Building Bridges between Symbolic Computation and Satisfiability Checking

Erika Ábrahám
RWTH Aachen University
52056 Aachen
Germany
abraham@cs.rwth-aachen.de

ABSTRACT

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. We discuss differences and similarities in their approaches, highlight potentials of combining their strengths, and discuss the challenges that come with this task.

Categories and Subject Descriptors

F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic; G.4 [**Mathematics of Computing**]: Mathematical Software

General Terms

Algorithms, Theory

Keywords

Arithmetic; SMT Solving; Symbolic Computation

ISSAC'15, July 6–9, 2015, Bath, United Kingdom.

DOI: http://dx.doi.org/10.1145/2755996.2756636.

1. INTRODUCTION

Formulas of *first-order logic over arithmetic theories* are logical combinations of linear or polynomial constraints over real- or integer-valued variables. This logic is a powerful modelling formalism frequently used to specify problems from the areas of scheduling, electronic design automation, product design optimisation, planning, controller synthesis, test-case generation, and the analysis of programs and probabilistic, timed, hybrid and cyber-physical systems, just to mention a few well-known examples. Once the problem is formalised, algorithms and their implementations are needed to check the validity or satisfiability of the formulas, and in case they are satisfiable, to identify satisfying solutions. Algorithms to solve this problem are called *decision procedures*.

Symbolic computation. The development of decision procedures for arithmetic theories started already in the early 20th century in mathematical logic. Though the satisfiability problem is known to be NP-hard already for the pure Boolean propositional logic (SAT) and even undecidable for non-linear integer arithmetic, the mathematical developments gave hope for practically applicable fully automated methods to solve arithmetic formulas. Therefore, an enormous effort was put into directing mathematical research towards practice, yielding powerful computer algebra systems in the area of *symbolic computation*.

Satisfiability checking. In the '60s, another line of research on *satisfiability checking* [6] for *propositional logic* started its career. The first idea used *resolution* for quantifier elimination [21], and had serious problems with the explosion of the memory requirements with increasing problem size. Another research line [20] suggested a combination of *enumeration* and *Boolean constraint propagation (BCP)*. A major improvement was achieved in the '90s by *combining* the two approaches, leading to *conflict-driven clause-learning* and *non-chronological backtracking* [37]. Later on, this impressive progress was continued by novel efficient implementation techniques (e.g., sophisticated decision heuristics, two-watched-literal scheme, restarts, cache performance, etc.), resulting in numerous powerful *SAT solvers*. Also different extensions are available, e.g. QBF solvers for quantified Boolean formulas, or Max-SAT solvers to find solutions which satisfy a maximal number of clauses.

Driven by this success, big efforts were made to enrich propositional SAT-solving with solver modules for different theories. Highly interesting techniques were implemented in *SAT-modulo-theories (SMT) solvers* for checking, e.g., equality logic with uninterpreted functions, array theory, bit-vector arithmetic and quantifier-free linear real and inte-

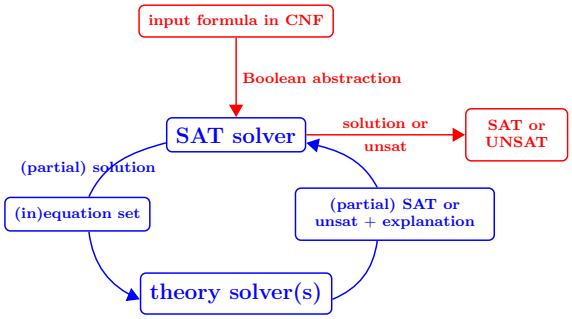

Figure 1: The functioning of SMT solvers

ger arithmetic (LRA/LIA), but the development for quantifier-free non-linear real and integer arithmetic (NRA/NIA) is still in its infancy. For further reading, see, e.g., [3, 35].

The prosperity of research on SAT- and SMT-solving is supported by a wide range of community activities. Examples are own *conferences* on satisfiability checking (the SAT conference is in its 18th edition in 2015), a forum *SatLive* to keep up-to-date with research, the development of *SMT standards*, *SAT solver competitions* since 2002, and *SMT solver competitions* since 2005.

Potentials and challenges. The research areas of SMT solving and symbolic computation are quite disconnected. 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 LRA and LIA problems, but just a few can handle NRA and NIA.

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 would 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. These challenging goals could get within reach, when supported by a stronger collaboration between these research areas, creating an infrastructure for dialogue and knowledge transfer.

