Solving Non-Linear Arithmetic

Dejan Jovanović[1] and Leonardo de Moura[2]

[1] New York University
[2] Microsoft Research

Abstract. We present a new algorithm for deciding satisfiability of non-linear arithmetic constraints. The algorithm performs a Conflict-Driven Clause Learning (CDCL)-style search for a feasible assignment, while using projection operators adapted from cylindrical algebraic decomposition to guide the search away from the conflicting states.

1 Introduction

From the early beginnings in Persian and Chinese mathematics until the present day, polynomial constraints and the algorithmic ways of solving them have been one of the driving forces in the development of mathematics. Though studied for centuries due to the natural elegance they provide in modeling the real world, from resolving simple taxation arguments to modeling planes and hybrid systems, we are still lacking a practical algorithm for solving a system of polynomial constraints. Throughout the history of mathematics, many brilliant minds have studied and algorithmically solved many of the related problems, such as root finding and factorization of polynomials. But, it was not until Alfred Tarski [26] showed that the theory of real closed fields admits elimination of quantifiers that it became clear that a general decision procedure for solving polynomial constraints was possible. Granted a wonderful theoretical result of landmark importance, with its non-elementary complexity, Tarski's procedure was unfortunately totally impractical.

As one would expect, Tarski's procedure consequently has been much improved. Most notably, Collins [10] gave the first relatively effective method of quantifier elimination by cylindrical algebraic decomposition (CAD). The CAD procedure itself has gone through many revisions [8]. However, even with the improvements and various heuristics, its doubly-exponential worst-case behavior has remained as a serious impediment. The CAD algorithm works by decomposing $\mathbb{R}^k$ into connected components such that, in each cell, all of the polynomials from the problem are sign-invariant. To be able to perform such a particular decomposition, CAD first performs a *projection* of the polynomials from the initial problem. This projection includes many new polynomials, derived from the initial ones, and these polynomials carry enough information to ensure that the decomposition is indeed possible. Unfortunately, the size of these projection sets grows exponentially in the number of variables, causing the projection phase, and its consequent impact on the search space, to be a key hurdle to CAD scalability.

We propose a new decision procedure for the existential theory of the reals that tries to alleviate the above problem. As in [16,20,17], the new procedure performs a backtracking search for a model in $\mathbb{R}$, where the backtracking is powered by a novel conflict resolution procedure. Our approach takes advantage of the fact that each conflict encountered during the search is based on the current assignment and generally involves only a few constraints, a *conflicting core*. When in conflict, we project only the polynomials from the conflicting core and explain the conflict in terms of the current model. This means that we use projection conservatively, only for the subsets of polynomials that are involved in the conflict, and even then we reduce it further. As another advantage, the conflict resolution provides the usual benefits of a Conflict-Driven Clause Learning (CDCL)-style [24] search engine, such as non-chronological backtracking and the ability to ignore irrelevant parts of the search space. The projection operators we use as part of the conflict resolution need not be CAD based and, in fact, one can easily adapt projections based on other algorithms (e.g [19,3]).

Due to the lack of space and the volume of algorithms and concepts involved, we concentrate on the details of the decision procedure in this paper and refer the reader to the existing literature for further information [7,8,9].[3]

2 Preliminaries

As usual, we denote the ring of integers with $\mathbb{Z}$, the field of rational numbers with $\mathbb{Q}$, and the field of real numbers as $\mathbb{R}$. Unless stated otherwise, we assume all polynomials take integer coefficients, i.e. a polynomial $f \in \mathbb{Z}[\boldsymbol{y}, x]$ is of the form

$$f(\boldsymbol{y}, x) = a_m \cdot x^{d_m} + a_{m-1} \cdot x^{d_{m-1}} + \cdots + a_1 \cdot x^{d_1} + a_0 \ ,$$

where $0 < d_1 < \cdots < d_m$, and the coefficients a_i are in $\mathbb{Z}[\boldsymbol{y}]$ with $a_m \neq 0$. We call x the *top variable*. The highest degree d_m is the *degree* of the polynomial f in variable x, and we denote it with $\mathsf{deg}(f, x)$. The set of coefficients of f is denoted as $\mathsf{coeff}(f, x)$. We call a_m the *leading coefficient* in variable x, and denote it with $\mathsf{lc}(f, x)$. If we exclude the first k terms of the polynomial f, we obtain the polynomial $\mathsf{R}_k(f, x) = a_{m-k}x^{d_{m-k}} + \cdots + a_0$, called the k-th *reductum* of f. We write $\mathsf{R}^*(f, x)$ for the set $\{\mathsf{R}_0(f, x), \ldots, \mathsf{R}_m(f, x)\}$ containing all reductums. We denote the set of variables appearing in a polynomial f as $\mathsf{vars}(f)$ and call the polynomial *univariate* if $\mathsf{vars}(f) = \{x\}$ for some variable x. Otherwise the polynomial is *multivariate*, or a constant polynomial (if it contains no variables). Given a set of polynomials $A \subset \mathbb{Z}[x_1, \ldots x_n]$, we denote with A_k the subset of polynomials in A that belong to $\mathbb{Z}[x_1, \ldots, x_k]$, i.e. $A_k = A \cap \mathbb{Z}[x_1, \ldots, x_k]$.

A number $\alpha \in \mathbb{R}$ is a *root of the polynomial* $p \in \mathbb{Z}[x]$ iff $f(\alpha) = 0$. We call a real number $\alpha \in \mathbb{R}$ *algebraic* iff it is a root of a univariate polynomial

[3] The website `http://cs.nyu.edu/~dejan/nonlinear/` contains a technical report, our prototype nlsat, and experimental results. The technical report contains additional examples, proofs of all main theorems, additional references, and implementation details.

$f \in \mathbb{Z}[x]$, and we denote the field of all real algebraic numbers by $\mathbb{R}_{\mathsf{alg}}$. We can represent any algebraic number α as $(l,u)_f$, with $l,u \in \mathbb{Q}$, where α is a root of a polynomial f, and the only root in the interval (l,u).

Example 1. Consider the univariate polynomial $f_1 = 16x^3 - 8x^2 + x + 16$. This polynomial has only one root, the irrational number $\alpha_1 \approx -0.840661$ and we can represent it as $(-0.9,-0.8)_{f_1}$.

