RWTH Aachen

Department of Computer Science
Technical Report

On Gröbner Bases in the Context of Satisfiability-Modulo-Theories Solving over the Real Numbers (Extended Version)

Sebastian Junges, Ulrich Loup, Florian Corzilius
and Erika Ábrahám

ISSN 0935–3232 · Aachener Informatik-Berichte · AIB-2013-08

RWTH Aachen · Department of Computer Science · June 2013

The publications of the Department of Computer Science of *RWTH Aachen University* are in general accessible through the World Wide Web.

`http://aib.informatik.rwth-aachen.de/`

On Gröbner Bases in the Context of Satisfiability-Modulo-Theories Solving over the Real Numbers (Extended Version)*

Sebastian Junges, Ulrich Loup, Florian Corzilius and Erika Ábrahám **

Chair of Computer Science 2
RWTH Aachen University, Germany
Email: sebastian.junges@rwth-aachen.de,
{loup, corzilius, abraham}@cs.rwth-aachen.de

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 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

* This is an extended version of a publication accepted at CAI'13 and available at www.springerlink.com.

** 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/).

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], implemented among others in the tool `QEPCAD`, also incomplete methods such as the *virtual substitution* (VS) method [15], supported, e.g., by the package `Redlog` of the computer algebra system `Reduce`, *simplex* [8] or *interval constraint propagation* [9], implemented, e.g., 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 [7] and [11]. In [13], a combination of Gröbner basis computations with the Fourier-Motzkin method is proposed. 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 $<$, and refer to $\mathbb{R}$ as the *real-closed field* (*RCF*). We sometimes omit the symbol $\cdot$ and write xy instead of $x \cdot y$.

In the following let K be a field and $\overline{x}$ a sequence $x_1, \ldots, x_n$ of variables for some $n \in \mathbb{N}$, $n \geq 1$. We call a product $m = \prod_{1 \leq i \leq n} x_i^{e_i}$ with $e_i \in \mathbb{N}$ a *monomial* in $\overline{x}$ having the *degree* $\deg(m) := \sum_{1 \leq i \leq n} e_i$. With $M_{\overline{x}}$ we denote the set of all monomials in $\overline{x}$. A product $a \cdot m$ with $a \in K$ and $m \in M_{\overline{x}}$ is called a *term* in $\overline{x}$ and a the *coefficient of* m.

A *polynomial* p in $\overline{x}$ is a sum of terms in $\overline{x}$ with pairwise different monomials. By $K[\overline{x}]$ we denote the *polynomial ring* over K in the variables $\overline{x}$. Let $p \in K[\overline{x}]$. We say that $x_i \in p$ if x_i occurs in p with a positive power. We define the *total degree* of p as $\mathrm{tdeg}(p) := \max\{\deg(m) \mid m \text{ monomial in } p\}$.

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 $m_1, m_2, m_3 \in M_{\overline{x}}$. The *leading monomial* $\mathrm{lm}(p)$ *of* p is the maximal monomial in p w.r.t. the current ordering. The *leading coefficient* $\mathrm{lc}(p)$ *of* p is the coefficient of $\mathrm{lm}(p)$. the product $\mathrm{lc}(p)\mathrm{lm}(p)$ is called *leading term* $\mathrm{lt}(p)$ *of* p.

We call $p \sim 0$ with $\sim \in \{=, >, \geq, \neq\}$ a (*polynomial*) *constraint over* p. A *set of constraints over* $P \subseteq K[\overline{x}]$ is a subset of $\{p \sim 0 \mid p \in P, \sim \in \{=, >, \geq, \neq\}\}$. For a set of constraints C, we define $\mathrm{pol}(C) := \{p \in K[\overline{x}] \mid p \sim 0 \in C\}$. Our input formulas are quantifier-free first-order formulas over polynomial constraints, i.e., constraints connected by the Boolean operators $\wedge$ and $\neg$. 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 $R = \mathbb{Q}[x_1, \ldots, x_n]$ with a fixed monomial ordering. Given a finite set $P \subseteq R$ of polynomials, 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. Let $\widetilde{K}$ be a field, by $\mathcal{V}_{\widetilde{K}}(P) := \{a \in \widetilde{K}^n \mid p(a) = 0 \text{ for all } p \in P\}$ we denote the $\widetilde{K}$ *variety of* P, i.e., the set of common zeros of P in $\widetilde{K}^n$.

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 $s \cdot \mathrm{lt}(f) = a_i m_i$ for some $i \in \{0, \ldots, k\}$ 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 $p' \in R$ and $f \in F$ with $p \xrightarrow{f} p'$ exists, p is *in normal form modulo* F. If $p \xrightarrow{F} \ldots \xrightarrow{F} p'$ and p' is in normal form modulo F then we call p' the *normal form of* P *modulo* F, denoted by $\mathrm{red}_F(p)$.

