SMT-Solving for the Real Algebra

Erika Ábrahám and Ulrich Loup

RWTH Aachen University, Germany

Abstract. SAT-solving is a highly actual research area with increasing success and plenty of industrial applications. SMT-solving, extending SAT with theories, has its main focus on linear real constrains. However, there are only few solvers going further to more expressive but still decidable logics like the first-order theory of the reals with addition and multiplication.
The main requests on theory solvers that must be fulfilled for their efficient embedding into an SMT solver are (a) incrementality, (b) the efficient computation of minimal infeasible subsets, and (c) the support of backtracking. For the first-order theory of the reals we are not aware of any solver offering these functionalities. In this work we address the possibilities to extend existing theory solving algorithms to come up with a theory solver suited for SMT.

1 Introduction

The *satisfiability problem* poses the question whether a given logical formula is satisfiable, i.e., whether we can assign values to the variables contained in the formula such that the formula becomes true. The development of efficient algorithms and tools (*solvers*) for satisfiability checking form an active research area in computer science. Such solvers are widely used not only in academical but also in industrial context, mainly for analysis purposes.

A lot of effort has been put into the development of fast solvers for the propositional satisfiability problem, called SAT. *SAT-solvers* are massively used for example in circuit analysis.

To increase expressiveness, extensions of the propositional logic with respect to first-order *theories* can be considered. The corresponding satisfiability problems are called SAT-modulo-theories problems, short SMT. *SMT-solvers* exist, e.g., for equality logic, uninterpreted functions, predicate logic, and linear arithmetic over the reals.

In contrast to the above-mentioned theories, less activity can be observed for SMT-solvers supporting the first-order theory of the real ordered field $(R, +, \cdot, <)$, what we call *real algebra*. Our research goal is to bridge this gap and develop an SMT-solver for real algebra, being capable

Dagstuhl Seminar Proceedings 09461
Algorithms and Applications for Next Generation SAT Solvers
http://drops.dagstuhl.de/opus/volltexte/2010/2508

of solving arbitrary Boolean combinations of polynomial constraints over the reals efficiently.

2 The Real algebra

Even though decidability of real algebra is known for a long time [Tar48] the first decision procedures were not yet practicable. Since 1974 it is known that the time complexity of deciding formulas of real algebra is in worst case doubly exponential in the number of variables contained in the formula [FR74]. In [BD07] the authors show that even simple input formulas can produce a double-exponentially large solution space to be searched.

Today, several methods are available which satisfy these complexity bounds, for example, cylindrical algebraic decomposition (CAD) , the Gröbner basis, and the virtual substitution method . An overview of these methods is given in [DSW97]. There are also tools available which implement these methods:

- The stand-alone application *QEPCAD* is a `C++` implementation of the CAD method [Bro03].
- The package *Redlog* of the computer algebra system *Reduce* based on `Lisp` offers an optimized combination of the methods in [DS97]. *Reduce* also provides a library for `C++` programs.

These algorithms are well-suited to check *conjunctions* of real constraints. However, *disjunctions* of real constraints are mostly solved by syntactic case splitting. For this reason, currently existing solvers are not suited to solve large formulas containing *arbitrary combinations* of real constraints. *Bounded Model Checking* formulas, for example, come under this issue.

3 SMT-solving for the Real Algebra

We plan to combine the advantages of highly tuned SAT-solvers and the most efficient techniques currently available for solving conjunctions of real constraints, by implementing an *SMT-solver for real algebra* that is capable to efficiently solve *abritrary Boolean combinations* of real constraints.

The basic scheme of DPLL-based SMT solving is roughly as follows (see Figure 1): The SMT-solver first creates a Boolean skeleton of the

input formula, replacing all real polynomial constraints contained in the input formula by fresh auxiliary Boolean variables. The resulting Boolean formula is passed to the SAT-solver, which searches for a satisfying assignment. If it does not succeed, the formula is unsatisfiable. Otherwise, the assignment found corresponds to certain truth values for the real polynomial constraints and has to be verified by the theory solver. If the constraints are satisfiable, then the original formula is satisfiable. Otherwise, if the theory solver detects that the conjunction of the corresponding real constraints is unsatisfiable, it then hands over an explanation for the unsatisfiability, a *minimal infeasible subset* of the real polynomial constraints, to the SAT-solver. The SAT-solver uses this piece of information to exclude the detected real conflict from further search. Afterwards, the SAT-solver computes again an assignment for the refined Boolean problem, which in turn has to be verified by the theory solver. Continuing this iteration in the end decides the satisfiability of the input formula.

