I-RiSC: An SMT-Compliant Solver for the Existential Fragment of Real Algebra[*]

Ulrich Loup and Erika Ábrahám

RWTH Aachen University, Germany
`{loup,abraham}@cs.rwth-aachen.de`

Abstract. This paper connects research in computer science in the field of SAT-modulo-theories (SMT) solving and research in mathematics on decision procedures for real algebra. We consider a real algebraic decision procedure computing all realizable sign conditions of a set of polynomials. We modify this procedure so that it satisfies certain requirements needed for the embedding into an SMT-solver.

Key words: SMT Solving, Real Algebra, I-RiSC, FO Logic, DPLL(T)

1 Introduction

Though the propositional satisfiability problem (SAT), where the variables range over the values 1 (true) and 0 (false), is NP-complete, SAT-solvers are quite efficient in practice due to a vast progress in SAT-solving over the last years. In particular, the DPLL algorithm [12] and its recent improvements such as clause learning or sophisticated decision heuristics made SAT-solving highly efficient for practical problems, what led to a break-through of SAT-solving also in industry.

SAT-modulo-theories (SMT) solving aims at embedding decision procedures for various first-order theories into the SAT-solving context [1,15] This combination yields highly efficient solvers, which are frequently applied, for example, in the formal analysis, verification, and synthesis of systems, even over a continuous domain. For the domain of the real numbers, research in the area of SMT solving concentrates on linear real arithmetic. Prominent examples of SMT-solvers for this logic are `Z3` [14], `Yices` [9], `MathSAT` [7] and `OpenSMT` [6]. Less emphasis is put on *real algebra*, the first-order logic with addition and multiplication over the reals, which we address in this paper. However, there is a growing interest in SMT-solving for real algebra. This drift is reflected, for instance, by 2010's SMT-competition [16], where the non-linear real arithmetic (NRA) division was introduced for the first time. The few existing SMT-solvers supporting non-linear real algebraic constraints are incomplete. For example, `Z3`, `CVC3` [2], `MiniSMT` [18] and `ABsolver` [4] can handle only fragments of real algebra, whereas the solver `iSAT` [10], based on interval arithmetic, allows even trigonometrical expressions but may terminate with the answer "unknown".

[*] The original publication is available at `http://www.springerlink.com`.

Several decision procedures were developed for real algebra since the 1940s, which are currently operational in some computer algebra systems. The most well-known approaches are the CAD method [8] and Gröbner bases computations. The textbook [3] comprises the state of the art for algorithms in real algebraic geometry. This book is also available online at

`http://perso.univ-rennes1.fr/marie-francoise.roy/bpr-ed2-posted2.pdf.`

However, the employment of these procedures in an SMT-solver is not straightforward, because SMT-solvers impose some requirements on the embedded decision procedures for the approach to be feasible in practice [1, 26.4.1]:

- First of all, for efficient SMT-solving we need decision procedures that work *incrementally*. That is, after the consistency check of a set of real algebraic constraints the procedure should be able to extend the set by adding new constraints, and reuse the previous computations for the check of the extended set.
- For an unsatisfiable set of constraints the decision procedure should be able to determine a *minimal infeasible subset*, i.e., an unsatisfiable subset which is minimal in the sense that removing any constraint makes it satisfiable.
- The ability to *backtrack* should allow to remove previously added constraints.

Unfortunately, current decision procedures for real algebra do not support the above functionalities. In this paper we describe how a method from [3], based on computing realizable sign conditions, can be modified to satisfy the requirements for SMT-solving. We call the modified new method I-RiSC(Incremental Realization of Sign Conditions).

2 Preliminaries

2.1 SMT-Solving

DPLL-based decision procedures [12] are also applicable to logics richer than propositional logic, by abstracting all non-propositional atomic constraints by propositional variables. This approach is called lazy *SAT-modulo-theories* (*SMT*) solving (cf. Fig. 1).

Full lazy (*off-line*) SMT-solvers first create a Boolean skeleton of the input formula, replacing all theory constraints by fresh Boolean variables. The resulting Boolean formula is passed to a SAT-solver, which searches for a satisfying assignment. If it does not succeed, the formula is unsatisfiable. Otherwise, the assignment found corresponds to certain truth values for the theory constraints and has to be verified by the theory solver. If the constraints are satisfiable, then the original formula is satisfiable. Otherwise, if the theory solver detects that the conjunction of the corresponding theory constraints is unsatisfiable, it then hands over a reason for the unsatisfiability, a *minimal infeasible subset* of the theory constraints, to the SAT-solver. The SAT-solver uses this piece of information to exclude the detected conflict from further search. Afterwards,

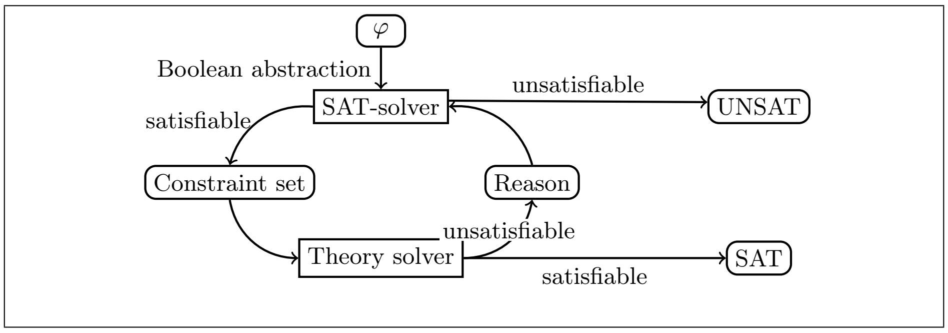


Fig. 1: The basic scheme of DPLL(T)-based SMT-solving

the SAT-solver computes again an assignment for the refined Boolean problem, which in turn has to be verified by the theory solver. Continuing this iteration in the end decides the satisfiability of the input formula.

Such full lazy check is often disadvantageous, since the SAT-solver may do a lot of needless work by extending an already (in the theory domain) contradictory partial assignment. *Less lazy* (*on-line*) DPLL(T) variants of the procedure call the theory solver more often, handing over constraints corresponding to partial assignments. To do so efficiently, the theory solver should accept constraints in an *incremental* fashion, where computation results of previous steps can be reused. In case of a conflict the theory solver should also be able to *backtrack*, i.e., remove the last asserted constraints.

Note that we strictly separate the satisfiability checks in the Boolean and in the theory domains, that means, we do not consider theory propagation embedded in the DPLL search like, e.g., Yices [9] does.

