

# On Gröbner Bases in the Context of Satisfiability-Modulo-Theories Solving over the Real Numbers<sup>\*</sup>

Sebastian Junges, Ulrich Loup, Florian Corzilius and Erika Ábrahám<sup>\*\*</sup>

RWTH Aachen University, Germany

**Abstract.** We address satisfiability checking for the first-order theory of the real-closed field (RCF) using *satisfiability-modulo-theories (SMT)* solving. SMT solvers combine a *SAT solver* to resolve the Boolean structure of a given formula with *theory solvers* to verify the consistency of sets of theory constraints.

In this paper, we report on an integration of *Gröbner bases* as a theory solver so that it conforms with the requirements for efficient SMT solving: (1) it allows the incremental adding and removing of polynomials from the input set and (2) it can compute an inconsistent subset of the input constraints if the Gröbner basis contains 1.

We modify Buchberger’s algorithm by implementing a new update operator to optimize the Gröbner basis and provide two methods to handle inequalities. Our implementation uses special data structures tuned to be efficient for huge sets of sparse polynomials. Besides solving, the resulting module can be used to simplify constraints before being passed to other RCF theory solvers based on, e.g., the cylindrical algebraic decomposition.

## 1 Introduction

Formulas of first-order logic over the theory of the *real-closed field* (RCF) are Boolean combinations of polynomial constraints with real-valued variables. Be it the analysis of real-time systems, the optimization of railway schedules or the computation of dense sphere packings in Euclidean space, many practical and theoretical problems can be expressed in this logic. Sophisticated decision procedures and increased computational power have led to efficient tools to analyze such formulas.

Boolean formulas are well-suited for the description of discrete systems, e.g., digital controllers. State-of-the-art *SAT solvers*, dedicated programs to determine the satisfiability of Boolean formulas, are highly tuned for efficiency. They

---

<sup>\*</sup> The original publication is available at <http://www.springerlink.com>.

<sup>\*\*</sup> This work has been partially supported by the German Research Council (DFG) as part of the Research Training Group “AlgoSyn” (GRK 1298, <http://www.algosyn.rwth-aachen.de/>).

can handle formulas with millions of literals and are frequently used not only in academic research but also in industry.

The success of SAT solvers has led to an approach called *satisfiability-modulo-theories* (SMT) solving for handling first-order logic over certain theories. This approach combines the high efficiency of SAT solvers to handle the Boolean structure with dedicated *theory solvers* to check sets of constraints from the given theory for consistency. For the optimal combination of these modules, theory solvers should be *SMT compliant*: they should support the extension of the constraint set (*incrementality*), the removal of constraints (*backtracking*) and the generation of small *infeasible subsets* in case of inconsistency [2][Ch. 26].

In this paper, we consider the existential fragment of the first-order logic over the theory of the RCF. Immense advances have been made in this area in the last decades. Besides complete decision procedures as the *cylindrical algebraic decomposition* (CAD) method [4], e.g. implemented in the tool QEPCAD, also incomplete methods such as the *virtual substitution* (VS) method [15], e.g. available in the package Redlog of the computer algebra system Reduce, *simplex* [7] or *interval constraint propagation* [8], e.g. implemented in iSAT, are available. In addition to such explicit methods working on the solution space, some symbolic approaches find application in SMT solving for preprocessing by using simple rules and basic Gröbner basis computations, or outside of SMT solvers in standalone tools, often based on some application of the Positivstellensatz [13] such as in the tool KeYmaera.

We aim to improve the integration of the *Gröbner bases* methodology in SMT solving, thereby enhancing speed and effectiveness. To reach this goal, we have to overcome several challenges. (1) The methodology has to be adapted to be *SMT compliant* and (2) to cope with typical *SMT-problem structures*, which often significantly differ from algebraically hard problems. (3) As we are solving over the RCF, we are more interested in the *real radical* than the ideal of our input polynomials. (4) Finally, we need to handle *inequalities* as well.

*Gröbner basis* computations are used for preprocessing in [6] and [11]. [13] proposes a combination of Gröbner basis computations with the Fourier-Motzkin method. However, this work is not directly related to SMT. Direct relation to SMT can be found in [10] for finding minimal infeasible subsets, and in [12] for coping with the special structure. Saturation to approximate the real radical is used in [11] and in [13].

We implement our approaches as a module in the SMT-solving framework SMT-RAT, which is a C++ toolbox allowing the combination of different theory solvers in a user-defined strategy. Our *Gröbner bases module* can be applied both as a preprocessing and as a solving technique.

Regarding (1), our Gröbner bases module supports the adding and removal of constraints as well as the computation of small infeasible subsets. The basic features of this module are the simplification of equations and the check whether there are common zeros of the input equations. To tackle (2), we utilize some ideas from [12] and [14] to develop data structures that can handle a large number of variables and huge sets of sparse input polynomials, not necessarily

of low degree, as they frequently occur in our setting. For (3), we further adapt Buchberger’s algorithm in that we prune polynomials without real zeros in the Gröbner basis. We implemented two different strategies to realize (4): firstly, we can encode all inequalities as equations and compute a Gröbner basis of the extended set of polynomials, or secondly, we reduce the polynomials belonging to the inequalities modulo the Gröbner basis for the equations.

The rest of the paper is structured as follows: In Section 2 we recall some basics for Gröbner bases. In Section 3 we describe our SMT framework before explaining our methods and their integration in Section 4. After giving some experimental results in Section 5, we conclude the paper in Section 6.

