SC 2 : Satisfiability Checking meets Symbolic Computation (Project Paper)

Eriká Abrahám, John Abbott, Bernd Becker, Anna Bigatti, Martin Brain, Bruno Buchberger, Alessandro Cimatti, James Davenport, Matthew England, Pascal Fontaine, et al.

To cite this version:

Eriká Abrahám, John Abbott, Bernd Becker, Anna Bigatti, Martin Brain, et al.. SC 2 : Satisfiability Checking meets Symbolic Computation (Project Paper). Intelligent Computer Mathematics, Jul 2016, Bialystok, Poland. <http://www.cicm-conference.org/2016/cicm.php>. <hal-01377655>

HAL Id: hal-01377655
https://hal.inria.fr/hal-01377655

Submitted on 7 Oct 2016

SC^2: Satisfiability Checking meets Symbolic Computation (Project Paper)

Erika Ábrahám[1], John Abbott[12], Bernd Becker[2], Anna M. Bigatti[3],
Martin Brain[11], Bruno Buchberger[4], Alessandro Cimatti[5],
James H. Davenport[6], Matthew England[7], Pascal Fontaine[9],
Stephen Forrest[10], Alberto Griggio[5], Daniel Kroening[11],
Werner M. Seiler[12], and Thomas Sturm[8,13]

[1] RWTH Aachen University, Aachen, Germany
[2] Albert-Ludwigs-Universität, Freiburg, Germany
[3] Università degli studi di Genova, Italy
[4] Johannes Kepler Universität, Linz, Austria
[5] Fondazione Bruno Kessler, Trento, Italy
[6] University of Bath, Bath, U.K.
[7] Coventry University, Coventry, U.K.
[8] CNRS, LORIA, Inria, Nancy, France
[9] LORIA, Inria, Université de Lorraine, Nancy, France
[10] Maplesoft Europe Ltd
[11] University of Oxford, Oxford, U.K.
[12] Universität Kassel, Kassel, Germany
[13] Max-Planck-Institut für Informatik, Saarbrücken, Germany

Abstract. *Symbolic Computation* and *Satisfiability Checking* are two research areas, both having their individual scientific focus but sharing also common interests in the development, implementation and application of decision procedures for arithmetic theories. Despite their commonalities, the two communities are rather weakly connected. The aim of our newly accepted SC^2 project (H2020-FETOPEN-CSA) is to strengthen the connection between these communities by creating common platforms, initiating interaction and exchange, identifying common challenges, and developing a common roadmap from theory along the way to tools and (industrial) applications. In this paper we report on the aims and on the first activities of this project, and formalise some relevant challenges for the unified SC^2 community.

Keywords: Logical Problems, Symbolic Computation, Computer Algebra Systems, Satisfiability Checking, Satisfiability Modulo Theories

1 Introduction

The use of advanced methods to solve practical and industrially relevant problems by computers has a long history. While it is customary to think that "computers are getting faster" (and indeed, they were, and are still getting more powerful in terms of multicores etc.), the progress in algorithms and software has been even greater. One of the leaders in the field of linear and mixed integer programming points out [9, slide 37] that you would be over 400 times better off running today's algorithms and software on a 1991 computer than you would running 1991 software on today's computer. The practice is heavily inspired by the theory: [9, slide 31] shows that the biggest version-on-version performance advance in software was caused by "mining the theory". *But* this progress has been in what is, mathematically, quite a limited domain: that of linear programming, possibly where some of the variables are integer-valued.

There has been also much progress in the use of computers to solve hard non-linear algebraic[14] problems. This is the area generally called *Symbolic Computation* (or *Computer Algebra*). It includes solving non-linear problems over both the real and complex numbers, though generally with very different techniques. This has produced many new applications and surprising developments: in an area everyone believed was solved, non-linear solving over the reals (using cylindrical algebraic decomposition — CAD) has recently found a new algorithm for computing square roots [35]. CAD is another area where practice is (sometimes) well ahead of theory: the theory [18, 29] states that the complexity is doubly exponential in the number of variables, but useful problems can still be solved in practice ([3] points out that CAD is the most significant engine in the "Todai robot" project).

Independently and contemporaneously, there has been a lot of practical progress in solving the SAT problem, i.e., checking the satisfiability of logical problems over the Boolean domain. The SAT problem is known to be NP-complete [27]. Nevertheless, the *Satisfiability Checking* [8] community has developed SAT solvers which can successfully handle inputs with millions of Boolean variables. Among other industrial applications, these tools are now at the heart of many techniques for verification and security of computer systems.

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* [6, 42] for checking easier theories, but the development for quantifier-free non-linear real and integer arithmetic[14] is still in its infancy.