2.2 Real Algebra

In SMT-solving we consider only a fragment of real algebra, containing existentially quantified conjunctions of real algebraic constraints c, which compare *polynomials* p to zero:

$$\begin{array}{rcccccccccc} p & ::= & 0 & \mid & 1 & \mid & x & \mid & (p+p) & \mid & (p \cdot p) \\ c & ::= & p=0 & \mid & p<0 & \mid & p>0 & & & & \end{array}$$

The operators $+$ and $\cdot$ have the standard semantics of addition and multiplication. We stick to a more algebraic point of view and refer to [13] or [3] for basic notions on real algebra.

Let $n \in \mathbb{N}$ with $n \geq 1$ and $\mathbb{Z}[x_1, \ldots, x_n]$ be the set of all polynomials in the real-valued variables $x_1, \ldots, x_n$, called the polynomial ring of multivariate polynomials with integer coefficients. For a polynomial $p \in \mathbb{Z}[x_1, \ldots, x_n]$ the *x_i-degree of p*, written $\deg_{x_i}(p)$, is the highest exponent at x_i in p, and $[x_i^d]p$ denotes the *coefficient of x_i^d in p*, which is a polynomial in $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n$. A polynomial p is called a *monomial* if $p = \prod_{i=1}^{n} x_i^{e_i}$ with $e_i \in \mathbb{N}$; and in this case we define its degree as $\deg(p) := \sum_{i=1}^{n} e_i$. For arbitrary multivariate polynomials p we set $\deg(p) := \max\{\deg(q) \mid q \text{ monomial of } p\}$. The power set of S is denoted by 2^S, and $\binom{S}{k} := \{S' \in 2^S \mid |S'| = k\}$.

A *system of m real algebraic constraints* is a sequence $p_1 \sim_1 0, \ldots, p_m \sim_m 0$ of real algebraic constraints with $m \geq 1$ and $p_i \in \mathbb{Z}[x_1, \ldots, x_n]$, $\sim_i \in \{=, <, >\}$ for $1 \leq i \leq m$. We define the *signs* $\sigma = (\sigma_1, \ldots, \sigma_m) \in \{-1, 0, 1\}^m$ of the polynomials $P = (p_1, \ldots, p_m)$ by $\operatorname{sgn}(p_i) = \sigma_i$ for all $1 \leq i \leq m$. We call (P, σ) a *sign condition on P*. The set

$$\Re eali_\sigma(P) = \{(a_1, \ldots, a_n) \in \mathbb{R}^n \mid \operatorname{sgn}(p_i(a_1, \ldots, a_n)) = \sigma_i, 1 \leq i \leq m\}$$

of real solutions to the sign condition (P, σ) is called the *realization of* (P, σ). Note that $\Re eali_\sigma(P)$ is also a special *semi-algebraic set*. A sign condition (P, σ) is called *realizable* if $\Re eali_\sigma(P) \neq \emptyset$ (cf. [3, Def. 2.25]). The sign condition $(P, \{0\}^m)$ is said to be *algebraic*.

The satisfiability problem for conjunctions of real algebraic constraints, which we address in this paper, can be formulated in an algebraic context as follows:

Problem 1 Satisfiability problem of real algebraic conjunctions

Input:	$n, m \in \mathbb{N}$, $m, n \geq 1$, $P \in \mathbb{Z}[x_1, \ldots, x_n]^m$, $\sigma \in \{-1, 0, 1\}^m$.
Problem:	Determine whether $\Re eali_\sigma(P) = \emptyset$ and compute an $a \in \Re eali_\sigma(P)$ in case the set is not empty.

A realization of a sign condition can be composed of several sign-invariant subsets which do not have to be connected among each other. The connected sign-invariant sets are defined as follows.

Definition 1 (Sign-invariance, region)
Let $n, m \in \mathbb{N}$, $m, n \geq 1$, $R \subseteq \mathbb{R}^n$, $P \in \mathbb{Z}[x_1, \ldots, x_n]^m$, and $\sigma \in \{-1, 0, 1\}^m$.

- *R is said to be* σ-invariant over P *if $R \subseteq \Re eali_\sigma(P)$.*
- *R is said to be* sign-invariant over P *or* P-sign invariant *if there is a $\sigma \in \{-1, 0, 1\}^m$ such that $R \subseteq \Re eali_\sigma(P)$.*
- *R is called a* region *if $R \neq \emptyset$ and R is connected, i.e., $R \neq (A \cap R) \cup (B \cap R)$ with $A \cap R \neq \emptyset$ and $B \cap R \neq \emptyset$ for any open, nonempty sets $A, B \subseteq \mathbb{R}^n$ with $A \cap B = \emptyset$.*

Considering the real line, the maximal sign-invariant regions have a simple structure: The P-sign invariant regions are confined by the roots and intersection points of the polynomials of P. This results in a decomposition of $\mathbb{R}$ into these points and the open intervals between the points. All points which can be solutions to a sign condition are called samples.

Definition 2 (Sample)
A set $S \subseteq \mathbb{R}^n$ is called set of samples for the sign condition (P, σ) *if $S \cap R \neq \emptyset$ for each maximal σ-invariant region $R \subseteq \mathbb{R}^n$.*

A natural approach to determine a real-valued point which satisfies a given semi-algebraic system of polynomials is to construct sample points for each maximal sign-invariant region and to search in the set of sample points for a solution point. The crucial part is the construction of the sample points. The CAD-method (partial-CAD), for example, tackles this problem by iteratively projecting the input set of polynomials until only univariate polynomials are left. Their roots, computed as algebraic numbers, and all points between the roots as well as one point below the smallest and one point above the largest root are the possible values for the current component of the sample points for the sign-invariant regions in a CAD.

If the set of samples for a given sign condition (P, σ) is empty, the sign condition is not realizable. In terms of SMT-solving, (P, σ) is then called an infeasible subset, which the SAT-solver gets back as a reason of unsatisfiability. We now define the notion of minimal infeasible subset precisely.

