09411 Abstracts Collection
Interaction versus Automation: The two Faces of Deduction
— Dagstuhl Seminar —

Thomas Ball[1], Jürgen Giesl[2], Reiner Hähnle[3] and Tobias Nipkow[4]

[1] Microsoft Corp. - Redmond, USA
tball@microsoft.com
[2] RWTH Aachen, D
giesl@informatik.rwth-aachen.de
[3] Chalmers UT - Göteborg, S
[4] TU München, D
nipkow@in.tum.de

Abstract. From 04.10. to 09.10.2009, the Dagstuhl Seminar 09411 "Interaction versus Automation: The two Faces of Deduction" was held in Schloss Dagstuhl – Leibniz Center for Informatics. During the seminar, several participants presented their current research, and ongoing work and open problems were discussed. Abstracts of the presentations given during the seminar as well as abstracts of seminar results and ideas are put together in this paper. The first section describes the seminar topics and goals in general. Links to extended abstracts or full papers are provided, if available.

Keywords. Formal Logic, Deduction, Artificial Intelligence

09411 Executive Summary – Interaction versus Automation: The two Faces of Deduction

This seminar was the ninth in the series of the Dagstuhl "Deduction" seminars held biennially since 1993. Its goal was to bring together the closely related but unnecessarily disjoint communities of researchers working in interactive and automatic program verification.

Keywords: Formal Logic, Deduction, Artificial Intelligence

Joint work of: Ball, Thomas; Giesl, Jürgen; Hähnle, Reiner; Nipkow, Tobias

Extended Abstract: http://drops.dagstuhl.de/opus/volltexte/2010/2421

LTL over Description Logic Axioms

Franz Baader (TU Dresden, DE)

Most of the research on temporalized Description Logics (DLs) has concentrated on the most general case where temporal operators can occur both within DL concepts and in front of DL axioms. In this setting, reasoning usually becomes quite hard. If rigid roles (i.e., roles whose interpretation does not vary over time) are allowed, then the interesting inference problems (such as satisfiability of concepts) become undecidable. Even if all symbols are interpreted as flexible (i.e., their interpretations can change arbitrarily from one time-point to the next), the complexity of reasoning is doubly exponential, i.e., one exponential higher than the complexity of reasoning in pure DLs such as $\mathcal{ALC}$. In this paper, we consider the case where temporal operators are allowed to occur only in front of axioms (i.e., ABox assertions and general concept inclusion axioms (GCIs)), but not inside concepts. As the temporal component, we use linear temporal logic (LTL) and in the DL component we consider $\mathcal{ALC}$. We show that reasoning becomes simpler in this setting.

Keywords: Description Logic, temporal logic, reasoning, complexity

Full Paper:
http://lat.inf.tu-dresden.de/research/papers/2008/BaaGhiLu-KR08.pdf

See also: Franz Baader, Silvio Ghilardi, and Carsten Lutz. LTL over Description Logic Axioms. In Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning (KR2008), 2008.

The SMT "Big Bang": Applications of Z3 in Microsoft

Thomas Ball (Microsoft Research - Redmond, US)

SMT solvers have been the focus of increased recent attention thanks to technological advances and an increasing number of applications. The Z3 solver from Microsoft Research is noteworthy both concerning technological advances and applications. I will describe several of the applications of Z3, such Pex (Program EXploration) for generating test cases for .NET binaries, SAGE for security testing, Static Driver Verifier version 2.0 (shipped with the Windows 7 Driver Development Kit), PREfix (enchanced with bit-precise reasoning to find integer overflows), SpecExplorer from the Protocol Test Team for generating test inputs from model-based test suites, as well as the HAVOC and VCC verifiers for C code.

Model Evolution with Built-In Theories

Peter Baumgartner (NICTA - Canberra, AU)

Many applications of automated deduction require reasoning modulo some form of integer arithmetic or other theories. Unfortunately, theory reasoning support in current approaches is sometimes too weak for practical purposes. This problem has been widely recognised and is currently a hot topic.

In previous work, we have shown how to add linear integer arithmetic reasoning to the Model Evolution calculus. The drawback of this approach is that, although complete, it builds on assumptions that make it difficult to apply it to other theories than linear integer arithmetic. In the talk we will present a new approach that solves this problem, is conceptually simpler and easier to implement.

