A Symbiosis of Interval Constraint Propagation and Cylindrical Algebraic Decomposition*

Ulrich Loup[1], Karsten Scheibler[2], Florian Corzilius[1], Erika Ábrahám[1], and Bernd Becker[2]**

[1] RWTH Aachen University, Germany
[2] University of Freiburg, Germany

Abstract. We present a novel decision procedure for non-linear real arithmetic: a combination of iSAT, an incomplete SMT solver based on interval constraint propagation (ICP), and an implementation of the complete cylindrical algebraic decomposition (CAD) method in the library GiNaCRA. While iSAT is efficient in finding unsatisfiability, on satisfiable instances it often terminates with an interval box whose satisfiability status is unknown to iSAT. The CAD method, in turn, always terminates with a satisfiability result. However, it has to traverse a double-exponentially large search space.
A symbiosis of iSAT and CAD combines the advantages of both methods resulting in a fast and complete solver. In particular, the interval box determined by iSAT provides precious extra information to guide the CAD-method search routine: We use the interval box to *prune the CAD search space* in both phases, the projection and the construction phase, forming a search "tube" rather than a search tree. This proves to be particularly beneficial for a CAD implementation designed to search a satisfying assignment pointedly, as opposed to search and exclude conflicting regions.

1 Introduction

The formal modeling of systems and their properties along with corresponding analysis and synthesis methods require the usage of appropriate logics. In addition, many algorithms from this area need decision procedures for *satisfiability checking*, i.e., algorithms to decide whether there exists an assignment of values to the variables occurring in a formula such that the formula evaluates to true. A typical example is bounded model checking [4], a technique to encode counterexamples of a certain length by formulas; a solution to such a formula provides a counterexample, which can be used for the correction of the erroneous system.

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

** This work has been partially supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "AVACS" (SFB/TR 14, `http://www.avacs.org/` and the Research Training Group "AlgoSyn" (GRK 1298, `http://www.algosyn.rwth-aachen.de/`)

In this context, propositional logic with *SAT solving* as a decision procedure is widely used for discrete systems. For more complex systems more expressive logics, e.g., fragments of first-order logic over some theories, are necessary. *Satisfiability-modulo-theories* (*SMT*) *solving* turned out to be a very successful technique, which combines SAT solving with theory decision procedures: the Boolean structure of a formula is handled by a SAT solver, whereas the consistency of sets of theory atoms is checked by a *theory solver*. In the last decade a lot of effort has been put into the development of efficient SMT solvers for, e.g., equality logic with uninterpreted functions and linear real arithmetic.

Recently, increasing interest is devoted to solvers for *quantifier-free non-linear real arithmetic* (*QFNRA*). However, available SMT solvers for this expressive, highly challenging logic are rare. One of the reasons is the complexity of the available methods for checking sets of polynomial constraints over the reals for consistency. This circumstance complicates an embedding into an *efficient* SMT-solving framework. `Z3` [12] and `iSAT` [9] are among the most prominent SMT solvers for QFNRA. `Z3` uses an elegant adaption of the *cylindrical algebraic decomposition* (*CAD*) method and is complete for QFNRA. `iSAT` is based on the efficient technique of *interval constraint propagation* (*ICP*), yielding a fast but, in its current version, incomplete tool.

In this paper we introduce an extension of `iSAT` with an adapted variant of the CAD method, turning `iSAT` into a practically efficient and complete SMT solver for QFNRA. Because of the high complexity of solving polynomial constraints, we implement a *full-lazy* interaction between ICP and CAD. The `iSAT` algorithm recursively splits the initially bounded search space into smaller boxes and applies ICP to reduce the box sizes by cutting down provably unsatisfying parts. Under certain conditions, `iSAT` can detect that all points in a box are solutions, or else, that no solutions are in a box. In all other cases, `iSAT` continues to split the respective box. To assure termination, `iSAT` stops this splitting process when the box size reaches a lower threshold. This is the point when we invoke our CAD solver: to decide whether the remaining box contains a satisfying solution.

In turn, we use such a box to restrict the CAD search space. This could easily be formulated by extending the original constraints with constraints representing the bounds to the variables as given by the box. However, this would only complicate the CAD computation. Instead, we adapt the CAD method itself to be able to use the given bounds to prune the search. To this end, we implement approaches similar to the ones proposed in [11], but kept simple enough for efficiency maintenance. The CAD method consists of two phases, the *projection* and the *construction* phase. For the projection phase, we propose a novel pruning operator. The composition of our pruning operator and Hong's improved projection operator [10] generalizes Hong's projection operator as well as the model-based projection operator introduced in [12]. For the construction phase, we modify the procedure for the computation of CAD cells to consider only those cells which have a non-empty intersection with the given box. These modifications together allow us to take full advantage of the interval boxes of `iSAT` to reduce the size of the CAD remarkably. Our tool and the benchmarks we used are available at `http://ginacra.sourceforge.net/cade2013.html`.

