Acta Univ. Sapientiae, Informatica **10,** 1 (2018) 5–25

DOI: 10.2478/ausi-2018-0001

Modular strategic SMT solving with SMT-RAT*

Gereon KREMER
RWTH Aachen University
Aachen, Germany
email:
gereon.kremer@cs.rwth-aachen.de

Erika ÁBRAHÁM
RWTH Aachen University
Aachen, Germany
email:
abraham@cs.rwth-aachen.de

Abstract. In this paper we present the latest developments in `SMT-RAT`, a tool for the automated check of quantifier-free real and integer arithmetic formulas for satisfiability. As a distinguishing feature, `SMT-RAT` provides a set of solving modules and supports their strategic combination. We describe our CArL library for arithmetic computations, the available modules implemented on top of CArL, and how modules can be combined to satisfiability-modulo-theories (SMT) solvers. Besides the traditional SMT approach, some new modules support also the recently proposed and highly promising model-constructing satisfiability calculus approach.

1 Introduction

The problem of checking the satisfiability of first-order logic formulas appears in many different areas like, e.g., program verification or synthesis approaches like planning or scheduling. On the one hand, solving arithmetic formulas has

* *This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No H2020-FETOPEN-2015-CSA 712689.*

Computing Classification System 1998: I.2.8

Mathematics Subject Classification 2010: 68-04

Key words and phrases: satisfiability modulo theories, polynomial arithmetic, strategic combination

deep historical roots in mathematical logic and *symbolic computation*. On the other hand, the last decades led to fruitful developments also in computer science, resulting in efficient *SAT* and *satisfiability-modulo-theories* (*SMT*) solvers. Whereas at the beginning SMT solvers focused on theories for equality logic and uninterpreted functions, bit-vectors, arrays and floating-point arithmetic, recently notable achievements were made also for arithmetic theories. For the support of polynomial constraints over the reals interesting symbiosis evolved between symbolic computation and satisfiability checking, learning from each other and mutually integrating successful techniques from the respective areas.

In this paper we describe recent developments for our SMT solver `SMT-RAT` [17, 16], which is mainly developed for checking the satisfiability of quantifier-free real and integer arithmetic formulas. A distinguishing feature of `SMT-RAT` is that it offers a library of decision procedure modules, which share a common interface structure such that they can be strategically combined based on user-defined specifications to efficient SMT solvers.

The majority of the decision procedures that are implemented in those `SMT-RAT` modules stem from the area of symbolic computation, for which implementations in *computer algebra systems* are available. Examples for such *theory solving* procedures are the simplex method, methods using Gröbner bases, the subtropical satisfiability method, the virtual substitution method or the cylindrical algebraic decomposition method. Thus it is natural to think of invoking their implementations provided by different computer algebra systems. However, there are several obstacles that hinder their direct embedding in SMT solvers. Firstly, SMT solvers for arithmetic problems need modules to check the consistency of sets of polynomial constraints in an *incremental* fashion, meaning that after the consistency of a constraint set has been determined, the consistency of an extended set needs to be checked. Such incremental consistency checks need to be executed frequently, therefore it is important that they are not done independently but re-use information from previous checks as much as possible. Secondly, in case of inconsistency, these modules need to return an *explanation* for unsatisfiability, e.g. in the form of an inconsistent subset of the constraints. Thirdly, SMT solvers explore the space of possible solutions in an enumerative manner, accompanied by smart propagation, resolution and learning procedures to avoid unnecessary work in unsatisfiable parts of the search space. Once it is detected that the currently considered part of the search space does not contain satisfying solutions, *backtracking* takes place, which requires also backtracking ability for the theory solver. Unfortunately, implementations in computer algebra systems do not

provide these functionalities. Therefore, we implemented adaptations for several such methods as `SMT-RAT` modules that satisfy the above requirements.

These implementations in `SMT-RAT` required support for basic arithmetic computations like algorithms for polynomial division or calculating the greatest common divisor of two (multivariate) polynomials. Today's commonly used computer algebra systems like `Maple`, `Mathematica`, `Singular`, `GAP` or `Reduce` offer a rich set of highly efficient algorithms on polynomials [28], while also putting the focus on the user interface and graphical capabilities. Therefore, we experimented with their usage for our `SMT-RAT` modules. However, we experienced a major communication bottleneck at the interface between `SMT-RAT` and external tools: the frequent exchange of large constraint sets is very time consuming. Additionally, as the two communication sides use different data types to represent numbers, arithmetic expressions and constraints, the frequent communication caused a serious overhead also for datatype conversion.

An optimal solution can be offered by a (preferably free and open-source) *library* for arithmetic computations, which offers an object-oriented, generic and modular data structure for polynomials or, even better, formulas over polynomial constraints. Though a few libraries exist that try to bridge this gap, for example `GiNaC` [6] and `CoCoALib` [1], they all have various downsides that led us to implement our own `C++` library `CArL`. It is clearly not one of our short- or medium-term goals to compete with the performance of these algorithms on arbitrary inputs, when implementing a polynomial arithmetic library from scratch. However, we hope that in the long-term our tool will provide helpful support to other research groups in the SMT community and beyond.

Previously we reported on our software developments in [17] in 2012 and in [16] in 2015. The novel contributions of this paper are (i) the introduction of the `CArL` library and (ii) the description of `SMT-RAT` with a special focus on three new solver modules in the latest release.

The rest of this paper is structured as follows. In Section 2 we give a short introduction to SAT and SMT solving and briefly explain the main ideas of some relevant algebraic decision procedures. In Section 3 we introduce our arithmetic library `CArL`, followed by a description of our SMT solver `SMT-RAT` and its new modules in Section 4. Finally we provide some experimental results in Section 5 before we conclude the paper in Section 6.

2 Preliminaries

Traditionally, *satisfiability checking* aims at the automated check of the satisfiability of quantifier-free first-order logic formulas over some theories, whereas recent developments extend the functionalities for satisfiability checking to, e.g., quantified formulas or optimization. In this paper we focus on checking the satisfiability of quantifier-free *real arithmetic* formulas, which are Boolean combinations of constraints comparing polynomials over real-valued variables to zero.