2. SMT SOLVING

SMT solvers were originally designed to solve *existentially quantified* (or quantifier-free) formulas. Though there are increasing activities and some SMT solvers are already able to handle quantified formulas of some theories, in the following we restrict ourselves to the solving of existential fragments of first-order logic over different theories.

SMT activities started for theories such as equality logic and uninterpreted functions, bit-vector arithmetic, and array theory. Later, these activities were further extended to (existential fragments of) arithmetic theories. The increasing variety of the theories considered by SMT solvers made an urgent need for a common input language. The `SMT-LIB` initiative [4] defined a *standard input format* for SMT solvers with a first release in 2004, and provides a large and still increasing number of *benchmarks*, systematically collected for all supported theories. `SMT-LIB` also enabled the start of *SMT competitions*; the first one took place in 2005 with 12 participating solvers in 7 divisions (theories, theory combinations, or fragments thereof) on 1360 benchmarks, which increased in 2014 to 20 solvers competing in 32 divisions on 67426 benchmarks.

The SMT-LIB standard and the competitions not only intensified the SMT research activities, but also gave visibility and acceptance for SMT solving in computer science and beyond. Once a problem is formulated in the `SMT-LIB` standard input language, the user can employ *any* SMT solver to solve his or her problem.

Modern *SMT solvers* typically combine a *SAT solver* with one or more *theory solvers* as illustrated in Figure 1. First the input formula is transformed into conjunctive normal form (CNF), a conjunction of disjunctions (clauses); this transformation can be done in linear time and space using Tseitin's transformation on the cost of additional variables. Next, the resulting CNF is abstracted to a pure Boolean propositional logic formula by replacing each theory constraint by a fresh Boolean proposition. Intuitively, the truth value of each fresh proposition defines whether the theory constraint, which it substitutes, holds. The SAT solver tries to find solutions for this propositional abstraction and during solving it consults the theory solver(s) to check the consistency of the theory constraints that should hold according to the current values of the abstraction variables. Whereas *full lazy* approaches search for a complete Boolean solution before invoking theory solvers, *less lazy* techniques consult the theory modules more frequently for partial solution candidates for the Boolean skeleton, and put special requirements on them. On the one hand the theory solvers only need to check *conjunctions (sets)* of theory constraints, instead of arbitrary Boolean combinations. On the other hand, theory solvers should have the following properties for being *SMT-compliant*:

- They should work *incrementally*, i.e., after they determine the consistency of a constraint set, they should be able to take delivery of some additional constraints and re-check the extended set, thereby making use of results from the previous check.
- In case of unsatisfiability, they should be able to return an *explanation* for inconsistency, e.g., by a preferably small inconsistent subset of the constraints.
- They should support *backtracking*, i.e., the removal of previously added constraints.

Optimally, theory solvers should also be able to provide

- a *satisfying solution*, if the problem is satisfiable, and
- a *proof of unsatisfiability* for the explanation, if the problem is unsatisfiable.

A great advantage of the SMT technology is that it can employ decision procedures not only in isolation, but also *in combination*. For example, solving non-linear arithmetic formulas can often be speeded up by first checking linear abstractions or linear problem parts using more efficient decision procedures, before applying heavier procedures. Additionally, theories can also be combined already in the input language of SMT solvers. For example, deductive program verification techniques generate verification conditions, which might refer to arrays, bit-vectors as well as integers; in such cases, dedicated SMT solvers can apply several decision procedures for different theories in combination.

When combining decision procedures, *incomplete* but *efficient* procedures are also valuable, if they guarantee termination but not necessarily return a conclusive answer. Such incomplete (but terminating) methods are frequently applied in SMT solving, a typical example being interval constraint propagation, based on interval arithmetic. Some solvers combine such incomplete methods with complete decision procedures, in order to guarantee the solution of the problem, while increasing efficiency. Other solvers even sacrifice completeness and might return a "don't know" answer, but still they are able to solve certain extremely large problems, which are out of reach for complete methods, very fast. Furthermore, incomplete procedures are the only way to support problems from undecidable theories, like formulas containing exponential or trigonometric functions.

SAT and SMT solvers are tuned for efficiency. Combining complete and incomplete decision procedures, making use of efficient heuristics, learning not only propositional facts but also (Boolean abstractions of) theory lemmas at the SAT level allow modern SMT solvers to solve relevant large-size problems with tens of thousands of variables, which could not be solved before by single decision procedures in isolation. For some example applications see, e.g., [2].