Given a set of variables $X = \{x_1,\ldots,x_n\}$, we call υ a *variable assignment* if it maps each variable x_k to a real algebraic number $\upsilon(x_k)$, the value of x_k under υ. We overload υ, as usual, to obtain the value of a polynomial $f \in \mathbb{Z}[x_1,\ldots,x_n]$ under υ and write it as $\upsilon(f)$. We say that a polynomial f *vanishes* under υ if $\upsilon(f) = 0$. We can update the assignment υ to map a variable x_k to the value α, and we denote this as $\upsilon[x_k \mapsto \alpha]$. Under a variable assignment υ that interprets the variables $\boldsymbol{y}$, some coefficients of a polynomial $f(\boldsymbol{y},x)$ may vanish. If a_k is the first non-vanishing coefficient of f, i.e., $\upsilon(a_k) \neq 0$, we write $\mathsf{R}(f,x,\upsilon) = a_k x^{d_k} + \cdots + a_0$ for the *reductum of f with respect to υ* (the non-vanishing part). Given any sequence of polynomials $\boldsymbol{f} = (f_1,\ldots,f_s)$ and a variable assignment υ we define the *vanishing signature* of $\boldsymbol{f}$ as the sequence $\mathsf{v\text{-}sig}(\boldsymbol{f},\upsilon) = (f_1,\ldots,f_k)$, where $k \leq s$ is the minimal number such that $\upsilon(f_k) \neq 0$, or s if they all vanish. For the polynomial $f(\boldsymbol{y},x)$ as above, we define the *vanishing coefficients signature* as $\mathsf{v\text{-}coeff}(f,x,\upsilon) = \mathsf{v\text{-}sig}(a_m,\ldots,a_0,\upsilon)$.

A *basic polynomial constraint* F is a constraint of the form $f \triangledown 0$ where f is a polynomial and $\triangledown \in \{<,\leq,=,\neq,\geq,>\}$. We denote the polynomial constraint that represents the *negation* of a constraint F with $\neg F$.[4] In order to identify the polynomial f of the constraint F, and the variables of F, we write $\mathsf{poly}(F)$ and $\mathsf{vars}(F)$, respectively. We normalize all constraints over constant polynomials to the dedicated constants true and false with the usual semantics. We write $\upsilon(F)$ to denote the evaluation of F under υ, which is the constraint $\upsilon(f) \triangledown 0$. If f does not evaluate to a constant under υ, then $\upsilon(F)$ evaluates to a new polynomial constraint F', where $\mathsf{poly}(F')$ can contain algebraic coefficients.

Borrowing from the extended Tarski language [4, Chapter 7], in addition to the basic constraints, we will also be working with *extended polynomial constraints*. An extended polynomial constraint F is of the form $x \triangledown_r \mathsf{root}(f,k)$, where $\triangledown_r \in \{<_r,\leq_r,=_r,\neq_r,\geq_r,>_r\}$, f is a polynomial in $\mathbb{Z}[\boldsymbol{y},\tilde{\mathrm{z}}]$, with $x \notin \mathsf{vars}(f)$, and the natural number $k \leq \mathsf{deg}(f,\tilde{\mathrm{z}})$ is the *root index*. Variable $\tilde{\mathrm{z}}$ is a distinguished free variable that cannot be used outside the root object. To be able to extract the polynomial of the constraint, we define $\mathsf{poly}(F) = f(\boldsymbol{y},x)$. Note that $\mathsf{poly}(F)$ replaces $\tilde{\mathrm{z}}$ with x. The semantics of the predicate $\triangledown_r$ under a variable assignment υ is the following. If the polynomial $\upsilon(f)$ is univariate, and υ assigns x to α, the (Boolean) value of the constraint can be determined as follows. If the univariate polynomial $\upsilon(f) \in \mathbb{R}_{\mathsf{alg}}[\tilde{\mathrm{z}}]$ has the roots $\beta_1 < \cdots < \beta_n$, with $k \leq n$, and $\alpha \triangledown \beta_k$ holds, then the predicate evaluates to true. Otherwise it evaluates to false. We denote the number of real roots of a univariate polynomial

[4] For example $\neg(x^2+1>0) \equiv x^2+1 \leq 0$.

f as $\mathsf{rootcount}(f)$. Naturally, if F is an extended polynomial constraint, so is the negation $\neg F$.[5]

A *polynomial constraint* is either a basic or an extended one. Given a set of polynomial constraints $\mathcal{F}$, we say that the variable assignment υ satisfies $\mathcal{F}$ if it satisfies each constraint in $\mathcal{F}$. If there is such a variable assignment, we say that $\mathcal{F}$ is *satisfiable*, otherwise it is *unsatisfiable*. A *clause* of polynomial constraints is a disjunction $C = F_1 \vee \ldots \vee F_n$ of polynomial constraints. We use $\mathsf{literals}(C)$ to denote the set $\{F_1, \neg F_1, \ldots, F_n, \neg F_n\}$. We say that the clause C is satisfied under the assignment υ if some polynomial constraint $F_j \in C$ evaluates to true under υ. Finally, a *polynomial constraint problem* is a set of clauses $\mathcal{C}$, and it is satisfiable if there is a variable assignment υ that satisfies all the clauses in $\mathcal{C}$. If the clauses of $\mathcal{C}$ contain the variables $x_1, \ldots, x_n$ then, for $k \leq n$, we denote with $\mathcal{C}_k$ the subset of the clauses that only contains variables $x_1, \ldots, x_k$.

3 An Abstract Decision Procedure

We describe our procedure as an abstract transition system in the spirit of Abstract DPLL [21]. The crucial difference between the system we present is that we depart from viewing the Boolean search engine and the theory reasoning as two separate entities that communicate only through existing literals. Instead, we allow the model that the theory is trying to construct to be involved in the search and in explaining the conflicts, while allowing new literals to be introduced so as to support more complex conflict analyses. The transition system presented here applies to non-linear arithmetic, but it can in general be applied to other theories.

The states in the transition system are indexed pairs of the form $\langle M, \mathcal{C} \rangle_n$, where M is a sequence (usually called a *trail*) of *trail elements*, and $\mathcal{C}$ is a set of clauses. The index n denotes the current *stage* of the state. Trail elements can be decided literals, propagated literals, or a variable assignment. A *decided literal* is a polynomial constraint F that we assume to be true. On the other hand, a *propagated literal*, denoted as $E \rightarrow F$, marks a polynomial constraint $F \in E$ that is implied to be true in the current state by the clause E (the *explanation*). In both cases, we say that the constraint F appears in M, and write this as $F \in M$. We denote the set of polynomial constraints appearing in M with $\mathsf{constraints}(M)$. We say M is *non-redundant* if no polynomial constraint appears in M more than once. A *trail variable assignment*, written as $x \mapsto \alpha$, is an assignment of a single variable to a value $\alpha \in \mathbb{R}_{\mathsf{alg}}$. Given a trail M, containing variable assignments $x_{i_1} \mapsto \alpha_1, \ldots, x_{i_k} \mapsto \alpha_k$, in order, we can construct an assignment $\upsilon[M] = \upsilon_0[x_{i_1} \mapsto \alpha_1] \ \ldots \ [x_{i_k} \mapsto \alpha_k]$, where υ_0 is an empty assignment that does not assign any variables.

[5] Note that, for example, $\neg(x <_r \mathsf{root}(f, k))$ is not necessarily equivalent to $x \geq_r \mathsf{root}(f, k)$.