Besides the related approaches [12,11] already mentioned, we are aware of [14,1,16], where approximation together with some form of validation is used to speed up the CAD computation.

2 Preliminaries

We use $\mathbb{Z}$ to denote the set of integers and $\mathbb{N}$ to denote the set of natural numbers including 0. We use the notation $]a, b[= \{c \in \mathbb{R} \mid a < c < b\}$ for open intervals, $[a, b] = \{c \in \mathbb{R} \mid a \leq c \leq b\}$ for closed intervals and define half-open intervals analogously. Furthermore, we permit unbounded open and half-open intervals by using ∞ or $-\infty$ as bounds. $\mathbb{I}_\mathbb{R}$ denotes the set of all intervals in $\mathbb{R}$. We call $I_1 \times \cdots \times I_n \in {\mathbb{I}_\mathbb{R}}^n$ an *interval box* composed of the intervals $I_j \in \mathbb{I}_\mathbb{R}$, $1 \leq j \leq n$. Given an interval $I \in \mathbb{I}_\mathbb{R}$, $\llcorner I$ denotes the *lower* or *left bound* and $I^\urcorner$ the *upper* or *right bound* of I.

2.1 Real arithmetic

We start with a formal definition of our input language, *quantifier-free non-linear real arithmetic* (*QFNRA*). QFNRA *formulas* φ are Boolean combinations of *constraints* c which compare *polynomials* p to 0. A polynomial p can be a *constant*, a *variable* x, or a sum, difference or product of polynomials:

$$\begin{array}{ccccccc} p & ::= & 0 & \mid\ 1 & \mid\ \ x & \mid (p+p) & \mid (p-p) \ \mid (p \cdot p) \\ c & ::= & p=0 & \mid p<0 & \mid\ p>0 & & \\ \varphi & ::= & c & \mid (\neg\varphi) & \mid (\varphi \wedge \varphi) & & \end{array}$$

Further operators such as disjunction $\vee$, implication $\rightarrow$ etc. and the weak relations $\leq$ and $\geq$ can be defined as syntactic sugar. We define $\cdot$ to bind stronger than $+,-$ and $\neg$ stronger than $\wedge$ stronger than $\vee$ and sometimes omit parentheses when causing no confusion. We use the standard semantics of QFNRA formulas.

Let $p = a_1 x_1^{e_{1,1}} \cdots x_n^{e_{n,1}} + \cdots + a_k x_1^{e_{1,k}} \cdots x_n^{e_{n,k}}$ be a polynomial with $a_j \in \mathbb{Z}$ and $e_{i,j} \in \mathbb{N}$ for $1 \leq i \leq n$ and $1 \leq j \leq k$. By $\deg(p) := \max_{1 \leq j \leq k}(\sum_{i=1}^n e_{i,j})$ we denote the *degree of* p. A formula φ is *linear* if $\deg(p) \leq 1$ for all polynomials p in φ, and *non-linear* otherwise. We denote the set of all polynomials with integer coefficients and variables $x_1, \ldots, x_n$ for some $n \geq 1$ by $\mathbb{Z}[x_1, \ldots, x_n]$. A polynomial $p \in \mathbb{Z}[x_1, \ldots, x_n]$ is called *univariate* if $n = 1$, and *multivariate* otherwise.

2.2 Satisfiability-modulo-theories (SMT) solving

In this paper, we tackle the following problem:

QFNRA Satisfiability Problem.	
Given:	QFNRA formula φ over the variables $x_1, \ldots, x_n$.
Question:	Is there an assignment $\alpha : \{x_1, \ldots, x_n\} \rightarrow \mathbb{R}$ satisfying φ?

To solve this problem we use the technique of *satisfiability-modulo-theories* (*SMT*) solving, whose basic scheme is depicted in Fig. 1.

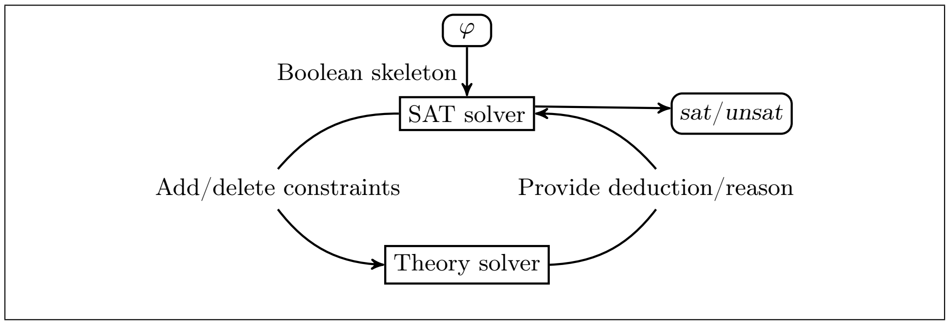

Fig. 1: Basic scheme of an SMT solver.