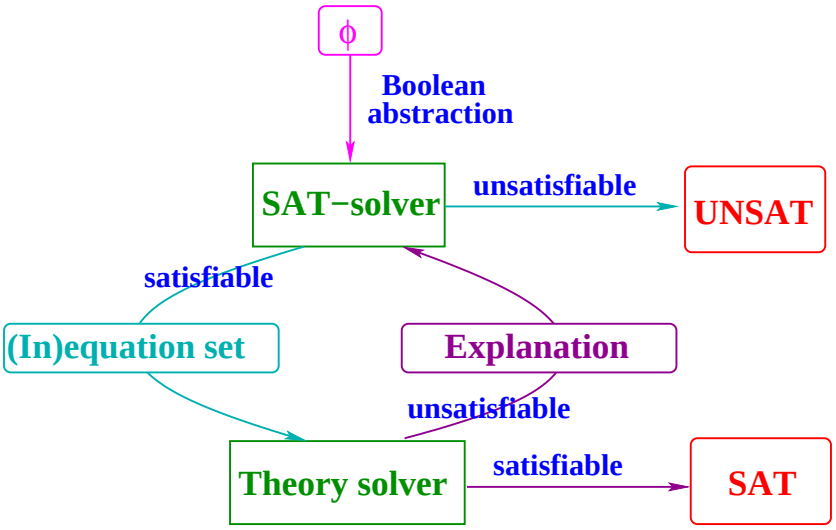

Fig. 1. The basic scheme of DPLL-based SMT solving

Above we described a *full lazy* procedure, where the theory solver checks constraints corresponding to a complete assignments only. In practice this is often disadvantageous, since the SAT-solver may do a lot of needless work by extending an already (in the real domain) contradictory partial assignment. *Less lazy* variants of the procedure call the theory solver more often, already handing over constraints corresponding to par-

tial assignments. To do so efficiently, the theory solver should accept constraints in an *incremental* fashion, where computation results of previous steps can be reused. Furthermore, in case of a conflict the theory solver should also be able to track back to previous computation steps.

Summarized, the implementation of an SMT-solver requires a theory solver to be able to

- work incrementally,
- compute minimal infeasible subsets, and
- backtrack on demand to previous computation steps.

To our knowledge, currently none of these functions are supported by the available theory solvers for real algebra.

We are only aware of the SMT-solvers *Z3* [dMB08], *HySAT* [FHT+07] and *ABsolver* [BPT07] which are able to handle arithmetic constraints. The algorithm implemented in *HySAT* uses interval arithmetic to check real constraints. The structures of *ABsolver* and *Z3* are more similar to our SMT-solver planned. However, *Z3* does not support full non-linear arithmetic. The authors of *ABsolver* do not address the issues of incrementality and backtracking. Though *ABsolver* computes minimal infeasible subsets, we did not find any information how they are generated.

4 Adaptation of the Theory Solvers

We consider the adaptation of two existing decision procedures for real algebra: the *virtual substitution* method and the *cylindrical algebraic decomposition (CAD)*, to support *incrementality*, *minimal infeasible subset computations* and *backtracking*.

Computation of *minimal infeasible subsets* is covered by a method which is independent of the underlying theory solver, and therefore presented separately in the current paragraph. We call constraints *dependent* if they share variables. In order to keep track of constraint dependencies, a dependency graph is updated whenever a constraint is added to the input of the theory solver. This graph yields some advantages for variable elimination order heuristics, furthermore it can be used to compute infeasible subsets: Whenever a constraint, sent to the theory solver, results in an unsatisfiable theory part this constraint together with all constraints which belong to the same component of the dependency graph build an infeasible subset. It can be minimized by using constraints from the same dependency graph component only.

4.1 Virtual substitution

The first procedure we want to tackle is the *virtual substitution* method, which is the most restricted but most simple and particularly the most efficient one. Virtual substitution in principle works by replacing each variable of the input formula by a term which does not include this variable anymore, thus eliminating all variables successively resulting in a formula which can be evaluated to "true" or "false" immediately. The substitution terms of one variable are mainly the solutions of linear, quadratic or cubic equations in this variable. Since there is mostly more then one solution of such an equation the size of the formula evolves exponentially during the substitution process – but not double-exponentially. Since such a solution formula exists for polynomials with an order up to 5 only, this method is not complete.

Incrementality The virtual substitution solver is modified to store a list of variables substituted so far, and a list of its substitution terms for each variable. Now assume the SAT-solver already sent several constraints to the virtual substitution solver, and this solver already found all substitution terms for these constraints. Then a new constraint, sent to the virtual substitution solver, is first solved for variables not considered yet, resulting in a formula containing only variables which already were substituted before the constraint was added. The remaining variables are substituted by using the original substitution term lists for each variable.