Figure 1 shows a non-exhaustive history of tool developments in these two areas. It illustrates nicely the historically deeper roots of computer algebra systems, but also the high intensity of research in both areas. The resulting tools are successfully applied in several academic and industrial areas, however, the current state is still not satisfactory, as described in [51]:

> "*Despite substantial advances in verification technology, complexity issues with classical decision procedures are still a major obstacle for formal verification of real-world applications, e.g., in automotive and avionic industries.*"

[14] It is usual in the SMT community to refer to these constraints as *arithmetic*. But, as they involve quantities as yet unknown, manipulating them is *algebra*. Hence both words occur, with essentially the same meaning, throughout this document.

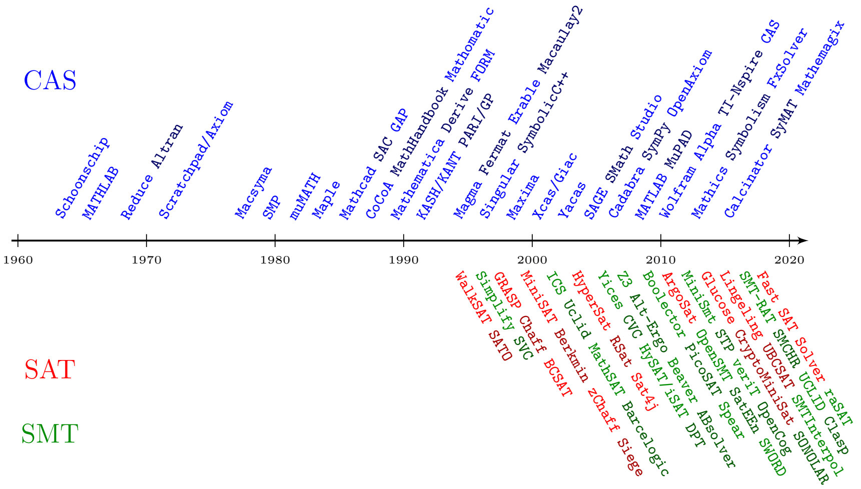

Fig. 1. History of some computer algebra systems and SAT/SMT solvers (not exhaustive; years approximate first release as far as known and as positioning allowed) [2]

Both communities address similar problems and share the challenge to improve their solutions to achieve applicability on complex large-scale applications. However, the Symbolic Computation community and the Satisfiability Checking community are largely in their own silos and traditionally do not interact much with each other.

To connect these communities, we successfully applied for a European Horizon 2020 *Coordination and Support Action*, with an envisaged project start in July 2016. The overall aim of this project is to create a new research community bridging the gap between Satisfiability Checking and Symbolic Computation, whose members will ultimately be well informed about both fields, and thus able to combine the knowledge and techniques of both fields to develop new research and to resolve problems (both academic and industrial) currently beyond the scope of either individual field. We call the new community SC^2, as it will join the communities of **S**atisfiability **C**hecking and **S**ymbolic **C**omputation.

The contributions of this paper are twofold: Firstly, we discuss the potentials of closer connection and more intensive exchange between the two communities, and list a number of challenges that are currently out of reach but could be tackled by a unified SC^2 community (Section 3). Secondly, we discuss what is needed to trigger and support these developments, and describe the actions of our project to satisfy these needs (Section 4).

2 Background

Before describing our project, we give a short description of the state-of-the-art in Satisfiability Checking and Symbolic Computation. Parts of this section are taken from [2].

2.1 Symbolic Computation and Computer Algebra Systems

Computer Algebra, the use of computers to do algebra rather than simply arithmetic, is almost as old as computing itself, with the first PhD theses [41, 50] dating back to 1953. This initial work consisted of programs to do one thing, but the focus soon moved on to 'systems', capable of doing a variety of tasks. One early such system was Collins' `SAC` [24], written in Fortran. Many of the early systems were written in LISP, largely because of its support for recursion, garbage collection and large integers. The group at M.I.T. developed `Macsyma` [45] in the 1960s. about the same time, Hearn developed `Reduce` [38], and shortly after a group at IBM Yorktown Heights produced `SCRATCHPAD`, then `AXIOM` [39], a system that attempted to match the generality of Mathematics with some kind of generic programming, to allow algorithms to be programmed in the generality in which they are conventionally stated, e.g., polynomials over a ring.

Symbolic Computation was initially seen as part of Artificial Intelligence, with major triumphs such as [54] being "A Heuristic Program that Solves Symbolic Integration Problems in Freshman Calculus", firmly in the AI camp. By the end of the 1960s, this approach to integration had been replaced by an algorithm [46], which had the great advantage that, when backed up with a suitable completeness theorem [52] it could *prove* unintegrability: "there is no formula made up of exponentials, logarithms and algebraic functions which differentiates to e^{-x^2}", in other words "e^{-x^2} is unintegrable".