The *Boolean skeleton* of a QFNRA formula replaces each polynomial constraint in the formula by a fresh Boolean variable, resulting in a *propositional logic* formula. The *propositional satisfiability problem* poses the question whether there exists an assignment to the propositions in such a formula rendering the formula `true`.

Most modern *SAT solvers*, offering efficient solutions to this problem, are based on the DPLL procedure [7]. The input formula is required to be in *conjunctive normal form (CNF)*, i.e., it is a conjunction of *clauses*, whereas each clause is a disjunction of *literals*, the latter being variables or their negations. The Tseitin transformation [17] can be used for an equisatisfiable transformation of propositional logic formulas into CNF with linear complexity (in the number of operators) on the cost of adding linearly many new variables.

Given an input propositional logic formula in CNF, DPLL-style SAT-solvers assign values to variables following some heuristics and apply *Boolean constraint propagation (BCP)* to detect implications of the assignments. Such implications stem from *unit* clauses with all literals but one being false, implying that the last literal must be true in order to satisfy the formula. When the propagation leads to a conflict, i.e., when all literals of a clause are false, the solver uses *conflict resolution* to derive a reason for the conflict. *Conflict-driven clause learning (CDCL)* [15] can be used to exclude this and similar conflicts from future search.

SMT solving generalizes SAT solving by allowing literals of the input formula to be atoms from some theories, in our case QFNRA constraints, or their negations. A SAT solver works on the Boolean skeleton of the underlying problem and assigns `true` or `false` to the theory atoms. The SAT solver is complemented by a *theory solver* offering a decision procedure for the conjunction of constraints from the underlying theory.

There are different approaches how to combine a theory solver and a SAT solver in an SMT application. In the *full lazy* approach, the SAT solver first searches for a complete satisfying assignment for the Boolean skeleton. If the skeleton is unsatisfiable then the input formula is also unsatisfiable. If a satisfying assignment is found, the SAT solver invokes the theory solver to check whether the conjunction of those theory atoms, which satisfy the clauses, is consistent. If this is the case then the input formula is satisfiable. Otherwise, the Boolean skeleton is refined with a conflict clause which forbids the current combination of the conflicting theory atoms for the future search. In the *less lazy* approach, the theory solver is invoked more frequently also for partial assignments. In some other solvers, the separation between SAT and theory solving is not so strict (see Section 3). More details on SMT solving can be found, e.g., in [5, Ch.26].

3 The SMT solver `iSAT`

In contrast to most other SMT solvers, in `iSAT` [9] the separation between the theory solver and the SAT solver is not so strict. Instead, `iSAT` tightly integrates *interval constraint propagation* (*ICP*) (see e.g. [3]) into the SAT framework. This deep integration has the advantage of sharing the common core of the search algorithms between the propositional and the theory-related part of the solver.

The `iSAT` solver allows Boolean and interval-bounded integer-valued and real-valued *variables*. Boolean assignments for propositions are extended with interval valuations $\rho : Var \rightarrow \mathbb{I}_{\mathbb{R}}$ assigning to each numerical variable from Var its current interval bound. Note that in `iSAT` all variables have initial intervals.

Beyond linear and non-linear integer and real arithmetic expressions, `iSAT` supports transcendental functions and a collection of further operators and functions. Examples for `iSAT` *theory atoms* are $x^2 + y^2 = z^2$, $|v - w| \leq \min(v, w)$ or $\sqrt[3]{x} + \sin y < e^z$. But in this paper we focus on non-linear problems only.

As a *preprocessing* step `iSAT` applies some basic arithmetic simplifications to the input formula (e.g., the expression $2x+3x$ is rewritten to $5x$). Furthermore, it tries to detect contradictions, tautologies and subsumptions (e.g., $x < 5 \wedge x < 7$ is simplified to $x < 5$, and $x < 5 \wedge x > 7$ is reduced to `false`). The preprocessed formula is then rewritten into CNF with the Tseitin transformation.