Minimal infeasible subsets Additionally to the above mentioned dependency graphs, book-keeping of a *substituion tree* would allow to backtrack the conflict for an unsatisfiability result to the constraints contributed to the conflict.

Backtracking The substitution procedure builds up a complete substitution tree which can – with an appropriate implementation – be traversed at will.

4.2 Cylindrical Algebraic Decomposition

At second we head for the *cylindrical algebraic decomposition* (CAD) method which, to our knowledge, is not yet embedded into any SMT solver. In contrast to the virtual substitution method, CAD provides a complete decision procedure. A combination of the virtual substituion with an underlying CAD solver to which the solving mechanism falls back if virtual substitution is not applicable, offers an efficient combination.

A CAD of the coordinate space underlying an input formula with polynomial constraints is a partition into cells such that the truth value of the input formula is constant on all points of a cell. The CAD method has two main phases: a projection phase and a construction phase. First a variable order is fixed, on the basis of which all variables are eliminated one by one. In the projection phase, first the set of all polynomials occurring in the input formula is built. Then a special projection operator is used, which maps sets of polynomials to sets of polynomials while satisfying the following property: given a CAD for a projection of a set of polynomials, a CAD for the original set of polynomials can be constructed. Now the initial set of polynomials is projected stepwise onto sets of polynomials each having the next variable in order eliminated. The final projection step yields a set of univariate polynomials. At this point the construction phase starts by calculating the real zeros of these polynomials. The zeros induce a partition of the real line into cells where the signs of the polynomials are constant. For each cell a sample point is computed, which is then used to compute new sample points of a CAD for the polynomials of the previous projection step. This sample point computation can be carried on until sample points of a CAD for the input polynomials are found. In a last evaluation step, these points are searched for one cell satisfying all constraints and hence, the input formula.

Backtracking Starting at the set of input polynomials, the projection steps are modified such that given a projected polynomial its projection preimage can be identified. This enhances the procedure by the possibility of tracking back the whole projection of an input polynomial. The CAD method is further modified to store in each construction phase step first the sample points, and second the information which polynomials produced the point. Thereby it is possible to identify the polynomials which produce a certain cell of a CAD occurring during the computation.

Incrementality Consider the situation wherein a CAD for a certain input formula has already been computed by the CAD method equipped with the backtracking modifications. In case a new constraint is added to the input, only the projection of just the new constraint must be computed. Existing decompositions are only refined by the sample points for the new input polynomial.

5 Conclusion

In this paper we discussed problems that arise when embedding a theory solver for the real algebra into DPLL-based SMT solving. We identified requirements the theory solving algorithms must provide, and showed up some possible solutions for two decision procedures, the virtual substitution and the cylindrical algebraic decomposition.

Of course, it is still a long way to go until an efficient SMT solver is developed. However, we are quite optimistic that SMT-solving for the real algebra can make solving much faster. We implemented a first full lazy prototype without minimal infeasible subset generation and without theory-solver backtracking, but a light version of incrementality for the theory solver. It is not surprising that first results show that the SAT-based search is superior to the syntactic case splitting.

References

[BD07] Christopher W. Brown and James H. Davenport. The complexity of quantifier elimination and cylindrical algebraic decomposition. In Dongming Wang, editor, *ISSAC*, pages 54–60. ACM, 2007.

[BPT07] Andreas Bauer, Markus Pister, and Michael Tautschnig. Tool-support for the analysis of hybrid systems and models. In *Proceedings of the 2007 Conference on Design, Automation and Test in Europe (DATE), Nice, France*, pages 924–929. European Design and Automation Association, 2007.

[Bro03] Christopher W. Brown. Qepcad b: a program for computing with semi-algebraic sets usingcads. *SIGSAM Bull.*, 37(4):97–108, 2003.

[dMB08] Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis (TACAS)*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340, Berlin, 2008. Springer-Verlag.

[DS97] Andreas Dolzmann and Thomas Sturm. REDLOG: Computer algebra meets computer logic. *SIGSAM Bulletin (ACM Special Interest Group on Symbolic and Algebraic Manipulation)*, 31(2):2–9, June 1997.

[DSW97] Andreas Dolzmann, Thomas Sturm, and Volker Weispfenning. Real quantifier elimination in practice, July 20 1997.

[FHT+07] Martin Fränzle, Christian Herde, Tino Teige, Stefan Ratschan, and Tobias Schubert. Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. *Journal on Satisfiability, Boolean Modeling and Computation*, 1:209–236, 2007.

[FR74] M. J. Fischer and M. O. Rabin. Super-exponential complexity of presburger arithmetic. Project MAC Tech. Memorandum 43, MIT, Cambridge, 1974.

[Tar48] Alfred Tarski. *A Decision Method for Elementary Algebra and Geometry.* University of California Press, 1948.