Report of Dagstuhl Seminar 15471

Symbolic Computation and Satisfiability Checking

Edited by

Erika Ábrahám[1], Pascal Fontaine[2], Thomas Sturm[3], and Dongming Wang[4]

1 **RWTH Aachen, DE,** abraham@cs.rwth-aachen.de
2 **LORIA – Nancy, FR,** pascal.fontaine@inria.fr
3 **MPI für Informatik – Saarbrücken, DE,** sturm@mpi-inf.mpg.de
4 **Beihang University – Beijing, CN,** dongming.wang@lip6.fr

Abstract

The seminar focused on satisfiability checking for combinations of first-order logic and subclasses thereof with arithmetic theories in a very liberal sense, also covering quantifiers and parameters. It gathered members of the two communities of symbolic computation (or computer algebra) and satisfiability checking (including satisfiability modulo theories). Up-to-now, these two communities have been working quite independently. We are confident that the seminar will initiate cross-fertilization of both fields and bring improvements for both satisfiability checking and symbolic computation, and for their applications.

Seminar November 15–20, 2015 – http://www.dagstuhl.de/15471
Keywords and phrases algorithmic algebra, arithmetic, automated reasoning, decision procedures, quantifier elimination, satisfiability checking, SMT solving, symbolic computation
Digital Object Identifier 10.4230/DagRep.5.11.71
Edited in cooperation with Tim King

1 Executive Summary

Erika Ábrahám
Pascal Fontaine
Thomas Sturm
Dongming Wang

The seminar focused on satisfiability checking for combinations of first-order logic and subclasses thereof with arithmetic theories in a very liberal sense, also covering quantifiers and parameters.

The development of decision procedures for corresponding theories started in the early 20th century in the area of mathematical logic. In the second half of the 20th century it played a prominent role within the development of algebraic model theory. Finally, around 1970, one important research line, viz. algebraic decision methods for real arithmetics, shifted its focus from theoretical results towards practically feasible procedures. That research line was one of the origins of an area known today as *symbolic computation* or *computer algebra*.

More recently, the *satisfiability checking* community, which originated from propositional SAT solving and which is surprisingly disconnected from symbolic computation, began to develop highly interesting results with a particular focus on existential decision problems, following the track of SAT solving towards industrial applications. Powerful *satisfiability*

modulo theories (SMT) solvers were developed, which enrich propositional SAT solving with components for different theories. We understand satisfiability checking in a broad sense, covering besides SMT solving also *theorem proving* with arithmetic.

The two communities of *symbolic computation* and *satisfiability checking* have been quite disjoint, despite strong reasons for them to discuss together. The communities share interests, e.g., examining arithmetic expressions, that are central to both. As a matter of fact, the symbolic computation community has been mostly unaware of basic insights in the satisfiability checking community, such as the efficiency of conflict-driven search with learning, as well as of their fundamental requirements, e.g., incrementality or explanations in the unsatisfiable case. Vice versa, researchers in satisfiability checking have adopted decision procedures from symbolic computation, such as CAD for real closed field, only quite naively, so that they do not really benefit from the considerable experience gained by the original community during 45 years. It is our hope that our seminar contribute to bringing the two communities together, and that they will be much stronger at tackling problems that currently defeat them both, separately.

The seminar offered its participants an opportunity to exchange knowledge about existing methods and applications, to push forward the communication of needs and interests, and to draw attention to challenging open research questions. The participants included researchers from all relevant research areas and with affiliations in academia and as well as in industry. The program was a balanced combination of presentations and tutorials, but also offering time for small group discussions and exchange of ideas.

To the best of our knowledge, the seminar was the first global meeting of the two communities of symbolic computation and satisfiability checking. We are confident that it will initiate cross-fertilization of both fields and bring improvements for both satisfiability checking and symbolic computation, and for their applications.

2 Table of Contents

3 Overview of Talks

3.1 Quick Intro to CoCoA/CoCoALib

John Abbott (Universität Kassel, DE)

Joint work of Anna Maria Bigatti
URL http://cocoa.dima.unige.it/

The CoCoA software suite implements many algorithms in the realm of Computational Commutative Algebra (especially ideals in multivariate polynomial rings). These implementations are accessible both via a user-friendly interactive system (called CoCoA-5), and more directly as functions in an open-source (GPL3) C++ library (called CoCoALib). There is also a prototype CoCoAServer which uses an OpenMath-like language.

Great emphasis has been placed on making the C++ library easy to use for mathematicians without requiring that they learn advanced features of the C++ language (though such features are used "invisibly" inside the CoCoALib implementations). CoCoALib comes with extensive documentation including around 100 illustrative example programs. CoCoALib has also been interfaced with a number of other specialized C++ libraries (including Normaliz, Frobby and GFan); the design of CoCoALib deliberately aims to accommodate such "collaborations".

To help understand how simple it is to use CoCoALib, we illustrate how to derive Heron's Formula for the area of a triangle, firstly using the CoCoA-5 interactive system (just 8 lines of program code), then using CoCoALib (just 9 lines of C++ beyond the standard "boilerplate" code). This particular example includes the computation of a Groebner basis.

For those with little experience of symbolic computation, we point out some of the strengths and weaknesses which seem most relevant to SMT computations.

