

Games and Decisions for Rigorous Systems Engineering

Edited by

Nikolaj Bjørner¹, Krishnendu Chatterjee², Laura Kovacs³, and
Rupak M. Majumdar⁴

1 Microsoft Research – Redmond, US, nbjorner@microsoft.com

2 IST Austria – Klosterneuburg, AT, Krishnendu.Chatterjee@ist.ac.at

3 TU Wien, AT, lkovacs@complang.tuwien.ac.at

4 MPI for Software Systems – Kaiserslautern, DE, rupak@mpi-sws.org

Abstract

This report documents the program and the outcomes of the Dagstuhl Seminar 12461 “Games and Decisions for Rigorous Systems Engineering”. The seminar brought together researchers working in rigorous software engineering, with a special focus on the interaction between synthesis and automated deduction. This event was the first seminar of this kind and a kickoff of a series of seminars organised on rigorous systems engineering. The theme of the seminar was close in spirit to many events that have been held over the last decades. The talks scheduled during the seminar naturally reflected fundamental research themes of the involved communities.

Seminar 11.–16. November, 2012 – www.dagstuhl.de/12461

1998 ACM Subject Classification C.1.4 Parallel Architectures, D.2.4 Software/Program Verification, D.2.5 Testing and Debugging, D.3.1 Formal Definitions and Theory, D.3.2 Language Classifications, F.1.1 Models of Computation, F.4.1 Mathematical Logic, I.2.2 Automatic Programming, I.2.3 Deduction and Theorem Proving

Keywords and phrases Systems Engineering, Software Verification, Reactive Synthesis, Automated Deduction

Digital Object Identifier 10.4230/DagRep.2.11.45

1 Executive Summary

Nikolaj Bjørner

Krishnendu Chatterjee

Laura Kovacs

Rupak M. Majumdar

License  Creative Commons BY-NC-ND 3.0 Unported license
© Nikolaj Bjørner, Krishnendu Chatterjee, Laura Kovacs, and Rupak M. Majumdar

Principled approaches to systems design offer several advantages, including developing safety-critical systems and scaling technological advances with multi-core processes and cloud computing. Rigorous mathematical techniques, such as model checking, decision procedures, and abstract interpretation, are dominantly used a posteriori in systems engineering: a program is formally analyzed after it has been developed. In the context of rigorous systems engineering, post-hoc verification is however very costly and error-prone. The explosion of concurrent computation in the new generation of embedded systems has therefore motivated the integration of established methods with novel techniques in the design process from day one.



Except where otherwise noted, content of this report is licensed
under a Creative Commons BY-NC-ND 3.0 Unported license

Games and Decisions for Rigorous Systems Engineering, *Dagstuhl Reports*, Vol. 2, Issue 11, pp. 45–65

Editors: Nikolaj Bjørner, Krishnendu Chatterjee, Laura Kovacs, and Rupak M. Majumdar



Dagstuhl Reports

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Such an integration has been materialized in using game theoretic synthesis of reactive systems from higher level design requirements. In many synthesis algorithms, it is better to work with symbolic representations, where the state space is modeled using logical formulas. This enables techniques to scale to potentially infinite models, but requires decision procedures for checking the validity of sentences in the pertinent logical theories. The increasingly complex integration of model checking with complementary techniques such as software testing has imposed new requirements on decision procedures, such as proof generation, unsatisfiable core extraction, and interpolation.

The main goal of the Dagstuhl Seminar 12461 “Games and Decisions for Rigorous Systems Engineering” was to bring together researchers working in the field of rigorous systems engineering, the tool-supported application of mathematical reasoning principles to the design and verification of complex software and hardware systems. The seminar had a special focus on developing systems (reactive, concurrent, distributed) using recent advances in game theoretic synthesis and in decision procedures and automated deduction techniques.

The seminar covered the following three main areas:

- software verification (reactive, concurrent, distributed);
- game theory and reactive synthesis;
- decision procedures (SAT, SMT, QBF) and theorem proving (first and higher order).

Within the scope of these areas, the seminar addressed tooling around software testing, model checking, interpolation, decision procedures, and model finding methods in automated theorem proving.

In the spirit of advancing tools and theory in related areas of theorem proving and model checking, the seminar schedule included tutorials on games, synthesis, theorem proving; research talks on recent results; and discussion sessions on applications and exchange formats for benchmarking tools.

The seminar fell on 5 days in the week of November 12–16, 2012. All together, 43 researchers participated (11 women and 32 men).

2 Table of Contents

Executive Summary

Nikolaj Bjørner, Krishnendu Chatterjee, Laura Kovacs, and Rupak M. Majumdar . 45

Overview of Talks

Lazy Abstraction with Interpolants for Arrays <i>Francesco Alberti</i>	49
Conditional Model Checking: A Technique to Pass Information between Verifiers <i>Dirk Beyer</i>	49
Efficient Controller Synthesis for Consumption Games with Multiple Resource Types <i>Tomas Brazdil</i>	50
Stochastic Program Synthesis with Smoothed Numerical Search <i>Swarat Chaudhuri</i>	50
Games in System Design: Tutorial and Survey <i>Laurent Doyen</i>	51
A new learning scheme for QDPLL solvers <i>Uwe Egly</i>	51
Inductive Data Flow Graphs <i>Azadeh Farzan</i>	51
Synthesis of reactive systems <i>Bernd Finkbeiner</i>	52
Deciding Floating-Point Logic with Systematic Abstraction <i>Alberto Griggio</i>	52
Concurrent Test Generation using Concolic Multi-Trace Analysis <i>Aarti Gupta</i>	53
VINTA: Verification with Interpolation and Abstract Interpretation <i>Arie Gurfinkel</i>	53
Proof Tree Preserving Interpolation <i>Jochen Hoenicke</i>	54
Underapproximation of Procedure Summaries for Integer Programs <i>Radu Iosif</i>	54
Proving Properties about Functional Programs <i>Moa Johansson</i>	54
Preprocessing for first-order logic with applications to hardware verification <i>Konstantin Korovin</i>	55
Asynchronous Games over Tree Architectures <i>Anca Muscholl</i>	55
A Tutorial on SAT and SMT <i>Albert Oliveras</i>	56
Decision Problems for Linear Recurrence Sequences <i>Joel Ouaknine</i>	56