Definition 1 (Gröbner basis)
Let $P \subseteq R \setminus \{0\}$. *A finite set* $G \subseteq \langle P\rangle$ *is called a* Gröbner basis (GB) of P *if* $\langle\{\mathrm{lt}(g) \mid g \in G\}\rangle = \langle\{\mathrm{lt}(p) \mid p \in P\}\rangle$. *Let* $\mathrm{lc}(p) = 1$ *for all* $p \in G$. *We call* G minimal *if* $\mathrm{lt}(g) \notin \langle\mathrm{lt}(\tilde{g}) \mid \tilde{g} \in G \setminus \{g\}\rangle$ *for all* $g \in G$, *and* reduced *if* $m \notin \langle\mathrm{lt}(\tilde{g}) \mid \tilde{g} \in G \setminus \{g\}\rangle$ *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 $\mathcal{V}_{\mathbb{R}}(\langle P\rangle) = \emptyset$, 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 $\mathrm{lm}(p) = \prod_{i=1}^{n} x_i^{e_i}$ and $\mathrm{lm}(q) = \prod_{i=1}^{n} x_i^{\tilde{e}_i}$, then the *least common multiple* of $\mathrm{lm}(p)$ and $\mathrm{lm}(q)$ is $\mathrm{lcm}(\mathrm{lm}(p), \mathrm{lm}(q)) = \prod_{i=1}^{n} x_i^{\max(e_i,\tilde{e}_i)} =: l$. We define $S(p,q) := \frac{l}{\mathrm{lt}(p)} \cdot p - \frac{l}{\mathrm{lt}(q)} \cdot q$ to be the *S-polynomial of* p *and* q. The S-polynomials of all pairs of input 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 $U : 2^R \times R \to 2^R$ an *update operator*, where 2^R denotes the power set of R. Buchberger's algorithm uses the *standard update operator* $\mathrm{U}_{\mathrm{std}}(G, s) = G \cup \{s\}$.

Listing 1: Buchberger's algorithm.

```
1  Input: Set of polynomials F, 0 ∉ F
2  Output: Gröbner basis G for ⟨F⟩
3
4
5  G := F
6  while true:
7    G' := G
8    for each {p,q} ⊆ G', p ≠ q:
9      s := red_G(S(p,q))
10     if s ≠ 0: G := U_std(G,s)
11   if G = G': break
12 return G
```

A reduced Gröbner basis can be obtained by 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.

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 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(`$C_{\texttt{rcv}}$`)`. For a given set C_{rcv} of RCF formulas, called *the set of received formulas*, the procedure either decides whether C_{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 φ to (from) it with `add(`φ`)` (`remove(`φ`)`). Since in the SMT embedding C_{rcv} is usually changed between two consecutive `check(`$C_{\texttt{rcv}}$`)` 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_{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_{pas} of RCF formulas, called *the set of passed formulas*, using the procedure `runBackends(`$C_{\texttt{pas}}$`)` which is controlled by the manager.

This paper presents the implementation of a new SMT-RAT module called $\texttt{M}_{\texttt{GB}}$ 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 $\texttt{M}_{\texttt{CNF}}$, a SAT solver $\texttt{M}_{\texttt{SAT}}$ and the modules $\texttt{M}_{\texttt{LRA}}$ for simplex, $\texttt{M}_{\texttt{VS}}$ for VS and $\texttt{M}_{\texttt{CAD}}$ 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 Ω is a set of conditions. Initially, the *manager* calls the method `check(`$C_{\texttt{rcv}}$`)` of the module instance given by the root of the strategy, where C_{rcv} is a set of RCF formulas. Whenever a module instance $m \in V$ calls `runBackends(`$C_{\texttt{pas}}$`)`, the manager calls `check(`$C_{\texttt{pas}}$`)` of each module m', for which an edge $(m, \omega, m') \in E$ exists such that ω holds for C_{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, ω, m) if $\omega = \text{True}$.

Usually, the root module $\texttt{M}_{\texttt{CNF}}$ transforms its set of received formulas $C_{\text{rcv}}^{\text{cnf}}$ to an equisatisfiable set of clauses $C_{\text{pas}}^{\text{cnf}}$ and calls `runBackends(`$C_{\texttt{pas}}^{\texttt{cnf}}$`)`. The back-

end is a SAT-solver module $\mathtt{M_{SAT}}$, which runs DPLL-style SAT-solving on the Boolean abstraction of the set of received clauses $C_{\text{rcv}}^{\text{sat}} = C_{\text{pas}}^{\text{cnf}}$. $\mathtt{M_{SAT}}$ might call `runBackends(`$C_{\text{pas}}^{\text{sat}}$`)` for partial Boolean assignments on the corresponding set of formulas $C_{\text{pas}}^{\text{sat}}$; we refer to such a backend call as a *theory call*. The Boolean abstractions of the obtained infeasible subsets and lemmas are stored as additional clauses. Infeasible subsets and lemmas, which contain only formulas from $C_{\text{pas}}^{\text{sat}}$, 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 $I_{\mathtt{M_{CNF}}}$, $I_{\mathtt{M_{SAT}}}$ and $I_{\mathtt{M_{CAD}}}$ and the edges $(I_{\mathtt{M_{CNF}}}, I_{\mathtt{M_{SAT}}})$ and $(I_{\mathtt{M_{SAT}}}, I_{\mathtt{M_{CAD}}})$.

4 Applying Gröbner bases

In this section we describe our `SMT-RAT` module $\mathtt{M_{GB}}$ 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 $\mathtt{M_{GB}}$ which makes it more suitable for preprocessing in Section 4.5.

In this section we assume C_{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 from C with a given comparison relation $\sim \in \{=, >, \geq, \neq\}$ 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.

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`.

For a given set C_{rcv} of received constraints we denote a (*GB module*) *state* by a tuple $(A, G) \in 2^{\mathbb{Q}[\bar{x}]} \times 2^{\mathbb{Q}[\bar{x}]}$ with $\langle A \cup G \rangle = \langle \text{pol}(C_{\text{rcv}}[=]) \rangle$ such that G is a Gröbner basis computed during the last consistency check for the polynomials received up to this point and $A \subseteq \text{pol}(C_{\text{rcv}}[=])$ is the set of polynomials added to $C_{\text{rcv}}[=]$ since then.

The *incremental* consistency check procedure is given in Listing 2. It operates on C_{rcv} and the state (A, G). The procedure possibly updates the state (A, G) and outputs, first, an answer as to whether C_{rcv} is *sat*, *unsat* or its consistency is *unknown*, and second, an infeasible subset $C_{\text{inf}} \subseteq C_{\text{rcv}}$ 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 6 in Listing 2).