An interesting strength is the ability to factorize polynomials. It is even not too costly to find all irreducible factors over the rationals (e.g. CoCoA takes about 1 millisecond for a degree 10 polynomial), so a strategy of "speculative factorization" might be worth considering. There are algorithms for factorizing over algebraic fields, but these are more costly.

Using symbolic computation we can compute exactly in algebraic extensions: numbers are represented as symbolic expressions such as $12\sqrt{2} - 17$, so with this approach there are no problems of "loss of precision". In contrast, arithmetic with such symbolic expressions is relatively costly (especially if the extension degree is high).

To help relate an algebraic number to the real world, symbolic computation also includes techniques for computing guaranteed approximations to solutions of systems of polynomial equations – the easiest case is just a single polynomial. Currently CoCoA can compute approximations only to real roots; it can handle polynomials of degree up to 100 in just a few seconds. Very close approximations can also be obtained relatively quickly.

One important point to note is that there is often a large difference in computation time between a result guaranteed to be absolutely correct, and a result which is correct with very high probability (e.g. 1-epsilon where epsilon is less than 10^{-15}): a typical example is testing a number for primality. Truly guaranteed results would require certified source code, a certified compiler, and certified hardware; with this viewpoint perhaps a probably correct result is more acceptable?

CoCoA also offers "heuristically guaranteed" floating-point arithmetic (called Twin-Floats). This can allow faster computation with real algebraic numbers, but the correctness of the final result is only "probable" – however greater probability incurs greater computational

cost. Note that, if asked, CoCoA can recognize when a twin-float represents a simple rational number; more generally it can also find the simplest rational number in a given real interval.

References

1 J. Abbott, A. Bigatti. CoCoALib: A C++ Library for Computations in Commutative Algebra ... and Beyond. In *Proceedings ICMS 2010*, pages 73–76, 2010.
2 J. Abbott, A. Bigatti. What Is New in CoCoA? In *Proceedings ICMS 2014*, pages 352–358, 2014.
3 J. Abbott, A. M. Bigatti, C. Soeger. Integration of Libnormaliz in CoCoALib and CoCoA-5. In *Proceedings ICMS 2014*, pages 647–653, 2014.

3.2 Building Bridges between Symbolic Computation and Satisfiability Checking

Erika Ábrahám (RWTH Aachen, DE)

Main reference E. Ábrahám, "Building Bridges between Symbolic Computation and Satisfiability Checking," in Proc. of the 2015 ACM on Int'l Symp. on Symbolic and Algebraic Computation (ISSAC'15), pp. 1–6, ACM, 2015.
URL http://dx.doi.org/10.1145/2755996.2756636

The satisfiability problem is the problem of deciding whether a logical formula is satisfiable. For first-order arithmetic theories, in the early 20th century some novel solutions in form of decision procedures were developed in the area of mathematical logic. With the advent of powerful computer architectures, a new research line started to develop practically feasible implementations of such decision procedures. Since then, symbolic computation has grown to an extremely successful scientific area, supporting all kinds of scientific computing by efficient computer algebra systems.

Independently, around 1960 a new technology called SAT solving started its career. Restricted to propositional logic, SAT solvers showed to be very efficient when employed by formal methods for verification. It did not take long till the power of SAT solving for Boolean problems had been extended to cover also different theories. Nowadays, fast SAT-modulo-theories (SMT) solvers are available also for arithmetic problems.

Due to their different roots, symbolic computation and SMT solving tackle the satisfiability problem differently. In this talk we illustrated SMT solving techniques to introduce them to the Symbolic Computation Community, discussed differences and similarities in their approaches, highlighted potentials of combining their strengths, and showed up the challenges that come with this task.

3.3 Open Non-uniform Cylindrical Algebraic Decomposition

Christopher W. Brown (U.S. Naval Academy – Annapolis, US)

Computation with real polynomial equalities and inequalities is an area that has already seen productive interaction between the SMT and computer algebra communities. Many presentations throughout the week highlighted this. For example, Viorica Sofronie-Stokkermans'

presentation described how computer algebra software for such computations were used as part of an SMT solving approach to problems in hybrid and reactive systems. This talk concerns interaction the other way, with ideas from the SMT community leading to new results in computer algebra, specifically new results in computing with real polynomial equalities and inequalities. In another presentation, Dejan Jovanovic described joint work with Leonardo de Moura that adapted Cylindrical Algebraic Decomposition (CAD) in a novel way as part of an SMT-style SAT solver for real polynomial equalities and inequalities. One of the key insights of that work was that in following an SMT-style approach, one incrementally builds a model (in this case a point in real space), and that model can be used to optimize certain aspects of CAD construction. This talk describes Non-uniform Cylindrical Algebraic Decomposition (NuCAD), a new kind of CAD that is inspired by their work. It takes the idea of the "model point" and how it can optimize certain aspects of CAD construction, and uses it to to construct NuCADs. As the talk describes, NuCADs can be constructed particularly efficiently, and they can represent solutions of sets of polynomial equalities and inequalities with far fewer cells than CADs.