Automated Game-theoretic Verification for Probabilistic Systems <i>David Parker</i>	56
Beluga: Programming proofs in context <i>Brigitte Pientka</i>	57
Proving termination of C-like programs using MAX-SMT <i>Albert Rubio</i>	57
Program verification as constraint solving (also for CTL* properties) <i>Andrey Rybalchenko</i>	58
QBFs and Certificates <i>Martina Seidl</i>	58
Incremental Upgrade Checking by Means of Interpolation-based Function Summaries <i>Natasha Sharygina</i>	58
Quantitatively Relaxed Concurrent Data Structures <i>Ana Sokolova</i>	59
Introduction to the Sketch Synthesis System <i>Armando Solar-Lezama</i>	59
A Semantic Account for Modularity in Multi-language Modelling of Search Problems <i>Eugenia Ternovska</i>	59
Secure Two-Party Computation in ANSI C <i>Helmut Veith</i>	60
Verification of Low Level List Manipulation <i>Tomas Vojnar</i>	60
First-order theorem proving and Vampire <i>Andrei Voronkov</i>	61
Labelled Interpolation Systems <i>Georg Weissenbacher</i>	61
Parameterized Model Checking of Fault-tolerant Distributed Algorithms <i>Josef Widder</i>	62
Complete Instantiation-Based Interpolation <i>Thomas Wies</i>	62
Working Groups	63
Discussions	63
Collaborations and Interaction	64
Participants	65

3 Overview of Talks

3.1 Lazy Abstraction with Interpolants for Arrays

Francesco Alberti (University of Lugano, CH)

License © © ⊖ Creative Commons BY-NC-ND 3.0 Unported license
© Francesco Alberti

Joint work of Alberti, Francesco; Bruttomesso, Roberto; Ghilardi, Silvio; Ranise, Silvio; Sharygina, Natasha

Main reference F. Alberti, R. Bruttomesso, S. Ghilardi, S. Ranise, N. Sharygina, “Lazy Abstraction with Interpolants for Arrays,” in Proc. of 18th Int’l Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR’12), LNCS, Vol. 7180, pp. 46–61, 2012.

URL http://dx.doi.org/10.1007/978-3-642-28717-6_7

Efficient and automatic model checking of software with unbounded data structure is a long standing scientific challenge. Abstraction/refinement techniques need to be carefully adapted when unbounded data structures come into play because of the need of quantified predicates. In this talk we will describe a recently proposed framework, “Lazy Abstraction with Interpolants for Arrays”, suited for reasoning about programs with unbounded arrays. The framework integrates a symbolic backward reachability analysis with an interpolation-based refinement procedure. It allows for an efficient handling of quantified formulas representing backward reachables states and exploiting of quantifier-free interpolation algorithms for refining predicates along spurious counterexamples. The talk will also describe SAFARI (SMT-based Abstraction For Arrays with Interpolants), a tool implementing this framework, and present “term abstraction”, a heuristic used to tune interpolation algorithms.

3.2 Conditional Model Checking: A Technique to Pass Information between Verifiers

Dirk Beyer (Universität Passau, DE)

License © © ⊖ Creative Commons BY-NC-ND 3.0 Unported license
© Dirk Beyer

Joint work of Beyer, Dirk; Henzinger, Thomas A.; Keremoglu, M. Erkan; Wendler, Philipp

Main reference D. Beyer, T.A. Henzinger, M.E. Keremoglu, P. Wendler, “Conditional Model Checking: A Technique to Pass Information between Verifiers,” in Proc. of the 20th ACM SIGSOFT Int’l Symp. on the Foundations of Software Engineering (FSE’12), ACM, 2012.

URL <http://dx.doi.org/10.1145/2393596.2393664>


URL http://www.sosy-lab.org/dbeyer/Publications/2012-FSE.Conditional_Model_Checking.pdf

Software model checking, as an undecidable problem, has three possible outcomes: (1) the program satisfies the specification, (2) the program does not satisfy the specification, and (3) the model checker fails. The third outcome usually manifests itself in a space-out, time-out, or one component of the verification tool giving up; in all of these failing cases, significant computation is performed by the verification tool before the failure, but no result is reported. We propose to reformulate the model-checking problem as follows, in order to have the verification tool report a summary of the performed work even in case of failure: given a program and a specification, the model checker returns a condition P —usually a state predicate— such that the program satisfies the specification under the condition P —that is, as long as the program does not leave the states in which P is satisfied. In our experiments, we investigated as one major application of conditional model checking the sequential combination of model checkers with information passing. We give the condition that one model checker produces, as input to a second conditional model checker, such that the verification problem for the second is restricted to the part of the state space that is

not covered by the condition, i.e., the second model checker works on the problems that the first model checker could not solve. Our experiments demonstrate that repeated application of conditional model checkers, passing information from one model checker to the next, can significantly improve the verification results and performance, i.e., we can now verify programs that we could not verify before.

3.3 Efficient Controller Synthesis for Consumption Games with Multiple Resource Types

Tomas Brazdil (Masaryk University, CZ)


License  Creative Commons BY-NC-ND 3.0 Unported license
© Tomas Brazdil

Joint work of Brazdil, Tomas; Chatterjee, Krishnendu; Kucera, Antonin; Novotny, Petr

We introduce consumption games, a model for discrete interactive system with multiple resources that are consumed or reloaded independently. More precisely, a consumption game is a finite-state graph where each transition is labeled by a vector of resource updates, where every update is a non-positive number or omega. The omega updates model the reloading of a given resource. Each vertex belongs either to player Box or player Diamond, where the aim of player Box is to play so that the resources are never exhausted. We consider several natural algorithmic problems about consumption games, and show that although these problems are computationally hard in general, they are solvable in polynomial time for every fixed number of resource types (i.e., the dimension of the update vectors) and bounded resource updates.