Keywords: Theorem proving, arithmetic, SMT-solver

Joint work of: Baumgartner, Peter; deMoura, Leonardo; Tinelli, Cesare

Proof Assistant vs. Extended Static Checking: The Two Faces of Program Verification

Bernhard Beckert (Universität Koblenz-Landau, DE)

This talk compares two paradigms/philosophies of program verification: using a proof assistant and using extended static checking. We discuss their differences, similarities, and what they can learn from each other.

Generalized Efficient Array Decision Procedures

Nikolaj Bjorner (Microsoft Research - Redmond, US)

The theory of arrays is ubiquitous in the context of automatic software and hardware verification and symbolic analysis. The basic array theory was introduced by McCarthy and allows to symbolically representing array updates.

To this date the theory of arrays itself remains fundamental to how modern program verification, test-case generation, and model-based program tools model program heaps and higher level data-types, such as sets and finite maps.

We present combinatory array logic, CAL, using a small, but powerful core of combinators, and reduce it to the theory of uninterpreted functions.

CAL allows expressing properties that go well beyond the basic array theory. CAL does not allow expressing the identity function I. CAL+I on the other hand allows encoding arbitrary lambda terms. We provide a new efficient decision procedure for the base theory as well as CAL. The efficient procedure serves a critical role in the performance of the state-of-the-art SMT solver Z3 on array formulas from applications.

Keywords: Decision Procedures, Automation

Joint work of: Bjorner, Nikolaj; de Moura; Leonardo

Full Paper:
http://research.microsoft.com/apps/pubs/?id=102329

Decision procedures with unsound inferences for software verification

Maria Paola Bonacina (Università degli Studi di Verona, IT)

Applications in software verification often require determining the satisfiability of first-order formulae, including quantifiers, with respect to some background theories. Superposition-based inference systems are strong at reasoning with equality, universally quantified variables, and Horn clauses. Satisfiability modulo theories (SMT) solvers are strong at reasoning with propositional logic, including non-Horn clauses, ground equalities and integrated theories such as linear arithmetic.

This talk presents an approach to combine these complementary strengths by integrating the superposition-based inference system in the SMT-solver.

Since during software development conjectures are usually false, it is desirable that the theorem prover terminates on satisfiable instances. In the integrated approach termination can be enforced by introducing additional axioms in such a way that the system detects and recovers from any ensuing unsoundness.

Joint work with Chris Lynch and Leonardo de Moura

Keywords: Satisfiability modulo theories; superposition-based theorem proving; model-based combination of theories

Joint work of: Bonacina, Maria Paola; Lynch, Christopher A.; de Moura Leonardo

Full Paper:
http://profs.sci.univr.it/∼bonacina/papers/CADE2009dpllSPutp.pdf

See also: Maria Paola Bonacina, Christopher A. Lynch and Leonardo de Moura. On deciding satisfiability by DPLL(Gamma+T) and unsound theorem proving. In Renate Schmidt (Ed.) Proceedings of the Twenty-Second International Conference on Automated Deduction (CADE), Montreal, Canada, August 2009. Springer, Lecture Notes in Artificial Intelligence 5663, 35-50, 2009.

Inductive Theorem Proving meets Dependency Pairs

Carsten Fuhs (RWTH Aachen, DE)

Current techniques and tools for automated termination analysis of term rewrite systems (TRSs) are already very powerful. However, they fail for algorithms whose termination is essentially due to an inductive argument.

Therefore, we show how to couple the dependency pair method for TRS termination with inductive theorem proving. As confirmed by the implementation of our new approach in the tool AProVE, now TRS termination techniques are also successful on this important class of algorithms.

Keywords: Termination, Term Rewriting, Dependency Pairs, Inductive Theorem Proving

Joint work of: Swiderski, Stephan; Parting, Michael; Giesl, Jürgen; Fuhs, Carsten; Schneider-Kamp, Peter

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2010/2422

Embedding Automated Deduction in a Question Answering System

Ulrich Furbach (Universität Koblenz-Landau, DE)

The deduction system E-KRHyper is used as the main inference machine in the question answering system Loganswer.

This talk focuses on aspects of interaction of other system components with E-KRHyper. In particular we will discuss how machine learning and deduction are co-operating in the task of handling a knowledge base of formulae, which represents 12 million sentences.