3.4 Computer Algebra for SAT People

James H. Davenport (University of Bath, GB)

Joint work of R. J. Bradford, M. England, S. McCallum, D. J. Wilson
Main reference M. England, R. J. Bradford, J. H. Davenport, "Improving the Use of Equational Constraints in Cylindrical Algebraic Decomposition," in Proc. of the 2015 ACM Int'l Symp. on Symbolic and Algebraic Computation (ISSAC'15), pp. 165–172, ACM, 2015.
URL http://dx.doi.org/10.1145/2755996.2756678

A brief introduction to computer algebra, focusing on basic tools and on cylindrical algebraic decomposition.

Complexity of dense polynomial arithmetic is well-understood, sparse (and more realistic) polynomial arithmetic less so [4]. Despite the theoretical advances, factoring is still best avoided, and implementations try to: replacing "irreducible" by "square-free and relatively prime" where possible

Cylindrical algebraic decomposition has its roots in [3] who first defined them and gave the first algorithm. Since then, many improvements on this ([5] for the latest), and alternative algorithms base don Regular Chains [2] or Comprehensive Gröbner Bases [6]. The complexity is doubly-exponential in the number of variables, and [1] shows this is inherent, but also that some examples might have this complexity for one variable order, and trivial for a different order.

References

1 C. W. Brown and J. H. Davenport. The Complexity of Quantifier Elimination and Cylindrical Algebraic Decomposition. In C. W. Brown, editor, *Proceedings ISSAC 2007*, pages 54–60, 2007.
2 C. Chen and M. Moreno Maza. An Incremental Algorithm for Computing Cylindrical Algebraic Decompositions. In R. Feng, W.-S. Lee, and Y. Sato, editors, *Proceedings Computer Mathematics ASCM 2009 and 2012*, pages 199–221, 2014.
3 G. E. Collins. Quantifier Elimination for Real Closed Fields by Cylindrical Algebraic Decomposition. In *Proceedings 2nd. GI Conference Automata Theory & Formal Languages*, pages 134–183, 1975.

4 J. H. Davenport and J. Carette. The Sparsity Challenges. In S. Watt *et al.*, editor, *Proceedings SYNASC 2009*, pages 3–7, 2010.
5 M. England, R. Bradford, and J. H. Davenport. Improving the Use of Equational Constraints in Cylindrical Algebraic Decomposition. In D. Robertz, editor, *Proceedings ISSAC 2015*, pages 165–172, 2015.
6 R. Fukasaku, H. Iwane, and Y. Sato. Real Quantifier Elimination by Computation of Comprehensive Gröbner Systems. In D. Robertz, editor, *Proceedings ISSAC 2015*, pages 173–180, 2015.

3.5 Introduction to Floating-Point Arithmetic

James H. Davenport (University of Bath, GB)

This was a brief introduction to floating-point arithmetic (restricted to IEEE [4], where numbers are $\pm m2^e$ and, generally, $m \in [1,2)$ is a 53-bit number and $-1022 \leq e \leq 1023$, so the largest finite number is $> 10^{308}$) from an algebraic point of view. If $\cdot$ is one of the arithmetic operators $\{+,-,\times,/\}$ and $\odot$ the floating-point equivalent, then $a \odot b$ is defined to be $[a \cdot b]$ where $[\cdots]$ denotes the nearest representable number (with precise rules for tie-breaking etc.). Therefore $\oplus$ and $\otimes$ are commutative, and $a \ominus b = \ominus(b \ominus a)$. Admittedly $a \oslash b \neq 1 \oslash (b \oslash a)$ and $a \otimes (b \otimes c) \neq (a \otimes b) \otimes c$, but the differences are trivial (at least for "sensibly-sized" numbers). What, then, are the problems? This list is not exhaustive, and the first three, at least, would hold for any floating-point system.

1. Computing other functions, e.g. $\odot$ when $\cdot \in \{\sin, \log, \ldots\}$, is much harder, as in general we have no idea how many extra places we need to compute with internally. This is a problem known as the "Table Maker's Dilemma" [3]. If we relax $[\cdots]$ to denotes the nearest representable number or its neighbor, then the problem is much more tractable for computation, but much messier for verification, as well as being non-deterministic.
2. $\oplus$ is wildly non-associative: $(1 \oplus 10^{20}) \oplus (-10^{20}) = 0$ but $1 \oplus (10^{20} \oplus (-10^{20})) = 1$. Slightly more subtly, $(5 \oplus 2^{53}) \oplus (-2^{53}) = 4$.

⋆ *Some* languages have rules about what re-arrangements are permissible, and *some* compilers adhere to these, at least with *some* compiler options [5].

3. "sensibly-sized" is far easier to prescribe than to define precisely, or check for.
4. Operations that we would generally think of as "errors" such as $10^{200} \otimes 10^{200}$, generate either representations of $\pm\infty$ ($e = 1024$, $m = 0$), or, as in $\infty - \infty$, "Not a Number" ($e = 1024$, $m \neq 0$) by default, rather than causing an error.

⋆ The speaker recalled seeing a supermarket where the "unit prices", instead of being, say `69 p/litre` were all `NaN p/litre`.

5. As we get very close to 0, there are concepts of "denormalised numbers" and "gradual underflow", which give results with less precision than usual, rather than just 0, which are generally good for the numerical results but hard to model formally (bit-blasting?).
6. Every number is signed, including zero. While this has theoretical advantages when treating complex functions and branch cuts [2], and means that $1 \oslash (1 \oslash x) = x$ when $x = \pm\infty$, it can complicate other areas: $xy \Rightarrow 1/x = 1 \oslash y$is not true when $x = +0$ and $y = -0$.