## 2 Preliminaries

We denote the set of *real*, *rational* and *natural* numbers by  $\mathbb{R}$ ,  $\mathbb{Q}$  and  $\mathbb{N}$  ( $0 \in \mathbb{N}$ ) respectively. We use  $\mathbb{R}$  and  $\mathbb{Q}$  also for the corresponding (*ordered*) *fields* over the arithmetic operations  $+$ ,  $\cdot$  and the ordering relation  $<$ . W.l.o.g., we refer to  $\mathbb{R}$  as the *real-closed field* (*RCF*). We omit the symbol  $\cdot$  when the context is clear. We abbreviate sequences of variables  $x_1, \dots, x_n$ ,  $n \geq 1$ , by  $\bar{x}$ .

Let  $K$  be a field.  $K[\bar{x}]$  denotes the *polynomial ring* over  $K$  in the variables  $\bar{x}$ . We call a product  $m = \prod_{1 \leq i \leq n} x_i^{d_i}$  with  $d_i \in \mathbb{N}$  a *monomial* having the *degree*  $\deg(m) := \sum_{1 \leq i \leq n} d_i$ . With  $M_{\bar{x}}$  we denote the set of all monomials in  $\bar{x}$ . A product  $a \cdot m$  with  $a \in K$  and  $m \in M_{\bar{x}}$  is called a *term* and  $a$  the *coefficient* of  $m$ . Hence, a polynomial  $p \in K[\bar{x}]$  is a sum of terms. We say that  $x_i \in p$  if  $x_i$  occurs in the polynomial  $p \in K[\bar{x}]$ . We define the *total degree* of  $p$  as  $\text{tdeg}(p) := \max\{\deg(m) \mid m \text{ monomial in } p\}$ . A *monomial ordering* is a linear well-ordering on monomials respecting multiplication of monomials, i.e., a linear ordering  $\prec$  with a minimal element such that  $m_1 \prec m_2$  entails  $m_1 m_3 \prec m_2 m_3$  for all  $m_1, m_2, m_3 \in M_{\bar{x}}$ . By  $\text{lm}(p)$  we denote the *leading monomial* of  $p$ , i.e., the maximal monomial w.r.t. the current ordering. Analogously, we define  $\text{lt}(p)$  to be the *leading term* of  $p$ . The coefficient of  $\text{lt}(p)$  is called *leading coefficient*, denoted by  $\text{lc}(p)$ . It holds that  $\text{lt}(p) = \text{lc}(p)\text{lm}(p)$  for all polynomials  $p \in K[\bar{x}]$ .

Let  $p \in \mathbb{Q}[\bar{x}]$ . We call  $p \sim 0$  a (*polynomial*) *constraint* over  $p$  if and only if  $\sim \in \{=, >, \geq, \neq\}$ . For  $P \subseteq K[\bar{x}]$  and  $C$  a set of constraints over  $P$  we define  $\text{pol}(C) := P$ . Our input formulas are quantifier-free first-order formulas over polynomial constraints, i.e., Boolean combinations connected by  $\wedge, \vee, \neg$  of constraints. We refer to such formulas as *RCF formulas*. Note that we only consider the existential fragment of the first-order theory of the RCF here.

### 2.1 Gröbner bases

We briefly introduce *Gröbner bases* and an application to solve real-algebraic constraint systems. More information can be found in [1].

Let  $R = \mathbb{Q}[\bar{x}]$  with a fixed monomial ordering. Given a finite set of polynomials  $P \subseteq R$ , we define the *ideal generated by  $P$*  as the set  $\langle P \rangle := \{\sum_{p \in P} r_p p \mid r_p \in R \text{ for each } p \in P\}$ . Note that the more general notion of an ideal is also

covered by our definition because, due to Hilbert's basis theorem, every ideal in  $R$  has a finite set of generators. By  $\mathcal{V}_K(\langle P \rangle) := \{a \in K \mid p(a) = 0 \text{ for all } p \in P\}$  we denote the  $K$  variety of  $P$ , i.e., the set of common zeros of  $P$  in  $K$ .

*Reduction.* Let  $p, p', f \in R$  with  $p, f \neq 0$ ,  $p = \sum_{i=0}^k a_i m_i$ ,  $k \in \mathbb{N}$  and let  $F \subseteq R$ . If  $p' = p - sf$  for some  $s \in R$  such that  $s \cdot \text{lt}(f) = a_i m_i$  for some  $i \in \{1, \dots, k\}$  then  $p$  reduces to  $p'$  modulo  $f$ , written  $p \xrightarrow{f} p'$ . We call  $f$  the *reductor* of  $p$ . We say that  $p$  reduces to  $p'$  modulo  $F$ , written  $p \xrightarrow{F} p'$ , if  $p \xrightarrow{f} p'$  for some  $f \in F$ . If no  $f \in F$  with  $p \xrightarrow{f} p'$  exists,  $p$  is in *normal form modulo  $F$* . If  $p \xrightarrow{F} \dots \xrightarrow{F} p'$  and  $p'$  is in normal form modulo  $F$  then we call  $p'$  the *normal form of  $P$  modulo  $F$* , denoted by  $\text{red}_F(p)$ .