Another aspect of embedding the prover, is the query relaxation mechanism, which is applied, whenever the deduction component does not yield an immediate answer.

This is joint work with Björn Pelzer and Ingo Glöckner.

Termination of Integer Term Rewriting

Juergen Giesl (RWTH Aachen, DE)

Recently, techniques and tools from term rewriting have been successfully applied to prove termination automatically for different programming languages.

The advantage of rewrite techniques is that they are very powerful for algorithms on user-defined data structures.

But in contrast to techniques for termination analysis of imperative programs, the drawback of rewrite techniques is that they do not support data structures like integer numbers which are pre-defined in almost all programming languages.

To solve this problem, we extend term rewriting by built-in integers and adapt the dependency pair framework to prove termination of integer term rewriting automatically. Our experiments show that this indeed combines the power of rewrite techniques on user-defined data types with a powerful treatment of predefined integers.

Keywords: Termination analysis, integers, term rewriting, dependency pairs

Joint work of: Fuhs, Carsten; Giesl, Jürgen; Plücker, Martin; Schneider-Kamp, Peter; Falke, Stephan

Extended Abstract: http://drops.dagstuhl.de/opus/volltexte/2010/2423

Formalized Automation

Georges Gonthier (Microsoft Research UK - Cambridge, GB)

Formal proofs can increase the robustness and adaptability of programs by providing a better understanding of how and on which data they operate. This rationale should apply to programs that build proofs, though it is often trumped by economy. However, we will show that higher-order logic offers a variety of tools that help support the formalization of proof techniques, ranging from dependent types and internal evaluation to type inference with value reconstruction. Combined, these can capture a set of proof techniques rich enough to support our work on the four-color theorem and on the odd order theorem.

Symbolic Execution and Partial Evaluation

Reiner Haehnle (Chalmers UT - Göteborg, SE)

Partial evaluation is a program specialisation technique that allows to optimize programs for which partial input is known. We show that partial evaluation can be used with advantage to speed up as well symbolic execution of programs. Interestingly, the input required for partial evaluation comes from symbolic execution itself which makes it natural to interleave partial evaluation and symbolic execution steps in a software verification setup.

Keywords: Symbolic execution, partial evaluation, software verification

Joint work of: Haehnle, Reiner; Bubel, Richard

Inching towards the completion of the Flyspeck Project

Thomas C. Hales (University of Pittsburgh, US)

Flyspeck is a large-scale proof formalization project. Its purpose is to give a formal proof of the Kepler conjecture on sphere packings. The project is now entering into its final stages. This talk will discuss the work of several people who have made significant recent contributions to the project. It will map out the final stages of this project.

Keywords: Proof assistant, formal proof

Comparing Unification Algorithms in First-Order Theorem Proving

Krystof Hoder (University of Manchester, GB)

Unification is one of the key procedures in first-order theorem provers.

Most first-order theorem provers use the Robinson unification algorithm.

Although its complexity is in the worst case exponential, the algorithm is easy to implement and examples on which it may show exponential behaviour are believed to be atypical. More sophisticated algorithms, such as the Martelli and Montanari algorithm, offer polynomial complexity but are harder to implement.

Very little is known about the practical perfomance of unification algorithms in theorem provers: previous case studies have been conducted on small numbers of artificially chosen problem and compared term-to-term unification while the best theorem provers perform set-of-terms-to-term unification using term indexing.

To evaluate the performance of unification in the context of term indexing, we made large-scale experiments over the TPTP library containing thousands of problems using the COMPIT methodology. Our results confirm that the Robinson algorithm is the most efficient one in practice.

They also reveal main sources of inefficiency in other algorithms.

We present these results and discuss various modification of unification algorithms.

Keywords: Unification, term indexing, first-order, theorem proving

Joint work of: Hoder, Krystof; Voronkov, Andrei

Full Paper:
http://www.springerlink.com/content/r168818720158m78/

See also: K. Hoder and A. Voronkov. Comparing unification algorithms in first-order theorem proving. In M. Hund B. Mertsching and Z. Aziz, editors, KI 2009: Advances in Artificial Intelligence, Proceedings of the 32nd German Conference on Artificial Intelligence, LNAI 5803. Springer, 2009.