Thereby we recompute the GB only if $A \neq \emptyset$, and we reuse G for the computation of the GB of $\mathrm{pol}(C_{\mathrm{rcv}}[=])$, what is possible because $\langle \mathrm{pol}(C_{\mathrm{rcv}}[=]) \rangle = \langle G \cup A \rangle$. If the Gröbner basis is $\{1\}$, the polynomials have no common real zeros; hence, we determine the infeasible subset C_{inf} as reason set of $1 = 0$ (details below) and return *unsat*. Otherwise, we call a module with the received inequations, and instead of the original equations, we pass equations formed by the Gröbner basis. This is done by the procedure `runBackends` described in Section 3.

Listing 2: GB module consistency check.

```
1  Input: C_rcv, state (A,G)
2  Output: (ans, C_inf), with C_inf ⊆ C_rcv
3    and ans ∈ {sat,unsat,unknown}
4
5  if A ≠ ∅:
6    G := Groebner(G ∪ A)
7    A := ∅
8    if G = {1}: return (unsat, C_rsn(1 = 0))
9  C_pas := (C_rcv \ C_rcv[=]) ∪ {p = 0 | p ∈ G}
10 (r, C'_inf) := runBackends(C_pas)
11 determine C_inf from C'_inf
12 return (r, C_inf)
```

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 $((A_1, G_1), \ldots, (A_k, G_k))$, $k \in \mathbb{N} \setminus \{0\}$, 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).

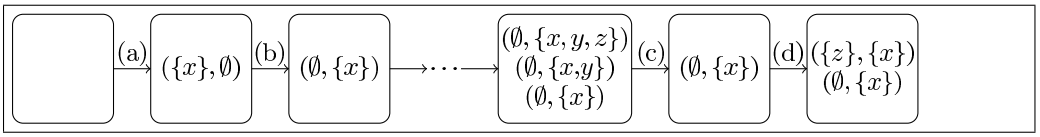

Fig. 1: The state stack in the GB module.

Infeasible subsets As argued before, the module is expected to return a subset $C_{\mathrm{inf}} \subseteq C_{\mathrm{rcv}}$ in case the set of received constraints C_{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 $(h_1, \ldots, h_n)$ such that for an ideal $I = \langle f_1, \ldots, f_n \rangle$ 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 full multivariate polynomial divisions, which is less efficient because we have to calculate and memorize

quotients for each input polynomial. As we do this calculation for all reason sets, we implemented a more naive way. The realization of small 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 adaptions 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, thus, the complexity of computing the union is linear in the number of polynomials.

Data structures. We base our implementation of data structures on [14] and 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 few terms only, each containing a small number of variables.

A term $a \cdot \prod_{i=1}^{n} x_i^{e_i}$ is represented as $(a, [(x_{i_1}, e_{i_1}), \ldots, (x_{i_k}, e_{i_k})], \sum_{i=1}^{n} e_i)$ with $\{i_1, \ldots, i_k\} = \{i \in \{1, \ldots, n\} \mid e_i > 0\}$ 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 adaption of the index structure from [12], which reduces the number of potential reductors. This can be done in two ways, but both indexing strategies are based on the observation from Appendix C. Instead of searching for a suitable reductor 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 the reduction of p we only search in an arbitrary l_x whereas $x \in \text{lm}(p)$, or we add each polynomial p to a single list l_x with $x \in \text{lm}(p)$ and during the 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. Since calculating the real radical is hard, [11] proposes the iterative application of simple rules to the ideal to *approximate* the real radical. We propose to take this one step further. Instead of alternatingly 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 the 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 $\mathcal{G}$ with input $P \subseteq \mathbb{Q}[\overline{x}]$ and output $\mathcal{G}(P) \subseteq \mathbb{Q}[\overline{x}]$ 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 of P.*

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

Definition 2 (Real-radical-preserving update operator). *Let U be an update operator and $\prec$ a monomial ordering. U is said to be* real-radical preserving *if for each $P \subseteq \mathbb{Q}[\overline{x}]$ and $s \in \mathbb{Q}[\overline{x}] \setminus \{0\}$, where s is in normal form modulo P, we have that $U(P, s) = P \cup Q$ for some $Q \subseteq \mathbb{Q}[\overline{x}]$ such that $\mathcal{V}_{\mathbb{R}}(Q) = \mathcal{V}_{\mathbb{R}}(\langle s \rangle)$ and all $q \in Q$ are in normal form modulo P.*

Note that if U is a real-radical-preserving update operator, $\mathcal{V}_{\mathbb{R}}(U(P, s)) = \mathcal{V}_{\mathbb{R}}(P \cup \{s\})$, i.e., U extends P by polynomials which have the same real roots as s. 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 Appendix D. In Appendix E we give some computationally cheap *rules* composing the real-radical-preserving update operator which is used in our implementation.

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 $q := \mathrm{red}_P(p)$ 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 $C_{\mathrm{rsn}}(C_{=}(P)) \cup \{p \sim 0\}$ 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* $\mathrm{RC}(p \sim 0) \subseteq \mathbb{Q}[\overline{x}] \times \mathbb{N}$, defined by the following cases.

- If a new inequality $p \sim 0$ is added, $\mathrm{RC}(p \sim 0)$ is empty and our stack is $((A_1,\ G_1), \ldots,\ (A_k,\ G_k))$, $k \geq 1$, then we set

  $$\mathrm{RC}(p \sim 0) = \{(p, 0)\} \cup \{(\mathrm{red}_{G_k}(p), k) \mid \mathrm{red}_{G_k}(p) \neq p\},$$

  i.e., we add the original polynomial p of the inequality with the index 0 and p reduced modulo the top-most Gröbner basis from the stack with index k.