**Definition 1 (Gröbner basis)**

Let  $P \subseteq R$ . A finite set  $G \subseteq \langle P \rangle$  is called a Gröbner basis (GB) of  $\langle P \rangle$  if  $\langle \{\text{lt}(g) \mid g \in G\} \rangle = \langle \{\text{lt}(p) \mid p \in P\} \rangle$ . Let  $\text{lc}(p) = 1$  for all  $p \in G$ . We call  $G$  minimal if  $\text{lt}(g) \notin \langle \text{lt}(\tilde{g}) \mid \tilde{g} \in G \setminus \{g\} \rangle$  for all  $g \in G$ , and reduced if  $m \notin \langle \text{lt}(\tilde{g}) \mid \tilde{g} \in G \setminus \{g\} \rangle$  for all monomials  $m$  of  $g$ .

We always regard a reduced GB, which is unique for a given monomial ordering. If the reduced Gröbner basis of  $\langle P \rangle$  is  $\{1\}$  then  $\mathcal{V}_R(\langle P \rangle) = \emptyset$ , i.e.,  $P$  has no common zeros.

*Buchberger's algorithm.* In his PhD thesis, Bruno Buchberger suggested a simple fixed-point iteration algorithm for computing a Gröbner basis [3] (see Listing 1). The most important tool in Buchberger's algorithm is the S-polynomial: Let  $p, q \in R$  with  $\text{lm}(p) = \prod_{i=1}^n x_i^{d_i}$  and  $\text{lm}(q) = \prod_{i=1}^n x_i^{\tilde{d}_i}$ , then the least common multiple of  $\text{lm}(p)$  and  $\text{lm}(q)$  is  $\text{lcm}(\text{lm}(p), \text{lm}(q)) = \prod_{i=1}^n x_i^{\max(d_i, \tilde{d}_i)} =: l$ . We define  $S(p, q) := \frac{l}{\text{lt}(p)} \cdot p - \frac{l}{\text{lt}(q)} \cdot q$  to be the *S-polynomial of  $p$  and  $q$* . All possible S-polynomials are computed during Buchberger's algorithm. We refer to a pair  $(p, q)$  whose S-polynomial is not yet computed as *S-pair*.

We call a mapping  $U : 2^R \times R \rightarrow 2^R$  an *update operator*, where  $2^R$  denotes the power set of  $R$ . Buchberger's algorithm uses the *standard update operator*  $U_{\text{std}}(G, s) = G \cup \{s\}$ .

A reduced Gröbner basis can be obtained by iteratively removing each polynomial whose leading term is a multiple of another leading term, and applying reduction modulo  $G \setminus \{p\}$  for the remaining  $p \in G$ , see [1][Table 5.5].

### 3 SMT-RAT

In this section, we give a short overview of our toolbox SMT-RAT [5], in which we embed our Gröbner bases implementation. The core procedure of Buchberger's algorithm and its underlying data structures are implemented in the extension GiNaCRA of the GiNaC library.

Listing (1) Buchberger’s algorithm.

```

1 Input: Set of polynomials  $P$ 
2 Output: Gröbner basis  $G$  for  $\langle P \rangle$ 
3
4
5  $G := P$ 
6 while true:
7    $G' := G$ 
8   for each  $\{p, q\} \subseteq G', p \neq q$ :
9      $s := \text{red}_G(\overline{S(p, q)})$ 
10    if  $s \neq 0$ :  $G := \text{U}_{\text{std}}(G, s)$ 
11    if  $G = G'$ : break
12 return  $G$ 

```

Listing (2) GB module consistency check.

```

1 Input:  $C_{\text{rcv}}$ , state  $(A, G)$ 
2 Output:  $(ans, C_{\text{inf}})$ , with  $C_{\text{inf}} \subseteq C_{\text{rcv}}$ 
3         and  $ans \in \{\text{sat}, \text{unsat}, \text{unknown}\}$ 
4
5 if  $A \neq \emptyset$ :
6    $G := \text{Groebner}(G \cup A)$ 
7    $A := \emptyset$ 
8   if  $G = \{1\}$ : return  $(\text{unsat}, C_{\text{rsn}}(1 = 0))$ 
9    $C_{\text{pas}} := (C_{\text{rcv}} \setminus C_{\text{rcv}}[=]) \cup \{p = 0 \mid p \in G\}$ 
10   $(r, C'_{\text{inf}}) := \text{runBackends}(C_{\text{pas}})$ 
11  determine  $C_{\text{inf}}$  from  $C'_{\text{inf}}$ 
12  return  $(r, C_{\text{inf}})$ 

```

*Framework.* SMT-RAT is a C++ library consisting of (1) a collection of SMT-compliant theory solver modules which can be used to extend an existing SMT solver to RCF and (2) an SMT solver in which these modules can be (and most of them are) integrated to tackle RCF. The latter is intended to be a testing environment for the development of SMT-compliant theory solvers, as the one presented in this paper. SMT-RAT defines three types of components (see [9, Appendix B]): *manager*, *strategy* and *module*. In the following we first describe the functionality of a module and show how the manager composes different modules according to a strategy to a solver.

*Modules.* The main procedure of a module is `check( $C_{\text{rcv}}$ )`. For a given set  $C_{\text{rcv}}$  of RCF formulas, called *the set of received formulas*, the procedure either decides whether  $C_{\text{rcv}}$  is satisfiable or not returning *sat* or *unsat*, respectively, or returns *unknown*. Note, that a set of formulas is semantically defined by their conjunction. We can manipulate the set of received formulas by adding (removing) formulas  $\varphi$  to (from) it with `add( $\varphi$ )` (`remove( $\varphi$ )`). Since in the SMT embedding  $C_{\text{rcv}}$  is usually changed between two consecutive `check( $C_{\text{rcv}}$ )` calls only by adding/removing constraints, the solver’s performance can be significantly improved if the modules can make use of the results of previous checks (*incrementality* and *backtracking*). In case that the module determines the unsatisfiability of  $C_{\text{rcv}}$ , it is expected to compute at least one preferably small *infeasible subset*  $C_{\text{inf}} \subseteq C_{\text{rcv}}$ . Moreover, a module has the possibility to name lemmas, which are RCF tautologies. These lemmas should encapsulate information which can be extracted from a module’s internal state and propagated among other SMT-RAT modules. Furthermore, SMT-RAT provides the feature that a module itself can ask other modules for the satisfiability of a set  $C_{\text{pas}}$  of RCF formulas, called *the set of passed formulas*, using the procedure `runBackends( $C_{\text{pas}}$ )` which is controlled by the manager.