We say that the sequence M is *stage increasing* when the sequence is of the form

$$M = [\![N_1, x_1 \mapsto \alpha_1, \ldots, x_{k-1} \mapsto \alpha_{k-1}, N_k, x_k \mapsto \alpha_k, \ldots, x_{n-1} \mapsto \alpha_{n-1}, N_n]\!] \ ,$$

where, for each $k \leq n$, the sequence N_k does not contain any variable assignments, each constraint $F \in \mathsf{constraints}(N_k)$ contains the variable x_k, and (optionally) the variables $x_1, \ldots, x_{k-1}$ (and $\tilde{z}$). In such a sequence M, we denote with $\mathsf{stage}(M) = n$ the *stage* of the sequence. If $\mathcal{F} = \mathsf{constraints}(M)$, we say that M is *feasible*, when the set of univariate polynomial constraints $\upsilon[M](\mathcal{F})$ has a solution. We write $\mathsf{feasible}(M)$ to denote the feasible set of $\upsilon[M](\mathcal{F})$. Given an additional polynomial constraint $F \in \mathbb{Z}[x_1, \ldots, x_n]$, we say that F is *compatible* with the sequence M, when $\mathsf{feasible}([\![M, F]\!]) \neq \emptyset$ and denote this with a predicate $\mathsf{compatible}(F, M)$. The technical report contains additional details on how these procedures are implemented.

Our transition system will work over states that are *well-formed*. Intuitively, in such a state, we commit to the variable assignment, but make sure that the current stage is consistent on the Boolean level. With this in mind, given a polynomial constraint F with $\mathsf{vars}(F) \subseteq \{x_1, \ldots, x_n\}$, and a state M with $\mathsf{stage}(M) = n$, we define the *state value* of F in M as

$$\mathsf{value}(F, M) = \begin{cases} \upsilon[M](F) & x_n \notin \mathsf{vars}(F) \ , \\ \mathsf{true} & F \in \mathsf{constraints}(M) \ , \\ \mathsf{false} & \neg F \in \mathsf{constraints}(M) \ , \\ \mathsf{undef} & \text{otherwise.} \end{cases}$$

Naturally, we overload value to also evaluate clauses of polynomial constraints, and sets of clauses, i.e. for a clause C we define $\mathsf{value}(C, M)$ to be true, if any of the literals evaluates to true, false if all literals evaluate to false, and undef otherwise.

Definition 1 (Well-Formed State). *We say a state $\langle M, \mathcal{C} \rangle_n$ is well-formed when M is non-redundant, stage increasing with $\mathsf{stage}(M) = n$, and all of the following hold.*

1. *Clauses up to stage n are satisfied, i.e. we have that $\mathsf{value}(\mathcal{C}_{n-1}, M) = \mathsf{true}$.*
2. *The state is consistent, i.e. $\mathsf{feasible}(M) \neq \emptyset$ and for each $F \in \mathsf{constraints}(M)$ we have that that $\mathsf{value}(F, M) = \mathsf{true}$.*
3. *Propagated literals $E \rightarrow F$ are implied, i.e. for all literals $F' \neq F$ in E, $\mathsf{value}(F', M) = \mathsf{false}$.*

We are now ready to define the transition system. We separate the transition rules into three groups: the search rules, the clause processing rules, and the conflict analysis rules. The *search rules* are the main driver of the procedure, with the responsibility for selecting clauses to process, creating the variable assignment while lifting the stages, and detecting Boolean conflicts. The search rules operate on well-formed states $\langle M, \mathcal{C} \rangle_n$. If the search rules select a clause

C to process, we switch to a state $\langle M, \mathcal{C}\rangle_n \models C$, where we can apply the set of *clause processing rules*. The notation $\models C$ designates that we are performing semantic reasoning in order to assign a value to a literal of C. If the search rules detect that in the current state some clause $C \in \mathcal{C}$ is falsified, we switch to a state $\langle M, \mathcal{C}\rangle_n \vdash C$, where we can apply the *conflict analysis rules*. The notation $\vdash C$ denotes that we are trying to produce a proof of why C is inconsistent in the current state.

Finally, given a polynomial constraint problem $\mathcal{C}$, with $\mathsf{vars}(\mathcal{C}) = \{x_1, \ldots, x_n\}$, the overall goal of the procedure is, starting from an initial state $\langle [\![]\!], \mathcal{C}\rangle_1$, and applying the rules, to end up either in a state $\langle v, \mathsf{sat}\rangle$, indicating that the initial set of clauses $\mathcal{C}$ is satisfiable where the assignment v is the witness, or derive unsat, which indicates that the set $\mathcal{C}$ unsatisfiable.

Search Rules. Fig 1 presents the set of search rules. The Select-Clause rule selects one of the clauses of the current stage, whose state value is still undetermined, and transitions into the clause processing mode that will hopefully satisfy the clause. The Conflict rule detects if there is a clause of the current stage that is inconsistent in the current state, and transitions into the conflict resolution mode that will explain the conflict and backtrack appropriately. On the other hand, if all the clauses of the current stage are satisfied, we can either transition to the next stage, using the Lift-Stage rule, or conclude that our problem is satisfiable, using the Sat rule. Since at this point the current stage is consistent, in addition to formally introducing the new stage, the Lift-Stage rule selects a particular value for the current variable from the feasible set of the current stage. Note that once we move to the next stage, all the clauses of previous stages have values in the state, and can never be selected by the Select-Clause or the Conflict rules. We conclude this set of rules with the Forget rule that can be used to eliminate any learnt clause (a clause added while analyzing conflicts) from the current set of clauses.

Clause Processing Rules. In this set of rules, presented in Fig 2, we are trying to assign a currently unassigned literal of the given clause C, hoping to satisfy the clause. When one of the clause processing rules is applied, we immediately switch back to the search rules. As usual in a CDCL-style procedure, the simplest way to satisfy the clause C is to perform the Boolean unit propagation, if applicable, by using the $\mathbb{B}$-Propagate rule. We restrict the application of this rule so that adding the constraint to the state keeps it consistent, i.e., it is $\mathsf{compatible}$ with the current set of constraints. If this is the case, we add the constraint to the state together with the explanation (clause C itself). To allow more complex propagations, the ones that are valid in $\mathbb{R}$ modulo the current state, we provide the $\mathbb{R}$-Propagate rule. This rule can propagate a constraint from the clause, if assuming the negation would be incompatible with the current state. The $\mathbb{R}$-Propagate rule is equipped with an explanation function $\mathsf{explain}$. The $\mathsf{explain}$ function, given a polynomial constraint F, and the trail M, returns the explanation clause $E = \mathsf{explain}(F, M)$ that is valid in $\mathbb{R}$, and implies the constraint F under the current assignment i.e., $F \in E$, and all literals in E but F are false