Arithmetic theories in SMT solvers. First extensions of SMT solving to arithmetic theories were designed for LRA and LIA. Some popular decision procedures to solve LRA formulas are the *simplex* algorithm, the *ellipsoid method* [34], the *Fourier-Motzkin variable elimination* algorithm, or the incomplete method of *interval constraint propagation* [30, 31].

The question whether there is a solution for an integer arithmetic formula is in general undecidable. However, its linear fragment is decidable. Some decision procedures for solving LIA are, e.g., the *Omega test* [38], *branch and bound* [26], *cutting planes* [25], *interval constraint propagation*, and *bit-blasting* [15].

Examples for solvers that can cope with LRA/LIA problems (either in a complete or in an incomplete manner) are `Alt-Ergo` [17], `CVC4` [1], `iSAT3` [29, 39], `MathSAT5` [14], `OpenSMT2` [12], `SMT-RAT` [19], `veriT` [9], `Yices2` [28], and `Z3` [24]. A further interesting SMT-approach for LIA is proposed in [10].

Much less activities can be observed for SMT solvers for NRA, which has an exponential time-complexity even for conjunctions of polynomial (in)equalities (in contrast to its linear fragment, in which conjunctions of inequalities are solvable in polynomial time). There are well-known complete as well as incomplete decision procedures for NRA, for example the *cylindrical algebraic decomposition* [16] method, methods based on *Gröbner bases* [41], the *realisation of sign conditions* [5], the *virtual substitution* [40] method, or *interval constraint propagation*.

There are some incomplete SMT-solvers for NRA like, e.g., `iSAT3`, which uses interval constraint propagation. The SMT solver `MiniSmt` [42] tries to reduce NRA problems to linear real arithmetic and can solve only satisfiable instances this way. We are aware of only two SMT solvers that are complete for non-linear real arithmetic: Firstly, the prominent `Z3` solver developed at Microsoft Research, which uses an elegant SMT-adaptation of the cylindrical algebraic decomposition method [32]. Secondly, our own SMT-solver based on our `SMT-RAT` (SMT Real-Arithmetic Toolbox) library [19], which we continuously improve and extend. The current release offers, besides some non-arithmetic components, the `CArL` [13] library for arithmetic datatypes and basic computations with them, and solver modules for simplex, the cylindrical algebraic decomposition [36], the virtual substitution method [18], Gröbner bases [33], interval constraint propagation, branch and bound, and their strategic combination [19].

Even less SMT solvers are available for NIA, which is undecidable in general. A linearisation approach was proposed in [8]. The SMT solving spin-off of `AProVE` [15] uses bit-blasting. To our knowledge, `Z3` implements a combination of linearisation and bit-blasting. `iSAT3` uses interval constraint propagation, whereas `Alt-Ergo` combines the idea of [7] with an axiom-based version of interval constraint propagation. Our `SMT-RAT`-based solver can tackle this theory using a generalised branch-and-bound technique, combined with careful selection of sample points in real-arithmetic procedures.

3. POTENTIALS AND OBSTACLES

Using arithmetic decision procedures in SMT solvers as theory modules is a promising symbiosis: highly efficient SAT solvers can handle the Boolean problem structure and learn from previous (SAT and theory) conflicts, and the expensive theory module calls only concern conjunctions of theory constraints.

The arithmetic decision procedures mentioned above are implemented in different tools. For example, a highly optimised implementation based on the cylindrical algebraic decomposition can be found in `QEPCAD` [11], whereas the `redlog` package [27] of the computer algebra system `Reduce` offers an optimised combination of the cylindrical algebraic decomposition with virtual substitution and Gröbner bases methods. *So why not use those existing implementations for SMT embedding?*

Though it sounds natural, the realisation of this idea is not at all straightforward. Available implementations of decision procedures are seldomly available as *libraries*, and even if they are, they are not SMT compliant. Thus, for SMT embedding, these mathematically complex decision procedures have to be adapted and extended before an SMT compliant implementation can be realised.