The 1960s and 70s also saw great advances in other areas. We had much more efficient algorithms to replace naive use of Euclid's algorithm for greatest common divisor computation (and hence the simplification of fractions), far better algorithms than the search-based methods for polynomial factorisation, and so on. All this meant that Symbolic Computation firmly moved into the camps of algorithmics and complexity theory, and the dominant question became "what is the worst-case complexity of this algorithm".

Gröbner bases. One great success of this period was the method of *Gröbner bases* [20]. This allows effective, and in many cases efficient, solution of many problems of polynomials over algebraically-closed fields (typically the complex numbers, though applications over finite fields and in cryptography abound). This notion paved the way for the discovery of numerous effective methods for polynomial ideals; many applications in other areas of Mathematics quickly followed. Buchberger's algorithm for computing a Gröbner basis is a prime example of the huge gulf that can separate an abstract algorithm from a usably efficient implementation. Over the fifty years since its initial publication, research into the algorithm's behaviour has produced several significant improvements: the modern refined version is typically thousands of times faster than the original. The search for further improvements continues today.

The remarkable computational utility of Gröbner bases prompted the development of a number of distinct, independent implementations of refined versions of Buchberger's algorithm. The main commercial general-purpose computer algebra systems (including `MAGMA` [12], `Maple` [43], `Mathematica` [58]) can all compute Gröbner bases; researchers needing the flexibility and ability to experiment with new algorithms also use computer algebra systems such as `CoCoA/CoCoALib` [1], `Macaulay/Macaulay2` [37] and `Singular` [32] and `Reduce` [38] which are freely downloadable from their respective websites.

Cylindrical algebraic decomposition. Another great success of the 1970s was the development of *cylindrical algebraic decomposition* (*CAD*) in [25]. This replaced the non-elementary complexity (no finite tower of exponentials bounds the complexity) of Tarski's method for real algebraic geometry, by a doubly exponential method. A CAD is a decomposition of $\mathbb{R}^n$ into cells arranged cylindrically (meaning their projections are equal or disjoint) and described by semi-algebraic sets. For a detailed description of modern CAD, see [15].

Hong created a C version of both the `SAC` library and the comprehensive CAD code, which is now open-source and freely available as `SACLIB` and `QEPCAD-B` [17]. Another example is the `Redlog` package [33] of the computer algebra system `Reduce`, which offers an optimised combination of the cylindrical algebraic decomposition with virtual substitution (see below) and Gröbner basis methods.

Virtual substitution. To mention a last algorithm, *virtual substitution* [57] focuses on non-linear real arithmetic formulas where the degree of the quantified variables is not too large. Although the method can be generalised to arbitrary degrees, current implementations are typically limited to input, where the total degree of the quantified variables does not exceed 2. In practice, this limitation is somewhat softened by employing powerful heuristics like systematic *degree shifts* or polynomial factorisation. One key idea is to eliminate existential quantifiers in favour of *finite* disjunctions plugging in test terms that are derived from the considered formula.

These methods and their numerous refinements belong to the usual tool box of state-of-the-art computer algebra systems, and enable them to tackle hard arithmetic problems.

2.2 Satisfiability Checking

In the 1960s, another line of research on *Satisfiability Checking* [8] for *propositional logic* started its career. The first idea used *resolution* for quantifier elimination [31], and had serious problems with the steeply increasing requirements on computational and memory resources with the increase of the problem size. Another research line [30] suggested a combination of *enumeration* and *Boolean constraint propagation (BCP)*. A major improvement was achieved in the 1990s by *combining* the two approaches, leading to *conflict-driven clause-learning* and *non-chronological backtracking* [44]. 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*.

Driven by this success, big efforts were made to enrich propositional SAT-solving with solver modules for different existentially quantified theories. Highly interesting techniques were implemented in *SAT-modulo-theories (SMT) solvers*

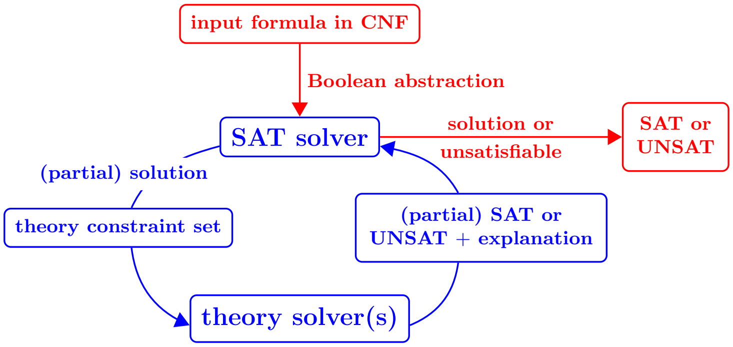

Fig. 2. The functioning of SMT solvers

for checking, e.g., equality logic with uninterpreted functions, array theory, bitvector arithmetic and quantifier-free linear real and integer arithmetic, but the development for quantifier-free non-linear real and integer arithmetic is still in its infancy. For further reading, see, e.g., [6,42].