Definition 3 (Minimal infeasible subset)
Let $n, m \in \mathbb{N}$, $m, n \geq 1$, $P \in \mathbb{Z}[x_1, \ldots, x_n]^m$ and $\sigma \in \{-1, 0, 1\}^m$. The sign condition

$$((P_{i_1}, \sigma_{i_1}), \ldots, (P_{i_k}, \sigma_{i_k})), \qquad \{i_1, \ldots, i_k\} \subseteq \{1, \ldots, m\}, 1 \leq k \leq m$$

is called infeasible subset *if $((P_{i_1}, \sigma_{i_1}), \ldots, (P_{i_k}, \sigma_{i_k}))$ is not realizable. If in this case*

$$((P_{j_1}, \sigma_{j_1}), \ldots, (P_{j_{k-1}}, \sigma_{j_{k-1}}))$$

is realizable for all $\{j_1, \ldots, j_{k-1}\} \subsetneq \{i_1, \ldots, i_k\}$ with $\{j_1, \ldots, j_{k-1}\} \neq \emptyset$, then $((P_{j_1}, \sigma_{j_1}), \ldots, (P_{j_k}, \sigma_{j_k}))$ is said to be minimal.

For the explanation of a recent approach for the sample construction in Section 3, we introduce algebraic computations with infinitesimal values. These are needed for several transformations of the input polynomials in order to obtain a set of polynomials whose realizable sign conditions can be computed much more efficiently than with the CAD-method.

Puiseux series. Since the abstraction to general fields is not needed in this paper, we introduce the notion of Puiseux series over the real field $\mathbb{R}$. Let ε be an infinitesimal element and $j, k \in \mathbb{Z}$ with $j > 0$. We call a series

$$a = \sum_{i \in \mathbb{Z}, i \geq k} a_i \varepsilon^{\frac{i}{j}}$$

with $a_i \in \mathbb{R}$ a *Puiseux series in ε*. Puiseux series are Laurent series in $\varepsilon^{\frac{1}{j}}$. If, for example, $j = 1$ and $k = 0$ then the Puiseux series a is nothing else than a Taylor series in ε. The additional parameters in a Puiseux series allow for the representation of elements *algebraic* over the *field of rational functions* $\mathbb{R}(\varepsilon)$, i.e. the representation of roots of polynomials in $\mathbb{R}(\varepsilon)[x]$. We denote the field of algebraic Puiseux series $\mathbb{R}\langle\varepsilon\rangle$. Given several infinitesimal elements $\varepsilon_1 < \ldots < \varepsilon_l$, we identify $\mathbb{R}\langle\varepsilon_1, \ldots, \varepsilon_l\rangle$ with $\mathbb{R}\langle\varepsilon_1\rangle \cdots \langle\varepsilon_l\rangle$. For $a \in \mathbb{R}\langle\varepsilon_1, \ldots, \varepsilon_l\rangle$ we denote $\lim_{\varepsilon_1, \ldots, \varepsilon_l} a = [\varepsilon_1^0 \cdots \varepsilon_l^0]a$, i.e. the part of a which is constant in $\varepsilon_1, \ldots, \varepsilon_l$.

3 Computing Realizable Sign Conditions

Let $m, n \in \mathbb{N}$ with $n \geq 1$ and $P \in \mathbb{Z}[x_1, \ldots, x_n]^m$ throughout this section.

We describe a method from [3] for determining all realizable sign conditions over P by computing samples for them. After presenting a first approach to solve this problem, in a second step we give an improved variant utilizing some state-of-the-art optimizations. Apart from these, both algorithms have a similar structure: They compute all or possibly a bounded number of subsets of the possibly modified input polynomials, and for each of these subsets they construct the samples for a specific algebraic sign condition. The union of all of these samples builds the samples for the original sign condition. The different subsets of polynomials need to be considered in order to transform strict sign conditions to just equations (see [3, Prop. 13.1]).

In this paper, we solely concentrate on the search structure of the procedure. Therefore, we decouple the subset computation from the construction of samples. Moreover, we make the sample construction a black box represented by the sub-procedure `samples(`Q`,`$(x_1 \ldots x_n)$`)` where Q is a tuple of $k \geq 1$ polynomials with coefficients possibly using infinitesimal elements δ and γ.

The method `samples(`Q`,`$(x_1 \ldots x_n)$`)` combines [3, Alg. 12.17] and [3, Alg. 13.2]. For readers familiar with real algebra, we give an intuitive description of how `samples(`Q`,`$(x_1 \ldots x_n)$`)` works in Table 1. We use the term `samples(`Q`,`$(x_1 \ldots x_n)$`)` to refer to both, the method and its output. `samples(`Q`,` $(x_1, \ldots, x_n)$`)` has the time complexity $d^{\mathcal{O}(n)}$ where $d = \max\{\deg(Q_i) \mid 1 \leq i \leq k\}$ (cf. [3, p. 512]).

Listing 1.1 shows the first variant of the algorithm for determining all realizable sign conditions over P suggested in [3, Rem. 13.3], combined with [3, Alg. 13.2].

Notation for lists. Empty lists are denoted by () and list concatenations by $\oplus$. We use L_i to refer to the ith element of a list L; the same notation is used for tuples.

Listing 1.1. First algorithm for computing realizable sign conditions.

```
1  Input: $m, n \in \mathbb{N}$, $n \geq 1$, $P \in \mathbb{Z}[x_1, \ldots, x_n]^m$
2  Output: samples $S \subseteq \mathbb{R}^n$ for every sign condition $(P, \sigma)$
3
4  $S$ := $\emptyset$;
5  for $1 \leq i \leq m$:
6    for $\{j_1, \ldots, j_i\} \in \binom{\{1,\ldots,m\}}{i}$:
7      $S$ := $S \cup$ samples($(P_{j_1}, \ldots, P_{j_i})$, $(x_1, \ldots, x_n)$);
8  return $S$;
```

The correctness of this algorithm follows from [3, Prop. 13.2].

Taking the complexity of the samples computation into account, the number of steps performed by Listing 1.1 is $2^m d^{\mathcal{O}(n)}$, because lines 5 and 6 define a search through every subset of the given m polynomials.

An optimized version of Listing 1.1 needs $m^{n+1} d^{\mathcal{O}(n)}$ steps to compute all realizable sign conditions. This method is given by [3, Alg. 13.1] combined

Table 1 Description of the method `samples(`Q`,(`$x_1 \dots x_n$`))`.