See, e.g., [3] in general, and [1] about verification.

References

1 M. Brain, C. Tinelli, P. Rümmer, and T. Wahl. An Automatable Formal Semantics for IEEE-754 Floating-Point Arithmetic. http://smtlib.cs.uiowa.edu/papers/BTRW14.pdf, 2014.
2 J. H. Davenport, R. Bradford, M. England, and D. Wilson. Program Verification in the presence of complex numbers, functions with branch cuts etc. In *Proceedings SYNASC 2012*, pages 83–88, 2012.
3 D. Goldberg. What Every Computer Scientist Should Know About Floating-Point Arithmetic. *ACM Comp. Surveys*, 23:5–48, 1991.
4 IEEE. IEEE Standard for Floating-Point Arithmetic (754-2008). *IEEE*, 2008.
5 Intel (Martyn Corden). Consistency of Floating-Point Results using the Intel® Compiler. http://software.intel.com/en-us/articles/consistency-of-floating-point-results-using-the-intel-compiler/, 2012.

3.6 Speed Maple

Jürgen Gerhard (Maplesoft – Waterloo, CA)

Main reference L. Bernardin, P. Chin, P. DeMarco, K. O. Geddes, D. E. G. Hare, K. M. Heal, G. Labahn, J. P. May, L. McCarron, M. B. Monagan, D. Ohashi, S. M. Vorkoetter, "Maple Programming Guide," Maplesoft, 2015.
URL http://www.maplesoft.com/documentation_center

A brief introduction to the internals of Maple's mathematical engine was given. Maple's architecture, the available data types, and the algorithms implemented for polynomial and semi-algebraic systems, and beyond, were summarized.

3.7 Adapting Real Quantifier Elimination Methods for Conflict Set Computation

Maximilian Jaroschek (MPI für Informatik – Saarbrücken, DE), Pascal Fontaine (LORIA – Nancy, FR), and Pablo Federico Dobal (LORIA – Nancy, FR)

Main reference M. Jaroschek, P. F. Dobal, P. Fontaine, "Adapting Real Quantifier Elimination Methods for Conflict Set Computation," in Proc. of the 10th Int'l Symp. on Frontiers of Combining Systems (FroCos'15), LNCS, Vol. 9322, pp. 151–166, Springer, 2015.
URL http://dx.doi.org/10.1007/978-3-319-24246-0_10

The satisfiability problem in real closed fields is decidable. In the context of satisfiability modulo theories, the problem restricted to conjunctive sets of literals, that is, sets of polynomial constraints, is of particular importance. One of the central problems is the computation of good explanations of the unsatisfiability of such sets, i.e. obtaining a small subset of the input constraints whose conjunction is already unsatisfiable. We adapt two commonly used real quantifier elimination methods, cylindrical algebraic decomposition and virtual substitution, to provide such conflict sets and demonstrate the performance of our method in practice.

3.8 Turning CAD Upside Down

Dejan Jovanovic (SRI – Menlo Park, US)

We present the NLSAT decision procedure for solving the existential fragment of non-linear arithmetic. The classic approach to the problem is to project the problem polynomials, followed by model construction (lifting). The new procedure performs the lifting optimistically, and performs focused model-based projection only on the polynomials that are relevant in inconsistencies. The new approach, and the leaner projection operator, are effective in practice, which we support with an extensive experimental evaluation of the implementations in Yices2 and Z3 SMT solvers.

3.9 Constructing a Single CAD Cell

Marek Kosta (MPI für Informatik – Saarbrücken, DE) and Christopher W. Brown (U.S. Naval Academy – Annapolis, US)

Main reference C. W. Brown, M. Kosta, "Constructing a single cell in cylindrical algebraic decomposition," Journal of Symbolic Computation, Vol. 70, pp. 14–48, 2015.
URL http://dx.doi.org/10.1016/j.jsc.2014.09.024

We present an algorithm which, given a point and a set of polynomials, constructs a single cylindrical cell containing the point, such that the polynomials are sign-invariant in the computed cell. To represent a single cylindrical cell, a novel recursive data structure is introduced. The algorithm works with the data structure and proceeds by incrementally merging the input polynomials into the cell. A merge procedure realizing this refinement is described. The merge procedure is based on McCallum's operator, but uses geometric information relative to the test point to reduce the projection set. The use of McCallum's operator implies the incompleteness of our algorithm in general. However, the algorithm is complete for well-oriented sets of polynomials. Moreover, the incremental approach described can be easily adapted to a different projection operator. Our cell construction is an alternative to the "model-based" method described by D. Jovanović during this seminar.

3.10 Symbol Elimination for Program Analysis

Laura Kovács (Chalmers UT – Göteborg, SE)

Joint work of Laura Kovács and Andrei Voronkov
Main reference L. Kovács, A. Voronkov, "Finding Loop Invariants for Programs over Arrays Using a Theorem Prover," in Proc. of the 12th Int'l Conf. on Fundamental Approaches to Software Engineering (FASE'09), LNCS, Vol. 5503, pp. 470–485, Springer, 2009.
URL http://dx.doi.org/10.1007/978-3-642-00593-0_33

I describe our new symbol elimination method [1, 2] for generating and proving properties about software systems. Symbol elimination uses first-order theorem proving and symbolic computation techniques to automatically discover non-trivial program properties, such as

loop invariants and loop bounds. Moreover, symbol elimination can be used as an alternative to interpolation for software verification. The talk will describe how symbol elimination can be used for polynomial and quantified invariant generation, by using Groebner basis computation, quantifier elimination and saturation-based first-order theorem proving. Symbol elimination is implemented in the award-winning first-order theorem prover Vampire and successfully evaluated on a large number of examples coming from academic and industrial benchmarks.