3.4 Stochastic Program Synthesis with Smoothed Numerical Search




Swarat Chaudhuri (Rice University, US)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Swarat Chaudhuri

Writing programs that behave optimally on probabilistic inputs is a highly challenging task. In this talk, I will describe a program synthesis procedure targeted to such tasks. The procedure takes an input an infinite-state program sketch annotated with a set of boolean assertions and a set of quantitative objectives. The procedure’s goal is to find an implementation that (a) satisfies the boolean assertions with probability above a certain bound; (b) in the expected behavior of the program, the quantitative objectives are minimized. We solve this problem using a combination of local numerical optimization and probabilistic abstract interpretation, called “smoothed numerical search”. The core idea of the algorithm is to approximate a program using a series of smooth mathematical functions. Each of these approximations is “unsound”; however, at the limit they converge to a sound abstraction.

3.5 Games in System Design: Tutorial and Survey




Laurent Doyen (CNRS, ENS – Cachan, FR)

License    Creative Commons BY-NC-ND 3.0 Unported license
© Laurent Doyen

We present a tutorial introduction on two-player games played on graphs, we discuss their fundamental properties, and basic ingredients for algorithmic solutions and complexity analysis. Along with applications in the design and formal verification of reactive systems, we survey recent results about games with quantitative objectives, combination of multiple objectives, and partial-observation games.

3.6 A new learning scheme for QDPLL solvers

Uwe Egly (TU Wien, AT)

License    Creative Commons BY-NC-ND 3.0 Unported license
© Uwe Egly

Joint work of Egly, Uwe; Lonsing, Florian; Van Gelder, Allen




Most of today's DPLL-based QBF solvers employ Q-resolution to learn clauses or cubes. The classical learning scheme from conflicts starts with the clause falsified by the current assignment and resolve upon existential variables in reverse assignment order using antecedent clauses stored in the implication graph during BCP.

We identify a class of formulas F_1, F_2, \dots for which (i) learning a single clause with the above scheme in an evaluation of F_k is exponential in k and (ii) F_k has a resolution proof of linear length. We propose a new learning scheme which avoids such efficiency problems. The new scheme employs resolution “from the decisions towards the conflict”, i.e., in the direction in which decisions have been propagated during QBCP. We discuss experimental results obtained with a very first implementation into depQBF.

This is joint work with Florian Lonsing (Vienna University of Technology) and Allen Van Gelder (University of California at Santa Cruz).

3.7 Inductive Data Flow Graphs

Azadeh Farzan (University of Toronto, CA)

License    Creative Commons BY-NC-ND 3.0 Unported license
© Azadeh Farzan


Joint work of Farzan, Azadeh; Kincaid, Zachary; Podelski, Andreas;
Main reference To appear in POPL 2013 (proceedings information not available yet).

The correctness of a sequential program can be shown by the annotation of its control flow graph with inductive assertions. We propose inductive data flow graphs, data flow graphs with incorporated inductive assertions, as the basis of an approach to verifying concurrent programs. An inductive data flow graph accounts for a set of dependencies between program actions in interleaved thread executions, and therefore stands as a representation for the set of concurrent program traces which give rise to these dependencies. The approach first constructs an inductive data flow graph and then checks whether all program traces are represented. The size of the inductive data flow graph is polynomial in the number of data

dependencies (in a sense that can be made formal); it does not grow exponentially in the number of threads unless the data dependencies do. The approach shifts the burden of the exponential explosion towards the check whether all program traces are represented, i.e., to a combinatorial problem (over finite graphs).

3.8 Synthesis of reactive systems

Bernd Finkbeiner (Universität des Saarlandes, DE)


License  Creative Commons BY-NC-ND 3.0 Unported license
© Bernd Finkbeiner

Joint work of Finkbeiner, Bernd; Schewe, Sven; Jacobs, Swen

More than fifty years after its introduction by Alonzo Church, the synthesis problem is still one of the most intriguing challenges in the theory of reactive systems. Synthesis is particularly difficult in the setting of distributed systems, where we try to find a combination of process implementations that jointly guarantee that a given specification is satisfied. A reduction from multi-player games shows that the problem is in general undecidable. Despite this negative result, there is a line of discoveries where the decidability of the synthesis problem was established for distributed systems with specific architectures, such as pipelines and rings, or other restrictions on the problem, such as local specifications. Encouraged by these findings, new specification languages like Coordination Logic aim for a comprehensive logical representation and a uniform algorithmic treatment of the decidable synthesis problems. In this talk, I will trace the progress from isolated decidability results towards universal synthesis logics and algorithms. I will demonstrate how the logical representation of the synthesis problem simplifies the identification of decidable cases and give an overview on the state of the art in decision procedures and strategy construction algorithms for the synthesis of reactive systems.

3.9 Deciding Floating-Point Logic with Systematic Abstraction

Alberto Griggio (Fondazione Bruno Kessler – Trento, IT)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Alberto Griggio

Joint work of Haller, Leopold; Griggio, Alberto; Brain, Martin; Kroening, Daniel;
Main reference L. Haller, A. Griggio, M. Brain, D. Kroening, “Deciding Floating-Point Logic with Systematic Abstraction,” in Proc. of the 12th Conf. on Formal Methods in Computer-Aided Design (FMCAD’12), 2012.

URL





URL <http://www.cs.utexas.edu/hunt/FMCAD/FMCAD12/fmcad2012.pdf>

We present a bit-precise decision procedure for the theory of binary floating-point arithmetic. The core of our approach is a non-trivial generalisation of the conflict analysis algorithm used in modern SAT solvers to lattice-based abstractions. Existing complete solvers for floating-point arithmetic employ bit-vector encodings. Propositional solvers based on the Conflict Driven Clause Learning (CDCL) algorithm are then used as a back-end. We present a natural-domain SMT approach that lifts the CDCL framework to operate directly over abstractions of floating-point values. We have instantiated our method inside MathSAT with the floating-point interval abstraction. The result is a sound and complete procedure for floating-point arithmetic that outperforms the state-of-the-art significantly on problems that

check ranges on numerical variables. Our technique is independent of the specific abstraction and can be applied to problems beyond floating-point satisfiability checking.