This paper presents the implementation of a new SMT-RAT module called  $M_{\text{GB}}$  based on Gröbner bases; the next section gives details on its implementation. SMT-RAT already contains various modules implementing, among others a conjunctive normal form transformer  $M_{\text{CNF}}$ , a SAT solver  $M_{\text{SAT}}$  and the modules  $M_{\text{LRA}}$  for simplex,  $M_{\text{VS}}$  for VS and  $M_{\text{CAD}}$  for CAD. Note that most of these procedures are

not complete. If a module cannot solve a problem then it either returns *unknown* or consults another module as explained below.

*Manager and strategy.* A *strategy* is a directed tree  $T := (V, E)$  with a set  $V$  of module instances as nodes and  $E \subseteq V \times \Omega \times V$ , where  $\Omega$  is a set of conditions. Initially, the *manager* calls the method  $\text{check}(C_{\text{rcv}})$  of the module instance given by the root of the strategy, where  $C_{\text{rcv}}$  is a set of RCF formulas. Whenever a module instance  $m \in V$  calls  $\text{runBackends}(C_{\text{pas}})$ , the manager calls  $\text{check}(C_{\text{pas}})$  of each module  $m'$ , for which an edge  $(m, \omega, m') \in E$  exists such that  $\omega$  holds for  $C_{\text{pas}}$ , and passes the results back to  $m$ . Furthermore, it also passes back the infeasible subsets and lemmas provided by the invoked modules. The module  $m$  can now benefit in its solving and reasoning process from this shared information. In the following we write short  $(m, m')$  for  $(m, \omega, m)$  if  $\omega = \text{True}$ .

Usually, the root module  $M_{\text{CNF}}$  transforms its set of received formulas  $C_{\text{rcv}}$  to an equisatisfiable set of clauses  $C_{\text{pas}}$  and calls  $\text{runBackends}(C_{\text{pas}})$ . The backend is a SAT-solver module  $M_{\text{SAT}}$ , which runs DPLL-style SAT-solving on the Boolean abstraction of the set of received clauses  $C_{\text{rcv}}$ .  $M_{\text{SAT}}$  might call  $\text{runBackends}(C_{\text{pas}})$  for partial Boolean assignments on the corresponding set of formulas  $C_{\text{pas}}$ ; we refer to such a backend call as *theory call*. The Boolean abstraction of the obtained infeasible subsets and lemmas are stored as additional clauses. Infeasible subsets and lemmas, which contain only formulas from  $C_{\text{rcv}}$ , prune the Boolean search space and hence the number of theory calls. Smaller infeasible subsets are usually more advantageous, because they make larger cuts in the search space. Other types of lemmas contain new formulas, so-called *inventive lemmas* (*non-inventive* otherwise) and might enlarge the Boolean search space, but they can reduce the complexity of later theory calls. This way we can compose SMT solvers for RCF, e.g., using the simple strategy defined by the nodes  $I_{M_{\text{CNF}}}$ ,  $I_{M_{\text{SAT}}}$  and  $I_{M_{\text{CAD}}}$  and the edges  $(I_{M_{\text{CNF}}}, I_{M_{\text{SAT}}})$  and  $(I_{M_{\text{SAT}}}, I_{M_{\text{CAD}}})$ .

## 4 Applying Gröbner bases

In this section we describe our SMT-RAT module  $M_{\text{GB}}$  applying Gröbner bases (GB) computations. In Section 4.1 we discuss how its design wraps a GB procedure such as Buchberger’s algorithm, while leaving the GB procedure itself untouched. In turn, Section 4.2 comprises how Buchberger’s algorithm can be adapted to work inside an SMT-RAT module. Moreover, we show how to treat inequalities in Section 4.3, how to realize a tighter SMT integration by giving lemmas in Section 4.4, and an extension to the GB module  $M_{\text{GB}}$  which makes it more suitable for preprocessing in Section 4.5.

In this section we assume  $C_{\text{rcv}}$  to be a set of constraints. Given a constraint  $c$ , a set of constraints  $C$  and a set of polynomials  $P$ , we use  $C[\sim] = \{p \sim 0 \mid p \sim 0 \in C\}$  to select constraints and  $C_{\sim}(P) := \{p \sim 0 \mid p \in P\}$  to construct constraints from polynomials. We call  $C_{\text{rsn}}(c) \subseteq C_{\text{rcv}}$  a *reason set of  $c$*  if  $\bigwedge_{r \in C_{\text{rsn}}(c)} r \implies c$  and  $C_{\text{rsn}}(C) = \bigcup_{c \in C} C_{\text{rsn}}(c)$  a reason set of  $C$ .