`Input:` $k, n \in \mathbb{N}$, $k, n \geq 1$, $Q \in \mathcal{R}[x_1, \dots, x_n]^k$ with $\mathcal{R} \in \{\mathbb{Z}, \mathbb{Z}[\delta, \gamma]\}$
`Output:` set of samples in $\mathbb{R}^k$ for $(Q_1^2 + \cdots + Q_k^2 + (\varepsilon(x_1, \dots, x_n, y) - 1)^2, \{0\}^k)$

(1) Define $q := Q_1^2 + \cdots + Q_k^2 + (\varepsilon(x_1, \dots, x_n, y) - 1)^2$. In Listing 1.1, the polynomials $Q_1, \dots, Q_k$ are a selection of the input polynomials $P_1, \dots, P_m$. However in the improved Listing 1.2, $Q_1, \dots, Q_k$ comprise a subset of perturbed versions of the input polynomials. We will give some more details on the perturbation below.
Squaring the polynomials $Q_1, \dots, Q_k$ and adding the term $(\varepsilon(x_1, \dots, x_n, y) - 1)^2$ applies a perturbation of the given set of polynomials. The common roots of the transformed set of polynomials are bounded. This is achieved by intersecting the cylinders based on the extension of sign-invariant regions of Q to $\mathbb{R}\langle\varepsilon\rangle$ with the k-dimensional sphere with center 0 and radius $\frac{1}{\varepsilon}$, as given by $(\varepsilon(x_1, \dots, x_n, y) - 1)^2$ and an appropriate projection eliminating y (see [3, 12.6]).

(2) Generate, based on q, a *special Gröbner basis* G containing n elements, by applying additional perturbations to q utilizing a fresh infinitesimal element ζ (see [3, Not. 12.46]). The special structure of this Gröbner basis assures a finite number of common roots with multiplicity 1 of the polynomials in G (see [3, Lem. 12.45]). In particular, the remainders modulo the ideal generated by G can be computed using a finite multiplication table; and, in addition to it, each of them represents exactly one common root of G.

(3) Apply [3, Alg. 12.9] on the input G to obtain the finite, special multiplication tables mentioned in the previous step.

(4) Apply [3, Alg. 12.14] (performing $\lim_\zeta$ or $\lim_{\gamma,\zeta}$) using the special multiplication tables to compute *univariate representations* (see [3, p. 465]) for the roots of q. Note that this representation can still contain infinitesimal elements.

(5) Apply [3, Alg. 11.20] (performing $\lim_\varepsilon$ or $\lim_{\delta,\varepsilon}$) to remove the remaining infinitesimal elements. Multiplication with the main denominator results in univariate representations for the roots of q in $\mathbb{R}$.

with [3, Alg. 13.2]. We now describe some details on this optimization and give the improved algorithm in Listing 1.2 by making use of the black box `samples(`Q`, `$(x_1, \dots, x_n)$`)` as introduced above.

Let ε, δ, γ be infinitesimal elements with $\varepsilon > \delta > \gamma > 0$.

Definition 4 (Perturbed general position polynomial)
Let $d, i, n \in \mathbb{N}$ with $n \geq 1$ and $1 \leq i \leq n$, $p \in \mathbb{Z}[x_1, \dots, x_n]$, and $\mu \in \{-1, 1, -\gamma, \gamma\}$ be a perturbation value, then

$$\mathrm{PG}^{\mu}_{n,i,d}(p) := (1-\delta)p + \delta\mu(1 + \sum_{1 \leq j \leq n} i^j x_j^d)$$

denotes the perturbed general position polynomial *of p w.r.t. n, i, d, and μ.*

This perturbation of P enables that not more than n polynomials need to be combined in order to compute the realizable sign conditions over P. More precisely, let $d = \max\{\deg(p_i) \mid 1 \leq i \leq m\}$ and for $1 \leq i \leq m$

$$\Gamma_i := \{\mathrm{PG}^{1}_{n,i,d}(p_i), \mathrm{PG}^{-1}_{n,i,d}(p_i), \mathrm{PG}^{\gamma}_{n,i,d}(p_i), \mathrm{PG}^{-\gamma}_{n,i,d}(p_i)\},$$

then any n polynomials from different Γ_i have at most a finite number of common roots; in particular, no $n+1$ polynomials from different Γ_i have a common root (see [3, Prop. 13.6]).

Note that there are only $\sum_{j=0}^{n} \binom{m}{j} 4^j = m^{n+1}$ combinations to consider, in contrast to 2^m of the first approach.

Listing 1.2. Improved algorithm for computing realizable sign conditions.

```
1  Input: $m, n \in \mathbb{N}$, $n \geq 1$, $P \in \mathbb{Z}[x_1, \dots, x_n]^m$
2  Output: samples $S \subseteq \mathbb{R}^n$ for every sign condition $(P, \sigma)$
3
4  $S$ := $\emptyset$;
5  $\Gamma$ := $()$;
6  for $1 \leq i \leq m$:
7    $\Gamma$ := $\Gamma \oplus (\emptyset)$;
8  $d$ := $\max\{\deg(p_i) \mid 1 \leq i \leq m\}$;
9  for $1 \leq i \leq m$:
10   $\Gamma_i$ := $\{\mathrm{PG}^{1}_{n,i,d}(p_i), \mathrm{PG}^{-1}_{n,i,d}(p_i), \mathrm{PG}^{\gamma}_{n,i,d}(p_i), \mathrm{PG}^{-\gamma}_{n,i,d}(p_i)\}$;
11 for $1 \leq i \leq n$:
12   for $\{j_1, \dots, j_i\} \in \binom{\{1,\dots,m\}}{i}$:
13     for $(p_{j_1}, \dots, p_{j_i}) \in \Gamma_{j_1} \times \cdots \times \Gamma_{j_i}$:
14       $S$ := $S \cup$ samples($(p_{j_1}, \dots, p_{j_i})$, $(x_1, \dots, x_n)$);
15 return $S$;
```

Observe that Listing 1.2 is similar to Listing 1.1, only the perturbation performed in lines 8 to 10 and the additional loop over the combinations of perturbed general position polynomials in line 13 are new.

The correctness of the improved algorithm follows from the correctness of the algorithm presented in 1.1 as well as [3, Cor. 13.8], which states that the sample computation also works for the perturbed general position polynomials.

Both algorithms can be implemented using only a polynomial amount of space [3, Rem. 13.10].

4 The I-RiSCSolver

In this section, we present an SMT-compliant solver for the satisfiability problem of real algebraic conjunctions, which especially supports adding sign conditions incrementally, providing minimal infeasible subsets as reason for unsatisfiability or an assignment as proof of satisfiability, and the possibility to backtrack the search by a given number of steps. We call this solver I-RiSCreferring to Incremental computation of Realizable Sign Conditions.

We pursue a class-based design of I-RiSCas it will be implemented in `C++` using `GiNaCRA`, which is a library providing real algebraic data structures [11].