The `iSAT` algorithm operates on CNFs containing only *simple bounds* and *primitive constraints* as theory atoms. Simple bounds compare a variable to a constant. Primitive constraints are equalities containing exactly one unary or binary operator. Constraints of other forms are decomposed by a Tseitin-like satisfiability-equivalent transformation into simple bounds and primitive constraints. E.g., the theory atom $x + y^2 > 0$ is replaced by $h_1 > 0$ and the unit clauses $h_1 = x + h_2$ and $h_2 = y^2$ are added to the clause set, where h_1 and h_2 are type-consistent fresh auxiliary variables.

The three basic elements decision, propagation and conflict resolution of the DPLL framework are also present in `iSAT`, but they are extended for the operation on integer- and real-valued intervals in addition to the Boolean domain. Deciding an integer- or real-valued variable corresponds to splitting its interval into two intervals and selecting the lower or upper one.

In the propagation phase, ICP is executed in addition to BCP. ICP tries to narrow the interval bounds of numerical variables based on simple bounds and primitive constraints. However, if a new bound has a negligible progress compared to the existing bound then the new bound is ignored to prevent infinite propagation sequences and thus to guarantee the termination of the ICP process. Furthermore, primitive constraints are checked whether they are still consistent with the current interval valuations of the variables. A primitive constraint c is *consistent* under an interval valuation iff there exist values in the intervals of the involved variables which evaluate c to true. Inconsistency can be detected when ICP deduces an empty interval for one of the involved variables.

ICP is incomplete, i.e., `iSAT` may terminate with an inconclusive answer, because in general equations like $x = y \cdot z$ can only be satisfied by point intervals. But reaching such point intervals by ICP cannot be guaranteed for real-valued variables. In such cases `iSAT` will return an interval box with a possible solution called a *candidate solution.* In [8] an approach is presented to check whether there exists a satisfying assignment in a candidate solution. Beside the fact that the approach is heuristic and thus may not succeed in all cases it is also unable to detect the unsatisfiability of an interval box. Using the CAD overcomes these two problems.

4 Cylindrical algebraic decomposition (CAD)

In this work, we use the *cylindrical algebraic decomposition* (*CAD*) method, introduced by Collins [6], to check the satisfiability of a set of non-linear real arithmetic constraints comparing polynomials to 0. Although the CAD method originally was designed to perform quantifier elimination, its adaption to our setting of solving constraint systems is marginal because the basic scheme of the CAD method is still the same. The CAD method decomposes the state space into a finite number of connected regions called *cells* in which the involved polynomials are sign-invariant, i.e., in which the truth values of the constraints are invariant. The method also allows to get a sample point from each of the cells. The satisfiability check of the constraint set can thus be done by checking whether one of the finitely many sample points satisfies all the constraints. The finite union of the satisfying cells builds the solution set of the given constraints.

Fig. 2 shows an example of a solution set in $\mathbb{R}^2$ for three constraints. In general, such solution sets are unions of open or closed connected subsets of $\mathbb{R}^n$ whose boundaries are defined by the real zeros of the considered polynomials over n variables. However, a CAD of the solution space is usually an even finer decomposition of $\mathbb{R}^n$, arranged in cylinders in each dimension. We describe the precise structure of a CAD by an explanation of the actual CAD computation. We use the constraint set defined in Example 1 in order to illustrate the method.

Example 1. We consider the constraint set $\{x^2 + y^2 > 4,\ x + 2 = y,\ x^2 y > 1\}$. A picture of its solution set in $\mathbb{R}^2$ is given in Fig. 2.

Let $P_n \subseteq \mathbb{Z}[x_1, \ldots, x_n]$ be a set of multivariate polynomials over $n \geq 1$ variables. The CAD method decomposes the state space $\mathbb{R}^n$ into a finite number

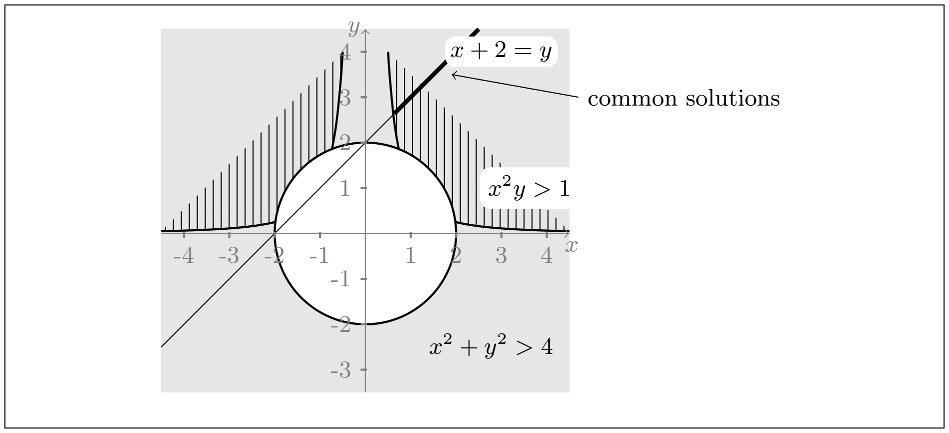

Fig. 2: Solution sets of the constraints $x^2+y^2>4$ (every point outside the circle around the center with radius 2), $x+2=y$ (angle bisecting line) and $x^2y>1$ (hatched area).