Software Verification using Liquid Types

Ranjit Jhala (University of California - San Diego, US)

ABSTRACT: We present Liquid Types, a new static program verification technique which combines the complementary strengths of automated deduction (SMT solvers), model checking (Predicate Abstraction), and type systems (Hindley-Milner inference).

We show how liquid types can be used to statically verify a variety of invariants like array-bound-safety, sortedness, balancedness, binary-search-ordering,

variable ordering, set-implementation, heap-implementation, and acyclicity of data structure libraries for list-sorting, union-find, splay trees, AVL trees, red-black trees, heaps, associative maps, extensible vectors, and binary decision diagrams. (Joint work with Patrick Rondon and Ming Kawaguchi)

Keywords: Refinement Types, Predicate Abstraction, SMT, Hindley-Milner

Full Paper:
http://pho.ucsd.edu/liquid/

Inductive Decidability Revisited

Deepak Kapur (University of New Mexico - Albuquerque, US)

In a CADE paper in 2000, Kapur and Subramaniam proposed a framework for deciding a subclass of inductive conjectures by placing restrictions on the definitions of recursive functions on freely-generated recursive data structures. The subclass of inductive conjectures that could be handled in that paper was subsequently extended by Kapur and Giesl presented at (IJCAR 2001) and (CADE 2003). However, the enlarged subclass still did not include any nonlinear conjectures or did not allow recursive definitions which could use some auxiliary functions. In this talk, those conditions are relaxed, thus deciding validity of nonlinear conjectures. Recursive definitions can also use other auxiliary functions defined elsewhere, or defined in a mutually recursive style. Definitions can use quantifier-free Presburger constraints as conditions. Previous work was done using the cover set induction method, a heuristic for implementing the explicit induction approach. This talk will illustrate how the framework also works for the socalled inductionless induction or implicit induction approach. A decision procedure for the enlarged class of conjectures is given. Its implementation demonstrates that syntactic checks on definitions and conjectures take much less time compared to the time spent on attempting proofs of conjectures in the enlarged class.

Keywords: Mechanization of induction, decision procedures, inductive conjectures

Joint work of: Falke, Stephan; Kapur, Deepak

Deductive Verification of Multi-threaded Java Programs by Symbolic Execution

Vladimir Klebanov (Universität Koblenz-Landau, DE)

We present an approach for deductive verification of multi-threaded Java programs. For this we have defined a Dynamic Logic and implemented a verification calculus in the KeY system. The calculus is based on symbolic execution, proves full functional properties and can handle unbounded multi-threading. We report on the arising deduction issues.

Finding Loop Invariants Using a Theorem Prover

Laura Kovacs (ETH Zürich, CH)

This talk presents how quantified loop invariants of programs over arrays can be automatically inferred using a first order theorem prover, reducing the burden of annotating loops with complete invariants.

Our approach allows one to generate first-order invariants containing alternations of quantifiers.

For doing so, we deploy symbolic computation methods to generate numeric invariants of the scalar loop variables, based on the software package Aligator, and then use update predicates of the loop. An update predicate for an array A expresses updates made to A. We observe that many properties of update predicates can be extracted automatically from the loop description and loop properties obtained by other methods such as a simple analysis of counters occurring in the loop, recurrence solving and quantifier elimination over loop variables. The first-order information extracted from the loop description can use auxiliary symbols, such as symbols denoting update predicates or loop counters. After having collected the first-order information, we run the saturation theorem prover Vampire to eliminate the auxiliary symbols and obtain loop invariants expressed as first-order formulas. When the invariants obtained in this way contain skolem functions, we de-skolemise them into formulas with quantifier alternations.

Our method does not require the user to give a post-condition, a predefined collection of predicates or any other form of human guidance and avoids inductive reasoning.

This is a joint work with Andrei Voronkov (University of Manchester, UK).

Keywords: Program verification, invariant generation, sumbolic computation, theorem proving

Full Paper:
http://www.springerlink.com/content/h4208k235552777q/

See also: @inproceedingsKovacsV09, author = L. Kovács and A. Voronkov, title = Finding Loop Invariants for Programs over Arrays Using a Theorem Prover, booktitle = FASE, year = 2009, pages = 470-485

Deciding Function Images and Recursive Functions over Data Types by BAPA Reduction

Viktor Kuncak (EPFL - Lausanne, CH)

I describe decision procedures for logics that supports sets, multisets, cardinality operator, as well as function and relation images. I also describe a parameterized decision procedure for classes of recursively defined functions on algebraic data types.