4.1 Data Structure

Let $m, n \in \mathbb{N}$ with $n \geq 1$, $P \in \mathbb{Z}[x_1, \ldots, x_n]^m$, and $\sigma \in \{-1, 0, 1\}^m$. When adding a sign condition, the I-RiSCsolver first chooses at most n of the sets of perturbed general position polynomials $\Gamma_1, \ldots, \Gamma_m$. The search tree for these selections can be represented as a binary tree of depth n where each inner node characterizes one decision whether to take a specific Γ_k or not. The left edge of an inner node labeled by 0 indicates that a specific Γ_k was not selected, the right edge labeled by 1 indicates that it was. The leaves of the tree are labeled with the bit words $w \in \{0, 1\}^n$ representing the path to the leaf, as shown in Fig. 2 for $n = 2$. A $w \in \{0, 1\}^n$ induces a choice of polynomials, denoted by

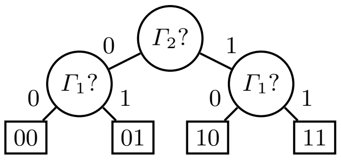


Fig. 2: I-RiSCsearch tree for two sets of polynomials Γ_1 and Γ_2.

$P_w = (P_i \mid w_i = 1)$, a choice of sets of perturbed general position polynomials, denoted by $\Gamma_w = (\Gamma_i \mid w_i = 1)$, and a choice of corresponding sign conditions, denoted by $\sigma_w = (\sigma_i \mid w_i = 1)$. The samples are generated only at the leafs of the search tree. The mapping $\alpha : \{0, 1\}^n \to 2^{\mathbb{R}^n}$ assigns a set of samples $\alpha(w)$ to each leaf $w \in \{0, 1\}^n$. We use the notation $|w|_1$ for the number of 1s in $w \in \{0, 1\}^n$; the empty word is denoted by ϵ. In order to traverse the search tree we use an order $\prec$ on bit words, defined as

$$w_1 \prec w_2 :\Leftrightarrow |w_1|_1 < |w_2|_1 \text{ or } (|w_1|_1 = |w_2|_1 \text{ and } w_1 <_{\text{lex}} w_2)$$

where $w_1 <_{\text{lex}} w_2$ compares the words lexicographically. For example, $00 <_{\text{lex}} 01 <_{\text{lex}} 10 <_{\text{lex}} 11$. Note that $\prec$ defines a strict total ordering on $\{0, 1\}^n$. This enables the definition of the $\text{next}_\prec$ operator providing the next element $\text{next}_\prec(w)$

of w. Moreover, we write $\prod(Q_1, \dots, Q_k)$ for the product $Q_1 \times \cdots \times Q_k$ of the sets $Q_1, \dots, Q_k$.

We now define a notion useful for the analysis of the I-RiSCmethods, especially the method for adding a new sign condition.

Definition 5 (Admissible and inadmissible leaves)
Let $w \in \{0,1\}^n$ be a leaf of the I-RiSCsearch tree. Let

$$S(w) := \{a \in \bigcup_{Q \in \prod \Gamma_w} \texttt{samples}(Q, (x_1 \dots x_n)) \mid a \textit{ is a sample for } (P_w, \sigma_w)\},$$

then w is called admissible *if $S(w) \neq \emptyset$, and* inadmissible *if $S(w) = \emptyset$.*

If a leaf $w \in \{0,1\}^n$ is admissible, then the set $S(w)$ can be reused in a later stage of the search. If, for example, the search is backtracked and the last i sign conditions, which did not contribute to the sample construction in w, are deleted, then $S(w) = S(w_{n-i+1} \dots w_n)$. Thus, we do not need to recompute the samples for $w_{n-i+1} \dots w_n$.

In the same fashion, we can reuse the result that $S(w)$ is empty. In particular, $S(w) = \emptyset$ entails that (P_w, σ_w) is an infeasible subset.

Lemma 1 (Minimal infeasible subset characterization)
Let $w \in \{0,1\}^n$ be a leaf of the I-RiSCsearch tree.
Then (P_w, σ_w) is a minimal infeasible subset iff w is inadmissible and for all $v \in \{0,1\}^n$ with $0 < |v|_1 < |w|_1$ it holds that v is admissible.

Proof. By Definition 3, (P_w, σ_w) is a minimal infeasible subset iff (P_v, σ_v) is realizable where $v = v_1 \dots v_{|w|}$ with $v_i \in \{w_i, 0\}$ for $1 \leq i \leq |w|$ such that not all v_i are 0 and not all v_i are w_i. Because of Definition 5, this is equivalent to v being admissible with $0 < |v|_1 < |w|_1$. □

Note that if all leaves of the I-RiSCsearch tree are traversed in order of $\prec$, then the first leaf w which proves to be inadmissible represents a minimal infeasible subset (P_w, σ_w), because only leaves with at least $|w|_1$ 1s would follow. The method `IRiSC.add` proceeds in this manner.

4.2 Class Design

In the following listing we specify the class containing the I-RiSCdata structure and methods for the interaction with the DPLL-based SAT-solver.

Table 2 contains annotations to the variables defined in Listing 1.3. Note that the type of the variables sometimes depends on m or n, which are instantiated at runtime. But this does not affect the realizability of the class in `C++` because the type can be implemented by means of dynamic data structures.

The methods in Listing 1.3 are described in Table 3.

Table 2 Annotations to the variables in Listing 1.3.

Variable	*Initial value*	*Description*
d	fixed	$d \geq \max\{\deg(q) \mid q \in P\}$ (preselected large enough)
m	0	current number of polynomials
n	fixed	dimension, i.e., the number of variables
P	$()$	current list of polynomials
σ	$()$	current list of signs
Γ	$()$	current list of sets of perturbed general position polynomials
W	$()$	list of search tree leafs such that W_k is the leaf where the search for the first k polynomials stopped
w	ϵ	current leaf of the search tree
α	$(\epsilon \mapsto \emptyset)$	set of samples not yet considered at the leaves of the search tree
S	$()$	list of m samples where the kth sample satisfies the first k sign conditions, in case they were satisfiable
r	1	flag determining the current state of w being a candidate for representing the reason of unsatisfiability ($r = 1$ means that w is still a candidate)
R	$()$	list of reasons for the unsatisfiability results where each reason represents a conjunction of constraints ($()$ in case of satisfiability)
K	$()$	list of satisfiability results such that W_k is the state of satisfiability (1 means satisfiable, 0 not) when the search for the first k polynomials stopped

Table 3 Annotations to the methods in Listing 1.3.