of disjoint subsets (cells) $\cup_{i=1}^{m} R_i$ such that in each cell R_i the polynomials in P_n are *sign invariant*, i.e., for all $p \in P_n$ either $p(a)>0$, $p(a)<0$ or $p(a)=0$ for all $a \in R_i$. Such a CAD can be used to decide consistency of a set of constraints comparing polynomials from P_n to 0 by substituting a sample point from each of the CAD cells into the constraints. The CAD is done in two phases, the *projection phase* and the *construction phase*, as illustrated in Fig. 3.

4.1 Projection phase

The left side of Fig. 3 depicts the projection phase. From the input set P_n, the CAD method computes another set of polynomials $\mathrm{proj}_{x_n}(P_n) \subseteq \mathbb{Z}[x_1,\ldots,x_{n-1}]$ so that the following holds: Let $R \subseteq \mathbb{R}^{n-1}$ be a connected set such that each $p \in \mathrm{proj}_{x_n}(P_n)$ is sign-invariant on R. Then there is a decomposition $R \times \mathbb{R} = \biguplus_{i=1}^{k} R \times S_i$ of the cylinder over R with $k \in \mathbb{N} \setminus \{0\}$ and $S_i \subseteq \mathbb{R}$ connected for $1 \le i \le k$ so that each $p \in P_n$ is sign-invariant on $R \times S_i$ for $1 \le i \le k$. We call the operation proj_{x_n} a (*CAD*) *projection operator*.

There are several implementations of projection operators in the literature. In this work, we use Hong's improved projection operator [10]. We illustrate such a CAD projection in Example 2: The given projection eliminates the variable x from the input set over the variables x and y. For a deeper insight into how the projection works, we refer to the before-mentioned articles on CAD [6,10].

Example 2 (CAD projection). We consider the polynomials corresponding to the constraint set of Example 1 $P=\{x^2+y^2-4,\ x+2-y,\ x^2y-1\} \subseteq \mathbb{Z}[x,y]$. Then $\mathrm{proj}_x(P)$ contains the 3 coefficients from the input polynomials $2-y$, y, y^2-4, but also 5 polynomials representing multiple or common roots, such as $-y^3+4y-1$, y^3-4y^2+4y-1, $y^6-8y^4+2y^3+16y^2-8y+1$.

$$\begin{tikzcd}
\to P_n \subseteq \mathbb{Z}[x_1, \ldots, x_n] \arrow[d, "\mathrm{proj}_{x_n}"'] & Z_n := \bigcup_{z \in Z_{n-1}} \{z\} \times \mathrm{samp}(\{p(z) \mid p \in P_n\}) \\
P_{n-1} := \mathrm{proj}_{x_n}(P_n) \arrow[d, "\mathrm{proj}_{x_{n-1}}"'] & \vdots \arrow[u, "\mathrm{samp}"'] \\
\vdots \arrow[d, "\mathrm{proj}_{x_2}"'] & Z_2 := \bigcup_{z \in Z_1} \{z\} \times \mathrm{samp}(\{p(z) \mid p \in P_2\}) \arrow[u, "\mathrm{samp}"'] \\
P_1 := \mathrm{proj}_{x_2}(P_2) \arrow[r, bend right, "\mathrm{samp}"'] & Z_1 := \mathrm{samp}(P_1) \arrow[u, "\mathrm{samp}"']
\end{tikzcd}$$

Fig. 3: The basic CAD method in a nutshell: the projection phase on the left, the construction phase on the right.

We call $P_i := \mathrm{proj}_{x_{i+1}}(P_{i+1}) \subseteq \mathbb{Z}[x_1, \ldots, x_i]$ the *ith successive projection* for $0 < i < n$. We start the construction phase with the set of univariate polynomials P_1.

4.2 Construction phase

In the construction phase, as sketched on the right side of Fig. 3, the CAD method starts with computing the set $Z_1 \subseteq \mathbb{R}$ of representatives for the cells of a CAD of the real line. Let $r_1 < \cdots < r_k$ be the real roots of the polynomials in P_1, then the maximal sets where each $p \in P_1$ is sign-invariant are the intervals $]-\infty, r_1[\,, [r_1, r_1], \,]r_1, r_2[\,, \ldots, [r_k, r_k], \,]r_k, \infty[$. This family of intervals is a *CAD of* $\mathbb{R}$. We call a representative of such an interval (*CAD*) *sample*, i.e., samples are real roots of univariate polynomials, the points between these roots, and one point less than the smallest and one greater than the largest root. The set of samples of a set of univariate polynomials U is denoted by $\mathrm{samp}(U)$.