3.10 Concurrent Test Generation using Concolic Multi-Trace Analysis

Aarti Gupta (NEC Laboratories America, Inc. – Princeton, US)

License     Creative Commons BY-NC-ND 3.0 Unported license
© Aarti Gupta





Joint work of Gupta, Aarti; Razavi, Noloofar; Kahlon, Vineet

Discovering concurrency bugs is inherently hard due to nondeterminism in multi-thread scheduling. Predictive analysis techniques have been used to find such bugs by observing given test runs, and then searching for other interesting thread interleavings. For sequential code, SMT-based concolic execution techniques have been used successfully to generate interesting test inputs to increase structural code coverage such as branch or statement coverage. In this talk, I will describe our recent work that targets increasing code coverage in multi-thread programs by using a concolic multi-trace analysis (CMTA) that combines elements of prediction with generation of new test inputs. We have implemented CMTA and show encouraging results on benchmark programs.

This is joint work with Niloofar Razavi, Franjo Ivancic, and Vineet Kahlon; to appear soon at the Asian Symposium on Programming Languages and Systems (APLAS 2012).

3.11 VINTA: Verification with Interpolation and Abstract Interpretation

Arie Gurfinkel (CMU – Pittsburgh, US)

License     Creative Commons BY-NC-ND 3.0 Unported license
© Arie Gurfinkel

Joint work of Gurfinkel, Arie; Albarghouthi, Aws; Chechik, Marsha


Main reference A. Gurfinkel, A. Albarghouthi, M. Chechik, “Craig Interpretation,” in Proc. of 19th Int’l Symp. on Static Analysis (SAS’12), LNCS, Vol. 7460, pp. 300–316, Springer, 2012.

URL http://dx.doi.org/10.1007/978-3-642-33125-1_21

Abstract interpretation (AI) is one of the most scalable automated program verification techniques. The scalability is achieved through aggressive abstraction in basic analysis steps (i.e., joins and widening). This leads to loss of precision. As such, AI is plagued by false alarms. In this talk, I will present VINTA, an algorithm that enriches AI with Abstraction Refinement techniques from Model Checking to alleviate the false alarms. VINTA is an iterative algorithm that uses Craig interpolants to refine and guide AI away from false alarms. VINTA is based on a novel refinement strategy that capitalizes on recent advances in SMT and interpolation-based Model Checking. On one hand, it can find concrete counterexamples to justify alarms produced by AI. On the other, it can strengthen invariants to exclude alarms that cannot be justified. The refinement process continues until either a safe inductive invariant is computed, a counterexample is found, or resources are exhausted. This strategy allows VINTA to recover precision lost in many AI steps. VINTA has been implemented as part of the UFO verification framework. It is a big contributor to the success of UFO in the 2nd International Software Verification Competition.

3.12 Proof Tree Preserving Interpolation

Jochen Hoenicke (Universität Freiburg, DE)


License  Creative Commons BY-NC-ND 3.0 Unported license
© Jochen Hoenicke

Craig interpolants are widely used in model checking and state space abstraction. Interpolants typically are extracted from proofs produced by theorem provers. While this extraction procedure is easy and well understood in the context of propositional logic, extracting interpolants from a proof generated by an SMT solver is more complex. In contrast to SAT solvers, SMT solvers create new literals, e.g., to combine multiple theories in a Nelson-Oppen style or to split the solution space using cuts. These literals might contain symbols local to different parts of the interpolation problem. Such literals are called mixed, or, sometimes, uncolorable. Resolution steps on mixed literals are the major difficulty when extracting interpolants from proofs from SMT solvers.

We present a technique to compute Craig interpolants in the theory of uninterpreted functions combined with the theory of linear arithmetic either over the reals or the integers. The interpolation scheme is based on a syntactical restriction of the partial interpolants and specialized rules to interpolate resolution steps on mixed literals. Contrary to existing approaches, this scheme neither limits the inferences done by the SMT solver, nor does it transform the proof tree before extracting interpolants. The interpolation scheme is used in the interpolating SMT solver SMTInterpol.

3.13 Underapproximation of Procedure Summaries for Integer Programs


Radu Iosif (VERIMAG – Gières, FR)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Radu Iosif

We show how to underapproximate the procedure summaries of recursive programs over the integers using off-the-shelf analyzers for non-recursive programs. The novelty of our approach is that the non-recursive program we compute may capture unboundedly many behaviors of the original recursive program for which stack usage cannot be bounded. Moreover, we identify a class of recursive programs on which our method terminates and returns the precise summary relations without underapproximation. Doing so, we generalize a similar result for non-recursive programs to the recursive case. Finally, we present experimental results of an implementation of our method applied on a number of examples.

3.14 Proving Properties about Functional Programs

Moa Johansson (Chalmers UT – Göteborg, SE)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Moa Johansson

Joint work of Claessen, Koen; Johansson, Moa; Rosen, Dan; Smallbone, Nicholas

HipSpec is an automatic inductive theorem prover for proving properties about Haskell programs. It implements a novel bottom-up approach to lemma discovery: potentially

interesting lemmas about available functions and datatypes are first synthesised creating a richer background theory for the prover.

HipSpec consists of several sub-systems: Hip is an inductive theorem prover. It translates Haskell function definitions to first order logic and applies induction to given conjectures. Resulting proof obligations are passed to an off the shelf prover (for instance E or Z3).

QuickSpec is responsible for generating candidate lemmas about available functions and datatypes. It generates terms which are divided up into equivalence classes using counterexample testing. From these equivalence classes, equations can be derived. These are passed to Hip for proof. Those that are proved are added to the background theory and may be used in subsequent proofs.

HipSpec is available for download from <https://github.com/danr/hipspec>

3.15 Preprocessing for first-order logic with applications to hardware verification

Konstantin Korovin (University of Manchester, GB)

License © ⓘ ⊖ Creative Commons BY-NC-ND 3.0 Unported license
© Konstantin Korovin

Joint work of Krystof Hoder, Zurab Khasidashvili, Konstantin Korovin and Andrei Voronkov

Main reference K. Hoder, Z. Khasidashvili, K. Korovin, A. Voronkov, “Preprocessing Techniques for First-Order Clausification,” in Proc. of the 12th Conf. on Formal Methods in Computer-Aided Design (FMCAD’12), pp. 44–51, 2012.