- If a new state (A_j, G_j) is added to the stack, we set

$$\mathrm{RC}(p \sim 0) = \mathrm{RC}(p \sim 0) \cup \{(\mathrm{red}_{G_j}(q), j)\}$$

  where $(q, m) \in \mathrm{RC}(p \sim 0)$ with $m = \max\{i \in \mathbb{N} \mid (q', i) \in \mathrm{RC}(p \sim 0)\}$, i.e., (q, m) is the entry with the largest index in the reduction chain $\mathrm{RC}(p \sim 0)$, $\mathrm{red}_{G_j}(q) \neq q$.
- If an equality is removed such that the new stack size is k', then we remove all (p, i), $i > k'$ from $\mathrm{RC}(p \sim 0)$.
- If $p \sim 0$ is removed, we simply delete $\mathrm{RC}(p \sim 0)$.

4.4 Learning

In the following we consider that a constraint $p \sim 0 \in C_{\mathrm{rcv}}$ is deduced from $C_{\mathrm{rcv}} \setminus \{p \sim 0\}$ by the Gröbner module using a subset P of a Gröbner basis. If we achieve a constant value, i.e. $\mathrm{red}_P(p) \in \mathbb{Q}$, and $\mathrm{red}_P(p) \sim 0$ holds, we obtain the non-inventive lemma $C_{\mathrm{rsn}}(C_{=}(P)) \to (p \sim 0)$. Note that $\mathrm{red}_P(p) \in \mathbb{Q}$ and $\mathrm{red}_P(p) \not\sim 0$ implies unsatisfiability. If $\mathrm{red}_P(p)$ is a linear polynomial and P contains at least one nonlinear constraint, we share the inventive lemma $C_{\mathrm{rsn}}(C_{=}(P) \cup \{P \sim 0\}) \to (\mathrm{red}_P(p) \sim 0)$. In successive theory calls, unsatisfiability may then be detected by a more efficient linear solver. Note, that linear solvers are usually capable of detecting such deductions if P consists of linear constraints only. Finally, if G is the obtained Gröbner basis and $q = \sum t_i x \in G$ for some terms t_i and a variable x such that $\mathrm{tdeg}(q)$ is sufficiently small, e.g., less then the maximum degree occurring in C_{rcv}, we learn the inventive lemma $C_{\mathrm{rsn}}(q) \to (x = 0 \vee \sum t_i = 0)$. 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 rules for the real radical into the GB procedure. However, some rules from [11] and [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 often contains polynomials of the form $t - x$, where t is a term not containing x. IVE removes $t - x$ from G and substitutes x by t in the remaining set, yielding $G' = (G \setminus \{t - x\})[t/x]$, which is in general not a GB. Then, it applies the GB procedure to obtain a GB and repeats these two steps until a fixpoint is reached. The strict embedding of this rule into the GB procedure is not straightforward, as potentially all GB elements are affected. In our GB module, we have to apply the encountered substitutions also to the 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 $c' := c[t/x]$ we obtained by applying a substitution to be $C_{\mathrm{rsn}}(c') := C_{\mathrm{rsn}}(c) \cup C_{\mathrm{rsn}}(t - x = 0)$ and identify infeasible subsets as before.

With IVE we are able to detect the infeasibitity 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 drawback of IVE is that it blows up the reason sets of the constraints and therefore leads to larger 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 $\mathtt{M_{GB}}$ settings with the SMT-RAT strategy $S := (V, E)$ where $V := \{I_{\mathtt{M_{CNF}}}, I_{\mathtt{M_{SAT}}}, I_{\mathtt{M_{LRA}}}, I_{\mathtt{M_{GB}}}, I_{\mathtt{M_{VS}}}, I_{\mathtt{M_{CAD}}}\}$ with I_M an instance of module M and $E := \{(I_{\mathtt{M_{CNF}}}, I_{\mathtt{M_{SAT}}}), (I_{\mathtt{M_{SAT}}}, I_{\mathtt{M_{LRA}}}), (I_{\mathtt{M_{LRA}}}, I_{\mathtt{M_{GB}}}), (I_{\mathtt{M_{GB}}}, I_{\mathtt{M_{VS}}}), (I_{\mathtt{M_{VS}}}, I_{\mathtt{M_{CAD}}})\}$. Since $\mathtt{M_{LRA}}$ performs significantly faster on many instances containing linear constraints, it is positioned before $\mathtt{M_{GB}}$. All $\mathtt{M_{GB}}$ settings implement the approaches explained in the Sections 4.1 and 4.2. The settings $\mathtt{GB_{np}}$ and $\mathtt{GB_{p}}$ reduce inequalities, $\mathtt{GB_{t}}$ transforms them. $\mathtt{GB_{np}}$ and $\mathtt{GB_{t}}$, however, set $C_{\text{pas}} = C_{\text{rcv}}$, while $\mathtt{GB_{p}}$ passes constraints as described in Section 4.1. $\mathtt{GB_{p}^{IVE}}$, $\mathtt{GB_{np}^{IVE}}$, $\mathtt{GB_{t}^{IVE}}$ 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 $V_{\text{ref}} := V \setminus \{I_{\mathtt{M_{GB}}}\}$ and $E_{\text{ref}} := (E \setminus \{(I_{\mathtt{M_{LRA}}}, I_{\mathtt{M_{GB}}}), (I_{\mathtt{M_{GB}}}, I_{\mathtt{M_{VS}}})\}) \cup \{(I_{\mathtt{M_{LRA}}}, I_{\mathtt{M_{VS}}})\}$.