The procedure to compute CAD samples is one of the most important parts of the CAD method. Hence, we shed some more light on algorithms for finding real roots of polynomials.

The heart of real root isolation is counting real roots of a univariate polynomial in a given interval. By p' we denote the formal derivative of a univariate polynomial p, and by $(p \mod q)$ the remainder of the division of the polynomials p by q. Given a univariate polynomial p, its *Sturm sequence* is $S = (p_0, \ldots, p_k)$ with $p_0 = p$, $p_1 = p'$, $p_i = -(p_{i-2} \mod p_{i-1})$ for $2 \leq i \leq k$, and p_k a rational constant. Let $S(t)$ for $t \in \mathbb{Q}$ denote the sequence of the non-zero $p_i(t)$ values, and let $\mathrm{Var}(S(t))$ be the number of sign changes in $S(t)$. Sturm's theorem (see e.g. [2, Theorem 2.50]) states that p has $\mathrm{Var}(S(a)) - \mathrm{Var}(S(b))$ many real roots in $\,]a,b[\,$ if a and b are no roots of p themselves. For example, the Sturm sequence of $p = x^2 - 2$ is $S = (x^2 - 2, 2x, 2)$. Therefore, p has $\mathrm{Var}(S(-2)) - \mathrm{Var}(S(2)) = 2 - 0 = 2$ real roots in $\,]-2,2[\,$.

In this example, a radius of 2 around 0 already suffices to capture all real roots of p. To obtain such an upper bound on the radius covering all roots of a univariate polynomial $p = \sum_{i=0}^{k} c_i x^i$, $c_k \neq 0$, we can use the *Cauchy bound* $\mathrm{C}(p) := \sum_{d=0}^{k} \left| \frac{c_d}{c_k} \right|$. In order to isolate all real roots of p by intervals, the interval $]-\mathrm{C}(p), \mathrm{C}(p)[\,$ can be successively split, e.g. by its midpoint, until intervals are found for which Sturm's theorem determines exactly one real root. An interval containing exactly one real root r of p is called an *isolating interval for* r, and $\langle p,\ ]a,b[\ \rangle$ an *interval representation for* r. Given two interval-represented real roots r_1 and r_2, we are able to effectively compute arithmetic operations such as $r_1 + r_2$, $-r_1$, $r_1 \cdot r_2$, r_1^{-1}, as well as relations such as $r_1 = r_2$ and $r_1 < r_2$ (see, e.g., [13, p. 327ff]).

Example 3 shows possible CAD samples including interval-represented roots.

Example 3 (CAD samples). Given $\mathrm{proj}_x(P)$ from Example 2, $\mathrm{samp}(\mathrm{proj}_x(P))$ contains 19 samples, e.g., the roots $r_1 := \langle -y^3 + 4y - 1,\ ]\frac{-17}{8}, \frac{-67}{32}[\ \rangle$, $r_2 := \langle -y^3 + 4y - 1,\ ]\frac{61}{160}, \frac{31}{80}[\ \rangle$, $r_3 := \langle -y^3 + 4y - 1,\ ]\frac{31}{80}, \frac{15}{8}[\ \rangle$ of $-y^3 + 4y - 1$ and corresponding intermediate points $-3 < r_1 < -1 < r_2 < \frac{1}{3} < r_3 < \frac{15}{8}$. The isolating intervals of the real roots in $\mathrm{samp}(\mathrm{proj}_x(P))$ are even tighter because of several refinements when computing the whole example; however, we omit these details for the sake of readability.

Hence we have $Z_1 = \mathrm{samp}(P_1)$ and continue with the computation of Z_{i+1} with $Z_{i+1} := \bigcup_{(a_1,\ldots,a_i) \in Z_i} \{(a_1,\ldots,a_i,a_{i+1}) \mid a_{i+1} \in \mathrm{samp}(\{p(a_1,\ldots,a_i,x_{i+1}) \mid p(x_1,\ldots,x_{i+1}) \in P_{i+1}\})\}$ for all $1 < i < n$. Note that $p(a_1,\ldots,a_i,x_{i+1})$ is indeed a univariate polynomial. We call a point $z \in Z_n \subseteq \mathbb{R}^n$ a *(CAD) sample point*.