The SMT-compliant adaptation of decision procedures has to assure that the algorithms work *incrementally*, and can generate *explanations* for unsatisfiable problems. *Backtracking* is a nice-to-have property, if it can be achieved on low

costs. Adapting decision procedures to be incremental can be tricky, as it requires the storage of additional bookkeeping information, and therefore a deliberate choice of efficient and well-suited data structures. Perhaps the most challenging aspect is the generation of explanations. Also this task entails the costly storage of additional information, where we need to make a careful choice of what and how to store. This leads to a healthy balance between the computational costs in space and time, and the size of the explanations we can extract from the stored information.

For the implementation of adapted algorithms, an *efficient modular library for basic computations with polynomials* is needed, which, if we want to have the door open for parallelisation, must be additionally *thread-safe*. Furthermore, on a given problem instance there might be big differences in the running times of different theory solver modules. Therefore, a *strategic combination* [23] of them should be supported.

What happens in practice is that research groups working on SMT solving do not make use of computer algebra systems, but stepwise adapt and re-implement established procedures from mathematical logic and the symbolic computation community. On the one hand, because active networking and collaboration between symbolic computation and SMT solving is still (surprisingly) quite restricted, much effort goes into re-investigation on the side of the SMT community. On the other hand, the symbolic computation community is not fully aware about the impressive achievements in SMT solving. More common projects would allow to join forces and commonly develop improvements on both sides.

4. SOME MORE CHALLENGES

SMT solvers are originally developed for checking satisfiability of existentially quantified fragments of first-order logic over different theories. However, there are further dimensions in which we could strengthen their power, and thereby increase their applicability and push forward their acceptance in other domains, including industrial applications. Research on the following extensions is interesting not only for the SMT community, but also in the context of symbolic computation.

- For the support for *linear and non-linear (global) optimisation* with respect to a possibly non-linear objective function and an arithmetic formula with an arbitrary logical structure, an interesting procedure using the cylindrical algebraic decomposition method was proposed [22], but currently available tools do not yet have optimisation functionalities.

- Another similarly complex task is the handling of *quantified arithmetic formulas*, which is already challenging for propositional logic. Though some SMT solvers like `Z3` and `CVC4` are able to handle quantified formulas, these techniques are restricted to certain logical fragments and are usually incomplete. Most SMT solvers do not support quantification at all.

- If a problem is detected to be satisfiable, SMT solvers should output *satisfying solutions*. However, solutions for non-linear real arithmetic problems might contain algebraic numbers, which can be hard to represent in a user-friendly way. This gives rise to the question, how we can aim at the generation of *user-friendly models*, at low cost. Furthermore, in some user context, not only one solution is required, but (a finite representation of) all models.

- If a problem is unsatisfiable, it is often helpful for the user to get an *unsatisfiable core*, i.e., a subset of the problem clauses that is already unsatisfiable. For example, if a formula models a realisability property, such an unsatisfiable core could explain, which parts of the system model are conflicting, and thus help to correct design errors. Furthermore, *proofs* for unsatisfiability would make the results more reliable. Last but not least, for large problems *minimal* unsatisfiable cores and proofs have a much stronger impact. Also *interpolant* generation features have great effect to extend the application domains.

- In SAT solving, some attempts were made to develop efficient *parallelisation techniques*. Unfortunately, it turned out that reaching a linear speed-up is hard to achieve. Perhaps this is the reason why parallelisation is not in the focus of the SAT- and SMT-solving communities. However, in the SMT context, parallel solutions have much more potentials. For non-linear arithmetic, even simple portfolio approaches can be very efficient, as the running times of different decision procedures or even just different heuristics might differ significantly: some procedures might succeed in milliseconds, whereas other ones might need hours or even days. Besides portfolio approaches, divide-and-conquer parallelisation inside and between theory modules, involving information exchange, are promising options. Like SAT solvers, also many theory solvers work on a tree-shaped search tree, but in the theory search space, sub-trees represent in general much harder problems. Therefore, it seems likely that additional communication effort would pay off for parallelisation. Currently available SMT parallelisation techniques mainly use the portfolio approach, as for example `CVC4` for linear arithmetic, or `SMT-RAT` for all of its modules.

- Each application has its own characteristics. Some involve symmetries; others are combined from several parts, some of which are static and others dynamically changing; sometimes single independent problems must be solved, sometimes a sequence of related problems appears. There is a wide range of possibilities to tune the solvers to be optimal for a given problem type. The applicability of SMT technology in industry could be increased by identifying typical problem patterns and implementing *dedicated SMT-solvers* for them, tuned for the given type of applications.