URL http://www.cs.man.ac.uk/~korovink/my_pub/fmcad_2012.pdf

We discuss several preprocessing techniques for simplifying first-order formulas aimed at improving clausification. These include definition inlining and merging, simplifications based on new data structure called quantified AIG, and its combination with OBDDs. These techniques were inspired by applications of first-order theorem proving to hardware verification.

3.16 Asynchronous Games over Tree Architectures


Anca Muscholl (Université Bordeaux, FR)

License © ⓘ ⊖ Creative Commons BY-NC-ND 3.0 Unported license
© Anca Muscholl

The control problem starts with a plant and asks to restrict its controllable actions in such a way that a given specification is met. Synthesis can be seen as a special case of control. We consider a distributed version of the control problem where both plant and controller are parallel compositions of finite-state processes communicating via shared variables (also known as Zielonka asynchronous automata). The most important aspect of this model is that the processes participating in a synchronization can exchange the complete information about their causal past. Thus, it is an intriguing open problem whether this control setting is decidable. We show decidability when the communication architecture is acyclic. In this case, if there is a solution then controllers exchange only bounded information. The complexity of our algorithm is l -fold exponential with l being the height of the tree representing the architecture. We show that this complexity is tight.

3.17 A Tutorial on SAT and SMT

Albert Oliveras (TU of Catalonia – Barcelona, ES)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Albert Oliveras


In this tutorial, an overview of SAT and SMT will be given.

In the first part, we will introduce the problem of SAT and its state-of-the-art techniques. Starting from an abstract presentation of the DPLL procedure, we then explain how several conceptual enhancements can be added to it giving the so-called CDCL algorithm for SAT solving.

After that, we will focus on the problem of SMT. We first overview the most common theories that SMT solvers deal with. Then, we focus on the two main approaches to SMT: the eager and the lazy approach, making special emphasis on the last one. In particular, we make clear which are the requirements that a theory solver needs to have to be used in a DPLL(T) system.

3.18 Decision Problems for Linear Recurrence Sequences

Joel Ouaknine (University of Oxford, GB)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Joel Ouaknine


Joint work of Ouaknine, Joel; Worrel, James; Daws, Matt

Linear recurrence sequences (such as the Fibonacci numbers) permeate a vast number of areas of mathematics and computer science, and also have many applications in other fields such as economics and theoretical biology. In the context of synthesis and verification, linear recurrence sequences arise in connection with linear programs, probabilistic systems, stochastic logics, and linear dynamical systems, among others.

In this talk, I will focus on three fundamental decision problems for linear recurrence sequences, namely the Skolem Problem (does the sequence have a zero?), the Positivity Problem (is the sequence always positive?), and the Ultimate Positivity Problem (is the sequence ultimately always positive?).

3.19 Automated Game-theoretic Verification for Probabilistic Systems

David Parker (University of Birmingham, GB)

License  Creative Commons BY-NC-ND 3.0 Unported license
© David Parker

Joint work of Chen, Taolue; Forejt, Vojtěch; Kwiatkowska, Marta; Parker, David; Simaitis, Aistis

Main reference T. Chen, V. Forejt, M. Kwiatkowska, D. Parker, A. Simaitis, “Automatic Verification of Competitive Stochastic Systems,” in Proc. of the 18th Int’l Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS’12), LNCS, Vol. 7214, pp. 315–330, Springer, 2012





URL http://dx.doi.org/10.1007/978-3-642-28756-5_22

We present automatic verification techniques for turn-based stochastic multi-player games, which model probabilistic systems containing components that can either collaborate or compete in order to achieve particular goals. We give model checking algorithms for a temporal logic called rPATL, which allows us to reason about the collective ability of a set of

players to achieve a goal relating to the probability of an event's occurrence or the expected amount of cost/reward accumulated. We implement our techniques in an extension of the PRISM model checker and use them to analyse and detect potential weaknesses in systems such as algorithms for energy management and collective decision making for autonomous systems.

3.20 Beluga: Programming proofs in context





Brigitte Pientka (McGill University – Montreal, CA)

License     Creative Commons BY-NC-ND 3.0 Unported license
© Brigitte Pientka

We routinely reason about the runtime behavior of software using formal systems such as type systems or logics for access control or information flow to establish safety and liveness properties. In this talk, I will give an overview of Beluga, a dependently typed programming and proof environment. It supports specifying formal systems in the logical framework LF and directly supports common and tricky routines dealing with variables, such as capture-avoiding substitution and renaming. Moreover, Beluga allows embedding LF objects together with their context in programs and types supporting inductive and coinductive definitions and we can manipulate contextual LF objects via pattern matching. Taken together these features lead to a powerful language which supports writing compact and elegant proofs.

3.21 Proving termination of C-like programs using MAX-SMT

Albert Rubio (UPC – Barcelona, ES)

License     Creative Commons BY-NC-ND 3.0 Unported license
© Albert Rubio


We show how MAX-SMT can be used for proving and disproving termination of C-like programs. MAX-SMT allow us to characterize our termination problem giving different weights to the needed conditions, providing a better notion of progress. This also makes it easier to combine the process of building the termination argument with the usually necessary process of generating invariants.

Our technique focuses on proving termination, but sometimes from the already generated invariants and partial termination arguments, we can prove non-termination or warn about potential cases of non-termination.

The method has been implemented in a prototype that has successfully been tested on a wide sample of examples.

3.22 Program verification as constraint solving (also for CTL* properties)

Andrey Rybalchenko (TU München, DE)


License  Creative Commons BY-NC-ND 3.0 Unported license
© Andrey Rybalchenko

Joint work of Beyene, Tewodros; Popeea, Corneliu; Rybalchenko, Andrey;

First, we review how proving reachability and termination properties of transition systems, procedural programs, multi-threaded programs, and higher-order functional programs can be reduced to constraint solving. Second, we show how CTL* properties can be proved using constraint-based setting. Finally, we discuss adequate solving algorithms and tools.