Modern *SMT solvers* typically combine a *SAT solver* with one or more *theory solvers* as illustrated in Figure 2. 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.

On the one hand, 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 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., [5].

Examples for solvers that are able to cope with linear arithmetic problems (either in a complete or in an incomplete manner) are `Alt-Ergo` [26], `CVC4` [4], `iSAT3` [36, 53], `MathSAT` [22], `OpenSMT2` [19], `SMT-RAT` [28], `veriT` [13], `Yices2` [34], and `Z3` [48]. A further interesting SMT-approach for linear integer arithmetic is proposed in [16].

Much less activity can be observed for SMT solvers for non-linear arithmetic. A few SMT tools embedded some (complete as well as incomplete) decision procedures. Such a solver is `iSAT3`, which uses interval constraint propagation. The SMT solver `MiniSmt` [59] tries to reduce non-linear real arithmetic 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 [40]. Secondly, `SMT-RAT` [28], using solver modules for simplex, the cylindrical algebraic decomposition, the virtual substitution method, Gröbner bases, interval constraint propagation, branch and bound, and their strategic combination [47].

Even fewer SMT solvers are available for non-linear integer arithmetic, which is undecidable in general. A linearisation approach was proposed in [11]. The SMT solving spin-off of `AProVE` [23] 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 [10] with an axiom-based version of interval constraint propagation. SMT-RAT can tackle this theory using a generalised branch-and-bound technique.

The increasing variety of the theories considered by SMT solvers created an urgent need for a common input language. The *SMT-LIB* initiative [7] defined a *standard input language* 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* language, the user can employ *any* SMT solver to solve the problem.

3 Some Scientific Challenges and Opportunities

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 checking 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 Satisfiability Checking community would definitely profit from further exploiting Symbolic Computation achievements and adapting and extending 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 not trivial.

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.

In this section we describe some ideas of how algorithms and tools from both communities could be made more powerful by exploiting scientific exchange and technology transfer.

3.1 Symbolic Computation Techniques for Satisfiability Checking

Many practical decision procedures, designed by the Symbolic Computation community, are implemented in computer algebra systems (e.g., linear real and integer arithmetic, non-linear real arithmetic, linear programming, quantified formulas, Gröbner and involutive bases). To use them in a Satisfiability Checking context, some scientific and engineering issues need solutions, notably to find new ways of *incremental* solving, *explaining* unsatisfiability and generating *lemmas.*

Whereas for linear real arithmetic useful procedures have been adapted to satisfy the requirements for SMT embedding, many opportunities remain to be explored for non-linear arithmetic. For example, there are (to the best of our knowledge) just two SMT solvers, `Z3` and `SMT-RAT`, which make use of the CAD method, but in a different way: `Z3` uses a very elegant solution to explore the state space by a close integration of theory decisions and theory propagation in the Boolean SAT search, and constructs CAD only partially to explain conflicts in the above search (more precisely, to compute a semi-algebraic description of CAD cells that do not satisfy a given sign condition). In contrast, `SMT-RAT` implements an incremental version of the CAD method, which works hand-in-hand with the search at the logical level. For this latter approach, the power of heuristics (variable and polynomial ordering for incremental projection, choice and order of sample points for lifting) and the generation of lemmas (most importantly the computation of explanations for unsatisfiability) is still far from being fully exploited.

There is still great potential for improvements not only for the SMT embedding of the CAD method, but also other non-linear arithmetic decision procedures like virtual substitution or Gröbner bases, and their strategic combination with each other and further light-weight methods such as interval constraint propagation.

Another important aspect is the Symbolic Computation community's expertise in simplification and preprocessing. The complexity of the problems this community handles is often extremely high, and no practical procedure would exist without significant techniques to prepare the input problems. Such techniques do exist for Satisfiability Checking, but they rather focus on easier theories. A transfer of the savoir-faire in simplification and preprocessing for non-linear real and integer arithmetic would certainly be highly profitable.