References

1 Laura Kovács and Andrei Voronkov. *Finding Loop Invariants for Programs over Arrays Using a Theorem Prover*. FASE 2009: 470-485.
2 Laura Kovács and Andrei Voronkov. *Interpolation and Symbol Elimination*. CADE 2009: 199-213.

3.11 SMT-RAT: An Open Source C++ Toolbox for Strategic and Parallel SMT Solving

Gereon Kremer (RWTH Aachen, DE) and Florian Corzilius (RWTH Aachen, DE)

Main reference F. Corzilius, G. Kremer, S. Junges, S. Schupp, E. Ábrahám, "SMT-RAT: An Open Source C++ Toolbox for Strategic and Parallel SMT Solving," in Proc. of the 18th Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT'15), LNCS, Vol. 9340, pp. 360–368, Springer, 2015.
URL http://dx.doi.org/10.1007/978-3-319-24318-4_26

During the last decade, popular SMT solvers have been extended step-by-step with a wide range of decision procedures for different theories. Some SMT solvers also support the user-defined tuning and combination of such procedures, typically via command-line options. However, configuring solvers this way is a tedious task with restricted options.

In this paper we present our modular and extensible C++ library SMT-RAT, which offers numerous parameterized procedure modules for different logics. These modules can be configured and combined into an SMT solver using a comprehensible whilst powerful strategy, which can be specified via a graphical user interface. This makes it easier to construct a solver which is tuned for a specific set of problem instances. Compared to a previous version, we have extended our library with a number of new modules and support for parallelization in strategies. An additional contribution is our thread-safe and generic C++ library CArL, offering efficient data structures and basic operations for real arithmetic, which can be used for the fast implementation of new theory-solving procedures.

3.12 An Invitation to Numerical Algebraic Geometry

Viktor Levandovskyy (RWTH Aachen, DE)

Joint work of Levandovskyy, Viktor; Hauenstein, Jonathan
Main reference J. D. Hauenstein, V. Levandovskyy, "Certifying solutions to square systems of polynomial-exponential equations," arXiv:1109.4547v1 [math.NA], 2011; to appear in Journal of Symbolic Computation.
URL http://arxiv.org/abs/1109.4547v1

In this ad-hoc contributed talk we present some basic facts from the emerging theory called "Numerical Algebraic Geometry". In particular, the results from the Smale's alpha-theory will be explained. There is an algorithm, allowing one to certify a solution of a system of nonlinear polynomials equations. Moreover, for some number of close candidate solutions, it is possible either to distinguish their associated solutions or to conclude that there is one associated solution with multiplicity greater than one. Since very recently there evolve more and more advanced algorithms, based on these two fundamental ones. These algorithms are often implemented in a freely available package "alphaCertified" and use a freely available system "bertini". A live demonstration of the former is presented as well.

3.13 A Presentation of Satisfiability Modulo Theory for Nonspecialists

David Monniaux (VERIMAG – Grenoble, FR)

Satisfiability modulo theory (SMT) extends propositional satisfiability (SAT) by allowing arithmetic predicates, or more generally predicates within a theory. $a \wedge (b \vee \neg c)$ is a SAT formula whose solutions are $(a = \text{true}, b = \text{true}, c = \text{true})$, $(a = \text{true}, b = \text{true}, c = \text{false})$, $(a = \text{true}, b = \text{false}, c = \text{false})$. $(x > 0) \wedge (y > 0) \wedge (x + y < 1)$ is an SMT formula over linear real arithmetic (LRA) whose solutions include e.g. $(x = \frac{1}{4}, y = \frac{1}{4})$, but the same formula admits no solution over linear integer arithmetic (LIA). $f(x) \neq f(y) \wedge x = y$, where f stands for an unspecified ("uninterpreted") function (UF) from integers to integers, has no solution since if $x = y$, for any function f, $f(x) = f(y)$. Examples of SMT theories include LRA, LIA, UF, and combinations thereof. SMT solvers may allow quantifiers inside formulas, though these quickly make the problem undecidable (e.g. UF+LIA+$\forall$ is undecidable).

The most common way of implementing SMT is by the "DPLL(T)" framework, in which a SAT solver based on the CDCL (constraint-driven clause learning) principle (a modern evolution of Davis-Putnam-Logemann-Loveland) interacts with a decision procedure for conjunctions of predicates from the theory. The formula is stripped to its propositional structure, e.g. $(x > 0) \wedge (y > 0) \wedge (x + y < 1 \vee y < 0)$ is replaced by $a \wedge b \wedge (c \vee d)$ where a stands for $x > 0$, b for $y > 0$, c for $x + y < 1$ and d for $y < 0$. An assignment $a = \text{true}$, $b = \text{true}$, $c = \text{true}$ is made but the decision procedure for LIA informs the SAT solver than this assignment is inconsistent. A clever decision procedure will strive to inform the SAT solver than a sub-part of the assignment only is inconsistent, since this will rule out a larger part of the propositional state space.