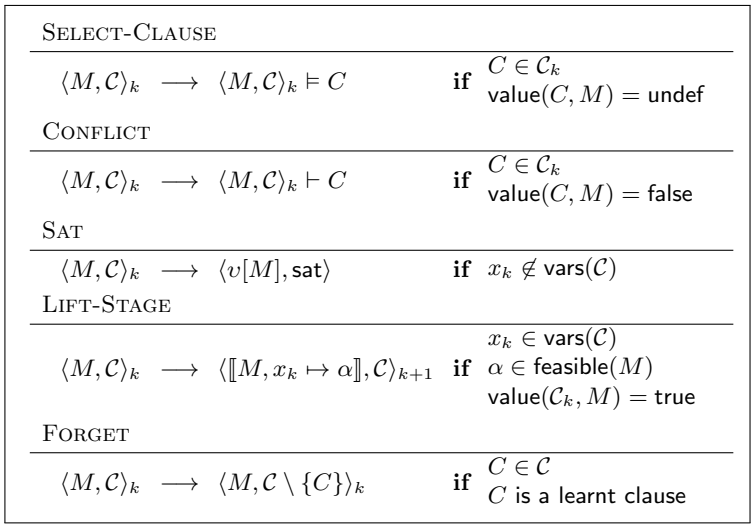


Fig. 1. The search rules.

in the state. The clause E may contain new literals that do not occur in $\mathcal{C}$, as long as they evaluate to false in the state. To simplify the presentation, in the $\mathbb{R}$-PROPAGATE rule, the explanation clause E is eagerly generated, but in our actual implementation, we compute them only if they are needed during conflict resolution. Finally, if we cannot deduce the value of an unassigned literal, we can assume a value for such a literal using the DECIDE-LITERAL rule.

Conflict analysis rules. The conflict analysis rules start from an initial proper state $\langle M, \mathcal{C}\rangle_n \vdash C$, where $C \in \mathcal{C}$ is the conflicting clause. The conflict analysis is a standard Boolean conflict analysis [24] with a model-based twist. As the rules move the state backwards, the goal is to construct a new resolvent clause R, that will explain the conflict and ensure progress in the search. This means that, when we backtrack the sequence M just enough, the addition of R will ensure progress in the search by eliminating the inconsistent part from the state, and thus forcing the search rules to change some of the choices made. On the other hand, if the conflict analysis backtracks the state all the way into an empty state, this will be a signal that the original problem is unsatisfiable. Once the conflict analysis backtracks enough and deduces the resolvent R, then we pass it to the clause processing immediately.[6]

Termination. Our decision procedure consists of all three sets of rules described above. Any derivation will proceed by switching amongst the three distinct modes. Proving termination in the basic CDCL(T) framework is usually a fairly straightforward task, as the new explanation and conflict clauses always contain

[6] This is crucial in order to ensure termination.

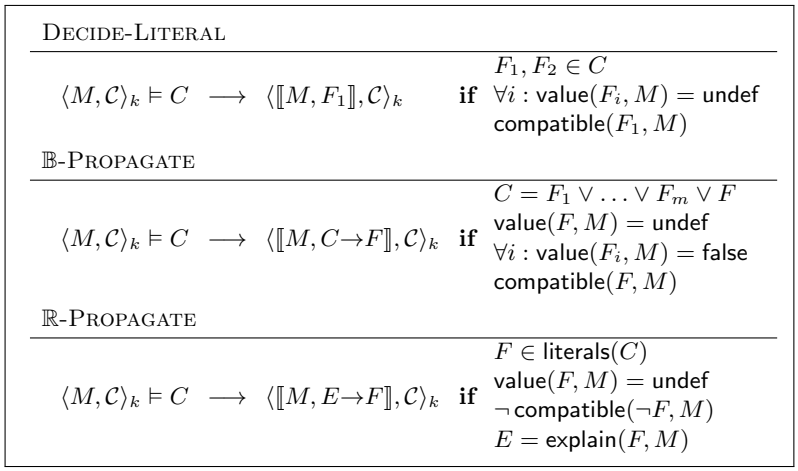


Fig. 2. The clause satisfaction rules.

only literals from the finite set of literals in the initial set of constraints. In our case, the main conundrum in proving termination is that we allow the explanations to contain fresh constraints, which, if we are not careful, could lead to non-termination. We therefore require the set of new constraints to be finite. We call an explanation function $\mathsf{explain}$ a *finite basis explanation function* with respect to a set of constraints $\mathcal{C}$, when there is a finite set of polynomial constraints $\mathcal{B}$ such that for any derivation of the proof rules, the clauses returned by applications of $\mathsf{explain}$ always contain only constraints from the basis $\mathcal{B}$. Having such an explanation function will therefore provide us with a termination argument, and we will provide one such explanation function for the theory of reals in the next section.

Theorem 1. *Given a set of polynomial constraints $\mathcal{C}$, and assuming a finite basis explanation function* $\mathsf{explain}$*, any derivation starting from the initial state $\langle [\![]\!], \mathcal{C}\rangle_1$ will terminate either in a state $\langle \upsilon, \mathsf{sat}\rangle$, where the assignment υ satisfies the constraints $\mathcal{C}$, or in the* unsat *state. In the later case, the set of constraints $\mathcal{C}$ is unsatisfiable in $\mathbb{R}$.*

4 Producing Explanations

Given a polynomial constraint F with $\mathsf{poly}(F) \in \mathbb{Z}[\boldsymbol{y}, x]$, and a trail M such that $\neg F$ is not compatible with M, the procedure $\mathsf{explain}(F, M)$ returns an explanation clause E that implies F in the current state. In principle, for any theory that admits elimination of quantifiers, it is possible to construct an explanation function $\mathsf{explain}$. In this section, we describe how to produce an $\mathsf{explain}$ procedure for theory of the reals based on cylindrical algebraic decomposition (CAD). Before that, we first make a short interlude into the world of CAD.

RESOLVE-PROPAGATION			
$\langle [\![M, E{\to}F]\!], \mathcal{C} \rangle_k \vdash C$	$\longrightarrow$	$\langle M, \mathcal{C} \rangle_k \vdash R$	**if** $\neg F \in C$, $R = \mathsf{resolve}(C, E, F)$
▷ resolve returns the standard Boolean resolvent			
RESOLVE-DECISION			
$\langle [\![M, F]\!], \mathcal{C} \rangle_k \vdash C$	$\longrightarrow$	$\langle M, \mathcal{C} \cup \{C\} \rangle_k \vDash C$	**if** $\neg F \in C$
CONSUME			
$\langle [\![M, F]\!], \mathcal{C} \rangle_k \vdash C$	$\longrightarrow$	$\langle M, \mathcal{C} \rangle_k \vdash C$	**if** $\neg F \notin C$
$\langle [\![M, E{\to}F]\!], \mathcal{C} \rangle_k \vdash C$	$\longrightarrow$	$\langle M, \mathcal{C} \rangle_k \vdash C$	**if** $\neg F \notin C$
DROP-STAGE			
$\langle [\![M, x_{k+1}{\mapsto}\alpha]\!], \mathcal{C} \rangle_{k+1} \vdash C$	$\longrightarrow$	$\langle M, \mathcal{C} \rangle_k \vdash C$	**if** $\mathsf{value}(C, M) = \mathsf{false}$
$\langle [\![M, x_{k+1}{\mapsto}\alpha]\!], \mathcal{C} \rangle_{k+1} \vdash C$	$\longrightarrow$	$\langle M, \mathcal{C} \cup \{C\} \rangle_k \vDash C$	**if** $\mathsf{value}(C, M) = \mathsf{undef}$
UNSAT			
$\langle [\![ ]\!], \mathcal{C} \rangle_1 \vdash C$	$\longrightarrow$	unsat	

Fig. 3. The conflict analysis rules.

