On the Implementation of Cylindrical Algebraic Coverings for Satisfiability Modulo Theories Solving

Gereon Kremer[1], Erika Ábrahám[2], Matthew England[3], and James H. Davenport[4]

[1] Stanford University
gkremer@cs.stanford.edu
[2] RWTH Aachen University
abraham@cs.rwth-aachen.de
[3] Coventry University
Matthew.England@coventry.ac.uk
[4] University of Bath
J.H.Davenport@bath.ac.uk

Abstract

We recently presented cylindrical algebraic coverings: a method based on the theory of cylindrical algebraic decomposition and suited for nonlinear real arithmetic theory reasoning in Satisfiability Modulo Theories solvers.

We now present a more careful implementation within cvc5, discuss some implementation details, and highlight practical benefits compared to previous approaches, i.e., NLSAT and incremental CAD. We show how this new implementation simplifies proof generation for nonlinear real arithmetic problems in cvc5 and announce some very encouraging experimental results that position cvc5 at the very front of currently available SMT solvers for `QF_NRA`.

1 Introduction

The Satisfiability Modulo Theories (SMT) problem is concerned with deciding satisfiability or unsatisfiability of first-order formulae over a given theory, or a combination of multiple theories. One theory of considerable interest is nonlinear real arithmetic (NRA), where atoms are sign constraints on polynomials with rational coefficients. This theory had already been studied extensively in the literature before the advent of SMT solving and a variety of methods exists. The only complete decision procedures that have found their way to practical use are all based on cylindrical algebraic decomposition (CAD) [5], although CAD has a worst-case running time that is doubly exponential in the number of variables [6].

All SMT solvers that treat nonlinear real arithmetic in a complete way are based on CAD: yices and z3 both use NLSAT [8] while SMT-RAT implements both NLSAT and an incremental variant of CAD [10]. Unfortunately, both these approaches have issues. The NLSAT framework is sufficiently different from the traditional CDCL(T) framework to make a seamless integration into an existing CDCL(T)-style SMT solver virtually impossible: NLSAT is mostly separated from the rest of the solver in SMT-RAT, yices, and z3. Meanwhile, implementing CAD in an incremental fashion [10] requires a great deal of machinery that may be prohibitive to implement.

A new decision procedure for NRA called *cylindrical algebraic coverings* was recently presented in [1]. It is heavily inspired by CAD and inherits its theoretic properties: it is complete, but also has doubly exponential worst-case running time. It can easily be used as a theory solver in the CDCL(T) framework (in contrast to NLSAT), but is considerably simpler to implement compared to the incremental CAD approach from [10]. It has also been observed that a trace of its computation for UNSAT is much closer to human reasoning compared to CAD [2], which makes it easier to understand and also simplifies the automatic generation of proofs of unsatisfiability.

Contribution

This paper presents an implementation of the cylindrical algebraic coverings method within cvc5 (the new successor to CVC4 [3]), its ability to generate proofs, and its performance compared to the approaches implemented in SMT-RAT, yices and z3. In the following, we assume some familiarity with [1].

2 Implementation

We have presented an initial implementation of the cylindrical algebraic coverings method within SMT-RAT in [1], which was still in an experimental state. Since then, we have produced a more stable implementation within cvc5 based on the algebraic routines from libpoly [9]. The new implementation directly follows the description in [1] with a few extensions: generation of infeasible subsets, partial theory checks, several variable ordering strategies, and dynamic exclusion of non-integral sample points for mixed-integer problems.

Most notably, though, we do not use our implementation as a self-contained theory solver but combine it with a linear arithmetic solver in the spirit of [4]: our method is only executed when the linearization of the input is satisfiable and a "linear model" for this linearization has been found. This not only avoids calling this comparably expensive method in many cases, but also allows for novel techniques like using the linear model as a seed for the nonlinear model we aim to construct. An extension to allow for an incremental processing of consistency checks is currently being worked on, but is still in an early experimental state.

Infeasible subsets. Generating small infeasible subsets is known to be an effective technique to speed up SMT solvers. The aim is to identify a small subset of the input constraints that is already unsatisfiable. We store all contributing constraints for every interval and simply collect all constraints for the intervals that cover the first dimension to obtain an infeasible subset. By checking constraints against sample points in a fixed order and regularly pruning redundant intervals, we hope to reduce this set of constraints. It is conceivable to perform an a-posteriori analysis to obtain an even smaller set of constraints, but this has not been done yet. Note that this task seems to be more involved than for a regular CAD as refuting a sample point by a different constraint most likely changes the generated interval and everything it was then used for.