More recently, alternatives to DPLL(T) such as MCSAT or the extension of DPLL to values outside of Booleans have been proposed. They have in common that they reason directly about rational, integer etc. values instead of going through a propositional abstraction.

Satisfiability testing is typically used in program analysis to prove that a finite sequence (including tests and other control flow) of statements is infeasible (in program verification), or to find concrete values going through that sequence (in automated test case generation). The use cases can be more complicated; for instance satisfying assignments may be used to automatically refine the search for inductive invariants in program verification.

Finally, Craig interpolation is another outcome of satisfiability testing. Given $A(x, y)$, $B(y, z)$ and $C(z, t)$, a Craig interpolant is a pair (I, J) such that for all x, y, z, t:

$$A(x, y) \implies I(y), I(y) \wedge B(y, z) \implies J(z), J(z) \wedge C(z, t) \implies \text{false}.$$

Such interpolants are used in program analysis to give "local arguments" why a trace with steps A,B,C is infeasible. Finding simple interpolants likely to generalize to other traces and becoming inductive is an active area of research.

3.14 Triangular Sets over F2 vs. Satisfiability Checking: A Potential Connection and Interaction?

Chenqi Mou (Beihang University – Beijing, CN) and Dongming Wang (Beihang University – Beijing, CN)

In this talk the concepts of triangular sets and triangular decomposition are presented. In particular, a typical splitting strategy in triangular decomposition and then a refined and easier one in Boolean rings are illustrated, with efforts to reveal the similarity between Boolean triangular sets and SAT solving.

References

1 X.-S. Gao and Z. Huang. Characteristic set algorithms for equation solving in finite fields In *Journal of Symbolic Computation, 47(6)*, pages 655–679, 2012.
2 M. Kalkbrener. A generalized Euclidean algorithm for computing triangular representations of algebraic varieties In *Journal of Symbolic Computation, 15(2)*, pages 143–167, 1993.
3 D. Wang. Decomposing polynomial systems into simple systems In *Journal of Symbolic Computation, 25(3)*, pages 295–314, 2000.
4 D. Wang. Computing triangular systems and regular systems In *Journal of Symbolic Computation, 30(2)*, pages 221–236, 2000.
5 W.-T. Wu. On zeros of algebraic equations: An application of Ritt principle In *Kexue Tongbao, 31(1)*, pages 1–5, 1986.
6 L. Yang and J.-Z. Zhang. Searching dependency between algebraic equations: An algorithm applied to automated reasoning In *Artificial Intelligence in Mathematics*, pages 147–156, 1994.

3.15 raSAT: SMT Solver for Nonlinear Arithmetic

Mizuhito Ogawa (JAIST – Ishikawa, JP)

We present the raSAT SMT solver for polynomial constraints, which aims to handle them over both reals and integers with simple unified methodologies: (1) raSAT loop for inequalities, which extends the interval constraint propagation with testing to accelerate SAT detection, and (2) a non-constructive reasoning for equations over reals, based on the generalized intermediate value theorem. raSAT has participated SMT-COMP 2015, and was 3rd (among 6) in QF_NRA and 2nd (among 7) in QF_NIA categories of main tracks.

3.16 Using Instantiation-Based Approaches for Quantifier Elimination in SMT

Andrew Joseph Reynolds (EPFL – Lausanne, CH)

This talk presents instantiation-based decision methods for determining the satisfiability of quantified formulas in first-order theories. Using this framework, we obtain decision procedures for linear real arithmetic (LRA) and linear integer arithmetic (LIA) formulas with one quantifier alternation. Our procedure can be integrated into the solving architecture used by typical SMT solvers. Experimental results on standardized benchmarks from model checking, static analysis, and synthesis show that our implementation of the procedure in the SMT solver CVC4 outperforms existing approaches for quantified linear arithmetic.

3.17 How Far We Can Eliminate Quantified Variables Using Equalities

Yosuke Sato (Tokyo University of Science, JP)

When a given quantified formula contains equalities not only explicitly but also implicitly, we can use them for eliminating quantifies. In order to use all such equalities we need computation of comprehensive Groebner bases. In the talk we introduce our work which gives a sufficiently practical such method.

3.18 Introduction to iSAT3

Karsten Scheibler (Universität Freiburg, DE)

Joint work of Herde, Christian; Kupferschmid, Stefan; Teige, Tino; Eggers, Andreas; Neubauer, Felix; Mahdi, Ahmed

We present iSAT3, a satisfiability checker for Boolean combinations of arithmetic constraints over real- and integer-valued variables. The solver supports constraints containing linear and non-linear arithmetic as well as transcendental functions. iSAT3 tightly integrates

Interval-Constraint-Propagation (ICP) into the Conflict-Driven Clause-Learning (CDCL) framework.

3.19 Hierarchical Reasoning for the Verification of Parametric Systems

Viorica Sofronie-Stokkermans (Universität Koblenz-Landau, DE)

We show how hierarchical reasoning and quantifier elimination can be used to automatically provide guarantees that given parametric systems satisfy certain safety or invariance conditions. Such guarantees can be expressed as constraints on parameters.