3.23 QBFs and Certificates

Martina Seidl (University of Linz, AT)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Martina Seidl

Joint work of Seidl, Martina; Niemetz, Aina; Preiner, Mathias; Lonsing, Florian; Biere, Armin


Main reference A. Niemetz, M. Preiner, F. Lonsing, M. Seidl, A. Biere, “Resolution-Based Certificate Extraction for QBF,” in Proc. of the 15th Int’l Conf. on Theory and Applications of Satisfiability Testing (SAT’12), LNCS, Vol. 7317, pp. 430–435, Springer, 2012.

URL http://dx.doi.org/10.1007/978-3-642-31612-8_33

A certificate of (un)satisfiability for a quantified Boolean formula (QBF) represents concrete assignments to the variables of the formula. Certificates are not only witnesses for the truth value returned by a QBF solver, but also represent the solutions for practical applications of QBF like formal verification and model checking. Recently, an approach has been presented, which can be directly built on top of DPLL based QBF solvers. Starting from resolution proofs produced by the solver during clause and cube learning, the certificates are constructed by certain syntactic properties of the proof tree. Based on our integrated set of tools realizing resolution-based certificate extraction for QBFs in prenex conjunctive normal form, in this talk, we discuss the state-of-the-art of QBF certification and point out future challenges.

3.24 Incremental Upgrade Checking by Means of Interpolation-based Function Summaries

Natasha Sharygina (University of Lugano, CH)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Natasha Sharygina

Joint work of Ondrej, Sery; Fedyukovich, Grigory; Sharygina, Natasha

Main reference S. Ondrej, G. Fedyukovich, N. Sharygina, “Incremental Upgrade Checking by Means of Interpolation-based Function Summaries,” in Proc. of the 12th Conf. on Formal Methods in Computer-Aided Design (FMCAD’12), 2012.

URL <http://www.verify.inf.unisi.ch/files/fmcad2012.pdf>

During its evolution, a typical software/hardware design undergoes a myriad of small changes. However, it is extremely costly to verify each new version from scratch. As a remedy to this problem, we propose to use function summaries to enable incremental verification of the evolving systems. During the evolution, our approach maintains function summaries derived using Craig’s interpolation. For each new version, these summaries are used to perform a

local incremental check. Benefit of this approach is that the cost of the check depends on the extent of the change between the two versions and can be performed cheaply for incremental changes without resorting to re-verification of the entire system. Our implementation and experimentation in the context of the bounded model checking for C confirms that incremental changes can be verified efficiently for different classes of industrial programs.

3.25 Quantitatively Relaxed Concurrent Data Structures

Ana Sokolova (Universität Salzburg, AT)

License © ⓘ ⊖ Creative Commons BY-NC-ND 3.0 Unported license
© Ana Sokolova

Joint work of Henzinger, Thomas A.; Kirsch, Christoph M.; Payer, Hannes; Sezgin, Ali; Sokolova, Ana
Main reference T.A. Henzinger, C.M. Kirsch, H. Payer, A. Sezgin, A. Sokolova, “Quantitative Relaxations of Concurrent Data Structures,” in Proc. POPL 2013, to appear.

This talk is about our recent work on relaxing the semantics of concurrent data structures, in a quantitative way, so that they allow better-performing implementations. By their nature, data structures are bottlenecks in the presence of concurrency, e.g. the top pointer of a stack is a point of contention for which all threads compete. As a consequence, implementations of concurrent data structures often show negative scalability. Recent trends in concurrency show that relaxing the semantics may be the way to better performance. In this work we provide a framework for quantitative relaxations of concurrent data structures, where the allowed “error” from the perfect semantics is quantified by a distance. During the talk, we will use a stack as a running example. We also present a new concurrent implementation of a quantitatively relaxed stack that performs well and shows positive scalability.

3.26 Introduction to the Sketch Synthesis System

Armando Solar-Lezama (MIT – Cambridge, US)

License © ⓘ ⊖ Creative Commons BY-NC-ND 3.0 Unported license
© Armando Solar-Lezama

I will provide a brief introduction to the Sketch language and highlight some of the recent applications and open problems both in terms of language design and decision procedures.

3.27 A Semantic Account for Modularity in Multi-language Modelling of Search Problems

Eugenia Ternovska (Simon Fraser University – Burnaby, CA)

License © ⓘ ⊖ Creative Commons BY-NC-ND 3.0 Unported license
© Eugenia Ternovska


Joint work of Ternovska, Eugenia; Tasharofi, Shahab; Wu, Xiongnan

With the increased applications of distributed communicating systems, there is a strong need in formalisms that support modularity. In such systems, the representation language of a module may not even be known outside of that module. I will describe a semantic approach to formal modelling of such systems, and analyze the expressive power of adding a loop

operator. I will then describe an algorithmic schema for synthesizing solutions of modular systems. The solutions agree with each of the interacting, collaborating, mutually dependent modules. The algorithmic schema is instantiated with oracle procedures specific to each module. It generalizes ideas underlying “combined” solving such as the DPLL(T) procedure, branch-and-cut ILP solvers and state-of-the-art combination of ASP and CP. Joint work with Shahab Tasharrofi and Xiongnan Wu.

3.28 Secure Two-Party Computation in ANSI C

Helmut Veith (TU Wien, AT)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Helmut Veith

Joint work of Andreas Holzer, Martin Franz, Stefan Katzenbeisser, and Helmut Veith

Main reference A. Holzer, M. Franz, S. Katzenbeisser, H. Veith, “Secure two-party computations in ANSI C,” in Proc. of the 2012 ACM Conf. on Computer and Communications Security (CCS’12), pp. 772–783, ACM, 2012.

URL <http://dx.doi.org/10.1145/2382196.2382278>


URL http://www.sosy-lab.org/~dbeyer/Publications/2012-FSE.Conditional_Model_Checking.pdf

The practical application of Secure Two-Party Computation is hindered by the difficulty to implement secure computation protocols. While recent work has proposed very simple programming languages which can be used to specify secure computations, it is still difficult for practitioners to use them, and cumbersome to translate existing source code into this format. Similarly, the manual construction of two-party computation protocols, in particular ones based on the approach of garbled circuits, is labor intensive and error-prone.