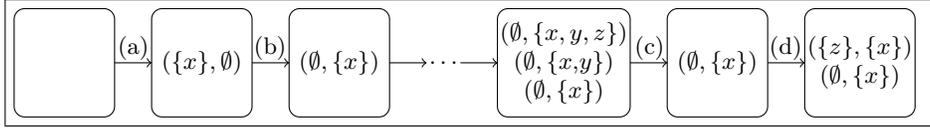


Fig. 1: The state stack in the GB module.

#### 4.1 SMT-compliant consistency checking

In this section we show how consistency checking in an SMT-RAT module based on a Gröbner bases core procedure can be accomplished. We do not further specify this core procedure here. It is thus possible to plug in an off-the-shelf GB procedure implementation such as the one in **Singular**.

The input consists of a set  $C_{\text{rcv}}$  of received constraints and the set of constraints arrived since the last consistency check. We call a tuple  $(A, G) \subseteq \mathbb{Q}[\bar{x}] \times \mathbb{Q}[\bar{x}]$  a (*GB module*) *state* if  $A \subseteq \text{pol}(C_{\text{rcv}})$  is the set of polynomials added since the last consistency check and  $G$  is a Gröbner basis for  $\langle \text{pol}(C_{\text{rcv}}[=]) \setminus A \rangle$ .

The *incremental* consistency check procedure is given in Listing 2. It operates on  $C_{\text{rcv}}$  and the state  $(A, G)$ . The procedure possibly updates the state  $(A, G)$  and outputs, first, an answer as to whether  $C_{\text{rcv}}$  is *sat*, *unsat* or its consistency is *unknown*, and second, a subset of  $C_{\text{rcv}}$  building an infeasible subset  $C_{\text{inf}}$  in case of the answer *unsat*. The first step in the procedure is the computation of a Gröbner basis of all polynomials appearing on the left-hand-side in  $C_{\text{rcv}}[=]$  (line 5 in Listing 2). Thereby we recompute the GB only if  $A \neq \emptyset$ . Then, we reuse  $G$  for the computation of the GB of  $\text{pol}(C_{\text{rcv}}[=])$ , what is possible because  $\langle \text{pol}(C_{\text{rcv}}[=]) \rangle = \langle G \cup A \rangle$ . If the Gröbner basis is  $\{1\}$ , the polynomials have no common real zeros; hence, we determine the infeasible subset  $C_{\text{inf}}$  as reason set of  $1 = 0$  (details below) and return *unsat*. Otherwise, we call a module with the same inequations, and instead of the original equations, we pass equations formed by the Gröbner basis. In the following, we describe the extensions around the algorithm in Listing 2 to provide the SMT compliance.

*Backtracking.* As in SMT solving constraints can be removed from theory solvers, we make bookkeeping of the GB module states. Because SAT solvers mostly use *chronological backtracking* we use a stack of states  $((A_0, G_0), \dots, (A_k, G_k))$ ,  $k \in \mathbb{N}$ , illustrated in Figure 1: We start with an empty stack. Whenever an equality is added, we add a state to the stack (a). After each consistency check, we update the topmost state from the stack (b). If an equality is removed, we remove all states from the stack which were added afterwards (c). Then, we add the polynomials which were added after the just removed equality iteratively, like a new equality (d).

*Infeasible subsets* As argued before, the module is expected to return a subset  $C_{\text{inf}} \subset C_{\text{rcv}}$  in case the set of received constraints  $C_{\text{rcv}}$  is inconsistent.

To determine such a subset, in [10] *certificates* for inconsistency were introduced. It was also shown that minimality of these certificates is a problem which

is as hard as calculating the Gröbner basis. These certificates are basically tuples of polynomials  $(h_1, \dots, h_n)$  such that for an ideal  $I = \langle f_1, \dots, f_n \rangle$  and a polynomial  $p \in I$  we have  $\sum_{i=1}^n h_i f_i = p$  for suitable  $h_i \in K[\bar{x}]$ . In the case of inconsistency, we have  $p = 1$ . Calculating certificates requires the reductions within the Gröbner basis calculation to be extended to ordinary divisions, which is certainly less efficient. As we do this calculation for all reason sets, we implemented a more naive way. The realization of smaller reason and infeasible sets is obvious under the assumption that our GB procedure returns reason sets for each  $p \in G$ , with  $G$  a GB.

## 4.2 Our Gröbner bases procedure

We describe the adaptations to Buchberger’s algorithm according to our setting of being called in an SMT-compliant way. The implementation is based on the description in [1].

*Incrementality.* As we call the GB procedure incrementally (cf. line 6 in Listing 2), we usually have to calculate Gröbner bases of  $G \cup A$  for some set of polynomials  $A$  where  $G$  is a GB already. Instead of using Buchberger’s algorithm from scratch, we skip all S-pairs  $(g_1, g_2)$ ,  $g_1, g_2 \in G$  as they reduce to zero.

*Reason set calculation.* We calculate the *origin set*  $C_{\text{org}}(p)$  of a polynomial  $p$  as follows: If  $p$  is added to our module,  $C_{\text{org}}(p) = \{p\}$ . Furthermore, for  $p = S(p_1, p_2)$  and  $p_1 \xrightarrow{p_2} p$ , we set  $C_{\text{org}}(p) = C_{\text{org}}(p_1) \cup C_{\text{org}}(p_2)$ . Then  $C_{\text{rsn}}(p = 0) = C_{\text{org}}(p)$ . The set representations are realized by bit vectors and therefore taking the union costs at most a couple of machine operations.

*Data structures.* We base our implementation of data structures on [14], e.g., we use a *compressed heap* during the reduction and for storing S-pairs. However, the term and ideal representations are adapted based on the following observations: The number of variables in the system is usually high and, due to incrementality, we do not have a fixed bound on the number of variables at initialization. However, most polynomials appearing are sparse, i.e., they consist of only few terms, each having small number of variables.