4.1 Cylindrical Algebraic Decomposition

A crucial role in the theory of CADs and in the construction of our explain procedure is the property of delineability. Following the terminology used in CAD, we say that a connected subset of $\mathbb{R}^k$ is a *region*. A set of polynomials $\{f_1, \dots f_s\} \subset \mathbb{Z}[\boldsymbol{y}, x]$, with $\boldsymbol{y} = (y_1, \dots, y_n)$, is said to be *delineable* in a region $S \subseteq \mathbb{R}^n$ if for every f_i (and f_j) from the set, the following properties are invariant for any $\alpha \in S$:

1. the *total number of complex roots* of $f_i(\alpha, x)$;
2. the *number of distinct complex roots* of $f_i(\alpha, x)$;
3. the *number of common complex roots* of $f_i(\alpha, x)$ and $f_j(\alpha, x)$.

Example 2. Consider the polynomial $f = x^2+y^2+z^2-1$, with zeros of f depicted in Fig 4(a) together with two squiggly regions of $\mathbb{R}^2$. In the region S_1 that does not intersect the sphere, polynomial f is delineable, as the number of complex (and real) roots of $f(\alpha, x)$ is 2 for any α in S_1. In the region S_2 that intersects the sphere, f is not delineable, as the number of real roots of f varies from 0 (α's outside the unit circle), 1 (on the circle), and 2 (inside the unit circle).

We will call a *projection operator* any map P that, given a variable x and set of polynomials $A \subset \mathbb{Z}[\boldsymbol{y}, x]$, transforms A into a set of polynomials $\mathsf{P}(A, x) \subset \mathbb{Z}[\boldsymbol{y}]$. We call $\mathsf{P}(A, x)$ the *projection* of A under P with respect to variable x. In his seminal paper [10], Collins introduced a projection operator which we denote with P_c. In order to define the operator P_c, we first need to define some "advanced" operations on polynomials, and we refer the reader to [18,3,6] for a more detailed exposition.

Let $f, g \in \mathbb{Z}[\boldsymbol{y}, x]$ be two polynomials with $n = \min(\mathsf{deg}(f, x), \mathsf{deg}(g, x))$. For $k = 0, \ldots, n-1$, we denote with $\mathsf{S}_k(f, g, x)$ the k-th *subresultant* of f and g. The k-th subresultant is defined as the determinant of the k-th Sylvester-Habicht matrix of f and g, and is a polynomial of degree $\leq k$ in x with coefficients in $\mathbb{Z}[\boldsymbol{y}]$. The matrix in question is a particular matrix containing as elements the coefficients of f and g. Additionally, we denote with $\mathsf{psc}_k(f, g, x)$ the k-th *principal subresultant coefficient* of f and g, which is the coefficient of x^k in the polynomial $\mathsf{S}_k(f, g, x)$, and define $\mathsf{psc}_n(f, g, x) = 1$. We denote the sequence of principal subresultant coefficients as $\mathsf{psc}(f, g, x) = (\mathsf{psc}_0(f, g, x), \ldots, \mathsf{psc}_n(f, g, x))$.

Definition 2. *Given a set of polynomials $A = \{f_1, \ldots, f_m\} \subset \mathbb{Z}[\boldsymbol{y}, x]$ the Collins projector operator $\mathsf{P}_c(A, x)$ is defined as*

$$\bigcup_{f \in A} \mathsf{coeff}(f, x) \cup \bigcup_{\substack{f \in A \\ g \in \mathsf{R}^*(f, x)}} \mathsf{psc}(g, g'_x, x) \cup \bigcup_{\substack{i<j \\ g_i \in \mathsf{R}^*(f_i, x) \\ g_j \in \mathsf{R}^*(f_j, x)}} \mathsf{psc}(g_i, g_j, x) \ .$$

Let $A = \{f_1, \ldots, f_m\} \subset \mathbb{Z}[\boldsymbol{y}]$ be a set of polynomials, where $\boldsymbol{y} = (y_1, \ldots, y_n)$, and S be a region of $\mathbb{R}^n$. If for any assignment υ such that $\upsilon(\boldsymbol{y}) = \boldsymbol{\alpha} \in S$, the polynomials in A have the same sign under υ, we say that A is *sign-invariant on S*.

Theorem 2 (Theorem 4 in [10]). *Given a finite set of polynomials $A \subset \mathbb{Z}[\boldsymbol{y}, x]$, where $\boldsymbol{y} = (y_1, \ldots, y_n)$, and let S be a region of $\mathbb{R}^n$. If $\mathsf{P}_c(A)$ is sign invariant on S, then A is delineable over S.*

A *sign assignment* for a set of polynomials A is a mapping σ, from polynomials in A to $\{-1, 0, 1\}$. Given a set of polynomials $A \subset \mathbb{Z}[\boldsymbol{y}, x]$, we say a sign assignment σ is *realizable* with respect to some $\boldsymbol{\alpha}$ in $\mathbb{R}^n$, if there exists a $\beta \in \mathbb{R}$ such that every $f \in A$ takes the sign corresponding to its sign assignment, i.e., $\mathsf{sgn}(f(\boldsymbol{\alpha}, \beta)) = \sigma(f)$. The function sgn maps a real number to its sign $\{-1, 0, 1\}$. We use $\mathsf{signs}(A, \alpha)$ to denote the set of realizable sign assignments of A with respect to α.

Lemma 1. *If a set of polynomials $A \subset \mathbb{Z}[\boldsymbol{y}, x]$ is delineable over a region S, then $\mathsf{signs}(A, \alpha)$ is invariant over S.*

4.2 Projection-Based Explanations

Suppose that we need to produce an explanation for propagating a polynomial constraint F, i.e. we are in a state such that $\neg\, \mathsf{compatible}(\neg F, M)$, with $\mathsf{poly}(F) \in \mathbb{Z}[\boldsymbol{y}, x]$, where $\boldsymbol{y} = (y_1, \ldots, y_n)$. To simplify the presentation, in the following, we write υ for $\upsilon[M]$. The explanation procedure $\mathsf{explain}(F, M)$ consists of the following steps.

IsolateCore: Find a minimal set $\mathcal{F}$ of literals in M such that $\upsilon(\mathcal{F} \cup \{\neg F\})$ does not allow a solution for x. We call the set $\mathcal{F} \cup \{\neg F\}$ set a *conflicting core*.

Project: Construct a region S of $\mathbb{R}^n$ where $A = \mathsf{poly}(\mathcal{F} \cup \{F\})$ is delineable, and $\upsilon(\boldsymbol{y})$ is in S. Note that, from Lemma 1, $\neg F$ is incompatible with $\mathcal{F}$ for any other $\boldsymbol{\alpha}'$ in S.