SAT solving The success story of satisfiability checking started with *SAT solving* for propositional logic. The main strength of SAT solving is a highly efficient heuristic combination of enumeration, propagation, resolution and learning [19, 43]. The input is a propositional logic formula, which is a Boolean combination (using operators for negation $\neg$, conjunction $\wedge$, disjunction $\vee$ etc.) of Boolean variables called *propositions*. The input formula is first transformed into *conjunctive normal form* (*CNF*) in linear time and space on the cost of additional variables using Tseitin's transformation [51]. The result is a conjunction of disjunctions of possibly negated propositions; propositions and negated propositions are called *literals* and their disjunctions *clauses*. Enumeration is used to explore possible solutions, *deciding* which values for which propositions should be tried first. After each such decision, which is actually a guess for satisfying variable values, *propagation* is applied to detect certain implications of the current assignments and thus to reduce the number of "wrong guesses". For example, if the CNF contains a clause $(a \vee b)$ and the value *false* is decided for a then propagation assigns *true* to b in order to satisfy the clause. However, propagation cannot always avoid running into an unsatisfiable assignment. For example, the previous assignments would lead to the violation of the clause $(a \vee \neg b)$. When such a *conflict* is detected, *resolution* is applied to determine a reason for the conflict; in our example, resolving the two clauses $(a \vee b)$ and $(a \vee \neg b)$ would result in the resolvent (a). *Learning* the reason for the conflict will protect the future search from running into conflicts with the same reason.

SMT solving The impressive success of SAT solving led to the idea to extend the technology to check satisfiability also for quantifier-free formulas over different theories. *Satisfiability modulo theories* (*SMT*) *solving* started for equalities and uninterpreted functions, theories relevant for program verification (arrays, bit-vectors, floating-point arithmetic etc.) and linear real and integer arithmetic. Nowadays, powerful tools exist that can handle also harder

theories like, e.g., nonlinear arithmetic formulas. Some popular SMT solvers for arithmetic theories are, e.g., `AProVE` [29], `CVC4` [4], `MathSAT5` [12], `raSAT` [39], `veriT` [7], `Yices2` [22], `Z3` [20] and our solver `SMT-RAT` [16].

SMT solving typically works in a *lazy* fashion, meaning that the solver prioritizes to satisfy the Boolean structure of the formula first and check consistency in the theory afterwards. To do so, the *Boolean skeleton* or *Boolean abstraction* of the input formula is generated by replacing each theory constraint by a fresh proposition, resulting in a propositional logic formula, which can be checked for satisfiability by a SAT solver. If the skeleton is unsatisfiable then the input formula is unsatisfiable, too. Otherwise, if the SAT solver has determined a Boolean solution for the skeleton then suitable *theory solvers* are invoked to check whether all constraints with true abstraction propositions and the negations of all constraints with false abstraction propositions are together consistent. If this is the case then a satisfying solution for the input formula is found. Otherwise, the theory solvers need to provide an *explanation* for the inconsistency, typically by returning an inconsistent subset of the considered constraints. Learning the Boolean abstraction of this explanation refines the Boolean abstraction, avoiding Boolean solutions with the same theory conflict in future search. Besides such *full lazy* approaches, *less lazy* variants check theory consistency more frequently (usually after the full propagation of each Boolean decision).

MCSAT A recent technique called *model constructing satisfiability calculus* (*MCSAT*) [21] generalizes the above approach by defining a set of derivation rules, which includes besides Boolean decision, propagation and conflict resolution also their counterparts for the theory. Especially, MCSAT provides the possibility to guess not only truth values for the theory constraints but also values for the theory variables, and use some theory propagation techniques to drive the search for further theory variable values away from unsatisfiable parts of the state space. For example, if we decide to try the value 0 for a theory variable x and a constraint $x > y$ should hold according to the Boolean search then we need to guess a negative value for y, e.g. -1. However, if also the constraint $x^2 > y^2$ should hold then we have run into a theory conflict, because for $x = 0$ there is no value for y that would satisfy both constraints. In such cases, the theory conflict needs to be *explained* by a lemma, which in the optimal case generalizes the current conflict and helps to exclude from further search not only the current assignment but also others with similar reasons for being unsatisfying. For our example, the solver could explain the conflict by returning $x^2 > y^2 \to x > 0$.

Theory decision procedures As mentioned above, SMT solvers use, besides SAT solvers, also theory solvers, which need to check *sets* or *conjunctions* of theory constraints for consistency. Furthermore, for efficiency reasons, in the less lazy setting theory solvers should work *incrementally*, meaning that if a set of constraints is found satisfiable and the set is extended by adding some further constraints then the theory solver should not check the consistency of the extended set from anew but re-use previous results as much as possible. Additionally, for inconsistent constraint sets the theory solvers must be able to provide *explanations*.

For this purpose, we use decision procedures for arithmetic theories developed in symbolic computation and implemented in computer algebra systems. The symbiosis of these methods with SAT solving is fruitful because these methods are good at checking sets of constraints for satisfiability but they are not designed for combinatorial checks in Boolean structures. In the following we describe some of these procedures in a nutshell.

- *Interval constraint propagation* (*ICP*) [27, 34] uses interval arithmetic to contract given variable domains under the assumption of certain constraints. For example, if $x \in [0, 2]$ and $y \in [1, 3]$ and $x = y$ should hold then ICP can imply that $x \in [1, 2]$ and $y \in [1, 2]$. ICP is very powerful and can be applied to nonlinear arithmetic involving also trigonometric and transcendental functions, but it is incomplete in general.

- The *simplex method* [18] is applicable to linear real arithmetic. Originally it was developed for optimization but in the SMT context we use it for satisfiability checking only. The main idea is to start from an initial variable assignment satisfying a set of equalities and modify this assignment step-wise to satisfy also additional variable bounds.