A term  $a \cdot \prod_{i=1}^n x_i^{d_i}$  is represented as  $(a, [(x_{i_1}, d_{i_1}), \dots, (x_{i_k}, d_{i_k})], \sum_{i=1}^n d_i)$  with  $d_{i_j} \neq 0$  for all  $1 \leq j \leq k$  and  $i_j < i_{j+1}$  for all  $1 \leq j < k$ . In our context, this representation seems more suitable than those from [14]. The degree is saved for fast access. For the ideal representation, we propose the adaptation of the index structure from [12], which reduces the number of potential reducers. This can be done in two ways, but both indexing strategies are based on the observation in [9, Appendix C]. Instead of searching for a suitable reducer in a single container of polynomials, we introduce lists  $l_x$  for each variable  $x$ . We have two possibilities to fill these lists. Either each  $l_x$  is filled with all polynomials  $p$  with  $x \in \text{lm}(p)$  and during reduction of  $p$  we only search in an arbitrary  $l_x$  where  $x \in \text{lm}(p)$ , or for each polynomial  $p$  we fill one arbitrary  $l_x$  with  $x \in \text{lm}(p)$  with  $p$  and during reduction of  $p$ , we search in all  $l_x$  where  $x \in \text{lm}(p)$ . To reduce the number of

terms which appear during the reduction, we order the polynomials in the index explained above according to the number of terms.

*Real radical.* Among others, [11] discusses the problem that calculating the real radical is hard. They both propose the iterative application of simple rules to the ideal and thereby approximating the real radical. We propose to take this one step further. Instead of alternately calculating the GB and applying such rules, we integrate the rules within the calculation of the GB. For a given set of polynomials  $P$ , such a procedure thus no longer yields a GB for the ideal. However, we neither require the procedure to calculate the real radical of  $P$ . We only require that it *preserves* the common real zeroes.

**Definition 1 (Real-radical preserving GB procedure).** *A procedure  $\mathcal{G}$  is called a real-radical preserving GB procedure if  $\mathcal{V}_{\mathbb{R}}(P) = \mathcal{V}_{\mathbb{R}}(\mathcal{G}(P))$  and  $\mathcal{G}(P)$  is a GB.*

To achieve such a procedure, we modify the update operator in Buchberger's algorithm (line 10, 1).

**Definition 2 (Real-radical preserving update operator).** *Let  $U$  be an update operator,  $<$  be a monomial ordering.  $U$  is said to be real-radical preserving if for  $P \subset \mathbb{Q}[\bar{x}]$  and  $s \in \mathbb{Q}[\bar{x}]$  we have that  $U(P, s) = P \cup Q$ , where  $Q \subset \mathbb{Q}[\bar{x}]$  such that  $\mathcal{V}_{\mathbb{R}}(Q) = \mathcal{V}_{\mathbb{R}}(\langle s \rangle)$  and  $q$  is normal form modulo  $P$  for all  $q \in Q$ .*

The following theorem formalizes the relation between the used update operator and the GB procedure.

**Theorem 1.** *If the update operator in the Buchberger algorithm is modified into a real-radical preserving update operator, then the modified Buchberger algorithm is a real-radical preserving GB procedure.*

The proof is included in [9, Appendix D]. In [9, Appendix E] we give some computationally cheap rules implemented.

### 4.3 The handling of inequalities

Our implementation offers two different approaches to deal with a received inequality  $p \sim 0$ . The first approach *equalizes* the inequation by introducing a new variable  $y$  according to the following valid equivalences [13]:

$$p \geq 0 \Leftrightarrow \exists y. p - y^2 = 0, \quad p > 0 \Leftrightarrow \exists y. py^2 - 1 = 0, \quad p \neq 0 \Leftrightarrow \exists y. py - 1 = 0$$

The resulting equation can then be handled as before.

In the second approach we *reduce*  $p$  to  $q := \text{red}_P(p)$  w.r.t. some subset  $P$  of a GB  $G$ . If  $q \in \mathbb{Q}$ , then either  $q \sim 0$  and we do not have to pass it to our backends, or  $q \not\sim 0$  and we obtain  $C_{\text{rsn}}(C_{=} (P)) \cup \{p \sim 0\}$  as infeasible subset and return *unsat*. In order to allow the correct interaction of the reduction of  $p \sim 0$  with the GB module stack, we store the most relevant reductions in a *reduction chain*

$\text{RC}(p \sim 0) \subseteq \mathbb{Q}[\bar{x}] \times \mathbb{N}$ : Assuming our stack is  $((A_0, G_0), \dots, (A_k, G_k))$ , then  $\text{RC}(p \sim 0) = \{(p, 0)\} \cup \{(\text{red}_{G_k}(p), k) \mid \text{red}_{G_k}(p) \neq p\}$ . If a new state  $(A_j, G_j)$  is added to the stack, we set  $\text{RC}(p \sim 0) = \text{RC}(p \sim 0) \cup \{(\text{red}_{G_j}(p_m), j) \mid \text{red}_{G_j}(p_m) \neq p_m\}$  where  $m = \max\{i \in \mathbb{N} \mid (p, i) \in \text{RC}(p \sim 0)\}$ . If an equality is removed such that the new stack size is  $k'$ , then we remove all  $(p, i)$ ,  $i > k'$  from  $\text{RC}(p \sim 0)$ . If  $p \sim 0$  is removed, we simply delete  $\text{RC}(p \sim 0)$ .