The presented procedures reduce the satisfiability problem to the satisfiability in BAPA (Boolean Algebra with Presburger Arithmetic). I outline a framework for BAPA reductions and show that, in addition to these new examples, some existing expressive logics such as weak monadic second-order logic of two successors and two-variable logic with counting fit into the framework. This allows us to combine expressive logics even when their signatures share set algebra operations.

Keywords: Decision procedure, set algebra, cardinality, algebraic data types, Tarskian set constraints

Simplex algorithm and formal proofs

Assia Mahboubi (Ecole Polytechnique - Palaiseau, FR)

Joint work with Pierre-Yves Strub (INRIA Rocquencourt - Tsinghua University, Beijing)

The simplex algorithm is both a standard method in optimization and the basis of decision methods for linear arithmetics, implemented for instance in various SMT tools. We propose to discuss our formalization inside the Coq system the theory of the simplex algorithm, ie. the correctness proof of the optimization algorithm. We will also present the computational counter-part of this work, eventually leading to efficient proof-producing decision procedures.

Keywords: Coq, decision procedures, linear arithmetic, formal proofs

Joint work of: Mahboubi, Assia; Strub, Pierre-Yves

Combination of automatic and interactive proving in program verification

Claude Marche (INRIA - Orsay, FR)

The Why platform (http://why.lri.fr) is a set of tools for deductive verification of Java and C source code. In both cases, the requirements are specified as annotations in the source, in a special style of comments. It is based on a unique, stand-alone, verification condition generator called Why, which is able to output goals in the native syntax of many provers, either automatic or interactive ones.

In this talk I will show the importance of combining several provers to perform program verification.

Keywords: Program verification, combination of provers

Decreasing Diagrams and Relative Termination

Aart Middeldorp (Universität Innsbruck, AT)

In this talk, based on joint work with Nao Hirokawa, we present a new confluence result for left-linear term rewrite systems based on critical pairs and relative termination. The proof is based on the decreasing diagrams technique. We further show how to encode the rule-labeling heuristic for decreasing diagrams as a satisfiability problem. We give experimental data for both methods.

Keywords: Confluence, automation, decreasing diagrams, relative termination

Sledgehammer: Judgment Day

Tobias Nipkow (TU München, DE)

Sledghammer is Isabelle's proof method that calls external first order provers E, Spass and Vampire. How well does it work? We present an empirical evaluation of Sledgehammer using existing Isabelle theories.

Joint work of: Nipkow, Tobias; Böhme, Sascha

Verifying C code in a microkernel

Michael Norrish (NICTA - Canberra, AU)

This paper presents the formal Isabelle/HOL framework we use to prove refinement between an executable, monadic specification and the C implementation of the seL4 microkernel. We describe the refinement framework itself, the automated tactics it supports, and the connection to our previous C verification framework. We also report on our experience in applying the framework to seL4. The characteristics of this microkernel verification are the size of the target (8,700 lines of C code), the treatment of low-level programming constructs, the focus on high performance, and the large subset of the C programming language addressed, which includes pointer arithmetic and type-unsafe code.

Keywords: Verification, interactive theorem-proving, micro-kernels

Full Paper:
http://nicta.com.au/__data/assets/pdf_file/0018/20961/tphols2009.pdf

See also: Simon Winwood, Gerwin Klein, Thomas Sewell, June Andronick, David Cock and Michael Norrish. Mind the Gap: A Verification Framework for Low-Level C. Proceedings of TPHOLs, Munich, 2009. LNCS 5674, pp500Ű515.

Beluga: programming with dependent types and higher-order data

Brigitte Pientka (McGill University - Montreal, CA)

The logical framework LF supports specifying formal systems and proofs about them using a simple, powerful technique, namely higher-order abstract syntax (HOAS). Recently, it has been for example successfully used to specify and verify guarantees about the run-time behavior of mobile code. However, incorporating logical framework technology into functional programming to directly allow programmers to describe and reason about properties of programs from within the programming language itself has been a major challenge.