The central contribution of the current paper is a tool which achieves Secure Two-Party Computation for ANSI C. Our work is based on a combination of model checking techniques and two-party computation based on garbled circuits. Our key insight is a nonstandard use of the bit-precise model checker CBMC which enables us to translate C programs into equivalent Boolean circuits. To this end, we modify the standard CBMC translation from programs into Boolean formulas whose variables correspond to the memory bits manipulated by the program. As CBMC attempts to minimize the size of the formulas, the circuits obtained by our tool chain are also size efficient; to improve the efficiency of the garbled circuit evaluation, we perform optimizations on the circuits. Experimental results with the new tool CBMC-GC demonstrate the practical usefulness of our approach.

3.29 Verification of Low Level List Manipulation

Tomas Vojnar (Brno University of Technology, CZ)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Tomas Vojnar

In the talk, we present an ongoing work related to the tool called Predator for verification of programs containing low level list manipulation. We first present some typical problems that arise in low-level list manipulating programs used in system software. Then, we briefly explain how these issues are tackled in Predator using a graph-based representation of sets of heaps (partially inspired by works on separation logic with higher order list predicates, but purely graph-based and significantly extended to cope with the low-level memory manipulation features).

3.30 First-order theorem proving and Vampire

Andrei Voronkov (University of Manchester, GB)

License © © ⊖ Creative Commons BY-NC-ND 3.0 Unported license
© Andrei Voronkov

Joint work of Krystof Hoder, Laura Kovács and Andrei Voronkov

In this tutorial we give a short introduction in first-order theorem proving and the use of the theorem prover Vampire.

I will discuss the the resolution and superposition calculus, introduce the saturation principle, present various algorithms implementing redundancy elimination, preprocessing and clause form transformation and demonstrate how these concepts are implemented in Vampire.

I will next also cover more advanced topics and features. Some of these features are implemented only in Vampire. This includes reasoning with theories, such as arithmetic, answering queries to very large knowledge bases, interpolation, and an original technique of symbol elimination, which allows one to automatically discover invariants in programs with loops.

3.31 Labelled Interpolation Systems

Georg Weissenbacher (TU Wien, AT)

License © © ⊖ Creative Commons BY-NC-ND 3.0 Unported license
© Georg Weissenbacher


Main reference G. Weissenbacher, “Interpolant Strength Revisited,” in Proc. of the 15th Conf. on Theory and Applications of Satisfiability Testing (SAT’12), LNCS, Vol. 7317, pp. 312–326, Springer, 2012.

URL http://dx.doi.org/10.1007/978-3-642-31612-8_24

Craig’s interpolation theorem has numerous applications in model checking, automated reasoning, and synthesis. The intrinsic properties of interpolants enable concise abstractions in verification and smaller circuit designs in synthesis. There is a variety of interpolation systems which derive interpolants from refutation proofs; these systems are ad-hoc and rigid in the sense that they provide exactly one interpolant for a given proof. In this talk, I will discuss how refutation-based interpolation techniques can be parametrised to remove this limitation. Labelled interpolation systems allow for the systematic variation of the logical strength and the size of Craig interpolants. In addition, they generalise a number of existing interpolation techniques for propositional and first-order logic. It is still an open question how applications can exploit the additional flexibility provided by this novel interpolation system.

3.32 Parameterized Model Checking of Fault-tolerant Distributed Algorithms

Josef Widder (TU Wien, AT)


License  Creative Commons BY-NC-ND 3.0 Unported license
© Josef Widder

Joint work of John, Annu; Konnov, Igor; Schmid, Ulrich; Veith, Helmut; Widder, Josef
Main reference A. John, I. Konnov, U. Schmid, H. Veith, J. Widder, “Counter Attack on Byzantine Generals: Parameterized Model Checking of Fault-tolerant Distributed Algorithms,” arXiv:1210.3846v2 [cs.LO].
URL <http://arxiv.org/abs/1210.3846>

We introduce a method for automated parameterized verification of fault-tolerant distributed algorithms. The distributed algorithms we consider are parameterized by both the number of processes and the assumed maximum number of Byzantine faulty processes. At the center of our technique is a parametric interval abstraction (PIA) where the interval boundaries are arithmetic expressions over parameters. Using PIA for both data abstraction and a new form of counter abstraction, we reduce the parameterized problem to finite-state model checking. We demonstrate the practical feasibility of our method by verifying several variants of the well-known distributed algorithm by Srikanth and Toueg. To the best of our knowledge, this is the first paper to achieve parameterized automated verification of Byzantine fault-tolerant distributed algorithms.

3.33 Complete Instantiation-Based Interpolation

Thomas Wies (New York University, US)

License  Creative Commons BY-NC-ND 3.0 Unported license
© Thomas Wies

Joint work of Wies, Thomas; Totla, Nishant

Craig interpolation has been a valuable tool for formal methods with interesting applications in program analysis and verification. Modern SMT solvers implement interpolation procedures for the theories that are most commonly used in these applications. However, many application-specific theories remain unsupported, which limits the class of problems to which interpolation-based techniques apply. In this talk, I present a generic framework to build new interpolation procedures via reduction to existing interpolation procedures. We consider the case where an application-specific theory can be formalized as an extension of a base theory with additional symbols and axioms. Our technique uses finite instantiation of the extension axioms to reduce an interpolation problem in the theory extension to one in the base theory. We identify a model-theoretic criterion that allows us to detect the cases where our technique is complete. We discuss specific theories that are relevant in program verification and that satisfy this criterion. In particular, we obtain complete interpolation procedures for theories of arrays and linked lists. The latter is the first complete interpolation procedure for a theory that supports reasoning about complex shape properties of heap-allocated data structures. We have implemented this procedure in a prototype on top of existing SMT solvers and used it to automatically infer loop invariants of list-manipulating programs. This is joined work with Nishant Totla.