3.2 Satisfiability Checking Techniques for Symbolic Computation

A key ingredient in the success of the Satisfiability Checking tools is the use of *learning* and *non-chronological backtracking* techniques to speed up the search through tree-shaped search structures. Traditionally (and in the majority of cases) CAD proceeds through a two stage process: first, projecting the problem through lower dimensions; then lifting: incrementally building a solution in increasing dimensions. An alternative approach using triangular decomposition was introduced in [21] where first the complex domain is cylindrically decomposed and then refined to a CAD of the real domain, where all data is in a tree-shaped structure.

Other techniques are certainly amenable for learning with the non-chronological backtracking approach. For instance, first prototypes integrating CDCLstyle learning techniques with virtual substitution for linear quantifier elimination have been successfully created and studied. Integration of learning techniques with the computation of comprehensive Gröbner bases [56] should also be investigated.

Incrementality, which played an important role in the success of Satisfiability Checking, may also be used to make Symbolic Computation techniques more efficient. The alternative CAD construction method described above is also incremental in nature [14] and so may offer one option here. An incremental CAD-based decision procedure for solving polymial constraint systems was proposed in [55]. There exist algorithms for computing Gröbner bases which exploit known mathematical facts about the ideal generated by the basis like its Hilbert function or some syzygies. Traditionally, this has been seen as a way to speed up computations. However, these approaches can naturally be adapted into incremental algorithms.

A central aspect of Satisfiability Modulo Theories is the combination frameworks for theories and decision procedures. Combining theories in Symbolic Computation (combined real/floating point theories, interval constraint propagation with other arithmetic theories) might also bring a number of advantages and possibilities, for instance, more expressive languages, or efficiency due to hierarchical reasoning. While combination of theories in Symbolic Computation are typically very specific and ad hoc, the SMT community systematically uses the generic Nelson–Oppen framework [49] for disjoint theories. Such a framework can of course not be used as it is, but it might be an inspiration for a modular approach in Symbolic Computation.

3.3 Standard Languages and Benchmarks

The initiation and maintenance of a common problem specification language standard *SMT-LIB* [7] and of competitions form an important part of the Satisfiability Checking community effort. Besides providing a stimulating event for tool developers to exhibit their systems, the competitions are also a vehicle for publishing practical progress. Competition results are advertised, and consulted by users to pick the best tools and techniques to solve problems.

The Symbolic Computation community does not have a similar tradition, and indeed, to quote one major system developer: "it is very hard to get any practical improvements published — the reviewers will often say this is not hard science". Although it is not good to only focus on a small library of benchmarks and have the competition as sole goal, competitions do have a tremendously positive effect on tools and techniques, as witnessed in the Satisfiability Checking community, especially if the competition challenges are concrete industrial challenges. Such driving forces could be also established in Symbolic Computation.

Though in Satisfiability Checking the standard input language allowed to provide large benchmark sets to the community, benchmarks for non-linear arithmetic theories are still rare, and harder to describe without ambiguity. Therefore, also the Satisfiability Checking community would profit from a common standard with an increased number of non-linear arithmetic benchmarks.

4 Project Actions

The solution of challenging problems, as mentioned in the previous section, could be within reach, when supported by a stronger collaboration between both SC^2 research areas, creating an infrastructure for dialogue and knowledge transfer. However, the research areas of Satisfiability Checking and Symbolic Computation are still quite disconnected, as reflected in their communication platforms and support structures. Symbolic Computation has its own conferences (ACA, CASC, ISSAC, etc.), several dedicated journals (e.g., AAECC, JSC, MSC), and the SIGSAM forum. Similarly, Satisfiability Checking is supported by its own conferences (CADE, IJCAR, SMT, etc.) and journals (e.g., JAR), the SatLive forum to keep up-to-date with research, SMT standards, and SAT- and SMT-solver competitions.

The main aims of our project are to create *communication platforms* and propose *standards* to enable the interaction between the two communities, and to use these platforms to *initiate discussions and cooperation* and to *identify potentials, challenges and obstacles* for future research and practical applications. In the following we shortly describe planned actions of our SC^2 project to achieve these goals.

Communication platforms To bridge the SC^2 communities, we will initiate platforms to support the interaction of the currently disjoint groups. We organised a Dagstuhl Seminar *Symbolic Computation and Satisfiability Checking*[15] 15-20th November, 2015, which already led to numerous interesting discussions and interactions. At CASC 2016, we will organise a *topical session* devoted to topics from the cross-community SC^2 area. Furthermore, we will establish a workshop series in the area of SC^2, covering the interests of both communities, and having its first edition affiliated with SYNASC 2016. These workshops will serve as platforms for scientific exchange, discussion and cooperation within and between the currently disjoint communities. To support and attract young new community members, we will organise a dedicated summer school aimed at interested young researchers from SC^2 areas, with courses specifically tailored to their needs.