Table 1 # instances more than δ ms faster/slower than SMT-RAT with S_{ref}.

Set (# instances)	δ	$\mathtt{GB_{np}}$	$\mathtt{GB_{np}^{IVE}}$	$\mathtt{GB_{p}}$	$\mathtt{GB_{p}^{IVE}}$	$\mathtt{GB_{t}}$	$\mathtt{GB_{t}^{IVE}}$	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

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 Appendix A, here we give a summary. Table 1 shows for each setting how many instances ran more than δ 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 in terms of running time by the $\mathtt{M_{GB}}$, 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, the $\mathtt{M_{GB}}$ has only effect if the resolved identities are passed by $\mathtt{GB_{p}^{IVE}}$. A heuristic choice of 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 integrated 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 certain problems are solved much faster using our new theory solver module.

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. Cox, D., Little, J., O'Shea, D.: Ideals, Varieties, and Algorithms: an Introduction to Computational Algebraic Geometry and Commutative Algebra. Undergraduate Texts in Mathematics, Springer (1997)
7. Dolzmann, A., Sturm, T.: Simplification of quantifier-free formulas over ordered fields. Journal of Symbolic Computation 24, 209–231 (1997)
8. 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)
9. 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)
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)

Appendix

A Experimental results

All benchmarks were run on an 2.1 GHz AMD Xeon core with a 200 second time-out per instance, as well as a memory limit of 4 GB. We did not experience any solver hitting the memory limit. In Table 2 the number of solved instances within the time limit are given, as well as their cumulative time. The results are split into the *sat* and the *unsat* instances respectively. In Table 3 we give an extended form of Table 1. All Gröbner bases computations were carried out with the graded reverse lexicographic order. The revised update operator was especially helpful for the $\mathtt{GB}_\mathtt{t}$ setting. In Bounce, no unsatisfiability was found by the Gröbner module. For Key, $\mathtt{GB}_\mathtt{t}$ detects unsatisfiability between inequalities relatively often, which accounts for its good performance. However, in cases with a majority of (mostly linear) inequalities, the transformation prevents a fast detection of trivial conflicts.

Table 2 Running times and solved instances.

	Bounce (180)		Met (8276)		Key (421)	
	#	*time*	#	*time*	#	*time*
S_{ref}	101	135.7	7107	16350.4	385	573.6
- sat	84	135.3	4780	8507.4	0	0.0
- unsat	17	0.5	2327	7843.0	385	573.6
$\mathtt{GB}_{\mathtt{np}}$	101	135.7	7088	16149.7	408	381.9
- sat	84	135.1	4774	7985.6	0	0.0
- unsat	17	0.6	2314	8164.2	408	381.9
$\mathtt{GB}_{\mathtt{np}}^{\mathtt{IVE}}$	101	138.4	7096	16167.4	407	376.1
- sat	84	137.8	4773	8077.0	0	0.0
- unsat	17	0.6	2323	8090.5	407	376.1
$\mathtt{GB}_{\mathtt{p}}$	93	73.7	7037	16484.2	405	68.6
- sat	76	73.3	4776	8195.6	0	0.0
- unsat	17	0.5	2261	8288.7	405	68.6
$\mathtt{GB}_{\mathtt{p}}^{\mathtt{IVE}}$	127	32.7	7037	15794.5	405	72.2
- sat	81	18.3	4777	7976.3	0	0.0
- unsat	46	14.4	2260	7818.2	405	72.2
$\mathtt{GB}_{\mathtt{t}}$	101	142.9	6850	17840.2	409	385.3
- sat	84	142.3	4671	9633.1	0	0.0
- unsat	17	0.6	2179	8207.0	409	385.3
$\mathtt{GB}_{\mathtt{t}}^{\mathtt{IVE}}$	101	136.0	6875	17767.9	412	377.8
- sat	84	135.4	4666	9768.6	0	0.0
- unsat	17	0.6	2209	7999.3	412	377.8
Any	137	95.8	7224	18463.7	414	595.1
- sat	88	67.4	4902	10102.9	0	0.0
- unsat	49	28.5	2322	8360.8	414	595.1

Table 3 # instances more than δ ms faster/slower than SMT-RAT with S_{ref}.

Set(# instances)	δ	GB_{np}	$\text{GB}_{\text{np}}^{\text{IVE}}$	GB_{p}	$\text{GB}_{\text{p}}^{\text{IVE}}$	GB_{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
	25	32/4	33/7	30/8	31/11	38/5	47/8	48/0
	500	29/0	29/1	28/5	27/6	31/2	35/0	36/0
MET (8276)	5	1201/1718	1017/1487	1483/1612	1371/1995	1203/2676	1263/2669	2644/182
	25	267/231	175/416	352/434	254/613	167/1410	239/1401	698/77
	500	37/118	59/96	96/206	80/206	103/820	122/807	251/17
BOUNCE (180)	5	34/37	6/69	59/25	117/9	1/87	11/63	128/1
	25	10/10	2/23	51/13	105/8	1/68	6/14	109/1
	500	0/0	0/1	10/11	77/7	0/0	1/0	78/0

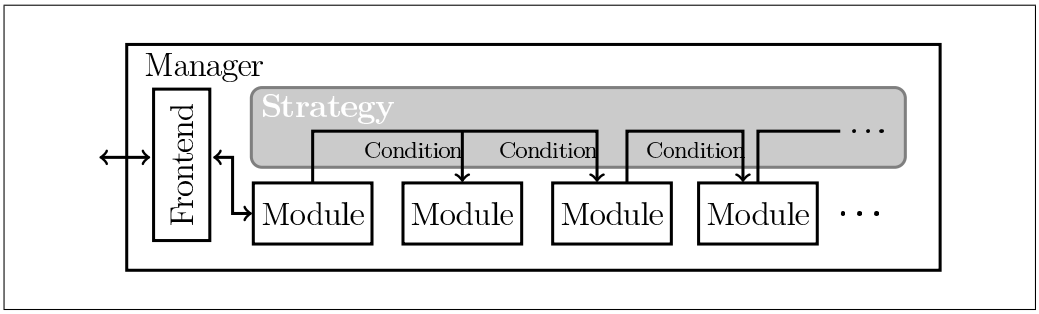

Fig. 2: A snapshot of an SMT-RAT composition.

