to ensure that critical properties are provided by the high-critical threads assuming they are schedulable; and (iii) Statistical Model Checking is used to ensure that the system satisfies desired probabilistic properties with required likelihood.
3. Build: Finally, C++ code is automatically generated from the AADL+DMPL description. The code links with the MADARA middleware for communication, uses the ZSRM scheduler to ensure correct runtime scheduling free of deadlocks, race conditions, and unbounded priority and criticality inversions, and also implements a synchronous model of computation, which enabled tractable verification, using barriers.

We have implemented a workbench available at https://github.com/cps-sei/dart to support our approach, and evaluated it on several model problems based on a multi-UAS missions. The result is a sound approach for producing high-assurance DART software involving multiple layers of the CPS stack. We conclude with open problems and directions for future work.

This research is part of a project on verifying DART systems at the CMU Software Engineering Institute. It is joint work with Bjorn Andersson, Dionisio de Niz, James Edmondson, Arie Gurfinkel, Charles Hammons, Jeffery Hansen, Scott Hissam, Mark Klein, David Kyle, and Gabriel Moreno.
Resequencing Decision Diagram Operations

Junaid Babar and Andrew S. Miner

Department of Computer Science, Iowa State University
Ames, IA 50011, USA
{junaid,asminer}@iastate.edu

Abstract. Decision Diagram manipulation often consists of applying the same operation to a collection of decision diagram nodes. Many of these operations are commutative and associative (such as set union and set intersection), and allow us to alter the order in which these operations are applied. In this preliminary work, we explore algorithms and heuristics that aim to improve performance by altering the sequence in which an operation is applied over a collection of decision diagrams.

Keywords: binary decision diagram, optimization
A Semantics for Attestation Protocols using
Session Types in Coq

Adam Petz

The Information and Telecommunication Technology Center,
University of Kansas

As our world becomes more connected, the average person must place
more trust in cloud systems for everyday transactions. However, estab-
lishing trust in these remote systems is not a trivial task, especially in the
diverse, distributed ecosystem of today's networked computers. Remote
Attestation is a mechanism for establishing trust in a remotely running
system where an appraiser requests information from a target that can
be used to evaluate its operational state. The target responds with evi-
dence providing configuration information, run-time measurements, and
authenticity meta-evidence used by appraiser to determine if it trusts the
target system. To apply Remote Attestation broadly, we must have ates-
tation protocols that perform operations on a collection of applications,
each of which must be measured differently. Verifying that these proto-
cols behave as expected and accomplish their diverse attestation goals is
a unique challenge. An important first step is to understand the struc-
tural properties and execution patterns they share. This talk will outline
a semantic framework for attestation protocol execution within the Coq
verification environment including a protocol representation based on
Session Types, a dependently typed model of perfect cryptography, and
an operational execution semantics. Within this framework, desirable
properties of protocol execution, such as progress and termination, hold
for all protocols by construction. The Coq environment is also a promis-
ing exploratory space to leverage the properties of messages exchanged
while proving properties of individual protocols and families of protocols.
Extensions of SMTCoq to Bit-Vectors and Arrays

Burak Ekici\textsuperscript{1}, Chantal Keller\textsuperscript{2}, Alain Mebsout\textsuperscript{1}, and Cesare Tinelli\textsuperscript{1}

\textsuperscript{1} The University of Iowa, Iowa City, IA, USA
\textsuperscript{2} LRI, Université Paris-Sud, Orsay, France

\textbf{Abstract.} SMTCoq is a communication tool between the Coq proof assistant and external SAT and SMT solvers. Based on a checker for generic first-order certificates implemented and proved correct in Coq, SMTCoq offers facilities both to check external SAT and SMT answers and to improve Coq’s automation using such solvers, in a safe way. It supports the SAT solver ZChaff, and the SMT solvers veriT and CVC4 for the quantifier free combination of a number of built-in theories. We will focus in particular on the extension of SMTCoq to the theories of fixed-width bit-vectors and functional arrays with extensionality.
A Mechanized Formalization of a Concurrent
Core Calculus with Modular Reasoning

Swarn Priya†, Mehdi Bagherzadeh⇤ and Hridesh Rajan†

Oakland University*, Rochester, Michigan, USA
mbagherzadeh@oakland.edu
Iowa State University†, Ames, Iowa, USA
spriya,hridesh@iastate.edu

Abstract. Most modern software systems are concurrent because con-
currency improves the responsiveness and performance by allowing simul-
taneous execution of multiple computations in a single program. Modular
reasoning about concurrent programs is important because it allows the
correctness of a module to be verified by just considering the imple-
mentation of the module and the interfaces of static types named in that
module, which is more scalable. Modular reasoning becomes difficult due
to the chances of interferences that may occur between any two instruc-
tions of a task (pervasive interference) and lack of information about the
behavior of interfering tasks (oblivious interference). Reasoning would
be easier if programmer modularly and statically knows the point at
which interference may happen (sparse interference) and also about the
behavior of interfering tasks (cognizant interference). In this work, we
present a mechanised formalization of a concurrent calculus Panini which
solves the problems of pervasive and oblivious interference thereby en-
abling modular reasoning about Panini programs. Panini supports an
abstraction called capsule which owns their states and communicates by
the asynchronous procedure invocation of their procedures and dynam-
ically transfers ownership. We have formalized Panini’s calculus and its
semantics in Coq proof assistant and proved its soundness.
Verification and Test Generation for Plan Executions in Autonomy Applications