4 Working Groups

The goal of the seminar was to explore synergies between game theoretic synthesis algorithms and decision procedures. To make the potential for collaboration clear and concrete, we identified a number of research questions to be addressed during our seminar:

Tools for Systems Design Which tools and techniques developed for software/hardware verification can be used to engineer next generation industrial-strength synthesis systems?

Solvers for Concurrency What further theory extensions are needed to design efficient analysis and synthesis procedures for concurrent programs? For example, what reasoning capabilities are required for the synthesis of concurrent data structures? For synthesis of parameterized systems? For reasoning in the presence of weak memory models?

Games Modulo Theories How to solve games over rich domains, for example, over infinite alphabets subject to a background theory? Do decision procedures provide practical abstraction mechanisms for synthesis problems? What of more expressive properties, such as those studied in stochastic and quantitative games?

Synthesis and New Capabilities for Decision Procedures What new capabilities must decision procedures expose to help in synthesis? For example, can interpolants be used in generating certificates for synthesis problems? Which fixed-point logic is best solved with which search method and/or deductive techniques?

Games and Systems Composition How can we decompose large specifications into smaller specifications that can be synthesized automatically? What is the best way to use game-theoretic solutions for composable language design?

This Dagstuhl Seminar 12461 “Games and Decisions for Rigorous Systems Engineering” brought together renowned as well as young aspiring researchers from three groups.

- The first group was formed by researchers developing new paradigms of concurrent software, such as multi-core, distributed computing, and software testing.
- The second group comprised researchers working on reactive synthesis, such as decompositions of large specifications, parameterised synthesis, game theoretic models and partial-information game models.
- The third group consisted of researchers who design and combine decision procedures for various logical formalisms, such as propositional satisfiability (SAT), satisfiability modulo theory (SMT), quantified boolean formulas, and first-order logic.

5 Discussions

Our seminar initiated discussions between experts and young researchers of exceptional talent from reactive synthesis and automated deduction. Moreover, the presence of academia and industry enabled to discuss problems that are challenging both from the theoretical and practical point of view. These problems included the use of decision procedures and automated deduction in automata-based synthesis; generalizations of model checking algorithms within game-theoretic frameworks; integration of model checking with complementary techniques such as software testing; and extending SMT solvers and theorem provers with proof generation, unsatisfiable core extraction, and Craig interpolation.

6 Collaborations and Interaction

An outcome of the seminar and interaction is a collection of software model checking benchmarks in Horn format. These benchmarks have now been added to the Software Verification Competition – SVCOMP repository and are available from <https://svn.sosy-lab.org/software/sv-benchmarks/trunk/clauses/>. The benchmarks include checking safety assertions and are coming from the following sources:

- Boolean programs extracted from the SDV/SLAM research distribution;
- C programs from SVCOMP 2013, provided by Arie Gurfinkel;
- numeric programs from the ARMC model checker, provided by Andrey Rybalchenko;
- heap manipulating programs from the SLayer tool, provided by Jael Kriener;
- driver programs, provided by Ken McMillan;
- Geometry design problems, provided by Zachary Kincaid;
- Liquid type checking problems from OCaml and Haskell, provided by Ranjit Jhala;
- Benchmarks from the Eldarica tool, provided by Hossein Hojjat, Philipp Rümmer, and Viktor Kuncak.

All together we collected around 10000 benchmarks published in the Horn format. The examples use domains ranging from Booleans, linear real arithmetic, linear integer arithmetic, and linear integer arithmetic combined with arrays. The benchmarks come from many different sources and problem domains, but share the characteristics that the queries amount to checking satisfiability of Horn clauses modulo theories. They are in the SMT-LIB interchange format, which is standardized on <http://smtlib.org>.

Participants

- Francesco Alberti
University of Lugano, CH
- Dirk Beyer
Universität Passau, DE
- Nikolaj Bjørner
Microsoft Res. – Redmond, US
- Tomas Brazdil
Masaryk University, CZ
- Krishnendu Chatterjee
IST Austria –
Klosterneuburg, AT
- Swarat Chaudhuri
Rice University, US
- Laurent Doyen
CNRS, ENS – Cachan, FR
- Uwe Egly
TU Wien, AT
- Azadeh Farzan
University of Toronto, CA
- Bernd Finkbeiner
Universität des Saarlandes, DE
- Alberto Griggio
Fondazione Bruno Kessler –
Trento, IT
- Aarti Gupta
NEC Laboratories America, Inc.
– Princeton, US
- Ashutosh Kumar Gupta
IST Austria –
Klosterneuburg, AT
- Arie Gurfinkel
CMU – Pittsburgh, US
- Jochen Hoenicke
Universität Freiburg, DE
- Radu Iosif
VERIMAG – Gières, FR
- Moa Johansson
Chalmers UT – Göteborg, SE
- Igor Konnov
TU Wien, AT
- Konstantin Korovin
University of Manchester, GB
- Laura Kovacs
TU Wien, AT
- Axel Legay
INRIA – Rennes, FR
- Rupak Majumdar
MPI for Software Systems –
Kaiserslautern, DE
- Anca Muscholl
Université Bordeaux, FR
- Albert Oliveras
TU of Catalonia – Barcelona, ES
- Joel Ouaknine
University of Oxford, GB
- David Parker
University of Birmingham, GB
- Brigitte Pientka
McGill Univ. – Montreal, CA
- Ruzica Piskac
MPI für Softwaresysteme –
Saarbrücken, DE
- Albert Rubio
UPC – Barcelona, ES
- Andrey Rybalchenko
TU München, DE
- Helmut Seidl
TU München, DE
- Martina Seidl
University of Linz, AT
- Natasha Sharygina
University of Lugano, CH
- Ana Sokolova
Universität Salzburg, AT
- Armando Solar-Lezama
MIT – Cambridge, US
- Eugenia Ternovska
Simon Fraser University –
Burnaby, CA
- Helmut Veith
TU Wien, AT
- Tomas Vojnar
Brno Univ. of Technology, CZ
- Andrei Voronkov
University of Manchester, GB
- Georg Weissenbacher
TU Wien, AT
- Josef Widder
TU Wien, AT
- Thomas Wies
New York University, US
- Florian Zuleger
TU Wien, AT