B SMT-RAT structure

As depicted in Figure 2, SMT-RAT consists of a *manager*, a *strategy* and *modules* as described in Section 3 and a frontend which (1) provides the interfaces to an external SMT solver or (2) parses the input file to an RCF formula.

C Observation regarding index structures for the ideal

Let $p, p' \in K[\bar{x}]$ be polynomials with $0 \neq p \neq p'$ and $P \subseteq K[\bar{x}]$ a set of polynomials.

For the below corollary of the properties of $\xrightarrow{P}$, we need another notion of reduction, which we also use in our implementation. We say that p *top-reduces to* p' *modulo* P, denoted by $p \overset{P}{\rightarrowtail} p'$, if $p' = p - sf$ for some $s \in K[\bar{x}]$ such that $s \cdot \text{lt}(f) = a_k m_k$ where $p = \sum_{i=0}^{k} a_i m_i$, $k \in \mathbb{N}$. In other words, top reduction is a reduction with regard to the leading term only. In particular, we implement the ordinary reduction as defined in Sec. 2.1 by top reduction (cf. [6, Proof of Theorem 3]).

Corollary 1. *1. We define the set of polynomials* $Q = \{q \in P \mid \forall x \in \text{lm}(q).x \in \text{lm}(p)\}$*, such that in the leading monomials of polynomials in* Q *only variables occur which also occur in* $\text{lm}(p)$*. Then it holds that* $p \overset{P}{\rightarrowtail} p'$ *implies* $p \overset{Q}{\rightarrowtail} p'$*.*

2. We define the set of polynomials $Q' = \{q \in P \mid \exists x \in \text{lm}(q).x \in \text{lm}(p)\}$*, such that in the leading monomials of polynomials in* Q' *at least one variable occurs which also occurs in* $\text{lm}(p)$*. Then it holds that* $p \overset{P}{\rightarrowtail} p'$ *implies* $p \overset{Q'}{\rightarrowtail} p'$*.*

We write $a \mid b$, for $a, b \in \mathbb{N}$, if there exists a $k \in \mathbb{N}$ such that $a \cdot k = b$.

Proof (Corrolary 1).

1. Assume $p \overset{P\setminus Q}{\rightarrowtail} p'$. Then $\mathrm{lm}(q) \mid \mathrm{lm}(p)$ for some $q \in P\setminus Q$. Let $\mathrm{lm}(q) = \prod_{i=1}^{n} x^{\alpha_i}$ and $\mathrm{lm}(p) = \prod_{i=1}^{n} x^{\beta_i}$. It follows from the definition that $\alpha_i \geq \beta_i$ for all $1 \leq i \leq |\bar{x}|$, and especially that $\beta_i > 0$ implies $\alpha_i > 0$. Now for an arbitrary $q \in P \setminus Q$ we know that there exists a $x \in \mathrm{lm}(p)$ with $x \notin \mathrm{lm}(q)$. Let j be the index of this x. Then $\beta_j > 0$, but $\alpha_j = 0$. Contradiction.
2. Clear from the fact that $Q \subseteq Q'$.

D The real radical preserving update routine

Let $(\mathbb{C}, +, \cdot)$ denote the algebraically closed field (ACF). Whenever possible, we omit the operators and write $\mathbb{C}$ to denote the ACF. With $\mathrm{lt}(P)$ we denote $\{\mathrm{lt}(p) \mid p \in P\}$.

Proof (Theorem 1).
We use the notation of Listing 1 and Definition 2.

Termination. For the main loop, we have that either $G = G'$ and the algorithm terminates, or $s = \mathrm{red}_G(S(p,q)) \neq 0$ for some $p, q \in G'$. Now, we argue that the latter case might happen only finitely often, i.e., the update operator is only called finitely often. We claim that $\langle \mathrm{lt}(G' \cup Q)\rangle \supseteq \langle \mathrm{lt}(G)\rangle$. Therefore, we notice that if $U(G', s)$ is called, we have that all $q \in Q$ are in normal form modulo G', and thus, $\mathrm{lt}(q) \notin \langle \mathrm{lt}(G')\rangle$. We thus get an ascending chain of ideals, which eventually stabilizes by the *ascending chain condition*.

Correctness. We have to show the following claims.

1. G is a Gröbner basis of F.
2. $\mathcal{V}_{\mathbb{R}}(G) = \mathcal{V}_{\mathbb{R}}(F)$.

Proof of 1.: Upon termination, for every $s = S(p,q)$, $p, q \in G$ we have $\mathrm{red}_G(s) = 0$. Thus by [1, Theorem 5.48] we have that G is a Gröbner basis.