Research roadmap The above platforms will initiate cross-community interactions, and help to clearly identify unused potentials. We aim at initiating discussions on what the communities can learn from each other, what are the common challenges which they can solve together, what Satisfiability Checking could learn from Symbolic Computation achievements, and which Satisfiability Checking results could be adapted to improve Symbolic Computation solutions.

Our long-term objective is to create a research roadmap of potentials and challenges, both to the two traditional subject silos, but also challenges that only the new joined SC^2 community can address. This roadmap should identify, within the problems currently faced in the industry, the particular points that can be expected to be solved by the SC^2 community in the short and middle term, and will provide recommendations for spin-off projects.

[15] http://www.dagstuhl.de/en/program/calendar/semhp/?semnr=15471

Standards, benchmarks, competitions We aim to create a standard problem specification language capable of representing common problems of the SC^2 community. We plan on extending the *SMT-LIB* language, which is already mature and fully accepted among the SMT (Satisfiability Checking) community, to handle features needed for the Symbolic Computation community. This will be done in a modular way, with a particular focus on extensibility for new features.

Agreeing on a common language, and being able to share challenging problems is an essential aspect for building a dynamic community. This will foster further discussions and uncover problems that can be solved by the SC^2 community altogether, set clear challenges on which various approaches can be evaluated, classify the approaches according to their strength and weaknesses on the various kinds of problems. Mixed approaches will naturally emerge, to tackle problems exhibiting several orthogonal difficulties. The standard could also serve as a communication protocol for platforms mixing tools, to build meta-tools to solve large and difficult problems out of reach of current techniques often specialised to just one kind of job.

How to become an associate? This project cannot reach its aims by involving just a small number of core project members. To be able to cover sufficiently wide research and application areas and to take into account their needs and interests, there are currently 37 SC^2 *associates* from both research communities as well as from industry. Our associates will be regularly informed about the project activities and they will be invited to take part in the corresponding events.

The SC^2 Coordination and Support Action will be an optimal platform for industrial and academic partners and associates to form smaller working groups and initiate specific projects. If you would like to participate in the project as an associate, please contact the Project Coordinator James Davenport[16].

5 Conclusions and Future Work

In this paper we gave a short description of the aims and actions of our upcoming EU Coordination and Support Action SC^2.

The SC^2 project will maintain a website (http://www.sc-square.org) making readily accessible all the public information of the project (e.g., contact information, details of past and forthcoming SC^2 workshops and other similar events).

Acknowledgements

We thank the anonymous reviewers for their comments. We are grateful for support by the H2020-FETOPEN-2016-2017-CSA project SC^2 (712689) and the ANR project ANR-13-IS02-0001-01 SMArT. Earlier work in this area was also supported by the EPSRC grant EP/J003247/1.