In this talk, I will present Beluga, a dependently-typed functional language which supports programming with data specified in the logical framework LF. First, I will show how to implement normalization for typed lambda-terms in Beluga, and highlight its theoretical foundation based on contextual types. Second, I will discuss practical challenges regarding type reconstruction for dependently typed systems and summarize the status of our implementation.

Keywords: Logical frameworks, dependent types, type theory

Automated Deduction for Hybrid Systems

Andre Platzer (Carnegie Mellon University - Pittsburgh, US)

Hybrid systems are models for complex physical systems and are defined as dynamical systems with interacting discrete transitions and continuous evolutions along differential equations. They arise frequently in many application domains, including aviation, automotive, railway, and robotics. As a theoretical and practical foundation for deductive verification of hybrid systems, we present a dynamic logic for hybrid systems: differential dynamic logic. Our verification approach for this logic is a compositional proof calculus combining results from symbolic logic, real algebraic geometry, differential algebra, and computer algebra. Our main result proves that this calculus axiomatizes the transition behavior of hybrid systems completely relative to differential equations. Our approach is implemented in the hybrid theorem prover KeYmaera for hybrid systems that supports both automatic and interactive verification of hybrid systems. With KeYmaera, we have verified several interesting case studies from ground and air transportation.

See `http://symbolaris.com/info/KeYmaera.html`

Keywords: Dynamic logic, differential equations, sequent calculus, axiomatisation, automated theorem proving, verification of hybrid systems

Full Paper:
http://symbolaris.com/pub/freedL.pdf

See also: André Platzer. Differential dynamic logic for hybrid systems. Journal of Automated Reasoning, 41(2), pages 143-189, 2008. See also André Platzer. Differential-algebraic dynamic logic for differential-algebraic programs. Journal of Logic and Computation, 2008

Static and Precise Detection of Concurrency Errors in Systems Code Using SMT Solvers

Zvonimir Rakamaric (University of British Columbia - Vancouver, CA)

Context-bounded analysis is an attractive approach to verification of concurrent programs. Bounding the number of contexts executed per thread not only reduces the asymptotic complexity, but also the complexity increases gradually from checking a purely sequential program.

Previous work provided a method for reducing the context-bounded verification of a concurrent boolean program to the verification of a sequential boolean program, thereby allowing sequential reasoning to be employed for verifying concurrent programs. In this work, we adapt the encoding to work for systems programs written in C with the heap and accompanying low-level operations such as pointer arithmetic and casts. Our approach is completely automatic: we use a verification condition generator and SMT solvers, instead of a boolean model checker, in order to avoid manual extraction of boolean programs and false alarms introduced by the abstraction. We evaluate our tool STORM on a set of real-world Windows device drivers.

Keywords: Software verification, software checking, concurrent programs, SMT solvers

Full Paper:
http://www.zvonimir.info/publications/cav2009-lqr.pdf

See also: S. Lahiri, S. Qadeer, Z. Rakamaric, "Static and Precise Detection of Concurrency Errors in Systems Code Using SMT Solvers", Proceedings of the 21st International Conference on Computer Aided Verification (CAV 2009), Lecture Notes in Computer Science, Springer, Vol. 5643, 2009, pp 509-524

Automated Synthesis of Tableau Calculi

Renate Schmidt (University of Manchester, GB)

We present a method for synthesising sound, complete and terminating tableau calculi. Given a well-defined specification of the formal semantics of a logic, the method generates a set of tableau inference rules which can then be used to reason within the logic. The method guarantees that the generated rules form a calculus, which is sound and constructively complete. If the logic can be shown

to admit finite filtration then adding a general blocking mechanism produces a terminating tableau calculus. The process of generating tableau rules can be completely automated and produces together with the blocking mechanism an automated procedure for generating tableau decision procedures for logics. The framework is intended to be as general as possible and covers a large class of logics including well-known description and modal logics. For illustration we show the workability of the approach on propositional intuitionistic logic.

Keywords: Synthesis, tableaux, decidability, blocking

Full Paper:
http://dx.doi.org/10.1007/978-3-642-02716-1_23

The Dependency Triple Framework for Termination Analysis of Logic Programs

Peter Schneider-Kamp (Univ. of Southern Denmark - Odense, DK)

There are two major approaches to termination analysis of logic programs (LPs).

In the global approach one tries to find one global ranking function, that proves termination of the given LP, while in the local approach, different ranking functions can be used for different loops of the LP. Some automated techniques in the global approach are based on a constraint-based framework to search for a suitable ranking function. In the local approach, most techniques use a given small set of functions which is either fixed or provided by the user.