- The *Fourier-Motzkin variable elimination method* allows to perform quantifier elimination on sets of linear real arithmetic constraints. The idea is that if two constraints define a lower and an upper bound respectively on a variable than satisfiability requires the lower bound to be smaller (or equal, depending on the comparison operator) than the upper bound. For example, $2y < x$ and $x < w$ requires $2y < w$. Collecting these requirements for all lower-upper-bound pairs on x allows to eliminate x from the constraint system.

- The original idea of the *virtual substitution method* [55] is to use solution equations to solve multivariate low-degree constraints symbolically

and substitute these solutions into the other constraints to eliminate variables. As these symbolic solutions might contain e.g. square roots, special virtual substitution rules are applied which produce standard arithmetic constraints after the substitution. The method requires the degree of the polynomials to be bounded and is thus incomplete in general.

- The *cylindrical algebraic decomposition* (*CAD*) *method* [13] is a decision procedure for real arithmetic. It is a quantifier elimination method, which decomposes the state space into a finite number of sign-invariant (or truth-invariant) regions, such that in each region either all points satisfy the input formula or none of them does so. Therefore, it is sufficient to take a single sample point from each region and check whether any of the sample points satisfies the formula. The CAD method is complete but in worst case it comes with doubly exponential solving effort.

- A recent incomplete but highly efficient method for finding solutions for real-arithmetic constraint sets is the *subtropical satisfiability method* [25]. It analyzes the exponent vectors of the monomials in the constraints and tries to find dominating monomials whose values can be made larger or smaller than all other monomial values. For example, $x^3y + x^2y^2 + y < 0$ is satisfiable because for any positive value for y we can find a sufficiently small value for x such that x^3y becomes the dominating monomial that makes the polynomial $x^3y + x^2y^2 + y$ negative.

- The incomplete *branch-and-bound method* [41] can be used to extend decision procedures for real arithmetic to check the satisfiability of integer arithmetic constraint sets. It first checks the real relaxation of the input constraints using some decision procedure for the reals, i.e., assuming the variables to be real-valued instead of integer-valued. If no solution exists in the real domain then there is no integer solution. If an integer solution is found by the real-valued search then the formula is satisfiable. Otherwise, if a real-valued solution v is found for an integer variable x then the search branches on values for x that are either less or equal than the largest integer below v or at least as large as than the smallest integer above v.

However, practical implementations of these procedures are usually not designed to work incrementally and they neither support the generation of explanations. Therefore, before their embedding in SMT solving they need to be adapted to satisfy these requirements.

3 CArL

For any project that aims to work on arithmetic formulas, some data structures are needed to represent numbers, polynomials and formulas. Data types for the exact representation of real numbers of arbitrary size in `C++` are provided e.g. by the libraries `gmp` and `cln`. Similar support is available for other languages like `Java` or `Python`. However, general-purpose libraries for the representation of real-algebraic numbers, polynomials, polynomial constraints and algebraic formulas, and efficient implementations of polynomial computations – ranging from addition and multiplication to pretty complex operations like greatest common divisor or factorization – are much more rare.

There are manifold reasons for this support gap. First of all, the range of algorithms that work on polynomials is extremely large and diverse, such that it is futile to attempt to exhaustively implement all algorithms. Furthermore, different representations tend to provide vastly different performance on different inputs and thus the application domain must be taken into consideration.

One attempt to provide a fairly generic `C++` library for polynomial arithmetic is `GiNaC` [6]. In contrast to computer algebra systems, `GiNaC` is designed as an open framework to be integrated in other tools, providing symbolic manipulations like arithmetic operations on polynomials. Its popularity reveals the urgent need for such a library. It is used for example for symbolic execution [3], probabilistic pointer analysis [11], and in the parametric probabilistic model checker PARAM [32].

However, `GiNaC` is not generic in the sense that it does not allow arbitrary coefficient types for the polynomials, and provides no possibility to influence the ordering of the variables and monomials in the polynomials. Both are crucial for the efficient implementation of many algorithms, for example decision procedures based on Gröbner bases [10] or the cylindrical algebraic decomposition method [13]. Furthermore, `GiNaC` lacks thread safety, thus it cannot be used safely in parallelized applications.

Another `C++` library is `CoCoALib` [1] which also provides many arithmetic operations on polynomials, though it is mostly tailored to the computation of Gröbner bases. Unfortunately, all `CoCoALib` polynomials are elements of some polynomial ring with a fixed variable ordering and polynomials of different rings are not directly compatible. This is a major obstacle whenever fresh variables are introduced or a certain operation is only performed on a small subset of the variables or on a different variable ordering. Furthermore `CoCoALib` does not offer all operations that are needed for methods like the cylindrical algebraic decomposition.

After having experimented with available libraries, we decided to develop a free and open-source *Computer Arithmetic Library* `CArL`[1] from scratch in `C++` to overcome these problems. The focus of `CArL` lies on efficient generic data types and algorithms for *polynomials and arithmetic formulas*, but also includes *bit vectors* and *uninterpreted variables and functions*.

Most of the data structures in `CArL` can be instantiated with different *number types*; it ships support for `gmp`, `cln` and native integers, as well as wrappers for `MPFR` and `Z3` rationals. For algebraic methods like the cylindrical algebraic decomposition method, `CArL` implements *real-algebraic numbers* – either in interval representation or using an encoding based on Thom's lemma. Furthermore, `CArL` implements an extension of the templated *`boost` intervals*, which allows open and closed bounds, and implements methods which are essential for interval constraint propagation techniques.

The library offers a variety of methods for computations with polynomials. Some of them provide basic functionalities like for example to get the list of variables of a polynomial, to check whether a polynomial is univariate, to iterate over the terms of a polynomial, to apply addition, subtraction, multiplication, substitution, comparison and evaluation, to normalize polynomials or to compute their derivatives. For univariate polynomials `CArL` can compute Cauchy and other bounds on the real zeros, Sturm sequences and their sign variations, resultants and sub-resultants, discriminants and real root isolations. Further methods for multivariate polynomials implement for example test for definiteness, sum of squares decomposition, polynomial (pseudo-)division and (pseudo-)remainder computation, factorization, computations of co-prime factors of coefficients, or S-polynomials. `CArL` even features its own implementation of Gröbner bases with a particular focus on the support of incrementality. Additionally to polynomial expressions, for methods like the virtual substitution `CArL` also offers data types for fractions and square root expressions.