Because $\mathrm{samp}(\{p(z) \mid p \in P_{i+1}\})$ contains all samples of a CAD of $\mathbb{R}$ for each $z \in Z_i$, we can conclude by induction on i, $1 \leq i < n$, that Z_n contains sample points for all sign-invariant connected components of $\mathbb{R}^n$. Note that the calculation of $p(z)$ for given $p \in P_{i+1}$, $z \in Z_i$, $1 \leq i < n$ can be performed by computing resultants [2, Sec. 4.2].

If $m = |P|$ and d be the maximum degree of all polynomials occurring in P and if we assume that $|\mathrm{proj}(P)| \leq m^2 d^2$ as in [10], then $|Z_n|$ is dominated by $\mathcal{O}(m^{2^n-1} d^{2^n+n-1})$. This worst case can occur even for linear input formulas, as showed in [18].

5 iSAT adaption

ICP works well for robust combinations of theory atoms, but may fail to give a conclusive answer for non-robust ones. Here, a set of theory atoms is meant to be robust, if the atoms maintain their truth value under small perturbations of the constants in the atoms. Nonetheless even a candidate solution contains useful information which can be used as a guiding path for a complete method like the CAD method.

Whereas iSAT operates on a CNF enriched with primitive constraints as unit clauses, the CAD method is fed with the non-decomposed theory atoms together with the current interval valuation of the variables occurring in those atoms in a full-lazy manner. The CAD method is used whenever iSAT is unable to give a conclusive answer by itself. As mentioned earlier the focus of this paper are non-linear problems. Note, that we could also allow the actual input logic of iSAT and would then simply avoid calling the CAD method when, e.g. transcendental functions, occur. In order to determine which theory atoms have to be passed to the CAD method, each simple bound originating from a decomposed theory atom keeps a link to its original non-decomposed form.

iSAT would stop its search with a candidate solution, because each primitive constraint is still consistent under the current interval valuation of the variables. However, it is unknown whether there exists a point interval for each variable satisfying all primitive constraints. In such a case, having the link back from the simple bounds to the original theory atoms makes it possible to determine the set of atoms to pass to the CAD method to get a definitive answer. The solver terminates with a satisfiable result when the CAD method finds a solution within the given interval box defined by the candidate solution. Otherwise a conflict clause excluding this set of atoms is created and the search continues. Note, that iSAT could call the CAD method even in case of unbounded problems making the extension of iSAT complete for QFNRA.

6 CAD adaption

In Section 4 we showed that the two phases of the CAD method, the projection and the construction phase, can be both extremely inefficient because of their doubly-exponential search space. However, because our CAD-method implementation is designed to traverse all sample points for a satisfying one, we can mainly avoid this worst case behavior when we assume that a possible solution lies within a given interval box.

At first, we formally fix the situation of a call to the CAD solver within the ICP solver. Let, throughout this section, $B = I_1 \times \cdots \times I_n \in \mathbb{I}_{\mathbb{R}}^n$ an interval box with $B \neq \emptyset$, $P = \{p_1, \ldots, p_m\}$ polynomials with $p_i \in \mathbb{Z}[x_1, \ldots, x_n]$ over $n \geq 1$ variables and $p_1 \sim_1 0, \ldots, p_m \sim_m 0$ be constraints with $\sim_i \in \{<, =, >\}$ for $1 \leq i \leq m$. We are interested in the satisfiability of

$$\bigwedge_{i=1}^{m} p_i(x_1, \ldots, x_n) \sim_i 0 \quad \wedge \quad \bigwedge_{i=1}^{n} x_i \in I_i. \tag{1}$$

The constraints representing the bounds can be formulated using the polynomials $P_B := \{x_i - b_i \mid b_i \in \{\llcorner I_i, I_i\urcorner\} \cap \mathbb{R}, 1 \leq i \leq n\}$. Note that the infinity bounds are excluded in this representation.

A straightforward approach to solve (1) is to construct a CAD of R^n w.r.t. $P \cup P_B$ and search a satisfying sample point. We refer to this solution approach as *lazy method* because the bounds are checked at the very end of the construction phase – lazily.

This section, however, is devoted to an approach we call *eager method*: the bounds are incorporated deeply into the CAD phases in order to restrict the search space w.r.t. B, as opposed to leaving it untouched and checking membership in B at the end. The adaption of the CAD method for the eager approach is two-fold: a reduction of the sets of polynomials computed in the projection phase and an early pruning of the CAD samples in the construction phase.

Projection phase. Based on the interval box B, we define a pruning operator prun_B so that $|\mathrm{prun}_B(P)| \leq |P|$.