Partial checks. The integration with the linear arithmetic solver was originally intended for an incremental linearization approach [4]. It is thus sufficient to issue a lemma that excludes the current linear model instead of performing a full consistency check. Our method can easily be adapted as every interval can be formulated as a lemma. We have implemented the possibility to terminate as soon as the first interval has been constructed in the first dimension. If we make sure that we sample the first variable according to the linear model, this interval yields a lemma suitable for [4]. Note that it seems unlikely that this technique has any benefits by itself, and an integration with the linearization is not trivial: the lemmas would need to be linear to actually help the linear solver, but the lemmas constructed from our intervals are usually not linear.

Variable ordering. Finding a good variable ordering is a notorious issue for all CAD-based approaches, as it has huge impacts on performance, with the right choice susceptible to slight changes to the input, and existing heuristics oftentimes not very robust. We implement a static ordering, as well as the heuristics commonly called "Brown" and "Triangular" [7]. Additionally, we experimented with a heuristic based on machine learning techniques, but with very limited success.

Mixed-integer problems. Most implementations for CAD-based approaches try to sample integer values whenever possible, simply to keep coefficients small. We do the same, and thus we only select a non-integer sample

when the surrounding interval has no integer. In such a case, we can simply exclude the interval in between the two neighbouring integer values and provide corresponding polynomials as characterization. Let for example $x = 1.3$ be a sample; then we exclude $(1; 2)$ and use $\{x - 1, x - 2\}$ as characterization.

Incremental consistency checks. Our integration with the linear solver allows to use the linear model as an initial model for our method. The hope would be that the linearization of the input describes the solution space sufficiently well to direct us into the vicinity of proper solutions by using the linear model. In practice, however, seeding our method with the linear model shows no benefit.

Additionally, we have an experimental implementation that is more in line with the common understanding of incremental consistency checks. Instead of only collecting the intervals locally, we maintain the intervals in an explicit tree that is retained across multiple theory calls. This allows us to reuse whole subtrees and possibly avoid a lot of work. In its current experimental state, though, this technique has not shown a substantial benefit.

3 Proof Generation

There is a new infrastructure in cvc5 to generate formal proofs for almost every aspect of its solving process. We use this to generate proofs for cylindrical algebraic coverings that roughly follow the ideas from [2]. In contrast to proofs generated in other parts of cvc5, they are not detailed enough to allow for an automatic verification by, e.g., Isabelle/HOL. They are already useful, though, in that they decompose the overall proof into meaningful proof steps and allow for consistency checks like connectedness and structural soundness. We intend to make these proofs more detailed in the future.

Consider the following formula:

$$(x > 0) \wedge (y < 0) \wedge (x \cdot y = 0).$$

The proof steps related to our approach are given below, while the actual proof generated by cvc5 contains a lot of additional reasoning steps. Note that it makes use of "root predicates" $x \sim root_k(p)$ that compare the kth root of p in x over the current assignment to x, and that the proof does not contain intervals that were generated, but ultimately are not necessary to build the proof.

- Consider $x > 0$, derive $x \notin (-\infty, 0]$ which is represented by $x > root_1(x)$.
- Guess $x = 1$ (abstracted by $x > root_1(x)$).
  - Consider $y < 0$, derive $y \notin [0, \infty)$ which is represented by $y < root_1(y)$.
  - Consider $x \cdot y = 0$, derive $y \notin (-\infty, 0)$ which is represented by $y \geq root_1(x \cdot y)$.
  - We combine $y < root_1(y)$ and $y \geq root_1(x \cdot y)$ to refute $x > root_1(x)$.
- We combine $x > 0$ and $\neg(x > root_1(x))$ to derive $false$.

The abstraction of $x = 1$ ($x > root_1(x)$) is not known when the solver guesses the value for x: it is computed when $x = 1$ is refuted and this branch of the proof is closed.