To easily borrow further advanced functionality from other libraries or compare against their implementation, `CArL` integrates `CoCoALib` and `GiNaC`, offering alternative implementations for e.g. polynomial factorization, multivariate polynomial greatest common divisor and Gröbner bases computations. A preliminary integration of `Maple` – given an existing `Maple` installation – is also available.

Moreover, `CArL` bundles a lot of utility functionality that we deem useful when implementing any kind of tool similar to an SMT solver.

[1] Available at https://github.com/smtrat/carl.

4 SMT-RAT

The efficient solving of some variant of the satisfiability problem is a cornerstone for many techniques in formal verification and numerous industrial applications. We focus on the satisfiability modulo theories (SMT) problem that combines Boolean satisfiability with one or more theories, for example nonlinear arithmetic. A number of open-source solvers exist that tackle this class of problems with great success, for example `CVC4` [4], `raSAT` [52], `veriT` [7], `Yices2` [22] or `Z3` [20].

Our goal is to combine various techniques for SMT solving and study the interaction of different approaches. This requires a framework that allows a user to compose an SMT solver from individual *modules* easily in a very flexible way. Though the aforementioned solvers all have different solution techniques and more or less powerful mechanisms to combine them, we want this strategic combination of modules to be very transparent to the user. This idea is at the core of our SMT solving library `SMT-RAT`[2].

`SMT-RAT` is a library meant for, but not limited to, SMT solving. Though it can be used *as-is* to compose a stand-alone SMT solver – and this is how we use it most of the time – it is intended to be employed in either other SMT solvers or even be used for other solving tasks. The focus on modularity and composability yields a framework where the individual solver modules have a well-defined interface and are completely decoupled otherwise. This allows for an easy extension of `SMT-RAT` by new solving techniques without knowledge about the overall architecture and other modules. Undergraduate students routinely implement new theory solver modules or extend existing ones for various logics in practical courses or as thesis projects [14, 37, 47, 50, 40, 49, 42, 45, 46, 53, 56, 30, 33, 44, 31].

4.1 Previous SMT-RAT solver modules

`SMT-RAT` holds a rich collection of solver modules that can be grouped into *preprocessing modules* applying simplifications, *decision procedure modules* that implement satisfiability checking procedures and *meta modules*, though the border is sometimes blurry.

Preprocessing modules Oftentimes input problems have a certain structure that allows for some simplifications. `SMT-RAT` contains a collection of preprocessing techniques to exploit such possibilities. The `ESModule` searches for

[2] Available at https://github.com/smtrat/smtrat.

variables occurring linearly in equations that must hold and eliminate them. The `GBPPModule` is based on Gröbner bases and uses nonlinear equalities to simplify inequalities. Certain patterns of circular inequalities can be exploited with the `ICEModule` to eliminate variables. Some encodings of a multiple choice scenario can be simplified to Boolean decisions using the `MCBModule`. The `PFEModule` uses bounds on arithmetic variables to identify and remove sign-invariant factors of polynomials. Finally, the `SymmetryModule` identifies and breaks symmetries on both Boolean and theory variables.

Decision procedure modules At the core of most modern SMT solvers, and `SMT-RAT` is not different here, is a SAT solver. `SMT-RAT` uses `MiniSat` [24] as its `SATModule`, which employs CDCL(T)-style SAT solving and forwards theory calls to its backends. As theory solvers, `SMT-RAT` offers implementations of several complete and incomplete decision procedures, each of them encapsulated in a module.

The theory of *bit-vectors* is handled by the `BVModule` [42] that encodes bit-vector constraints to propositional logic similar to [26]. Another variant of theory constraints supported by `SMT-RAT` are *pseudo-Boolean* constraints – arithmetic constraints over Boolean variables. These are transformed by the `PBPPModule` [30] to either propositional or arithmetic formulas.

Linear arithmetic can be solved using the `FouMoModule`, which implements a variant of Fourier-Motzkin variable elimination, or the `LRAModule` which implements the simplex method in the spirit of [23], including support for linear integer arithmetic. Additionally the `CubeLIAModule` provides an incomplete but fast test for linear integer arithmetic inspired by [8].

The focus of our research lies however on *nonlinear arithmetic*. The cylindrical algebraic decomposition method is implemented in the `CADModule` [16, 41, 54], the only complete `SMT-RAT` module for nonlinear real arithmetic, which is complemented by different incomplete solver modules. The `GBModule` [38] uses a variant Gröbner bases to determine satisfiability over the reals. The virtual substitution method with various optimizations is implemented in the `VSModule` [15]. Another very popular method is interval constraint propagation that is available through the `ICPModule` [49]. All these modules can also work on nonlinear integer arithmetic problems using the branch-and-bound technique, though all of them are incomplete due to the undecidability of nonlinear integer arithmetic. Additionally the `IntBlastModule` [42, 41] encodes bounded nonlinear integer problems in bit-vector arithmetic, similar to what most other solvers do for this kind of problems.

Meta modules These solver modules do not implement any solving technique by themselves, but connect other modules. Meta modules extend what the user can do with the strategy framework: they allow to keep the strategy formalism comparably simple, as more complicated and technical components can be implemented as meta modules that encapsulate a specific strategic feature.

One such example is the `FPPModule` which applies a given strategy of preprocessing techniques to simplify a formula multiple times until a fixed-point is reached and no further simplifications can be done.

4.2 New modules

There are three new decision procedure modules in the latest `SMT-RAT` release.

Subtropical satisfiability The `STropModule` implements a quick check for satisfiability following the idea of [25] as briefly explained in Section 2. This method is very fast but incomplete: either it finds a satisfying solution or returns `unknown`, but it is not able to determine unsatisfiability.