5. ACKNOWLEDGEMENTS

I would like to thank many of my colleagues for fruitful discussions, especially Florian Corzilius, Pascal Fontaine, Gereon Kremer, Ulrich Loup, Stefan Schupp, and Thomas Sturm.

6. REFERENCES

[1] C. Barrett, C. L. Conway, M. Deters, L. Hadarean, T. K. D. Jovanović, A. Reynolds, and C. Tinelli.

CVC4. In *Proc. CAV*, volume 6806 of *Lecture Notes in Computer Science*, pages 171–177. Springer, 2011.

[2] C. Barrett, D. Kroening, and T. Melham. Problem solving for the 21st century: Efficient solvers for satisfiability modulo theories. Technical Report 3, London Mathematical Society and Smith Institute for Industrial Mathematics and System Engineering, June 2014. Knowledge Transfer Report.

[3] C. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli. Satisfiability modulo theories. In *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, chapter 26, pages 825–885. IOS Press, 2009.

[4] C. Barrett, A. Stump, and C. Tinelli. The satisfiability modulo theories library (SMT-LIB). `www.SMT-LIB.org`, 2010.

[5] S. Basu, R. Pollack, and M. Roy. *Algorithms in Real Algebraic Geometry*. Springer, 2010.

[6] A. Biere, A. Biere, M. Heule, H. van Maaren, and T. Walsh. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[7] F. Bobot, S. Conchon, E. Contejean, A. Mahboubi, A. Mebsout, and G. Melquiond. A simplex-based extension of Fourier-Motzkin for solving linear integer arithmetic. In *Proc. IJCAR*, volume 7364 of *Lecture Notes in Computer Science*, pages 67–81. Springer, 2012.

[8] C. Borralleras, S. Lucas, R. Navarro-Marset, E. Rodriguez-Carbonell, and A. Rubio. Solving non-linear polynomial arithmetic via SAT modulo linear arithmetic. In *Proc. CADE*, volume 5663 of *Lecture Notes in Computer Science*, pages 294–305. Springer, 2009.

[9] T. Bouton, D. C. B. de Oliveira, D. Déharbe, and P. Fontaine. veriT: An open, trustable and efficient SMT-solver. In *Proc. CADE*, volume 5663 of *Lecture Notes in Computer Science*, pages 151–156. Springer, 2009.

[10] M. Bromberger, T. Sturm, and C. Weidenbach. Linear integer arithmetic revisited. *CoRR*, abs/1503.02948, 2015. Accepted at CADE-25.

[11] C. W. Brown. QEPCAD B: A program for computing with semi-algebraic sets using CADs. *SIGSAM Bulletin*, 37(4):97–108, 2003.

[12] R. Bruttomesso et al. The OpenSMT solver. In *Proc. TACAS*, volume 6015 of *Lecture Notes in Computer Science*, pages 150–153. Springer, 2010.

[13] `CArL`. Project homepage. `http://smtrat.github.io/carl/`.

[14] A. Cimatti, A. Griggio, B. Schaafsma, and R. Sebastiani. The MathSAT5 SMT solver. In *Proc. TACAS*, volume 7795 of *Lecture Notes in Computer Science*, pages 93–107. Springer, 2013.

[15] M. Codish, Y. Fekete, C. Fuhs, J. Giesl, and J. Waldmann. Exotic semi-ring constraints. In *Proc. SMT*, volume 20 of *EPiC Series*, pages 88–97. EasyChair, 2013.

[16] G. E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Automata Theory and Formal Languages*, volume 33 of *Lecture Notes in Computer Science*, pages 134–183. Springer, 1975.

[17] S. Conchon, M. Iguernelala, and A. Mebsout. A collaborative framework for non-linear integer arithmetic reasoning in Alt-Ergo. In *Proc. SYNASC*, pages 161–168. IEEE Computer Society, 2013.

[18] F. Corzilius and E. Ábrahám. Virtual substitution for SMT solving. In *Proc. FCT*, volume 6914 of *Lecture Notes in Computer Science*, pages 360–371. Springer, 2011.

[19] F. Corzilius, U. Loup, S. Junges, and E. Ábrahám. SMT-RAT: An SMT-compliant nonlinear real arithmetic toolbox. In *Proc. SAT*, volume 7317 of *Lecture Notes in Computer Science*, pages 442–448. Springer, 2012.

[20] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

[21] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, July 1960.