[16] Email contact: J.H.Davenport@bath.ac.uk

References

1. Abbott, J., Bigatti, A.M., Lagorio, G.: `CoCoA-5`: A system for doing computations in commutative algebra. http://cocoa.dima.unige.it
2. Ábrahám, E.: Building bridges between symbolic computation and satisfiability checking. In: Proceedings ISSAC 2015. pp. 1–6. ACM (2015)
3. Arai, N.H., Matsuzaki, T., Iwane, H., Anai, H.: Mathematics by machine. In: Proceedings ISSAC 2014. pp. 1–8. ACM (2014)
4. Barrett, C., Conway, C.L., Deters, M., Hadarean, L., D. Jovanović, T.K., Reynolds, A., Tinelli, C.: `CVC4`. In: Proceedings CAV 2011. LNCS, vol. 6806, pp. 171–177. Springer (2011)
5. Barrett, C., Kroening, D., Melham, T.: Problem solving for the 21st century: Efficient solvers for satisfiability modulo theories. Tech. Rep. 3, London Mathematical Society and Smith Institute for Industrial Mathematics and System Engineering (2014), http://www.cs.nyu.edu/ barrett/pubs/BKM14.pdf, knowledge Transfer Report
6. Barrett, C., Sebastiani, R., Seshia, S.A., Tinelli, C.: Satisfiability modulo theories. In: Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185, chap. 26, pp. 825–885. IOS Press (2009)
7. Barrett, C., Stump, A., Tinelli, C.: The satisfiability modulo theories library (*SMT-LIB*). www.SMT-LIB.org (2010)
8. Biere, A., Biere, A., Heule, M., van Maaren, H., Walsh, T.: Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press (2009)
9. Bixby, R.E.: Computational progress in linear and mixed integer programming. Presentation at ICIAM 2015 (2015)
10. Bobot, F., Conchon, S., Contejean, E., Mahboubi, A., Mebsout, A., Melquiond, G.: A simplex-based extension of Fourier-Motzkin for solving linear integer arithmetic. In: Proceedings IJCAR 2012. LNCS, vol. 7364, pp. 67–81. Springer (2012)
11. Borralleras, C., Lucas, S., Navarro-Marset, R., Rodriguez-Carbonell, E., Rubio, A.: Solving non-linear polynomial arithmetic via SAT modulo linear arithmetic. In: Proceedings CADE-22, LNCS, vol. 5663, pp. 294–305. Springer (2009)
12. Bosma, W., Cannon, J., Playoust, C.: The Magma algebra system. I. The user language. J. Symbolic Comput. 24(3-4), 235–265 (1997), http://dx.doi.org/10.1006/jsco.1996.0125, computational algebra and number theory (London, 1993)
13. Bouton, T., Caminha, D., de Oliveira, B., Déharbe, D., Fontaine, P.: `veriT`: An open, trustable and efficient SMT-solver. In: Proceedings CADE-22. LNCS, vol. 5663, pp. 151–156. Springer (2009)
14. Bradford, R.J., Chen, C., Davenport, J.H., England, M., Moreno Maza, M., Wilson, D.J.: Truth table invariant cylindrical algebraic decomposition by regular chains. In: Proceedings CASC 2014. LNCS, vol. 8660, pp. 44–58. Springer (2014)
15. Bradford, R., Davenport, J., England, M., McCallum, S., Wilson, D.: Truth Table Invariant Cylindrical Algebraic Decomposition. J. Symbolic Computation 76, 1–35 (2016)
16. Bromberger, M., Sturm, T., Weidenbach, C.: Linear integer arithmetic revisited. In: Proceedings CADE-25. LNCS, vol. 9195, pp. 623–637. Springer (2015)
17. Brown, C.W.: `QEPCAD B`: A program for computing with semi-algebraic sets using CADs. ACM SIGSAM Bulletin 37(4), 97–108 (2003)

18. Brown, C.W., Davenport, J.H.: The complexity of quantifier elimination and cylindrical algebraic decomposition. In: Proceedings ISSAC 2007. pp. 54–60. ACM (2007)
19. Bruttomesso, R., Pek, E., Sharygina, N., Tsitovich, A.: The `OpenSMT2` solver. In: Proceedings TACAS 2010. LNCS, vol. 6015, pp. 150–153. Springer (2010)
20. Buchberger, B.: Ein Algorithmus zum Auffinden des basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal. Ph.D. thesis, University of Innsbruck (1965), english translation: *J. Symbolic Computation*, 41:475–511, 2006
21. Chen, C., Moreno Maza, M., Xia, B., Yang, L.: Computing cylindrical algebraic decomposition via triangular decomposition. In: Proceedings ISSAC 2009. pp. 95–102. ACM (2009)
22. Cimatti, A., Griggio, A., Schaafsma, B., Sebastiani, R.: The `MathSAT5` smt solver. In: Proceedings TACAS 2013. LNCS, vol. 7795, pp. 93–107. Springer (2013)
23. Codish, M., Fekete, Y., Fuhs, C., Giesl, J., Waldmann, J.: Exotic semi-ring constraints. In: Proceedings SMT 2013. EPiC Series, vol. 20, pp. 88–97. EasyChair (2013)
24. Collins, G.E.: The SAC-1 system: An introduction and survey. In: Proceedings SYMSAC 1971. pp. 144–152. ACM (1971)
25. Collins, G.E.: Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In: Automata Theory and Formal Languages. LNCS, vol. 33, pp. 134–183. Springer (1975)
26. Conchon, S., Iguernelala, M., Mebsout, A.: A collaborative framework for non-linear integer arithmetic reasoning in `Alt-Ergo`. In: Proceedings SYNASC 2013. pp. 161–168. IEEE (2013)
27. Cook, S.A.: The complexity of theorem-proving procedures. In: Proceedings STOC 1971. pp. 151–158. ACM (1971), http://doi.acm.org/10.1145/800157.805047
28. Corzilius, F., Kremer, G., Junges, S., Schupp, S., Ábrahám, E.: `SMT-RAT`: An open source C++ toolbox for strategic and parallel SMT solving. In: Proc. SAT 2015. LNCS, vol. 9340, pp. 360–368. Springer (2015)
29. Davenport, J.H., Heintz, J.: Real quantifier elimination is doubly exponential. J. Symbolic Computation 5, 29–35 (1988)
30. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Communications of the ACM 5(7), 394–397 (1962)
31. Davis, M., Putnam, H.: A computing procedure for quantification theory. Journal of the ACM 7(3), 201–215 (1960)
32. Decker, W., Greuel, G.M., Pfister, G., Schönemann, H.: `Singular` 4-0-2 — A computer algebra system for polynomial computations. http://www.singular.uni-kl.de (2015)
33. Dolzmann, A., Sturm, T.: `Redlog`: Computer algebra meets computer logic. ACM SIGSAM Bulletin 31(2), 2–9 (1997)
34. Dutertre, B., de Moura, L.: A fast linear-arithmetic solver for DPLL(T). In: Proc. CAV 2006. LNCS, vol. 4144, pp. 81–94. Springer (2006)
35. Eraşcu, M., Hong, H.: Synthesis of optimal numerical algorithms using real quantifier elimination (Case study: Square root computation). In: Proceedings ISSAC 2014. pp. 162–169. ACM (2014)
36. Fränzle, M., Herde, C., Teige, T., Ratschan, S., Schubert, T.: 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)
37. Grayson, D.R., Stillman, M.E.: `Macaulay2`, a software system for research in algebraic geometry. http://www.math.uiuc.edu/Macaulay2/