Cylindrical algebraic decomposition The new version of `SMT-RAT` offers a complete re-implementation of the CAD method in the solver module `NewCADModule`. The advantage of this new CAD module lies in its data stuctures. The CAD method consists of two phases (projection and construction), both of them spanning a tree-structured search. Compared to the original `CADModule` the data structures for the projection and construction phases are more modular and allow for more flexibility. For example, the sample point construction can be performed in any heuristic order and more advanced optimizations e.g. for equational constraints could be integrated.

MCSAT-style SMT Solving As briefly mentioned in Section 2, an approach called *model-constructing satisfiability calculus* (*MCSAT*) [21] was proposed recently, firstly instantiated for nonlinear arithmetic which is called NLSAT [36]. Given its great success on nonlinear arithmetic problems, we developed a new `SMT-RAT` module to support MCSAT-style SMT solving based on the cylindrical algebraic decomposition.

As of now, `SMT-RAT` features the core solving engine for MCSAT-style solving and explanation functions based on CAD in the spirit of NLSAT and a first version using Fourier-Motzkin variable elimination for linear problems. Current work includes a more powerful implementation based on Fourier-Motzkin that also handles certain nonlinear cases, explanations based on virtual substitution for conflicts of bounded degree as described in [2] and an implementation

of the OneCell [9] approach. As for CDCL(T)-style SMT solving, we work on combining the different explanation functions in a meaningful way.

The current version of `SMT-RAT` can be compiled to an MCSAT-style SMT solver using an NLSAT-style explanation and it seems to work reliably on the SMT-LIB benchmark set. It is rather premature still and should be considered work-in-progress. Because MCSAT-style reasoning requires a close interaction between the SAT solver and the theory module, our MCSAT module is currently integrated in the SAT module and can be activated through a dedicated module configuration. In a later release we will improve the modularity for MCSAT support.

4.3 Strategic combination of solver modules

The fundamental idea of `SMT-RAT` is to use a *strategic combination* of solver modules for SMT solving. A *module* encapsulates a single solving technique and multiple modules can be composed to form a *strategy*. The *manager* takes care of parsing the input formula and executing the strategy on it, possibly exploiting opportunities to execute multiple branches of the strategy in parallel.

Every module works on a set of *received formulas* C_{rcv} that are the input to the solving technique. The module can be asked to check the consistency of the (conjunction of the) received formulas, to which the module can return `sat`, `unsat` or `unknown`. In case of satisfiability, the module may be asked to construct a *satisfying assignment* while unsatisfiability must be proven with an *infeasible subset* of C_{rcv}. Common extensions like the generation of *theory lemmas* or *multiple* infeasible subsets are supported as well.

While working on some received formula, a module may ask other modules for help by adding formulas to the set of *passed formulas* C_{pass} and ask its *backends* to decide upon the satisfiability of this (sub-)problem. The backends of a module are defined by the strategy and can be annotated with *conditionals* that specify when a backend should be used. These conditionals could check whether the current formula is linear, argues over bit-vectors or contains weak inequalities. When a module calls its backends, the manager collects all backends whose conditionals evaluate to true on C_{pass} and executes them sequentially or in parallel, according to the strategy specification. For these backend modules, C_{rcv} is identical to C_{pass} of the calling module.

We illustrate a possible strategy in Figure 1 that can solve formulas over bit-vectors and arithmetic theories. It starts with the meta-module `FPPModule` that uses the strategy `PPStrategy` to employ a series of preprocessing mod-

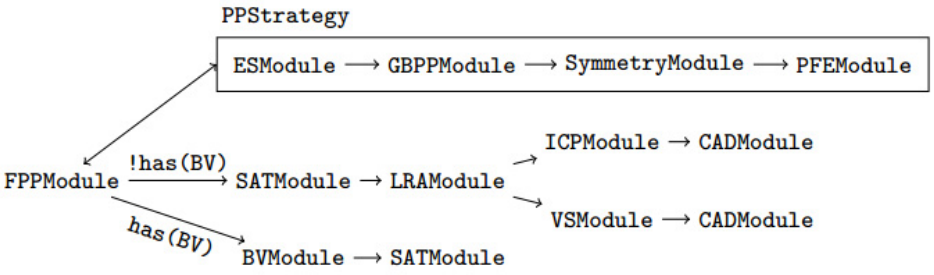

Figure 1: An example SMT-RAT strategy

ules: ESModule, GBPPModule, SymmetryModule and PFEModule. The preprocessed input is then forwarded either to the BVModule or to a SATModule that forwards theory calls to the LRAModule module. The LRAModule module tries to determine satisfiability and if it fails then it forwards the formula to its backends that use interval constraint propagation (ICPModule) or the virtual substitution (VSModule) method which are both incomplete and use the cylindrical algebraic decomposition method (CADModule) as a fallback. Note that the interval constraint propagation and the virtual substitution methods are called unconditionally and are thus both executed in parallel, the result of the first one to finish being returned to the LRAModule module.

All modules are thread-safe and can be used multiple times, for example the SATModule and the CADModule each have two instances that are completely independent of each other. In particular, different instances can be executed using different *configurations* that specify certain heuristics. The strict separation of procedures into modules that are sealed from each other is a great asset as it allows a high degree of flexibility and modularity and also significantly simplifies the implementation of new modules. This means, however, also that all modules should accept all possible input formulas as input. Thus if a module is called with a formula whose solution is not supported by the module than it should either call backend modules or return unknown.

5 Experimental results

The presented software, both CArL and SMT-RAT, are meant to be used within other projects in a community that emphasizes performance, albeit not as

much as correctness. It is therefore important that it performs reasonably well on practical problems, in particular as competitors exist. We want to point out that we do not aim to be the fastest solver for any given logic for two reasons: firstly, we consider `SMT-RAT` a framework to allow for low-threshold research on novel SMT-related research; secondly, beating all the other solvers in a particular logic would exhaust too many resources on our side. Nevertheless, we need to perform reasonably well such that using `CArL` or `SMT-RAT` makes sense at all. We therefore present a couple of experiments for both `CArL` and `SMT-RAT` to give a feeling for the level of performance that can be expected.