Explain: Define the region S using extended polynomial constraints, obtaining a set of constraints $\mathcal{E}$. Then, we define $\mathsf{explain}(F, M) \equiv (\mathcal{E} \wedge \mathcal{F}) \implies F$.

We focus here on the second step of the procedure. To obtain the region S we will use a projection operator which, with insights of Theorem 2, will ensure delineability. Since our procedure requires a region S that contains the current assignment $\upsilon(\boldsymbol{y}) = \boldsymbol{\alpha}$, we add the assignment υ as an additional argument to the projection operator, and call such a projection operator *model-based*. Given a variable assignment υ, we denote the vanishing signature of a principal subresultant sequence as $\mathsf{v\text{-}psc}(f, g, x, \upsilon) = \mathsf{v\text{-}sig}(\mathsf{psc}_0(f, g, x), \ldots, \mathsf{psc}_n(f, g, x), \upsilon)$, and define our model-based projection operator $\mathsf{P}_m(A, x, \upsilon)$ as follows.

Definition 3. *Given a set of polynomials $A = \{f_1, \ldots, f_m\} \subset \mathbb{Z}[\boldsymbol{y}, x]$ and a variable assignment υ, the model-based Collins projector operator $\mathsf{P}_m(A, x, \upsilon)$ is defined as*

$$\bigcup_{f \in A} \mathsf{v\text{-}coeff}(f, x, \upsilon) \cup \bigcup_{\substack{f \in A \\ g = \mathsf{R}(f, x, \upsilon)}} \mathsf{v\text{-}psc}(g, g'_x, x, \upsilon) \cup \bigcup_{\substack{i<j \\ g_i = \mathsf{R}(f_i, x, \upsilon) \\ g_j = \mathsf{R}(f_j, x, \upsilon)}} \mathsf{v\text{-}psc}(g_i, g_j, x, \upsilon) \ .$$

Example 3. Consider the variable assignment υ, with $\upsilon(x) = 0$, and the set A containing two polynomials $f_2 = x^2 + y^2 - 1$ and $f_3 = -4xy - 4x + y - 1$. The projection operator P_m maps the set A into $\mathsf{P}_m(A, y, \upsilon)$

$$\{\ (\underbrace{16x^3 - 8x^2 + x + 16}_{f_1})x,\ -4x + 1,\ 4(x+1)(x-1),\ 2,\ 1\ \}\ , \tag{1}$$

where f_1 is the polynomial from Ex. 1. The zeros of f_2 and f_3 are depicted in Fig. 4(b), together with a set of important points $\{-1, \alpha_1, 0, \frac{1}{4}, 1\}$, where α_1 is the algebraic number from Ex. 1. These points are exactly the roots of the projection polynomials (1). It is easy to see that f_2 and f_3 are delineable in the intervals defined by these points. But, considering a polynomial $f_4 = x^3 + 2x^2 + 3y^2 - 5$, we can see that it is not delineable on the interval $(1, +\infty)$.

We will use the projection operator P_m to compute the required region S, and show that A is delineable in S. First, we close the set of polynomials $A \subset \mathbb{Z}[y_1, \ldots, y_n, x]$ under the application of a projection operator P_m. We compute this closure by computing sets of polynomials $\mathcal{P}^n, \ldots, \mathcal{P}^1$ iteratively, starting from $\mathcal{P}^n = \mathsf{P}_m(A, \upsilon, x)$, and then for $k = n, \ldots, 2$, compute the subsequent ones as $\mathcal{P}^{k-1} = \mathsf{P}_m(\mathcal{P}^k, y_k, \upsilon) \cup (\mathcal{P}^k \cap \mathbb{Z}[y_1, \ldots, y_{k-1}])$. Each set of polynomials $\mathcal{P}^k \subseteq \mathbb{Z}[y_1, \ldots, y_k]$ is obtained by projecting the previous set $\mathcal{P}^{k+1}$ and adding all the polynomials from $\mathcal{P}^{k+1}$ that do not involve the variable y_{k+1}.

Now, we can build the region S inductively, in a bottom-up fashion, by constructing a sequence of regions $S^k \subset \mathbb{R}^k$ such that each $\mathcal{P}^k$ is sign invariant in

S^k, and $\mathcal{P}^{k+1}$ is delineable in S^k. Assume that S^{k-1}, and its defining constraints $\mathcal{E}^{k-1}$, have already been constructed. Now, consider the set of root objects

$$R^k = \{\ \mathsf{root}(f,i)\ \mid\ f \in \mathcal{P}^k,\ 1 \leq i \leq \mathsf{rootcount}(v(f))\ \}\ .$$

Under the assignment v each of the root objects $\mathsf{root}(f,i)$ is defined and evaluates to some value $\omega_f^i \in \mathbb{R}_{\mathsf{alg}}$. The values ω_f^i partition the real line into maximal intervals where the polynomials $f \in \mathcal{P}^k$ are sign invariant. We pick the one interval that contains $v(y_k) = \alpha_k$ and construct the defining constraints $\mathcal{E}^k$ of the region S^k by selecting one of the appropriate cases

$$\begin{aligned}
&\alpha_k \in (\omega_f^i, \omega_g^j) && \implies \mathcal{E}^k = \mathcal{E}^{k-1} \cup \{\ y_k >_r \mathsf{root}(f,i), y_k <_r \mathsf{root}(g,j)\ \}\ ,\\
&\alpha_k \in (-\infty, \omega_f^i) && \implies \mathcal{E}^k = \mathcal{E}^{k-1} \cup \{\ y_k <_r \mathsf{root}(f,i)\ \}\ ,\\
&\alpha_k \in (\omega_f^i, +\infty) && \implies \mathcal{E}^k = \mathcal{E}^{k-1} \cup \{\ y_k >_r \mathsf{root}(f,i)\ \}\ ,\\
&\alpha_k = \omega_f^i && \implies \mathcal{E}^k = \mathcal{E}^{k-1} \cup \{\ y_k =_r \mathsf{root}(f,i)\ \}\ .
\end{aligned}$$

Finally, we guarantee that $\mathcal{P}^{k+1}$ is delineable in S^k because polynomials in $\mathcal{P}^* = \mathcal{P}^1 \cup \ldots \cup \mathcal{P}^k$ are by construction sign invariant in S^k. Once we have computed the regions $S^1, \ldots, S^n$, we can use the region $S = S^n$ and the corresponding constraints $\mathcal{E} = \mathcal{E}^n$ to explain why $\neg F$ is incompatible with $\mathcal{F}$. Thus, we set $\mathsf{explain}(F, M) \equiv (\mathcal{E} \wedge \mathcal{F}) \implies F$.