Proof of 2.: We show that in each iteration it holds that $\mathcal{V}_{\mathbb{R}}(\langle G'\rangle) = \mathcal{V}_{\mathbb{R}}(\langle G' \cup Q\rangle)$. First, notice that all $S(p,q)$ with $p, q \in G'$ are in $\langle G'\rangle$, and so is $\mathrm{red}_{G'}(p)$ for $p \in G'$. Thus, for any s in $U(G', s)$ we have that $s \in \langle G'\rangle$, thus $\langle s\rangle \subseteq \langle G\rangle$. Therefore, we get $\mathcal{V}_{\mathbb{C}}(\langle s\rangle) \supseteq \mathcal{V}_{\mathbb{C}}(\langle G'\rangle)$ and $\mathcal{V}_{\mathbb{R}}(\langle s\rangle) \supseteq \mathcal{V}_{\mathbb{R}}(\langle G'\rangle)$. From the definition, $\mathcal{V}_{\mathbb{R}}(\langle Q\rangle) \supseteq \mathcal{V}_{\mathbb{R}}(\langle G'\rangle)$. Now for $\langle G'\rangle \cup \langle Q\rangle = I$, we get $\mathcal{V}_{\mathbb{R}}(I) = \mathcal{V}_{\mathbb{R}}(\langle G'\rangle) \cap \mathcal{V}_{\mathbb{R}}(\langle Q'\rangle)$ and thus $\mathcal{V}_{\mathbb{R}}(I) = \mathcal{V}_{\mathbb{R}}(\langle G'\rangle)$.

E Embedded rules

We give three rules, in order of practical relevance on the tested benchmarks. Furthermore, for any field K' and variety $V_{K'}$, we define the *vanishing ideal of* $V_{K'}$ as $\mathcal{I}(V_{K'}) = \{p \in K[\bar{x}] \mid p(\bar{a}) = 0 \text{ for all } \bar{a} \in V_{K'}\}$.

1. Calculating the separable part is not so cheap, but for single monomials it is trivial.
$$\frac{\prod_{i=0}^{n} {x_i}^{e_i} \in \langle P\rangle \qquad J = \{i \mid e_i > 0\}}{\prod_{i\in J} x_i \in \mathcal{I}(\mathcal{V}_{\mathbb{R}}(\langle P\rangle))}$$

2. This rule is based on the positive semi-definiteness of sums of squares. It was also applied in [13].

$$\frac{\sum s_i^2 \in \langle P \rangle}{s_i \in \mathcal{I}(\mathcal{V}_{\mathbb{R}}(\langle P \rangle))}$$

3. A rather special rule which helped very little. This rule was used differently in [11].

$$\frac{x_i^{2m+1} - x_j^{2n+1} \in \langle P \rangle \qquad (2n+1) \mid (2m+1) \qquad i \neq j}{x^{\frac{2m+1}{2n+1}} - x_j \in \mathcal{I}(\mathcal{V}_{\mathbb{R}}(\langle P \rangle))}$$

Aachener Informatik-Berichte

This list contains all technical reports published during the past three years. A complete list of reports dating back to 1987 is available from:

`http://aib.informatik.rwth-aachen.de/`

To obtain copies please consult the above URL or send your request to:

Informatik-Bibliothek, RWTH Aachen, Ahornstr. 55, 52056 Aachen,
Email: `biblio@informatik.rwth-aachen.de`

2010-01 * Fachgruppe Informatik: Jahresbericht 2010
2010-02 Daniel Neider, Christof Löding: Learning Visibly One-Counter Automata in Polynomial Time
2010-03 Holger Krahn: MontiCore: Agile Entwicklung von domänenspezifischen Sprachen im Software-Engineering
2010-04 René Wörzberger: Management dynamischer Geschäftsprozesse auf Basis statischer Prozessmanagementsysteme
2010-05 Daniel Retkowitz: Softwareunterstützung für adaptive eHome-Systeme
2010-06 Taolue Chen, Tingting Han, Joost-Pieter Katoen, Alexandru Mereacre: Computing maximum reachability probabilities in Markovian timed automata
2010-07 George B. Mertzios: A New Intersection Model for Multitolerance Graphs, Hierarchy, and Efficient Algorithms
2010-08 Carsten Otto, Marc Brockschmidt, Christian von Essen, Jürgen Giesl: Automated Termination Analysis of Java Bytecode by Term Rewriting
2010-09 George B. Mertzios, Shmuel Zaks: The Structure of the Intersection of Tolerance and Cocomparability Graphs
2010-10 Peter Schneider-Kamp, Jürgen Giesl, Thomas Ströder, Alexander Serebrenik, René Thiemann: Automated Termination Analysis for Logic Programs with Cut
2010-11 Martin Zimmermann: Parametric LTL Games
2010-12 Thomas Ströder, Peter Schneider-Kamp, Jürgen Giesl: Dependency Triples for Improving Termination Analysis of Logic Programs with Cut
2010-13 Ashraf Armoush: Design Patterns for Safety-Critical Embedded Systems
2010-14 Michael Codish, Carsten Fuhs, Jürgen Giesl, Peter Schneider-Kamp: Lazy Abstraction for Size-Change Termination
2010-15 Marc Brockschmidt, Carsten Otto, Christian von Essen, Jürgen Giesl: Termination Graphs for Java Bytecode
2010-16 Christian Berger: Automating Acceptance Tests for Sensor- and Actuator-based Systems on the Example of Autonomous Vehicles
2010-17 Hans Grönniger: Systemmodell-basierte Definition objektbasierter Modellierungssprachen mit semantischen Variationspunkten
2010-18 Ibrahim Armaç: Personalisierte eHomes: Mobilität, Privatsphäre und Sicherheit
2010-19 Felix Reidl: Experimental Evaluation of an Independent Set Algorithm
2010-20 Wladimir Fridman, Christof Löding, Martin Zimmermann: Degrees of Lookahead in Context-free Infinite Games

2011-01 * Fachgruppe Informatik: Jahresbericht 2011

2011-02 Marc Brockschmidt, Carsten Otto, Jürgen Giesl: Modular Termination Proofs of Recursive Java Bytecode Programs by Term Rewriting