`IRiSC`	Constructor fixing the dimension, the maximal degree of input polynomials, and the initial values for the variables as given in Table 2.
`add`	Adds a constraint in the form of a sign condition (p, σ) to the system of polynomial equations and inequalities.
`backjump`	Jumps back to a previous state of the search, by undoing the last j additions of sign conditions.
`satisfiable`	Asks whether the current sign condition is satisfiable.
`reason`	Returns a reason of unsatisfiability as list of sign conditions if appropriate.
`assignment`	Returns a satisfying assignment if appropriate.

Listing 1.3. The class IRiSC.

```
class IRiSC
{
  const VAR $\delta$, $\gamma$, $\epsilon$;
  $d, m, n \in \mathbb{N}$;
  $P \in \mathbb{Z}[x_1, \ldots, x_n]^m$;
  $\sigma \in \{-1, 0, 1\}^m$;
  $\Gamma \in (2^{\mathbb{Z}}[x_1, \ldots, x_n])^m$;
  $W \in (\{0, 1\}^m)^m$;
  $w \in \{0, 1\}^n$;
  $\alpha : \{0, 1\}^n \to 2^{\mathbb{R}^n}$;
  $S \in (\mathbb{R}^n)^m$;
  bool $r$;
  $R \in 2^{\mathbb{Z}[x_1, \ldots, x_n]}$;
  $K \in \{0, 1\}^m$;

  IRiSC($d, n \in \mathbb{N}$);

  void add($p \in \mathbb{Z}[x_1, \ldots, x_n]$, $\varsigma \in \{-1, 0, 1\}$);
  void backjump($i \in \mathbb{N} : 1 \leq i \leq m$);
  bool satisfiable(){ return $K_m$; };
  $2^{\mathbb{Z}[x_1, \ldots, x_n]}$ reason(){ return $\{(R_m)_j \mid 1 \leq j \leq |R_m|\}$; };
  $\{x_1, \ldots, x_n\} \to \mathbb{R}$ assignment() { return $(x_j \mapsto (S_m)_j \mid 1 \leq j \leq n)$; };
}
```

4.3 Methods

The methods `IRiSC.add` and `IRiSC.backjump` are presented in Listings 1.4 and 1.5.

IRiSC.add. In lines 3 to 12, the new sign condition (p, ς) is stored in P and σ, the corresponding set of perturbed general position polynomials is attached to Γ, and w and α are adapted to the new search tree. In particular, the arriving sign condition is the new root of the search tree, and the old search tree is the new left successor of the root. This is illustrated in Fig. 3.

In case the sign conditions added so far are not realizable, the arriving sign condition does not change that result. Thus, the solver does not have to construct new samples and may abort the search immediately, copying the previous entries of W, S, R, and K. This case is captured by lines 13 to 16.

If the previously added sign conditions are realizable, the search for a sample satisfying all sign conditions continues at the currently smallest leaf w.r.t. $\prec$ which has an empty sample set. Since only those leaves w matter, which choose at least one polynomial for the sample construction ($|w|_1 > 0$), we must exclude the cases where $|w|_1 = 0$ (lines 17 and 18). Lines 19 to 45 comprise the outer loop, enumerating all leaves $w \in \{0, 1\}^n$ with $|w|_1 > 0$ in order of $\prec$.

Listing 1.4. IRiSC.add.

```
void IRiSC.add($p \in \mathbb{Z}[x_1, \ldots, x_n]$, $\varsigma \in \{-1, 0, 1\}$)
{
  $m$ := $m + 1$;
  $P$ := $P \oplus (p)$; $\sigma$ := $\sigma \oplus (\varsigma)$;
  $\Gamma$ := $\Gamma \oplus (\{\mathrm{PG}^{1}_{n,k,d}(p), \mathrm{PG}^{-1}_{n,k,d}(p), \mathrm{PG}^{\gamma}_{n,k,d}(p), \mathrm{PG}^{-\gamma}_{n,k,d}(p)\})$;
  $\widetilde{\alpha} : \{0,1\}^n \to 2^{\mathbb{R}^n}$;
  for $v \in \{0,1\}^{m-1}$: {
    $\widetilde{\alpha}(0v)$ := $\alpha(v)$;
    $\widetilde{\alpha}(1v)$ := $\emptyset$;
  }
  $\alpha$ := $\widetilde{\alpha}$;
  $w$ := $\min_{\prec}($ $w$, $\min_{\prec}\{v \in \{0,1\}^n \mid \alpha(v) = \emptyset\}$ );
  if $K_{m-1} = 0$: {
    $W$ := $W \oplus (w)$; $S$ := $S \oplus (1)$; $R$ := $R \oplus (R_{m-1})$; $K$ := $K \oplus (0)$;
    return;
  }
  if $|w|_1 = 0$:
    $w$ := $\mathrm{next}_{\prec}(w)$;
  while true: {
    if $\alpha(w) = \emptyset$: {
      $\alpha(w)$ := $\bigcup_{Q \in \prod \Gamma_w}$ samples($Q$, $(x_1, \ldots, x_n)$);
      $r$ := true;
    }
    else:
      $r$ := false;
    $S'$ := $\emptyset$;
    for $a \in \alpha(w)$: {
      if $a \in \Re eali_{\sigma_w}(P_w)$: {
        $r$ := false;
        $S'$ := $S' \cup \{a\}$;
        if $a \in \Re eali_{\sigma}(P)$: {
          $W$ := $W \oplus (w)$; $S$ := $S \oplus (a)$; $R$ := $R \oplus (())$;
          $K$ := $K \oplus (1)$; $\alpha(w)$ := $\alpha(w) \cup S'$;
          return;
        }
      }
      $\alpha(w)$ := $\alpha(w) \setminus \{a\}$;
    }
    if $r$ = true: {
      $W$ := $W \oplus (w)$;$S$ := $S \oplus (1)$;$R$ := $R \oplus ((P_w, \sigma_w))$;$K$ := $K \oplus (0)$;
      return;
    }
    $\alpha(w)$ := $S'$;
    $w$ := $\mathrm{next}_{\prec}(w)$;
  }
}
```

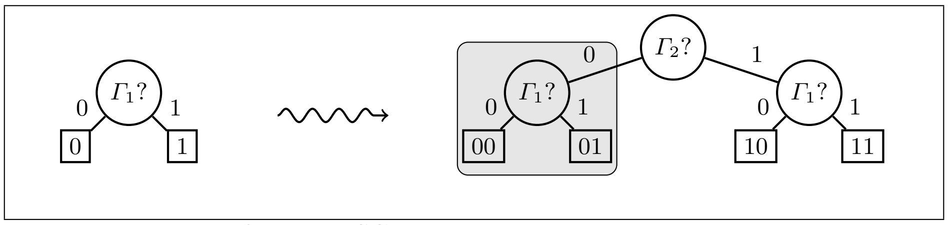


Fig. 3: Evolution of the I-RiSCsearch tree when adding a new sign condition with its corresponding set of perturbed general position polynomials Γ_2.