[22] L. de Moura and G. O. Passmore. Computation in real closed infinitesimal and transcendental extensions of the rationals. In *Proc. CADE*, volume 7898 of *Lecture Notes in Computer Science*, pages 178–192. Springer, 2013.

[23] L. de Moura and G. O. Passmore. The strategy challenge in SMT solving. In *Automated Reasoning and Mathematics*, pages 15–44. Springer, 2013.

[24] L. M. de Moura and N. Bjørner. Z3: An efficient SMT solver. In *Proc. TACAS*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008.

[25] I. Dillig, T. Dillig, and A. Aiken. Cuts from proofs: A complete and practical technique for solving linear inequalities over integers. In *Proc. CAV*, volume 5643 of *Lecture Notes in Computer Science*, pages 233–247. Springer, 2009.

[26] A. G. Doig, B. H. Land, and A. G. Doig. An automatic method for solving discrete programming problems. *Econometrica*, 28:497–520, 1960.

[27] A. Dolzmann and T. Sturm. REDLOG: Computer algebra meets computer logic. *SIGSAM Bulletin*, 31(2):2–9, 1997.

[28] B. Dutertre and L. de Moura. A fast linear-arithmetic solver for DPLL(T). In *Proc. CAV*, volume 4144 of *Lecture Notes in Computer Science*, pages 81–94. Springer, 2006.

[29] M. Fränzle, C. Herde, T. Teige, S. Ratschan, and T. Schubert. Efficient solving of large non-linear arithmetic constraint systems with complex Boolean structure. *Journal on Satisfiability, Boolean Modeling and Computation*, 1(3-4):209–236, 2007.

[30] S. Gao, M. Ganai, F. Ivančić, A. Gupta, S. Sankaranarayanan, and E. M. Clarke. Integrating ICP and LRA solvers for deciding nonlinear real arithmetic problems. In *Proc. FMCAD*, pages 81–90. IEEE Computer Society, 2010.

[31] S. Herbort and D. Ratz. Improving the efficiency of a nonlinear-system-solver using a componentwise Newton method. Technical Report 2/1997, Inst. für Angewandte Mathematik, University of Karlsruhe, 1997.

[32] D. Jovanović and L. de Moura. Solving non-linear arithmetic. In *Proc. IJCAR*, volume 7364 of *Lecture Notes in Artificial Intelligence*, pages 339–354. Springer, 2012.

[33] S. Junges, U. Loup, F. Corzilius, and E. Ábrahám. On Gröbner bases in the context of satisfiability-modulo-theories solving over the real numbers. In *Proc. CAI*, volume 8080 of *Lecture Notes in Computer Science*, pages 186–198. Springer, 2013.

[34] L. C. Khačiyan. Polynomial algorithm for linear programming. *Soviet Doklady*, 244:1093–1096, 1979. Typed translation.

[35] D. Kroening and O. Strichman. *Decision Procedures: An Algorithmic Point of View*. Springer, 2008.

[36] U. Loup, K. Scheibler, F. Corzilius, E. Ábrahám, and B. Becker. A symbiosis of interval constraint propagation and cylindrical algebraic decomposition. In *Proc. CADE*, volume 7898 of *Lecture Notes in Computer Science*, pages 193–207. Springer, 2013.

[37] J. P. Marques-silva and K. A. Sakallah. Grasp: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48:506–521, 1999.

[38] W. Pugh. The Omega test: A fast and practical integer programming algorithm for dependence analysis. *Communications of the ACM*, 8:4–13, 1992.

[39] K. Scheibler, S. Kupferschmid, and B. Becker. Recent improvements in the SMT solver iSAT. In *Proc. MBMV*, pages 231–241. Institut für Angewandte Mikroelektronik und Datentechnik, Fakultät für Informatik und Elektrotechnik, Universität Rostock, 2013.

[40] V. Weispfenning. Quantifier elimination for real algebra – The quadratic case and beyond. *Applicable Algebra in Engineering, Communication and Computing*, 8(2):85–101, 1997.

[41] V. Weispfenning. A new approach to quantifier elimination for real algebra. In *Quantifier Elimination and Cylindrical Algebraic Decomposition*, Texts and Monographs in Symbolic Computation, pages 376–392. Springer, 1998.

[42] H. Zankl and A. Middeldorp. Satisfiability of non-linear (ir)rational arithmetic. In *Proc. LPAR*, volume 6355 of *Lecture Notes in Artificial Intelligence*, pages 481–500. Springer, 2010.