Theorem 3. *The explanation function* $\mathsf{explain}(F, M)$ *is a finite-basis explanation function for the existential theory of real closed fields.*

Example 4. Consider the polynomial $f = x^2 + y^2 + z^2 - 1$, from Ex. 2, and the constraint $f < 0$ corresponding to the interior of the sphere in Fig. 4(a). Under an assignment v with $v(x) = \frac{3}{4}$ and $v(y) = -\frac{3}{4}$ this constraint does not allow a solution for z (it evaluates to $z^2 < -\frac{1}{8}$). In order to explain it, we can compute the projection closure of $A = \{f\}$, using P_m, obtaining $\mathcal{P}_3 = A$ and

$$\mathcal{P}_2 = \{\ 4x^2 + 4y^2 - 4,\ 2,\ 1\ \}\ , \qquad \mathcal{P}_1 = \{\ 256x^2 - 256,\ 8,\ 4,\ 2,\ 1\ \}\ .$$

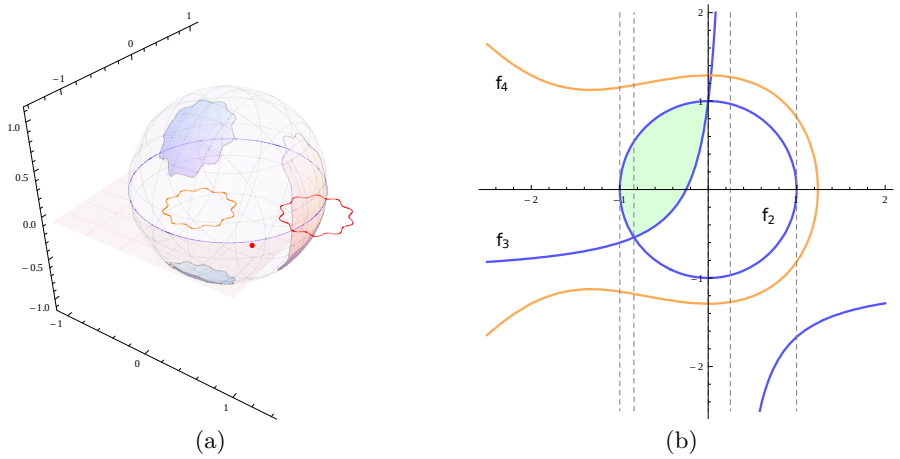
The sets of root objects under v are then

$$\begin{aligned}
R^2 &= \{\ \mathsf{root}(\tilde{z}^2 + x^2 - 1, 1),\ \mathsf{root}(\tilde{z}^2 + x^2 - 1, 2)\ \}\ ,\\
R^1 &= \{\ \mathsf{root}(\tilde{z}^2 - 1, 1),\ \mathsf{root}(\tilde{z}^2 - 1, 2)\ \}\ .
\end{aligned}$$

Since $v(x) = \frac{3}{4} = 0.75$ and the root objects of R_1 evaluate to -1 and 1, respectively, the constraints corresponding to the region S^1 are $(x > -1)$ and $(x < 1)$. The root objects of R_2 evaluate to $-\frac{\sqrt{7}}{4} \approx -0.6614$ and $\frac{\sqrt{7}}{4} \approx 0.6614$. Since $v(y) = -\frac{3}{4} = -0.75$, and we describe the region S^2 with the additional constraint $(y < \mathsf{root}(\tilde{z}^2 - x^2 - 1, 1))$. Using the constraints defining the region S^2 we construct the explanation $\mathsf{explain}(f < 0, v)$ as

$$(x \leq -1) \vee (x \geq 1) \vee \neg(y < \mathsf{root}(\tilde{z}^2 - x^2 - 1, 1)) \vee (f \geq 0)\ .$$

The explanation clause states that, in order to fix the conflict under v, we must change v so as to exit the region $-1 < x < 1$ below (in y) the unit circle.



(a) (b)

Fig. 4. (a) The sphere corresponding to the roots of $x^2 + y^2 + z^2 - 1$, and regions of Ex 2 and Ex 4. (b) Solutions of $f_2 = x^2 + y^2 - 1 = 0$, $f_3 = -4xy - 4x + y - 1 = 0$, and $f_4 = x^3 + 2x^2 + 3y^2 - 5 = 0$, with the solution set of $\{f_2 < 0, f_3 > 0, f_4 < 0\}$ emphasized. The dashed lines represent the zeros of the projection set (1).

.

Isolating the conflicting core. Given a constraint F incompatible with a trail M, we can compute a minimal set of constraints $\mathcal{F}$ from M that is not compatible with F by taking the constraints that that caused the inconsistency and then refine it by trying to eliminate the constraints one by one.

Example 5. Consider the set of polynomial constraints $\mathcal{C} = \{f_2 < 0, f_3 > 0, f_4 < 0\}$, where the polynomials f_2 and f_3 are from Ex. 3. The roots of these polynomials and the feasible region of $\mathcal{C}$ are depicted in Fig. 4(b). Assume the transition is in the state $\langle [\![x \mapsto 0, (f_2 < 0), (f_4 < 0), E \rightarrow (f_3 \leq 0)]\!], \mathcal{C} \rangle_2$, and we need to compute the explanation E of the last propagation. Although the propagation was based on the inconsistency of $\mathcal{C}$ under M, we can pick the subset $\{f_2 < 0, f_3 > 0\}$ to produce the explanation. It is a smaller set, but sufficient, as it is also inconsistent with M. Doing so we reduce the number of polynomials we need to project, which, in CAD settings, is always an improvement.

5 Related Work and Experimental Results

In addition to CAD, a number of other procedures have been developed and implemented in working tools since the 1980s, including Weispfenning's method of virtual term substitution (VTS) [28] (as implemented in Reduce/Redlog), and the Harrison-McLaughlin proof producing version of the Cohen-Hörmander method [19]. Abstract Partial Cylindrical Algebraic Decomposition [22] combines fast, sound but incomplete procedures with CAD. Tiwari [27] presents

an approach using Gröbner bases and sign conditions to produce unsatisfiability witnesses for nonlinear constraints. Platzer, Quesel and Rümmer combine Gröbner bases with semidefinite programming [23] for the real Nullstellensatz.

In order to evaluate the new decision procedure we have implemented a new solver nlsat, the implementation being a clean translation of the decision procedure described in this paper. We compare the new solver to the following solvers that have been reported to perform reasonably well on fragments of nonlinear arithmetic: the z3 3.2 [11], cvc3 2.4.1 [2], and MiniSmt 0.3 [29] SMT solvers; the quantifier elimination based solvers Mathematica 8.0 [25], QEPCAD 1.65 [5], Redlog-CAD and Redlog-VTS [12]; and the interval based iSAT [13] solver.[7]

We ran all the solvers on several sets of benchmarks, where each benchmark set has particular characteristics that can be problematic for a non-linear solver. The meti-tarski benchmarks are proof obligations extracted from the Meti-Tarski project [1], where the constraints are of high degree and the polynomials represent approximations of the elementary real functions being analyzed. The keymaera benchmark set contains verification conditions from the Keymaera verification platform [23]. The zankl set of problems are the benchmarks from the QF_NRA category of the SMT-LIB library, with most problems originating from attempts to prove termination of term-rewrite systems [14]. We also have two crafted sets of benchmarks, the hong benchmarks, which are a parametrized generalization of the problem from [15], and the kissing problems that describe some classic kissing number problems, both sets containing instances of increasing dimensions.