5.1 Computations with polynomials

First we compare the implementation of multivariate polynomials from `CArL`, `CoCoALib` and `GiNaC`. We start with basic operations to compare, multiply and divide multivariate polynomials. Furthermore we compute the pseudo-remainder and resultant of multivariate polynomials and substitute individual variables by polynomials.

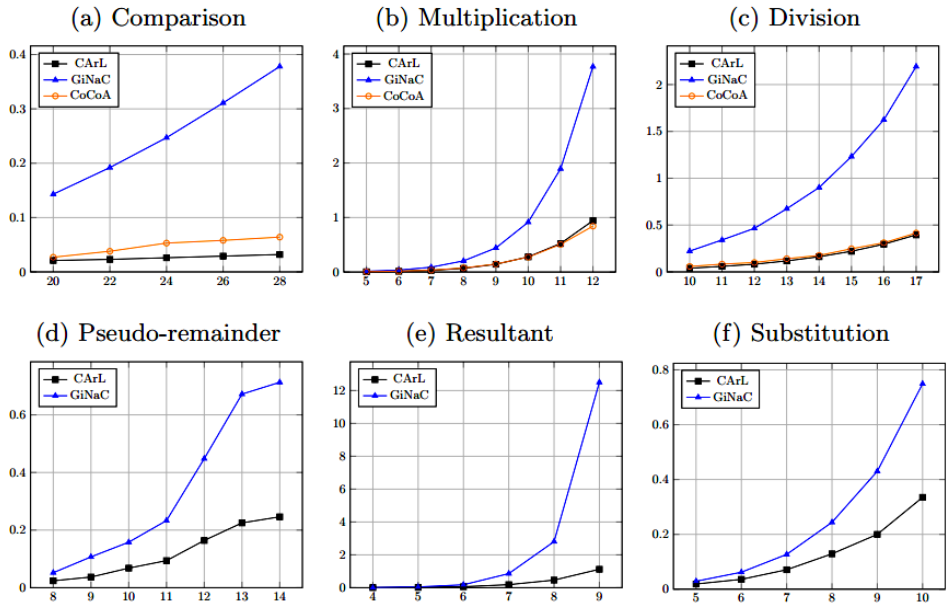

Figure 2: Experimental results for polynomial computations

Figure 2 shows some experimental results for these operations. We constructed a reasonably large set of random inputs (100 for multiplication, pseudo-remainder and resultant, 1000 for comparison, division and substitution) for every operation of the degree depicted on the x axis and give the cumulative computation time in seconds. Note that all three libraries run on the exact same inputs and the conversions from one representation to another are not included in the results. We have been unable to find an implementation for the pseudo-remainder, resultant and substitution in `CoCoALib` and therefore only compare with `GiNaC` in these cases. We can see that `CArL` significantly outperforms `GiNaC` on all operations shown here and is comparable to `CoCoALib`.

Note that some more challenging algorithms like multivariate greatest common divisor of multivariate factorization are not implemented in `CArL` directly, but instead `CArL` provides a seamless integration of either `GiNaC` or `CoCoALib`. Comparisons of these methods are therefore not meaningful.

5.2 SMT solving

Based on the finding that at least our fundamental polynomial procedures are reasonably fast, we want to investigate whether `SMT-RAT` is competitive with other state-of-the-art solvers. Past publications have shown that `SMT-RAT` usually performs pretty good, in particular on nonlinear real arithmetic [52, 16] and nonlinear integer arithmetic [41, 35].

Three methods we have recently worked on are the `CADModule`, `STropModule` and our version of MCSAT-style solving. All three of them are targeted towards nonlinear real arithmetic and we show an overview about the current status in Figure 3 on the SMT-LIB [5] `QF_NRA` benchmark set, which contains 11354 problem instances from 10 different applications. The table shows the number of instances that could be solved (as `sat` or `unsat`) and those that could not be solved due to time or memory limits (as `resout`), in our case 60 seconds and 4 GB. The `STropModule` can not be applied to certain problems which is shown as `unknown`.

For the first solver the `STropModule` is used as the sole theory solver, while it can use other theory modules as backends in the second configuration, namely the `ICPModule`, `VSModule` and `CADModule`. As for the `CADModule`, we analyzed the impact of exploiting incrementality and different heuristics. $\texttt{CADModule}_{naive}$ uses no incrementality while $\texttt{CADModule}_{A}$ and $\texttt{CADModule}_{B}$ only differ in the order used for projecting polynomials. Using the exact same basic data structures as the CAD from `CADModule`, the `MCSATModule` imple-

	sat	unsat	solved	unknown	resout
STropModule	1372	2605	3977	5620	1757
STropModule + Backends	4260	4289	8549	–	2805
CADModule $_{naive}$	2872	2699	5571	–	5783
CADModule $_{A}$	4263	3873	8136	–	3218
CADModule $_{B}$	4271	3803	8074	–	3280
MCSATModule	4297	4455	8752	–	2602

Figure 3: Experimental SMT solving results for different strategies on QF_NRA

ments a variant of the NLSAT approach. Note that these are preliminary results and none of the solvers uses our preprocessing techniques yet because we currently focus on these methods on their own. As a rough comparison, the leading solvers solved almost 9950 of the QF_NRA benchmarks at last year's SMT competition [48], though in 20 minutes instead of 60 seconds.

6 Conclusion

The implementation of formal approaches to handle arithmetic problems is highly challenging and extremely time consuming. In this paper we presented our CArL library for arithmetic computations, whose development required a serious effort. We also presented the latest version of SMT-RAT, whose development started in 2009. It required six years of work till we were able to participate in the SMT competition in 2015 the first time. Since then, our solver was enriched by further important modules like MCSAT-support based on the CAD method and a module for the subtropical satisfiability. All this work resulted in free and open-source software libraries that can be used not only in SMT-RAT but also in other software projects. Further optimizations like reduced projection in the CAD for equality constraints and further modules like MCSAT support based on the virtual substitution method are currently being implemented and will hopefully further strengthen applicability and efficiency.