38. Hearn, A.C.: REDUCE: The first forty years. In: Proceedings A3L. pp. 19–24. Books on Demand GmbH (2005)
39. Jenks, R.D., Sutor, R.S.: AXIOM: The Scientific Computation System. Springer (1992)
40. Jovanović, D., de Moura, L.: Solving non-linear arithmetic. In: Proceedings IJCAR 2012. LNAI, vol. 7364, pp. 339–354. Springer (2012)
41. Kahrimanian, H.G.: Analytic Differentiation by a Digital Computer. Master's thesis, Temple University Philadelphia (1953)
42. Kroening, D., Strichman, O.: Decision Procedures: An Algorithmic Point of View. Springer (2008)
43. Maple, http://www.maplesoft.com/
44. Marques-Silva, J.P., Sakallah, K.A.: GRASP: A search algorithm for propositional satisfiability. IEEE Trans. Computers 48, 506–521 (1999)
45. Martin, W.A., Fateman, R.J.: The Macsyma system. In: Proceedings SYMSAC 1971. pp. 59–75. ACM (1971)
46. Moses, J.: Symbolic Integration. Ph.D. thesis, M.I.T. & MAC TR-47 (1967)
47. de Moura, L., Passmore, G.O.: The strategy challenge in SMT solving. In: Automated Reasoning and Mathematics, pp. 15–44. Springer (2013)
48. de Moura, L.M., Bjørner, N.: Z3: An efficient SMT solver. In: Proceedings TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer (2008)
49. Nelson, G., Oppen, D.C.: Simplifications by cooperating decision procedures. ACM Transactions on Programming Languages and Systems 1(2), 245–257 (1979)
50. Nolan, J.: Analytic Differentiation on a Digital Computer. Master's thesis, M.I.T. (1953)
51. Platzer, A., Quesel, J.D., Rümmer, P.: Real world verification. In: Proceedings CADE-22. pp. 485–501. ACM (2009)
52. Risch, R.H.: The problem of integration in finite terms. Transactions of the American Mathematical Society 139, 167–189 (1969)
53. Scheibler, K., Kupferschmid, S., Becker, B.: Recent improvements in the SMT solver iSAT. In: Proceedings MBMV 2013. pp. 231–241. Institut für Angewandte Mikroelektronik und Datentechnik, Fakultät für Informatik und Elektrotechnik, Universität Rostock (2013)
54. Slagle, J.: A Heuristic Program that Solves Symbolic Integration Problems in Freshman Calculus. Ph.D. thesis, Harvard University (1961)
55. Strzeboński, A.: Solving polynomial systems over semialgebraic sets represented by cylindrical algebraic formulas. In: Proceedings ISSAC'12. pp. 335–342. ACM (2012)
56. Weispfenning, V.: Comprehensive Gröbner bases. J. Symbolic Computation 14(1), 1–29 (1992)
57. Weispfenning, V.: Quantifier elimination for real algebra – The quadratic case and beyond. Applicable Algebra in Engineering, Communication and Computing 8(2), 85–101 (1997)
58. Wolfram Research, Inc.: Mathematica, version 10.4. Wolfram Research, Inc., Champaign, Illinois (2016)
59. Zankl, H., Middeldorp, A.: Satisfiability of non-linear (ir)rational arithmetic. In: Proceedings LPAR 2010. LNAI, vol. 6355, pp. 481–500. Springer (2010)