This loop can be exited in two cases, each of which also terminates: firstly, in case a sample satisfying *all* sign conditions was found; secondly, if w is inadmissible. The samples are constructed for all combinations polynomials of different sets of perturbed general position polynomials in line 21. This construction is only performed if no samples were constructed for this node beforehand. The inner loop in lines 27 to 38 then iterates over all samples for the current leaf w. The test of satisfiability is two-fold: first, the current sample is checked for satisfying (P_w, σ_w) in line 28; second, it is tested against the complete sign condition (P, σ) in line 31. This proceeding enables on the one hand the storage of the set $S(w)$, by the variable S' which later is assigned to $\alpha(w)$ in line 43. On the other hand, it allows for recognizing an inadmissible leaf, by using the variable r: r is only set to `true` in line 22 where w's samples are constructed newly, and r is switched to `false` immediately in case one sample satisfies the current local sign condition (P_w, σ_w), i.e., if w proves to be admissible. If otherwise all checks against (P_w, σ_w) fail, then w must be inadmissible; and by Lemma 1, (P_w, σ_w) is a minimal infeasible subset, which is stored as such in line 40. Note that r is set to `false` in line 25, where w is identified to be admissible.

IRiSC.backjump. The method `IRiSC.backjump(`$i \in \mathbb{N} : 1 \leq i \leq m$`)` removes the last i entries from P, σ, Γ, W, S, R, and K. It also updates the mapping α.

Listing 1.5. IRiSC.backjump.

```
void IRiSC.backjump($i \in \mathbb{N} : 1 \leq i \leq m$)
{
  $\widetilde{\alpha} : \{0,1\}^{m-i} \to 2^{\mathbb{R}^n}$;
  for $w \in \{0,1\}^m$:
    if $w = 0 \dots 0\, w_{i+1} \dots w_m$:
      $\widetilde{\alpha}(w_{i+1} \dots w_m)$ := $\alpha(w)$;
  $\alpha$ := $\widetilde{\alpha}$; $\widetilde{P}$ := $()$; $\widetilde{\sigma}$ := $\epsilon$; $\widetilde{\Gamma}$ := $()$;
  $\widetilde{W}$ := $()$; $\widetilde{S}$ := $()$; $\widetilde{R}$ := $()$; $\widetilde{K}$ := $()$;
  for $1 \leq k \leq m-i$: {
    $\widetilde{P}$ := $\widetilde{P} \oplus (P_k)$; $\widetilde{\sigma}$ := $\widetilde{\sigma} \oplus (\sigma_k)$; $\widetilde{\Gamma}$ := $\widetilde{\Gamma} \oplus (\Gamma_k)$;
    $\widetilde{W}$ := $\widetilde{W} \oplus (W_k)$; $\widetilde{S}$ := $\widetilde{S} \oplus (S_k)$; $\widetilde{R}$ := $\widetilde{R} \oplus (R_k)$; $\widetilde{K}$ := $\widetilde{K} \oplus (K_k)$;
  }
  $P$ := $\widetilde{P}$; $\sigma$ := $\widetilde{\sigma}$; $\Gamma$ := $\widetilde{\Gamma}$;
  $W$ := $\widetilde{W}$; $S$ := $\widetilde{S}$; $R$ := $\widetilde{R}$; $K$ := $\widetilde{K}$; $m$ := $m-i$; $w$ := $W_m$;
}
```

Fig. 4 shows the execution of `IRiSC.backjump(1)` using the example search tree of Fig. 3.

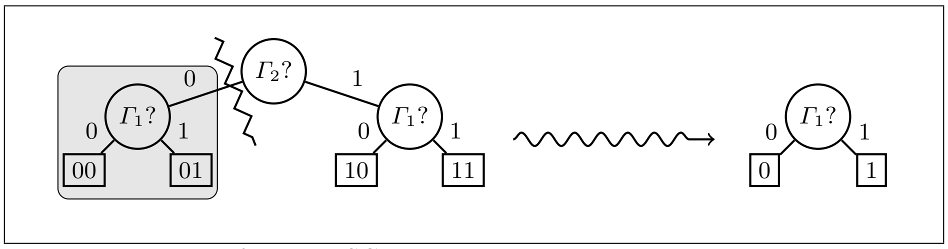


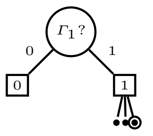
Fig. 4: Evolution of the I-RiSCsearch tree when backtracking, undoing the last step.

4.4 Example

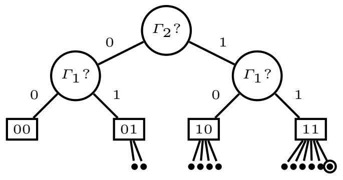
We explain the functioning of I-RiSCby an example, illustrating how the I-RiSCsearch tree evolves when adding the sign conditions $((p_1),(\sigma_1))$, $((p_2),(\sigma_2))$, $((p_3),(\sigma_3))$ one by one, then backtracking 2 steps, and finally add a last sign condition $((p_4),(\sigma_4))$.

We assume $((p_1,p_2),(\sigma_1,\sigma_2))$ and $((p_1,p_3),(\sigma_1,\sigma_3))$ being satisfiable, but $((p_2,p_3),(\sigma_2,\sigma_3))$ is not. We depict sample points constructed for one combination of a leaf by •, and circle the sample satisfying the current sign condition.