We present a new local approach, the dependeny triple framework, which is both modular on the level of loops and allows for constraint-based automated generation of ranking functions. With this new framework, one can combine arbitrary termination techniques for LPs (even including transformations from LPs to term rewrite systems) in a completely modular way.

Keywords: Logic programming, termination analysis, ranking functions

Joint work of: Schneider-Kamp, Peter; Giesl, Jürgen; Nguyen, Manh Thang

Automated reasoning in extensions of theories of constructors with recursively defined functions and homomorphisms

Viorica Sofronie-Stokkermans (MPI für Informatik - Saarbrücken, DE)

We study possibilities of reasoning about extensions of base theories with functions which satisfy certain recursion and homomorphism properties. Our focus is on emphasizing possibilities of hierarchical and modular reasoning in such extensions and combinations thereof.

(1) We show that the theory of absolutely free constructors is local, and locality is preserved also in the presence of selectors. These results are consistent with existing decision procedures for this theory (e.g. by Oppen).
(2) We show that, under certain assumptions, extensions of the theory of absolutely free constructors with functions satisfying a certain type of recursion axioms satisfy locality properties, and show that for functions with values in an ordered domain we can combine recursive definitions with boundedness axioms without sacrificing locality.
    We also address the problem of only considering models whose data part is the *initial* term algebra of such theories.
(3) We analyze conditions which ensure that similar results can be obtained if we relax some assumptions about the absolute freeness of the underlying theory of data types, and illustrate the ideas on an example from cryptography.

The locality results we establish allow us to reduce the task of reasoning about the class of recursive functions we consider to reasoning in the underlying theory of data structures (possibly combined with the theories associated with the co-domains of the recursive functions).

As a by-product, the methods we use provide a possibility of presenting in a different light (and in a different form) locality phenomena studied in cryptography; we believe that they will allow to better separate rewriting from proving, and thus to give simpler proofs.

Keywords: Decision procedures, recursive datatypes, recursive functions, homomorphisms, verification, cryptography

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2010/2424

Interactive Verification of Application Specific Security Protocols

Kurt Stenzel (Universität Augsburg, DE)

Formal verification can give more confidence in the security of cryptographic protocols. Application specific security properties like “The service provider does not loose money” can give even more confidence than standard properties like secrecy or authentication.

This yields more complex security properties that are proven using interactive verification. The verification of this kind of application-specific property is part of the SecureMDD approach, a model-driven development method for secure smart card applications.

However, it is surprisingly easy to get a meaningful property slightly wrong. The result is that an insecure protocol can be ‘proven’ secure.

We illustrate the problem with a very small example, and propose a solution that incorporates more of the real-world application into the formal model.

Keywords: Interactive verification, security protocols

Full Paper:
http://www.informatik.uni-augsburg.de/lehrstuehle/swt/se/projects/secureMDD

See also: Formal Verification of Application-Specific Security Properties in a Model-Driven Approach; Nina Moebius, Kurt Stenzel, Wolfgang Reif; Proceedings of ESSoS 2010 - International Symposium on Engineering Secure Software and Systems; Springer LNCS 2010; Pitfalls in Formal Reasoning about Security Protocols; Nina Moebius, Kurt Stenzel, Wolfgang Reif; Proceedings of ARES 2010 - The Fifth International Conference on Availability, Reliability and Security; IEEE Press 2010;

Dependently Typed Programming with Mutable State

Aaron Stump (University of Iowa - Iowa City, US)

The Guru verified-programming language combines a dependently typed pure functional programming language with a logic for machine-checked proofs about its programs. Imperative abstractions like mutable arrays or aliased linked data structures are supported by specifying a pure functional model, to use for proving properties of programs using the abstractions; and an imperative implementation in C, which the compiler inserts during compilation in place of the pure functional model. Linear types help ensure that the functional model's behavior matches the imperative implementation's. This talk will survey some imperative abstractions implemented using this approach.

Keywords: Dependent types, aliasing, program verification

Integrating SAT and SMT into the Coq proof assistant

Laurent Thery (INRIA - Sophia Antipolis, FR)

In this talk, we are going to report on our ongoing effort to integrate SAT/SMT technologies inside Coq.