We present several examples (from the verification of reactive (or discrete time) systems and of linear hybrid automata) and point out some challenges we encountered when using existing systems for quantifier elimination.

3.20 The Rodin Platform and SMT Solvers

Laurent Voisin (SYSTEREL Aix-en-Provence, FR)

Event-B is a formal notation for modelling discrete transition systems. It is based on first-order predicate calculus with equality completed with typed set theory and integer arithmetics. The system is modelled by a state and its properties as invariants. Transitions are described by guarded events that can fire atomically as soon as their guard holds.

The Rodin platform is the reference tool for modelling with the Event-B notation and proving the model correct. It has been developed for more than ten years and is freely available from http://event-b.org under an Eclipse license. The main design principles of the platform are openness (open source and open syntax), extensibility (more than thirty plug-ins are available) and a reactive modelling process (tools are run automatically in the background).

The interactive prover of the Rodin platform is tactic based. The prover just maintains a proof tree in the sequent calculus. The rules of the proof tree are not predefined, but produced by reasoners (which are invoked by the tactics). The reasoners can be either internal (provided by the core platform) or external (provided by plug-ins). In order to foster reuse after model modification (which always happens), the proof rules shall be as small as possible.

The SMT plug-in works as follows: It first translates an Event-B sequent to its negation in the SMT-LIB format, then invokes an SMT solver. If the solver returns SAT, then the sequent is not valid and the reasoner fails. If the solver returns UNSAT, then the sequent is valid and the UNSAT core provided by the solver is used to build the proof rule.

Two examples are particularly interesting. One is a simple property of a barycenter. It

consists in proving that

$$\begin{array}{ll} & 0 \le N \ \land \ 0 \le a \ \land \ 0 \le b \\ & x \in 0 \mathinner{..} N \ \land \ y \in 0 \mathinner{..} N \\ \vdash & (a * x + b * y) \div (a + b) \in 0 \mathinner{..} N \end{array}$$

No solver currently connected to the Rodin platform can prove this property, as it uses non-linear arithmetic. Consequently, one has to prove it manually by unfolding the definitions of the operators, which is quite tedious.

A second example consists in stating that an acylic relation is non-reflexive:

$$\begin{array}{ll} & r \in S \leftrightarrow S \\ & \forall p \cdot p \subseteq r^{-1}[p] \implies p = \emptyset \\ \vdash & \mathrm{id} \cap r = \emptyset \end{array}$$

Surprisingly, this example is proved by the CVC3 solver, although it needs second order instantiation to find the right instance for p.

3.21 The SMT Theory of Floating-Point Arithmetic

Christoph M. Wintersteiger (Microsoft Research UK – Cambridge, GB)

URL http://smtlib.cs.uiowa.edu/theories-FloatingPoint.shtml

The Satisfiability Modulo Theories (SMT) community recently added a standard for floating-point arithmetic (SMT FP) to its set of theories. This enables many verification techniques that already build on SMT solvers to add support for floating-point arithmetic at a very low cost. In this presentation I summarize and demonstrate the scope of the new theory; it is, to a large extent, based on the IEEE-754 standard, but there are some intricacies in which SMT FP departs from IEEE-754 in an effort to simplify and streamline common problems, and I will point some of those intricacies out. Support for SMT FP in actual SMT solvers exists, but is not commonplace yet; those solvers that support it are either pure translation to Boolean logic (bit-blasters), or they employ abstraction refinement schemes that promise to be more efficient on many problems. At the time, the solvers with support for SMT FP are MathSAT, Sonolar, CVC4, and Z3.

3.22 Stability of Parametric Decomposition

Kazuhiro Yokoyama (Rikkyo University – Tokyo, JP)

Main reference Kazuhiro Yokoyama, "Stability of Parametric Decomposition," in Proc. of the 2nd Int'l Congress on Mathematical Software (ICMS'06), LNCS, Vol. 4151, pp. 391–402, Springer, 2006.
URL http://dx.doi.org/10.1007/11832225_39

We deal with ideals generated by polynomials with parametric coefficients, and introduce "stabilities on ideal structures" based on stability of forms of Gröbner bases. Then, we extend those stabilities to radicals and irreducible decompositions and show the computational tractability on those computations by integrating existing techniques.

Making these computational realizations efficient and practical is still ongoing, but it is very challenging for Computer Algebra not only in theory, but also in application. It will certainly help to develop the abilities of Computer Algebra and widen its application, including Quantifier Elimination and Satisfiability Checking.

4 Panel discussions

4.1 SMT Solvers and Computer Algebra Systems: Potentials of Technology Transfer

Erika Ábrahám (RWTH Aachen, DE)

During and after the talk "Building Bridges between Symbolic Computation and Satisfiability Checking", we discussed different issues of connecting algorithms and implementations rooted in the two communities of Symbolic Computation and Satisfiability Checking.

On the one hand, SMT solving has its strength in efficient techniques for exploring Boolean structures, learning, combining solving techniques, and developing dedicated heuristics, but its current focus lies on easier theories and it makes use of symbolic computation results only in a rather naive way. There are fast SMT solvers available for the satisfiability check of linear real and integer arithmetic problems, but just a few can handle non-linear arithmetic.