Table 1. Experimental results.

	meti-tarski (1006)		keymaera (421)		zankl (166)		hong (20)		kissing (45)		all (1658)	
solver	solved	time (s)	solved	time (s)	solved	time (s)	solved	time (s)	solved	time (s)	solved	time (s)
nlsat	1002	343	**420**	**5**	**89**	**234**	10	170	13	95	**1534**	**849**
Mathematica	**1006**	**796**	420	171	50	366	9	208	6	29	1491	1572
QEPCAD	991	2616	368	1331	21	38	6	43	4	5	1390	4036
Redlog-VTS	847	28640	419	78	42	490	6	3	10	275	1324	29488
Redlog-CAD	848	21706	363	730	21	173	6	2	4	0	1242	22613
z3	266	83	379	1216	21	0	1	0	0	0	667	1299
iSAT	203	122	291	16	21	24	**20**	**822**	0	0	535	986
cvc3	150	13	361	5	12	3	0	0	0	0	523	22
MiniSmt	40	697	35	0	46	1370	0	0	**18**	**44**	139	2112

All tests were conducted on an Intel Pentium E2220 2.4 GHz processor, with individual runs limited to 2GB of memory and 900 seconds. The results of our experimental evaluation are presented in Table 1. The rows are associated with the individual solvers, and columns separate the problem sets. For each problem

[7] We ran the solvers with default settings, using the `Resolve` command of Mathematica, the `rlcad` command for Redlog-CAD, and the `rlqe` for Redlog-VTS.

set we write the number of problems that the solver managed to solve within the time limit, and the cumulative time (rounded) for the solved problems.

The results are both revealing and encouraging. On this set of benchmarks, except for nlsat and the quantifier elimination based solvers, all other solvers that we've tried have a niche problem set where they perform well (or reasonably well), whereas on others they perform poorly. The new nlsat solver, on the other hand, is consistently one of the best solvers for each problem set, with impressive running times, and, overall manages to solve the most problems, in much faster time.

6 Conclusion

We proposed a new procedure for solving systems of non-linear polynomial constraints. The new procedure performs a backtracking search for a model, where the backtracking is powered by a novel conflict resolution procedure. In our experiments, our first prototype was consistently one of the best solvers for each problem set we tried, and, overall manages to solve the most problems, in much faster time. We expect even better results after several missing optimizations in the core algorithms are implemented. We see many possible improvements and extensions to our procedure. We plan to design and experiment with different explain procedures. One possible idea is to try explain procedures that are more efficient, but do not guarantee termination. Heuristics for reordering variables and selecting a value from the feasible set should also be tried. Integrating our solver with a Simplex-based procedure is another promising possibility.

Acknowledgements. We would like to thank Grant Passmore for providing valuable feedback, the Meti-Tarski benchmark set, and so many interesting technical discussions. We also would like to thank Clark Barrett for all his support.

References

1. B. Akbarpour and L. C. Paulson. MetiTarski: An automatic theorem prover for real-valued special functions. *Journal of Automated Reasoning*, 44(3):175–205, 2010.
2. C. Barrett and C. Tinelli. CVC3. In *CAV 2007*, pages 298–302. Springer, 2007.
3. S. Basu, R. Pollack, and M.-F. Roy. *Algorithms in real algebraic geometry*. Springer, 2006.
4. C. W. Brown. *Solution formula construction for truth invariant CAD's*. PhD thesis, University of Delaware, 1999.
5. C. W. Brown. QEPCAD B: a program for computing with semi-algebraic sets using CADs. *ACM SIGSAM Bulletin*, 37(4):97–108, 2003.
6. W. S. Brown and J. F. Traub. On Euclid's algorithm and the theory of subresultants. *Journal of the ACM*, 18(4):505–514, 1971.
7. B. Buchberger, G. E. Collins, R. Loos, and R. Albrecht, editors. *Computer algebra*. Symbolic and algebraic computation. Springer, 1982.

8. B. F. Caviness and J. R. Johnson, editors. *Quantifier Elimination and Cylindrical Algebraic Decomposition*. Texts and Monographs in Symbolic Computation. Springer, 2004.
9. H. Cohen. *A Course in Computational Algebraic Number Theory*. Springer, 1993.
10. G. E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Automata Theory and Formal Languages*, pages 134–183. Springer, 1975.
11. L. de Moura and N. Bjørner. Z3: An efficient SMT solver. *TACAS 2008*, pages 337–340, 2008.
12. A. Dolzmann and T. Sturm. Redlog: Computer algebra meets computer logic. *ACM SIGSAM Bulletin*, 31(2):2–9, 1997.
13. M. Fränzle, C. Herde, T. Teige, S. Ratschan, and T. Schubert. 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.
14. C. Fuhs, J. Giesl, A. Middeldorp, P. Schneider-Kamp, R. Thiemann, and H. Zankl. Sat solving for termination analysis with polynomial interpretations. *Theory and Applications of Satisfiability Testing*, pages 340–354, 2007.
15. H. Hong. Comparison of several decision algorithms for the existential theory of the reals. 1991.
16. D. Jovanović and L. de Moura. Cutting to the chase: Solving linear integer arithmetic. In *CADE 2011*, pages 338–353. Springer, 2011.
17. K. Korovin, N. Tsiskaridze, and A. Voronkov. Conflict resolution. *Principles and Practice of Constraint Programming*, pages 509–523, 2009.
18. R. Loos. Generalized polynomial remainder sequences. *Computer Algebra: Symbolic and Algebraic Computation*, pages 115–137, 1982.
19. S. McLaughlin and J. R. Harrison. A proof-producing decision procedure for real arithmetic. In *CADE 2005*, page 295. Springer, 2005.
20. K. L. McMillan, A. Kuehlmann, and M. Sagiv. Generalizing DPLL to richer logics. In *CAV 2009*, pages 462–476. Springer, 2009.
21. R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT modulo theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). *Journal of the ACM*, 53(6):937–977, 2006.
22. G. O. Passmore. *Combined Decision Procedures for Nonlinear Arithmetics, Real and Complex*. PhD thesis, University of Edinburgh, 2011.
23. A. Platzer, J.-D. Quesel, and P. Rümmer. Real world verification. In *CADE 2009*, pages 485–501. Springer, 2009.
24. J. P. M. Silva and K. A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999.
25. A. W. Strzeboński. Cylindrical algebraic decomposition using validated numerics. *Journal of Symbolic Computation*, 41(9):1021–1038, 2006.
26. A. Tarski. A decision method for elementary algebra and geometry. Technical Report R-109, Rand Corporation, 1951.
27. A. Tiwari. An algebraic approach for the unsatisfiability of nonlinear constraints. In *Computer Science Logic*, pages 248–262. Springer, 2005.
28. V. Weispfenning. Quantifier elimination for real algebra - the quadratic case and beyond. *AAECC*, 8:85–101, 1993.
29. H. Zankl and A. Middeldorp. Satisfiability of non-linear (ir)rational arithmetic. In *LPAR 2010*, pages 481–500. Springer, 2010.