Our hope is that other research groups can make use of SMT-RAT for their own research in their own tools. We have seen time and time again that SMT solvers are used as black-boxes and thus researchers cannot understand or modify the inner workings of the solver in question. We want to provide the opportunity to change that and make the customization and extension of a

reasonably good solver for a specific class of problems accessible to non-experts that are so far forced to use a monolithic black-box solver.

References

[1] J. Abbott, A. M. Bigatti, CoCoALib: A C++ library for computations in commutative algebra ... and beyond. *Proc. of ICMS'10* (2010), vol. 6327 of *LNCS*, Springer, pp. 73–76. ⇒7, 12

[2] E. Ábrahám, J. Nalbach, G. Kremer, Embedding the virtual substitution method in the model constructing satisfiability calculus framework. *Proc. of SC-square'17* (2017), vol. 1974 of *CEUR Workshop Proceedings*, CEUR-WS.org. ⇒16

[3] A. Albarghouthi, A. Gurfinkel, O. Wei, M. Chechik, Abstract analysis of symbolic executions. *Proc. of CAV'10* (2010), vol. 6174 of *LNCS*, Springer, pp. 495–510. ⇒12

[4] C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanović, T. King, A. Reynolds, C. Tinelli, CVC4. *Proc. of CAV'11* (2011), vol. 6806 of *LNCS*, Springer, pp. 171–177. ⇒9, 14

[5] C. Barrett, P. Fontaine, C. Tinelli, The Satisfiability Modulo Theories Library (SMT-LIB). `http://www.SMT-LIB.org`, 2016. ⇒20

[6] C. Bauer, A. Frink, R. Kreckel, Introduction to the GiNaC framework for symbolic computation within the C++ programming language, *Journal of Symbolic Computation* **33,** 1 (2002) 1–12. ⇒7, 12

[7] T. Bouton, D. C. B. de Oliveira, D. Déharbe, P. Fontaine, veriT: An open, trustable and efficient SMT-solver, *Proc. of CADE-22* (2009), vol. 5663 of *LNCS*, Springer, pp. 151–156. ⇒9, 14

[8] M. Bromberger, C. Weidenbach, Fast cube tests for LIA constraint solving. *Proc. of IJCAR'16* (2016), Springer, pp. 116–132. ⇒15

[9] C. W. Brown, M. Košta, Constructing a single cell in cylindrical algebraic decomposition, *Journal of Symbolic Computation* **70** (2015) 14–48. ⇒17

[10] B. Buchberger, Gröbner bases: Applications. in: *The Concise Handbook of Algebra*. Kluwer Academic Publishers, 2002, pp. 265–268. ⇒12

[11] P.-S. Chen, Y.-S. Hwang, R. D.-C. Ju, J.-K. Lee, Interprocedural probabilistic pointer analysis, *IEEE Trans. Parallel Distrib. Syst.* **15,** 10 (2004) 893–907. ⇒12

[12] A. Cimatti, A. Griggio, B. Schaafsma, B., R. Sebastiani, The MathSAT5 SMT solver. *Proc. of TACAS'13*, vol. 7795 of *LNCS*. Springer, 2013, pp. 93–107. ⇒9

[13] G. E. Collins, Quantifier elimination for real closed fields by cylindrical algebraic decomposition, *Automata Theory and Formal Languages* (1975), vol. 33 of *LNCS*, Springer, pp. 134–183. ⇒11, 12

[14] F. Corzilius, *Virtual substitution in SMT solving*, Diploma thesis, RWTH Aachen University, 2011. ⇒14

[15] F. Corzilius, *Integrating Virtual Substitution into Strategic SMT Solving*. PhD thesis, RWTH Aachen University, 2016. ⇒15

[16] F. Corzilius, G. Kremer, S. Junges, S. Schupp, E. Ábrahám, SMT-RAT: An open source C++ toolbox for strategic and parallel SMT solving. *Proc. of SAT'15* (2015), vol. 9340 of *LNCS*, Springer, pp. 360–368. ⇒6, 7, 9, 15, 20

[17] F. Corzilius, U. Loup, S. Junges, S., E. Ábrahám, SMT-RAT: An SMT-compliant nonlinear real arithmetic toolbox. *Proc. of SAT'12* (2012), vol. 7317 of *LNCS*, Springer, pp. 442–448. ⇒6, 7

[18] G. B. Dantzig, *Linear programming and extensions*. Princeton University Press, 1963. ⇒10

[19] M. Davis, H. Putnam, A computing procedure for quantification theory. *Journal of the ACM* **7,** 3 (1960) 201–215. ⇒8

[20] L. de Moura, N. Bjørner, Z3: An efficient SMT solver. *Proc. of TACAS'08* (2008), vol. 4963 of *LNCS*, Springer, pp. 337–340. ⇒9, 14

[21] L. M. de Moura, D. Jovanovic, A model-constructing satisfiability calculus. *Proc. of VMCAI'13* (2013), vol. 7737 of *LNCS*, Springer, pp. 1–12. ⇒9, 16

[22] B. Dutertre, Yices 2.2. *Proc. of CAV'14* (2014), vol. 8559 of *LNCS*, Springer, pp. 737–744. ⇒9, 14

[23] B. Dutertre, L. M. de Moura, A fast linear-arithmetic solver for DPLL(T). *Proc. of CAV'06* (2006), vol. 4144 of *LNCS*, Springer, pp. 81–94. ⇒15

[24] N. Eén, N. Sörensson, An extensible SAT-solver. *Proc. of SAT'03* (2004), vol. 2919 of *LNCS*, Springer, pp. 502–518. ⇒15

[25] P. Fontaine, M. Ogawa, T. Sturm, T., X. T. Vu, Subtropical satisfiability. *Proc. of FroCoS'17* (2017), Springer, pp. 189–206. ⇒11, 16

[26] C. Fuhs, J. Giesl, A. Middeldorp, P. Schneider-Kamp, R. Thiemann, H. Zankl, SAT solving for termination analysis with polynomial interpretations. *Proc. of SAT'07* (2007), Springer, pp. 340–354. ⇒15