4 Experiments

We have already shown in Section 7 of [1] that a very preliminary implementation of the cylindrical algebraic coverings approach in SMT-RAT outperforms a rather elaborate incremental CAD implementation and is almost competitive with the NLSAT variant from SMT-RAT. We now show that a more careful implementation of the cylindrical algebraic coverings method can compete with other state-of-the art implementations. We use the `QF_NRA` benchmark set from SMT-LIB and use Intel Xeon E5-2620 processors with a timeout of ten minutes.

We compare the current implementation within cvc5 with z3 4.8.10, yices 2.6.2, the implementation of cylindrical algebraic coverings within SMT-RAT, the incremental linearization approach for `QF_NRA` that was already present in CVC4, and cvc5 without any nonlinear reasoning technique.

Solver	sat	unsat	overall
cvc5 (cdcac)	5021	5377	10398
yices	4904	5437	10341
z3	5093	5195	10288
SMT-RAT	4438	4435	8873
cvc5 (inc. lin.)	3283	5385	8668
cvc5 (no nl)	2203	3271	5474

Figure 1: Experimental results

Figure 1 shows that the new approach significantly outperforms the incremental linearization approach, and that it even has a small lead over both yices and z3, the two best solvers in the SMT competition 2020. We verify that it is indeed the new approach that makes cvc5 so powerful for QF_NRA by comparing to two other configurations of cvc5 for QF_NRA.

5 Summary and Future

We have described some implementation details for cylindrical algebraic coverings in cvc5, and how they can simplify automatic proof generation. The experimental results look very promising, in particular as some common optimization techniques are not used yet.

We aim to further improve our implementation by: generating better infeasible subsets; better integrate with the existing linearization approach to make use of partial checks; find better variable orderings; and improve the handling of incremental consistency checks.

References

[1] Erika Ábrahám, James H. Davenport, Matthew England, and Gereon Kremer. Deciding the consistency of non-linear real arithmetic constraints with a conflict driven search using cylindrical algebraic coverings. *Journal of Logical and Algebraic Methods in Programming*, 119(100633), 2021. doi:10.1016/j.jlamp.2020.100633.

[2] Erika Ábrahám, James H. Davenport, Gereon Kremer, and Zak Tonks. New opportunities for the formal proof of computational real geometry? In *Satisfiability Checking and Symbolic Computation Workshop*, volume 2752 of *CEUR Workshop Proceedings*, 2020. URL: http://ceur-ws.org/Vol-2752/paper13.pdf.

[3] Clark W. Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanovic, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In *Computer Aided Verification*, volume 6806 of *LNCS*, pages 171–177, 2011. doi:10.1007/978-3-642-22110-1_14.

[4] Alessandro Cimatti, Alberto Griggio, Ahmed Irfan, Marco Roveri, and Roberto Sebastiani. Incremental linearization for satisfiability and verification modulo nonlinear arithmetic and transcendental functions. *ACM Transactions on Computational Logic*, 19:1–52, 2018. doi:10.1145/3230639.

[5] George E. Collins. Quantifier Elimination for Real Closed Fields by Cylindrical Algebraic Decomposition. In *Proceedings 2nd. GI Conference Automata Theory & Formal Languages*, pages 134–183, 1975.

[6] James H. Davenport and Joos Heintz. Real Quantifier Elimination is Doubly Exponential. *J. Symbolic Comp.*, 5:29–35, 1988.

[7] Matthew England, Russell Bradford, James H. Davenport, and David Wilson. Choosing a variable ordering for truth-table invariant cylindrical algebraic decomposition by incremental triangular decomposition. In *International Congress on Mathematical Software*, volume 8592 of *LNCS*, 2014. doi:10.1007/978-3-662-44199-2_68.

[8] Dejan Jovanović and Leonardo de Moura. Solving non-linear arithmetic. In *Automated Reasoning*, volume 7364 of *LNCS*, pages 339–354, 2012. doi:10.1007/978-3-642-31365-3_27.

[9] Dejan Jovanović and Bruno Dutertre. Libpoly: A library for reasoning about polynomials. In *Satisfiability Modulo Theories*, volume 1889 of *CEUR Workshop Proceedings*, 2017. URL: http://ceur-ws.org/Vol-1889/paper3.pdf.

[10] Gereon Kremer and Erika Ábrahám. Fully incremental cylindrical algebraic decomposition. *Journal of Symbolic Computation*, 100:11–37, 2020. doi:10.1016/j.jsc.2019.07.018.