2011-03 Lars Noschinski, Fabian Emmes, Jürgen Giesl: A Dependency Pair Framework for Innermost Complexity Analysis of Term Rewrite Systems

2011-04 Christina Jansen, Jonathan Heinen, Joost-Pieter Katoen, Thomas Noll: A Local Greibach Normal Form for Hyperedge Replacement Grammars

2011-06 Johannes Lotz, Klaus Leppkes, and Uwe Naumann: dco/c++ - Derivative Code by Overloading in C++

2011-07 Shahar Maoz, Jan Oliver Ringert, Bernhard Rumpe: An Operational Semantics for Activity Diagrams using SMV

2011-08 Thomas Ströder, Fabian Emmes, Peter Schneider-Kamp, Jürgen Giesl, Carsten Fuhs: A Linear Operational Semantics for Termination and Complexity Analysis of ISO Prolog

2011-09 Markus Beckers, Johannes Lotz, Viktor Mosenkis, Uwe Naumann (Editors): Fifth SIAM Workshop on Combinatorial Scientific Computing

2011-10 Markus Beckers, Viktor Mosenkis, Michael Maier, Uwe Naumann: Adjoint Subgradient Calculation for McCormick Relaxations

2011-11 Nils Jansen, Erika Ábrahám, Jens Katelaan, Ralf Wimmer, Joost-Pieter Katoen, Bernd Becker: Hierarchical Counterexamples for Discrete-Time Markov Chains

2011-12 Ingo Felscher, Wolfgang Thomas: On Compositional Failure Detection in Structured Transition Systems

2011-13 Michael Förster, Uwe Naumann, Jean Utke: Toward Adjoint OpenMP

2011-14 Daniel Neider, Roman Rabinovich, Martin Zimmermann: Solving Muller Games via Safety Games

2011-16 Niloofar Safiran, Uwe Naumann: Toward Adjoint OpenFOAM

2011-17 Carsten Fuhs: SAT Encodings: From Constraint-Based Termination Analysis to Circuit Synthesis

2011-18 Kamal Barakat: Introducing Timers to pi-Calculus

2011-19 Marc Brockschmidt, Thomas Ströder, Carsten Otto, Jürgen Giesl: Automated Detection of Non-Termination and NullPointerExceptions for Java Bytecode

2011-24 Callum Corbett, Uwe Naumann, Alexander Mitsos: Demonstration of a Branch-and-Bound Algorithm for Global Optimization using McCormick Relaxations

2011-25 Callum Corbett, Michael Maier, Markus Beckers, Uwe Naumann, Amin Ghobeity, Alexander Mitsos: Compiler-Generated Subgradient Code for McCormick Relaxations

2011-26 Hongfei Fu: The Complexity of Deciding a Behavioural Pseudometric on Probabilistic Automata

2012-01 Fachgruppe Informatik: Annual Report 2012

2012-02 Thomas Heer: Controlling Development Processes

2012-03 Arne Haber, Jan Oliver Ringert, Bernhard Rumpe: MontiArc - Architectural Modeling of Interactive Distributed and Cyber-Physical Systems

2012-04 Marcus Gelderie: Strategy Machines and their Complexity

2012-05 Thomas Ströder, Fabian Emmes, Jürgen Giesl, Peter Schneider-Kamp, and Carsten Fuhs: Automated Complexity Analysis for Prolog by Term Rewriting

2012-06 Marc Brockschmidt, Richard Musiol, Carsten Otto, Jürgen Giesl: Automated Termination Proofs for Java Programs with Cyclic Data

2012-07 André Egners, Björn Marschollek, and Ulrike Meyer: Hackers in Your Pocket: A Survey of Smartphone Security Across Platforms

2012-08 Hongfei Fu: Computing Game Metrics on Markov Decision Processes

2012-09 Dennis Guck, Tingting Han, Joost-Pieter Katoen, and Martin R. Neuhäußer: Quantitative Timed Analysis of Interactive Markov Chains

2012-10 Uwe Naumann and Johannes Lotz: Algorithmic Differentiation of Numerical Methods: Tangent-Linear and Adjoint Direct Solvers for Systems of Linear Equations

2012-12 Jürgen Giesl, Thomas Ströder, Peter Schneider-Kamp, Fabian Emmes, and Carsten Fuhs: Symbolic Evaluation Graphs and Term Rewriting — A General Methodology for Analyzing Logic Programs

2012-15 Uwe Naumann, Johannes Lotz, Klaus Leppkes, and Markus Towara: Algorithmic Differentiation of Numerical Methods: Tangent-Linear and Adjoint Solvers for Systems of Nonlinear Equations

2012-16 Georg Neugebauer and Ulrike Meyer: SMC-MuSe: A Framework for Secure Multi-Party Computation on MultiSets

2013-01 * Fachgruppe Informatik: Annual Report 2013

2013-02 Michael Reke: Modellbasierte Entwicklung automobiler Steuerungssysteme in Klein- und mittelständischen Unternehmen

2013-03 Markus Towara and Uwe Naumann: A Discrete Adjoint Model for OpenFOAM

2013-04 Max Sagebaum, Nicolas R. Gauger, Uwe Naumann, Johannes Lotz, and Klaus Leppkes: Algorithmic Differentiation of a Complex C++ Code with Underlying Libraries

2013-05 Andreas Rausch and Marc Sihling: Software & Systems Engineering Essentials 2013

2013-06 Marc Brockschmidt, Byron Cook, and Carsten Fuhs: Better termination proving through cooperation

* These reports are only available as a printed version.

Please contact `biblio@informatik.rwth-aachen.de` to obtain copies.