#### 4.4 Learning

In the following we consider that a constraint  $p \sim 0$  is deduced from  $C_{\text{rcv}}$  by the module. If we achieve a constant value, i.e.  $q \in \mathbb{Q}$  and  $q \sim 0$  holds, we obtain the non-inventive lemma  $C_{\text{rsn}}(C_{=}(P)) \rightarrow (p \sim 0)$ . If  $q$  is a linear polynomial and  $P$  contains at least one nonlinear constraint, we share the inventive lemma  $C_{\text{rsn}}(C_{=}(P)) \rightarrow (q = 0)$ . Successive theory calls might then be solved by a more efficient linear solver. Note, that linear solvers are usually capable of detecting such deductions where  $P$  consists only of linear constraints. Finally, if  $q := \sum t_i x$  and  $\text{tdeg}(q)$  is sufficiently small, for instance less than the maximum degree occurring in  $C_{\text{rcv}}$ , we learn the inventive lemma  $C_{\text{rsn}}(C_{=}(P)) \rightarrow (x = 0 \vee \sum t_i = 0)$ . It forms a case splitting and at least one case reduces the complexity of the subsequent theory call significantly.

#### 4.5 Iterative variable elimination

In Section 4.2 we have discussed the embedding of saturation rules for the real radical into the GB procedure. However, some saturation rules from [11][13] are not (yet) suitable for this kind of integration, e.g., rules involving a case splitting, which is optimally resolved by learning as discussed in the previous subsection.

Another example is the *iterative variable elimination* (IVE) as introduced in [13]. In practice, a GB  $G$  contains a lot of identities of the form  $t - x$ , where  $t$  is a term not containing  $x$ . IVE removes the respective identity and substitutes  $x$  by  $t$  in  $G$ , in symbols  $G[t \setminus x]$ , yielding  $G' = (G \setminus \{t - x\})[t \setminus x]$ , which is in general not a GB. Then, it applies the GB procedure to obtain a GB and repeats these two steps until we reach a fixpoint. The strict embedding of this saturation rule into the GB procedure is not straightforward, as potentially all GB elements are affected. Furthermore, we apply the encountered substitutions to the GB module's received inequalities.

When applying IVE, we have to preserve the module's SMT compliance, which turns out to be rather straightforward for the provided mechanisms. The incrementality can be guaranteed as all substitutions can be applied to the polynomials of added constraints belatedly. In order to provide backtrackability, we add the substitutions to the stored module state. We define the reason set of a constraint  $c' := c[t \setminus x]$  we obtained by applying a substitution to be  $C_{\text{rsn}}(c') := C_{\text{rsn}}(c) \cup C_{\text{rsn}}(t - x = 0)$  and identify infeasible subsets as before.

With IVE we are able to detect the infeasibility of a set of constraints more often. Moreover, the constraints we pass to our backends contain less variables by the cost of an in general higher complexity in the remaining variables. A

**Table 1** # instances more than  $\delta$  ms faster/slower than SMT-RAT with  $S_{\text{ref}}$ .

Set (# instances)	$\delta$	$\text{GB}_{\text{np}}$	$\text{GB}_{\text{np}}^{\text{IVE}}$	$\text{GB}_{\text{p}}$	$\text{GB}_{\text{p}}^{\text{IVE}}$	$\text{GB}_{\text{t}}$	$\text{GB}_{\text{t}}^{\text{IVE}}$	Any
KEY (421)	5	102/36	120/44	110/46	119/51	183/45	178/56	252/4
	500	29/0	29/1	28/5	27/6	31/2	35/0	36/0
MET (8276)	25	267/231	175/416	352/434	254/613	167/1410	239/1401	698/77
BOUNCE (180)	500	0/0	0/1	10/11	77/7	0/0	1/0	78/0

drawback of IVE is that it blows up the reason sets of the constraints and therefore leads to greater infeasible subsets.

## 5 Experimental results

The symbolic computations we present in this paper can significantly improve the performance of an SMT-RAT solver instance. We tested six different  $M_{\text{GB}}$  settings with the SMT-RAT strategy  $S := (V, E)$  where  $V := \{I_{M_{\text{CNF}}}, I_{M_{\text{SAT}}}, I_{M_{\text{LRA}}}, I_{M_{\text{GB}}}, I_{M_{\text{VS}}}, I_{M_{\text{CAD}}}\}$  with  $I_M$  an instance of module  $M$  and  $E := \{(I_{M_{\text{CNF}}}, I_{M_{\text{SAT}}}), (I_{M_{\text{SAT}}}, I_{M_{\text{LRA}}}), (I_{M_{\text{LRA}}}, I_{M_{\text{GB}}}), (I_{M_{\text{GB}}}, I_{M_{\text{VS}}}), (I_{M_{\text{VS}}}, I_{M_{\text{CAD}}})\}$ . Since  $M_{\text{LRA}}$  performs significantly faster on many instances containing linear constraints, it is positioned before  $M_{\text{GB}}$ . All  $M_{\text{GB}}$  settings implement the approaches explained in the Sections 4.1 and 4.2. The settings  $\text{GB}_{\text{np}}$  and  $\text{GB}_{\text{p}}$  reduce inequalities,  $\text{GB}_{\text{t}}$  transforms them.  $\text{GB}_{\text{np}}$  and  $\text{GB}_{\text{t}}$ , however, set  $C_{\text{pas}} = C_{\text{rcv}}$ , while  $\text{GB}_{\text{p}}$  passes constraints as described in Section 4.1.  $\text{GB}_{\text{p}}^{\text{IVE}}$ ,  $\text{GB}_{\text{np}}^{\text{IVE}}$ ,  $\text{GB}_{\text{t}}^{\text{IVE}}$  are the extensions of the aforementioned settings by IVE. The computational effort and thus the room for optimization stepwise increases with enabling transformation and IVE. Passing the constraints has a major influence on the backends. We compared all settings with the reference strategy  $S_{\text{ref}} := (V_{\text{ref}}, E_{\text{ref}})$  where  $V_{\text{ref}} := V \setminus \{I_{M_{\text{GB}}}\}$  and  $E_{\text{ref}} := (E \setminus \{(I_{M_{\text{LRA}}}, I_{M_{\text{GB}}}), (I_{M_{\text{GB}}}, I_{M_{\text{VS}}})\}) \cup \{(I_{M_{\text{LRA}}}, I_{M_{\text{VS}}})\}$ . We regard three example sets: BOUNCE is an extension of examples introduced in [5]. KEY and MET originate from the tools `KeyMaera` and `MetiTarski`. Details of our benchmarks can be found in [9, Appendix A], here we give a summary.