On the other hand, Symbolic Computation is strong in providing powerful procedures for sets (conjunctions) of arithmetic constraints, but it does not exploit the achievements in SMT solving for efficiently handling logical fragments, using heuristics and learning to speed-up the search for satisfying solutions.

The SMT-solving community could definitely profit from further exploiting Symbolic Computation achievements and adapt and extend them to comply with the requirements on embedding in the SMT context. However, it is a highly challenging task, as it requires a deep understanding of complex mathematical problems, whose embedding in SMT solving is far from trivial and needs their adaptation and extension. Symmetrically, Symbolic Computation could profit from exploiting successful SMT ideas, but it requires expertise in efficient solver technologies and their implementation, like dedicated data structures, sophisticated heuristics, effective learning techniques, and approaches for incrementality and explanation generation in theory solving modules.

We discussed how we could overcome these problems: on the one hand, how to adapt implementations of decision procedures in computer algebra systems to satisfy the requirements for SMT-embedding, supporting incrementality, the generation of models for satisfiable problems and explanations for unsatisfiable problem instances, and to offer suitable interfaces for SMT calls; on the other hand, how to adapt successful SAT and SMT technologies to improve the efficiency of computer algebra systems, making use of efficient handling of logical structures, learning, and developing dedicated search heuristics.

4.2 SMT-LIB: An Introduction

Pascal Fontaine (LORIA – Nancy, FR)

Joint work of Barrett, Clark; Ranise, Silvio; Stump, Aaron; Tinelli, Cesare
Main reference C. Barrett, P. Fontaine, C. Tinelli, "The SMT-LIB Standard: Version 2.5," 2015.
URL http://smtlib.cs.uiowa.edu/language.shtml
URL http://www.SMT-LIB.org

The Satisfiability Modulo Theories (see [2] for a survey on SMT) community is built around the SMT-LIB initiative (http://smtlib.cs.uiowa.edu/). The aim of this initiative was at first to collect a library of benchmarks. The emergence of the SMT-LIB language quickly followed, as a necessity for exchanging benchmarks and unambiguously interpreting them. This also involved being able to describe precisely the underlying concepts behind SMT, i.e. the theories and their combinations. The SMT-LIB standard is constantly improving, and aims at tackling always richer languages while at the same time keeping the standard simple. Nowadays, SMT-LIB is supported by all the main SMT solvers. It is used as the interface language of many tools (e.g. verification platforms) with their SMT solver backends. It is also the official language of the SMT-COMP (http://www.smtcomp.org/), the annual competition of solvers.

We discussed the SMT-LIB language version 2.5 [1], with a focus on arithmetic theories and logics. The discussion was also the opportunity to compare its features with other languages in use in the Symbolic Computation community. The SMT-LIB language version 2.5 is not extendable but version 3.0 should be more flexible, and could accommodate some of the needs for a language suitable in a larger context.

References

1 Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The SMT-LIB Standard: Version 2.5, 2015. Available at http://www.SMT-LIB.org.

2 Clark Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability modulo theories. In Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, chapter 26, pages 825–885. IOS Press, February 2009.

Participants

- John Abbott
Universität Kassel, DE
- Erika Ábrahám
RWTH Aachen, DE
- Bernd Becker
Universität Freiburg, DE
- Martin Bromberger
MPI für Informatik – Saarbrücken, DE
- Christopher W. Brown
U.S. Naval Academy – Annapolis, US
- Shaowei Cai
Chinese Academy of Sciences – Beijing, CN
- Florian Corzilius
RWTH Aachen, DE
- James H. Davenport
University of Bath, GB
- Pascal Fontaine
LORIA – Nancy, FR
- Stephen Forrest
Maplesoft Europe GmbH, DE
- Jürgen Gerhard
Maplesoft – Waterloo, CA
- Maximilian Jaroschek
MPI für Informatik – Saarbrücken, DE
- Dejan Jovanovic
SRI – Menlo Park, US
- Tim A. King
Google Inc. – Mountain View, US
- Konstantin Korovin
University of Manchester, GB
- Marek Kosta
MPI für Informatik – Saarbrücken, DE
- Laura Kovács
Chalmers UT – Göteborg, SE
- Gereon Kremer
RWTH Aachen, DE
- Wolfgang Küchlin
Universität Tübingen, DE
- Viktor Levandovskyy
RWTH Aachen, DE
- Klaus Meer
BTU Cottbus, DE
- David Monniaux
VERIMAG – Grenoble, FR
- Chenqi Mou
Beihang University – Beijing, CN
- Mizuhito Ogawa
JAIST – Ishikawa, JP
- Andrew Joseph Reynolds
EPFL – Lausanne, CH
- Yosuke Sato
Tokyo University of Science, JP
- Karsten Scheibler
Universität Freiburg, DE
- Tobias Schubert
Universität Freiburg, DE
- Viorica Sofronie-Stokkermans
Universität Koblenz-Landau, DE
- Thomas Sturm
MPI für Informatik – Saarbrücken, DE
- Laurent Voisin
SYSTEREL Aix-en-Provence, FR
- Christoph M. Wintersteiger
Microsoft Research UK – Cambridge, GB
- Patrick Wischnewski
Logic4Business – Saarbrücken, DE
- Kazuhiro Yokoyama
Rikkyo University – Tokyo, JP