Jason Biatek and Sanjai Rayadurgam
{biatek,rsanjai}@cs.umn.edu

Dept. of Computer Science and Engineering, University of Minnesota

Abstract. With autonomy increasingly becoming a key aspect of several safety-critical applications, such as driverless vehicles, UAVs, and surgical robots, there is an emerging need to demonstrate sufficient assurance in the correct and safe behavior of these systems. PLEXIL (Plan Execution Interchange Language), a language for representing plans for automation, is being used at NASA for various autonomy applications. In the context of an ongoing project to demonstrate the feasibility of developing autonomously piloted aircrafts, PLEXIL is used to model pilot procedures as described in the Aeronautical Information Manual. To support the verification and validation of plans expressed in PLEXIL, we have developed TPlex, a framework for verification and test case generation for plans expressed in PLEXIL. This framework performs semantics preserving transformation of PLEXIL plans, along with some optimizations, into models that can be analyzed by model-checkers such as JKind and Java-PathFinder. Automated test generation to meet structural criteria such as execution state coverage and transition coverage for PLEXIL plans is achieved by transforming the plans into the Lustre synchronous dataflow language and expressing test coverage obligations as properties to be refuted by the JKind model-checker. The resulting counter-examples are translated back to PLEXIL test scripts, which achieve the desired coverage when executed on the plan. A challenging aspect of test generation for PLEXIL plans is that long execution paths may be frequently needed to exercise reach important parts of the plan. In recent work, we have leveraged incremental test generation techniques for Lustre to significantly increase the coverage achieved by automatically generated tests. Instead of an unbounded search to generate tests that meet all the test obligations, the search for test cases is confined to a fixed bound on length. As tests are generated, further search for tests to satisfy unmet obligations continue from the partially concrete states already seen. While a simple incremental strategy in itself significantly improves the coverage achieved by the generated tests, we are also investigating various heuristic approaches to guide the search for achieving better coverage of test obligations. Further, to generate tests that are meaningful in the application domain, constraints on the plans execution environment need to be taken into account when generating tests.

Acknowledgement. This is based on work supported by NASA under Grant NNX13AR47G.
Practical Verification - Insights from a Formal Verification Study of the Linux Kernel

Suresh Kothari, Payas Awadhutkar and Ahmed Tamrawi

Electrical and Computer Engineering Department
Iowa State University
Email: {kothari, payas, ahmed}@iastate.edu

Abstract. Formal verification of large software has been a difficult target, riddled with the problem of scalability. Even if the obstacle of scale may be cleared, major challenges remain to adopt formal verification in practice.

This talk presents an empirical study using a top-rated formal verification tool for Linux, and draws insights from the study to discuss the intrinsic challenges for adopting formal verification in practice.

We discuss challenges that focus on practical needs: (a) facilitate cross-checking of verification results, (b) enable the use of formal verification for certification, and (c) integrate formal methods in a development environment. Leaning on visionary papers by Turing Award recipients, the talk will present ideas for advancing formal verification in new directions that would address practical needs.

The talk will present:
1. Real-world context and scenarios for verification
2. A motivating case study
3. Practical Needs
4. New directions for advancing verification methods to meet the practical needs
5. Graph models to facilitate verification
6. A demo of our platform to build verification and other tools
Formal Methods for Certification: Why and How?

Lucas Wagner
Rockwell Collins, Cedar Rapids, IA 52402
lucas.wagner@rockwellcollins.com

Abstract. Formal methods tools have been established as useful bug-finding tools in the avionics industry. However, certification guidance provided by DO-178B relegated their use for obtaining certification credit as alternate methods. Historically, alternate methods have been avoided due to their inherent uncertainty. Recent updates to the guidance resulted in the promotion of formal methods as a first-class citizen. The new DO-333 Formal Methods technology supplement to the newly revised DO-178C provides clear and explicit guidance on how to (and how not to) use formal methods to obtain certification credit. Further, the DO-330 Software Tool Qualification Considerations document provides comprehensive guidance for tool qualification. Together, these two documents provide much clearer guidance and assurance that formal methods tools can be successfully used for certification credit.

NASA has funded Rockwell Collins on two separate efforts to further investigate the use of formal methods tools to obtain certification credit. The first effort focused on identifying how formal methods tools could be used to satisfy certification objectives by following DO-333 guidance. The second, and ongoing effort is focused on investigating qualification of formal methods tools under the new DO-330 guidance. Both efforts focus on developing case studies that potential formal methods tool users may find useful when developing their own certification and qualification strategies. This talk will walk through these two projects and how conference attendees may find the developed case studies useful.
Verification of a Molecular Watchdog Timer

Samuel J. Ellis
Department of Computer Science, Iowa State University, Ames, IA, USA,
{sjeiellis@iastate.edu}

Abstract. Molecular programming creates and programs a wide variety of synthetic, self-assembling nanosystems to achieve desired structures and behaviors. When developing a molecular system, it can be difficult to determine whether the design satisfies the desired behavior. Formal modeling and verification techniques help determine the correctness of a system. This talk describes the use of mathematical analysis and probabilistic model checking to ensure the correctness of a designed molecular system. The talk shows how we derived the requirements for a Molecular Watchdog Timer, inspired by a device commonly used in safety-critical software systems, and represented its requirements in Continuous Stochastic Logic (CSL). Using an incremental design process, we developed three components to satisfy the requirements—an Absence Detector, a Threshold Filter, and an Amplifier—and used Stochastic Chemical Reaction Networks as a high-level programming language for their design. Use of the PRISM probabilistic model checker helped verify that the design model satisfies the requirements. (This is joint work with Eric R. Henderson, Titus H. Klinge, James I. Lathrop, Jack H. Lutz, Robyn R. Lutz, Divita Mathur, and Andrew S. Miner. The research is supported in part by NSF grants 1247051 and 1545028 and by the ISU LAS College Graduate Fellowship.)