Table 1 shows for each setting how many instances ran more than  $\delta$  milliseconds faster/slower than the reference solver. In the last column, we give results for a hypothetical optimal solver, which always takes the setting yielding the best running time. Although many instances are not significantly influenced by  $M_{\text{GB}}$  in terms of running time, we observe a critical speed-up on specific instances. For KEY, improvements are gained by detecting unsatisfiability, which in most cases occurs during the reduction of inequalities. Here the received constraints are more suitable for passing. For BOUNCE,  $M_{\text{GB}}$  has only effect if the resolved identities are passed by  $\text{GB}_{\text{p}}^{\text{IVE}}$ . A heuristic choosing the right setting increases the overall performance, and is essential for MET.

## 6 Conclusion and future work

In this work, we made use of the strength of traditional computer algebra procedures to resolve weaknesses of SMT solving for RCF. In particular, we inte-

grated Gröbner bases computations in a module of an SMT solver. Moreover, we adapted the implementation of the Buchberger algorithm and its data structures to reflect differences in treated problems. To meet our requirement of real solutions, we embedded saturation rules for the real radical within the Buchberger algorithm, which makes the module more powerful. Experimental results show that selected instances are solved a lot faster.

As a next step we want to optimize the heuristics used in our Gröbner bases module and do other improvements, e.g., by developing new saturation rules or by algorithmic improvements tailored towards special input problem structures. We are also interested in integrating further methods based on (lexicographic) Gröbner bases, and especially in realizing applications of the Positivstellensatz. Another open point is the choice of the SMT-RAT strategy. For instance, the interplay between the GB and the CAD module could be much more dynamic as compared to one fixed strategy with fixed CAD settings.

## References

1. Becker, T., Weispfenning, V., Kredel, H.: Gröbner bases: a computational approach to commutative algebra. Graduate texts in mathematics, Springer (1993)
2. Biere, A., Heule, M.J.H., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press (2009)
3. Buchberger, B.: Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal. Ph.D. thesis, University of Innsbruck (1965)
4. 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)
5. Corzilius, F., Loup, U., Junges, S., Ábrahám, E.: SMT-RAT: An SMT-compliant nonlinear real arithmetic toolbox. In: Proc. of SAT’12. LNCS, vol. 7317, pp. 442–448. Springer (2012)
6. Dolzmann, A., Sturm, T.: Simplification of quantifier-free formulas over ordered fields. Journal of Symbolic Computation 24, 209–231 (1997)
7. Dutertre, B., de Moura, L.: A fast linear-arithmetic solver for DPLL(T). In: Proc. of CAV’06. LNCS, vol. 4144, pp. 81–94. Springer (2006)
8. Gao, S., Ganai, M.K., Ivancic, F., Gupta, A., Sankaranarayanan, S., Clarke, E.M.: Integrating ICP and LRA solvers for deciding nonlinear real arithmetic problems. In: Proc. of FMCAD’10. pp. 81–89. IEEE (2010)
9. Junges, S., Loup, U., Corzilius, F., Ábrahám, E.: On Gröbner bases in the context of satisfiability-modulo-theories solving over the real numbers. Tech. Rep. AIB-2013-08, RWTH Aachen University (May 2013), <http://aib.informatik.rwth-aachen.de/2013/2013-08.pdf>
10. de Moura, L., Passmore, G.O.: On locally minimal Nullstellensatz proofs. In: Proc. of SMT’09. pp. 35–42 (2009)
11. Passmore, G.O.: Combined Decision Procedures for Nonlinear Arithmetics, Real and Complex. Ph.D. thesis, University of Edinburgh (2011)
12. Passmore, G.O., de Moura, L., Jackson, P.B.: Gröbner basis construction algorithms based on theorem proving saturation loops. In: Decision Procedures in Software, Hardware and Bioware. No. 10161 in Dagstuhl Seminar Proc. (2010)

13. Platzer, A., Quesel, J.D., Rümmer, P.: Real world verification. In: Proc. of CADE-22. LNCS, vol. 5663, pp. 485–501. Springer (2009)
14. Roune, B.H., Stillman, M.: Practical Gröbner basis computation. In: Proc. of ISSAC'12. pp. 203–210. ACM (2012)
15. Weispfenning, V.: Quantifier elimination for real algebra – the quadratic case and beyond. AAECC 8(2), 85–101 (1997)