Certifying Termination Proofs: Interaction and Automation

Rene Thiemann (Universität Innsbruck, AT)

Current termination analyzer for term rewrite systems are powerful tools for generating termination proofs in a fully automatic way.

However, sometimes the resulting proofs are incorrect due to some bug in one of these analyzers. A human inspection of the proofs is often not feasible, since the resulting proofs reach sizes of several megabytes.

To solve this problem, we show how one can use an interactive theorem prover to develop a fully automatic certified algorithm which can check the correctness of proofs from a termination analyzer.

We also illustrate some similarities and differences between termination analyzers and the certification algorithm.

Our implementation IsaFoR / CeTA supports several termination techniques, including dependency pairs, usable rules, the subterm-criterion, semantic labeling, size-change termination, and matrix interpretations. This variety allows us to certify a large class of proofs that are found by current termination analyzers.

Keywords: Termination, certification, theorem proving, term rewrite systems

Joint work of: Thiemann, René; Sternagel, Christian; Krauss, Alexander; Winkler, Sarah; Zankl, Harald

Full Paper:
http://cl-informatik.uibk.ac.at/software/ceta/

Interaction vs. Automation: The Automated Theorem Proving View or A Kind of User's Guide to Automatic Provers

Christoph Weidenbach (MPI für Informatik - Saarbrücken, DE)

Although SAT provers have seen impressive progress in previous years their behavior is not "robust" in various aspects: the formulation of a problem matters, small changes to formulas may drastically change the behaviour of an afterwards applied prover, no single prover/algorithm dominates all others on all problems etc. This is not surprising because SAT is a hard problem.

If SAT is not robust in the above sense then this applies as well to provers for more expressive logics such as SMT or first-order logic. Therefore, it is not suprising that a black box usage of such provers does not always lead to satisfactory results. In such cases it may make a lot of sense to think of an overall tool chain starting from the selection of a problem over its formalization down to the eventual proof tasks already in terms of the eventually used automated prover.

I illustrate the above arguments by a bunch of examples.

On prover interaction and integration with Isabelle/Scala

Makarius Wenzel (TU München, DE)

After several decades, most proof assistants are still centered around tty-bound interaction in a tight read-eval-print loop.

Even well-known Emacs modes for such provers follow this synchronous model based on single commands. There have also been some attempts at re-implementing prover interfaces in big IDE frameworks, while keeping the old interaction model. Can we do better than that?

Already 10 years ago, the Isabelle/Isar proof language has emphasized the idea of "proof document" (structured text) instead of "proof script" (sequence of commands), although the implementation was still emulating tty interaction in order to be able to work with the existing Proof General interface. After some recent reworking of Isabelle internals, in order to support parallel processing of theories and proofs, the original idea of structured document processing has surfaced again.

Isabelle2009 already provides some support for interactive proof documents with asynchronous / parallel checking, which awaits to be connected to a suitable editor framework (or "IDE"). The remaining problem is how to do that systematically, without having to specify and implement complex protocols or middle-ware for prover integration.

This is the point where we introduce the new Isabelle/Scala layer, which is meant to expose certain aspects of Isabelle/ML to the outside world. The Scala language (by Martin Odersky) is sufficiently close to ML in order to model well-known prover concepts conveniently, but Scala also runs on the JVM and can access existing Java libraries without requiring additional glue code. By building more and more external system wrapping for Isabelle in Scala, we shall eventually reach the point where we can integrate the prover seamlessly into existing IDEs (say Netbeans). Although current experiments are focused on relatively simple editor functionality in jEdit, the emerging Isabelle/Scala library is targeting general tool integration.

For example, an Isabelle/Isar proof method might want to call into the Scala/JVM world to invoke some ATP or model finder that happens to be available there. Or some external tools might want to invoke Isabelle as a "service", by interacting directly with a stylized Scala API, instead of generating adhoc text files.

Two automation challenges

Freek Wiedijk (Radboud University Nijmegen, NL)

We present two challenges for the automation of mathematics.

The first challenge is to automate the mathematical reasoning steps that in a non-formalized mathematical textbook are just explained with the word "therefore". The second challenge is to automate mathematics at the high school level. We propose a project to combine these challenges and make them specific.

Keywords: Proof assistants, decision procedures, computer algebra