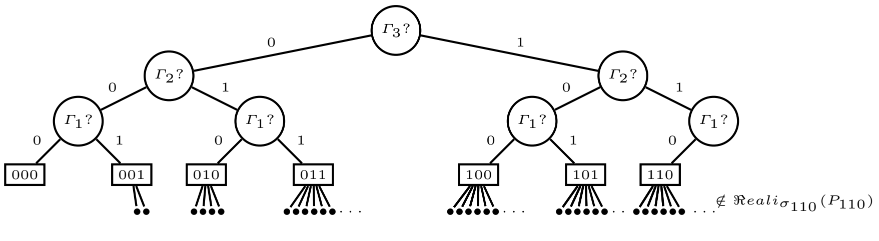
1. Search tree after adding (p_1,σ_1):



2. Search tree after adding (p_2,σ_2):



3. Search tree after adding (p_3,σ_3):

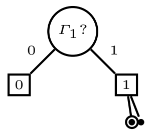


In the first step, I-RiSCconstructs the set Γ_1 of perturbed general position polynomials for p_1 and, using Γ_1, constructs three samples for the given sign condition, where the third sample satisfies it. The second condition is added by first constructing the node "Γ_2", which represents the choice of Γ_2, and second, appending the previous search tree as left successor of the new root node "Γ_2". The samples of Γ_1 are discarded, except for two of the two of them, which satisfy (p_1, σ_1) but not $((p_1, p_2), (\sigma_1, \sigma_2))$. I-RiSCproceeds by trying samples for Γ_2 and for (Γ_1, Γ_2). The latter yields a satisfying sample after computing 6 of the possible 16 choices of sets of general position polynomials. In the last step, (p_3, σ_3) is added. Now, following $\prec$ back to the next leaf without samples, Γ_3 is used for sample construction. In this example, the samples generated at the leaves 100, 011, or 101 satisfy their respective sign conditions (P_{100}, σ_{100}), (P_{011}, σ_{011}), or (P_{101}, σ_{101}), but not all sign conditions. The next condition (P_{110}, σ_{110}) is satisfied by none of its respective samples. This allows I-RiSCto stop the search process and store the sign condition (P_{110}, σ_{110}) as reason of unsatisfiability.

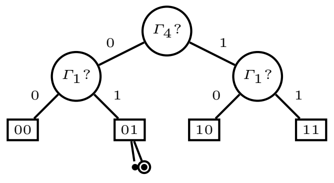
This reason is now removed by performing `IRiSC.backjump(2)`. Thereby, the search tree of the very first step is restored, still containing some of the samples computed for Γ_1. The final step of this example shows the adding of $((p_4), (\sigma_4))$, which here is satisfied already by the next sample, computed for Γ_1.

The resulting search trees may look as follows.

1. Search tree after backtracking 2 steps:



2. Search tree after adding (p_4, σ_4):



4.5 Optimizations

There are several ways to optimize the implementation of I-RiSC. Here, we list three of them.

- The method `IRiSC.add` does not have to compute all the complete set $\prod \Gamma_w$ and all samples for a given w in advance, but could store the current position in the $\prod \Gamma_w$ and the corresponding sample computation. This would at best avoid $4^{|w|_1} - 1$ calls of the method `IRiSC.samples` as well as save the memory consumed by the samples and the tuples of $\prod \Gamma_w$.
- I-RiSCdoes not yet support theory propagation. As discussed in [15], theory propagation can considerably speed-up the solving procedure. We plan first to implement the I-RiSCsolver as proposed in this paper, before further improving it by adding theory propagation.
- Last but not least, since good search heuristics are very important for the practical applicability of I-RiSC, we will investigate them in the near future.

5 Conclusion

In this paper we introduced a modification of an existing decision procedure for real algebra from [3]. Our modified algorithm I-RiSCsatisfies the requirement to be embedded into an efficient SMT-solver.

Currently, we work on the implementation of the I-RiSCmethod. The implementation, based on the `C++` library `GiNaCRA` [11], will allow to develop a complete SMT-solver for real algebra.

There are several points for further optimization of the proposed algorithm. In addition, I-RiSCcan be combined with techniques applied in other decision procedures for real algebra. One example is the reduction of the dimension of the input polynomials by variable elimination techniques. E.g., if the multivariate input polynomials are at most quadratic in one variable, then solution formulas can be used for variable elimination, as done by the virtual substitution [17].

References

1. Barrett, C., Sebastiani, R., Seshia, S., Tinelli, C.: Satisfiability Modulo Theories, chap. 26, pp. 825–885. Vol. 185 of Frontiers in Artificial Intelligence and Applications [5] (2009)
2. Barrett, C., Tinelli, C.: CVC3. In: CAV'07. LNCS, vol. 4590, pp. 298–302. Springer-Verlag (2007)
3. Basu, S., Pollack, R., Roy, M.: Algorithms in Real Algebraic Geometry. Springer-Verlag (2010)
4. Bauer, A., Pister, M., Tautschnig, M.: Tool-support for the analysis of hybrid systems and models. In: DATE 2007. pp. 924–929. European Design and Automation Association (2007)
5. Biere, A., Heule, M., van Maaren, H., Walsh, T.: Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press (2009)
6. Bruttomesso, R., Pek, E., Sharygina, N., Tsitovich, A.: The OpenSMT solver. In: TACAS 2010. LNCS, vol. 6015, pp. 150–153. Springer-Verlag (2010)
7. Bruttomesso, R., Cimatti, A., Franzén, A., Griggio, A., Sebastiani, R.: The mathSAT 4SMT solver. In: CAV 2008. LNCS, vol. 5123, pp. 299–303. Springer-Verlag (2008)
8. Collins, G.E.: Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In: Automata Theory and Formal Languages 1975. LNCS, vol. 33, pp. 134–183. Springer-Verlag, Berlin (1975)
9. Dutertre, B., Moura, L.D.: A fast linear-arithmetic solver for DPLL(T). In: CAV 2006. LNCS, vol. 4144, pp. 81–94. Springer-Verlag (2006)
10. Fränzle, M., Herde, C., Teige, T., Ratschan, S., Schubert, T.: Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. Journal on Satisfiability, Boolean Modeling and Computation 1(3-4), 209–236 (2007)
11. Loup, U., Ábrahám, E.: GiNaCRA: A C++ library for real algebraic computations. In: NFM 2011. LNCS, vol. 6617, pp. 512–517. Springer-Verlag (2011)
12. Marques-Silva, J., Lynce, I., Malik, S.: Conflict-Driven Clause Learning SAT Solvers, chap. 4, pp. 131–153. Vol. 185 of Frontiers in Artificial Intelligence and Applications [5] (2009)

13. Mishra, B.: Algorithmic Algebra. Texts and Monographs in Computer Science, Springer-Verlag (1993)
14. de Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer-Verlag (2008)
15. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). Journal of the ACM 53, 937–977 (2006)
16. SMT competition 2010. `http://www.smtcomp.org/2010/`
17. Weispfenning, V.: Quantifier elimination for real algebra – The quadratic case and beyond. Applicable Algebra in Engineering, Communication and Computing 8(2), 85–101 (1997)
18. Zankl, H., Middeldorp, A.: Satisfiability of non-linear (ir)rational arithmetic. In: LPAR-16. LNCS, vol. 6355, pp. 481–500. Springer-Verlag (2010)