[27] S. Gao, M. Ganai, F. Ivančić, A. Gupta, S. Sankaranarayanan, E. M. Clarke, Integrating ICP and LRA solvers for deciding nonlinear real arithmetic problems. *Proc. of FMCAD'10* (2010), IEEE, pp. 81–90. ⇒10

[28] K. O. Geddes, S. R. Czapor, G. Labahn, *Algorithms for Computer Algebra*. Kluwer Academic Publishers, 1992. ⇒7

[29] J. Giesl, M. Brockschmidt, F. Emmes, F. Frohn, C. Fuhs, C. Otto, M. Plücker, P. Schneider-Kamp, T. Ströder, S. Swiderski, R. Thiemann, Proving termination of programs automatically with AProVE. *Proc. of IJCAR'14* (2014), vol. 8562 of *LNAI*, Springer, pp. 184–191. ⇒9

[30] M. Grobelna, SAT-modulo-theories solving for pseudo-Boolean constraints. Bachelor's Thesis, RWTH Aachen University, 2017. ⇒14, 15

[31] R. Haehn, Using equational constraints in an incremental CAD projection. Master's thesis, RWTH Aachen University, 2017. ⇒14

[32] E. M. Hahn, H. Hermanns, B. Wachter, L. Zhang, PARAM: A model checker for parametric Markov models. *Proc. of CAV'10* (2010), vol. 6174 of *LNCS*, Springer, pp. 660–664. ⇒12

[33] W. Hentze, Infeasible subsets for nonlinear SMT. Bachelor's Thesis, RWTH Aachen University, 2017. ⇒14
[34] S. Herbort, D. Ratz, Improving the efficiency of a nonlinear-system-solver using a componentwise Newton method. Tech. Rep. 2/1997, Inst. für Angewandte Mathematik, University of Karlsruhe, 1997. ⇒10
[35] D. Jovanović, Solving nonlinear integer arithmetic with MCSAT. *Proc. of VMCAI'17* (2017), Springer, pp. 330–346. ⇒20
[36] D. Jovanović, L. de Moura, Solving non-linear arithmetic. *Proc. of IJCAR'12* (2012), vol. 7364 of *LNAI*, Springer, pp. 339–354. ⇒16
[37] S. Junges, On Gröbner bases in SMT-compliant decision procedures. Bachelor's Thesis, RWTH Aachen University, 2012. ⇒14
[38] S. Junges, U. Loup, F. Corzilius, E. Ábrahám, E. On Gröbner bases in the context of satisfiability-modulo-theories solving over the real numbers. *Proc. of CAI'13* (2013), vol. 8080 of *LNCS*, Springer, pp. 186–198. ⇒15
[39] T. V. Khanh, X. Vu, M. Ogawa, raSAT: SMT for polynomial inequality. *Proc. of SMT'14* (2014), p. 67. ⇒9
[40] G. Kremer, Isolating real roots using adaptable-precision interval arithmetic. Master's thesis, RWTH Aachen University, 2013. ⇒14
[41] G. Kremer, F. Corzilius, E. Ábrahám, A generalised branch-and-bound approach and its application in SAT modulo nonlinear integer arithmetic. *Proc. of CASC'16* (2016), vol. 9890 of *LNCS*, Springer, pp. 315–335. ⇒11, 15, 20
[42] A. Krüger, Bitvectors in SMT-RAT and their application to integer arithmetics. Master's thesis, RWTH Aachen University, 2015. ⇒14, 15
[43] J. P. Marques-Silva, K. A. Sakallah, Grasp: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers* **48** (1999) 506–521. ⇒8
[44] J. Nalbach, Embedding the virtual substitution in the MCSAT framework. Bachelor's Thesis, RWTH Aachen University, 2017. ⇒14
[45] L. Netz, Using Horner schemes to improve the efficiency and precision of interval constraint propagation. Bachelor's Thesis, RWTH Aachen University, 2015. ⇒14
[46] L. Neuberger, Generation of infeasible subsets in less-lazy SMT-solving for the theory of uninterpreted functions. Bachelor's Thesis, RWTH Aachen University, 2015. ⇒14
[47] J. Redies, An extension of the GiNaCRA library for the cylindrical algebraic decomposition. Bachelor's Thesis, RWTH Aachen University, 2012. ⇒14
[48] SMT-COMP 2017 result summary. `http://smtcomp.sourceforge.net/2017/results-toc.shtml`, 2017. ⇒21
[49] S. Schupp, Interval constraint propagation in SMT compliant decision procedures. Master's thesis, RWTH Aachen University, 2013. ⇒14, 15
[50] D. Scully, Preprocessing for solving non-linear real-arithmetic formulas. Bachelor's Thesis, RWTH Aachen University, 2012. ⇒14

[51] G. S. Tseitin, On the complexity of derivation in propositional calculus. in: *Automation of Reasoning*. Springer, 1983, pp. 466–483. ⇒8

[52] V. X. Tung, T. Van Khanh, M. Ogawa, raSAT: An SMT solver for polynomial constraints. *Formal Methods in System Design* **51,** 3 (2017), 462–499. ⇒14, 20

[53] T. Viehmann, Projection operators for the CAD. Bachelor's Thesis, RWTH Aachen University, 2016. ⇒14

[54] T. Viehmann, G. Kremer, E. Ábrahám, Comparing different projection operators in the cylindrical algebraic decomposition for SMT solving. *Proc. of SC-square'17* (2017), vol. 1974 of *CEUR Workshop Proceedings*, CEUR-WS.org. ⇒15

[55] V. Weispfenning, Quantifier elimination for real algebra – the quadratic case and beyond. *Appl. Algebra Eng. Commun. Comput.* **8,** 2 (1997), 85–101. ⇒10

[56] T. Winkler, Using Thom's lemma for real algebraic numbers in the CAD. Bachelor's Thesis, RWTH Aachen University, 2016. ⇒14

Received: May 15, 2018 • Revised: July 23, 2018