Definition 1 (Interval-based polynomial pruning). *Let $Q \subseteq \mathbb{Z}[x_1, \ldots, x_n]$. We call the operation* $\mathrm{prun}_B(Q) := \{p \in Q \mid p(r) = 0$ *for some* $r \in B\}$ interval-based polynomial pruning*.*

The set $\mathrm{prun}_B(P)$ consists of those polynomials from P which have a real root in the given interval box. Thus, $\mathrm{prun}_B(P) \subseteq P$.

Note that the composition $\mathrm{prun}_B \circ \mathrm{proj}_{x_i}$ is an implementation of the projection operator introduced in [11], generalizing the *model-based* projection operator described in [12] where the authors prune the output of the operator using numbers, possibly interval-represented, instead of arbitrary intervals.

The polynomial pruning $\mathrm{prun}_B(Q)$ can be straightforwardly implemented by testing whether $p \in Q$ has a root inside B by constructing a CAD. Because this approach can be very inefficient, we first compute an over-approximation of $\{p(a) \mid a \in B\}$ by interval arithmetic and check whether $(0, \ldots, 0)$ is contained. If not, we can safely prune p. Otherwise we compute a CAD w.r.t. p and prune p if no satisfying sample for $p = 0$ is found. Else, we have to keep p for the computation of the CAD of P and, as an optimization, we can reuse all projection polynomials computed for p.

We demonstrate the "tube" effect of the interval-based polynomial pruning in the projection phase by two examples depicted in Fig. 4 and Example 4.

Example 4 (Interval-based polynomial pruning). Let P be as in Ex. 2. We consider the two interval boxes B_1, $B_2 \in \mathbb{I}_{\mathbb{R}}{}^2$ with $B_1 = [-3, 0]^2$ (cf. left picture in Fig. 4), $B_2 = \,]1, 4[\, \times \,]2, 4[$ (cf. right picture in Fig. 4) and P_{B_1}, P_{B_2} their polynomial representations each having 4 additional input polynomials.

The projection $\mathrm{proj}_x(P \cup P_{B_1})$ has 14 elements. Fortunately, $\mathrm{prun}_{B_1}(P \cup P_{B_1})$ already removes $x^2y - 1$ resulting in a smaller projection $\mathrm{prun}_{B_1}(\mathrm{proj}_x(P \cup P_{B_1}))$ of 9 elements. The projection $\mathrm{proj}_x(P \cup P_{B_2})$ contains 18 elements and $\mathrm{prun}_{B_2}(P \cup P_{B_2})$ leaves just $x + 2 - y$ so that $\mathrm{prun}_{B_2}(\mathrm{proj}_x(P \cup P_{B_2}))$ consists only of 8 linear elements.

The example shows that B restricts the set of projections in such a way that the problem can become significantly smaller, e.g., for B_2 the initial problem is reduced to a linear polynomial only.

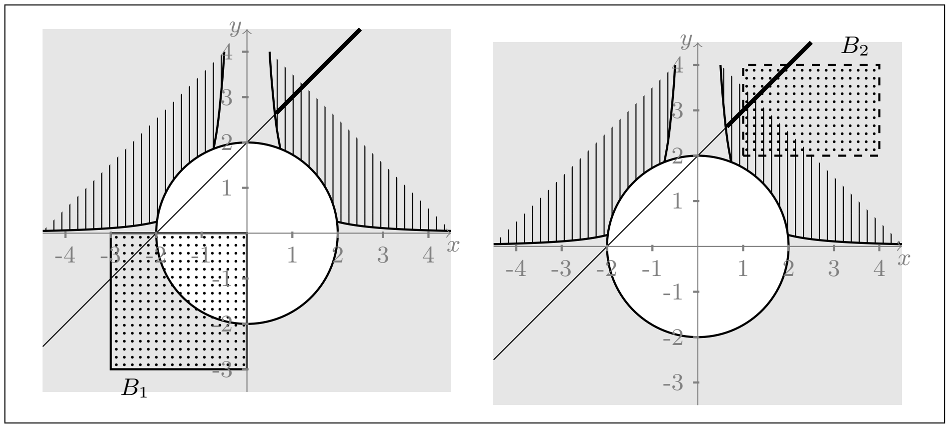

Fig. 4: Solution sets of the constraints $x^2 + y^2 > 4$, $x + 2 = y$ and $x^2 y > 1$ intersected with the boxes $B_1 = [-3, 0]^2$ and $B_2 =]1, 4[\times]2, 4[$.

Construction phase. We can use the interval box B to restrict also the sample construction for each dimension.

Definition 2 (Interval-based samples). *Let $U \subseteq \mathbb{Z}[x_i]$ be a set of univariate polynomials. We define the set* $\mathrm{samp}_B(U)$ *by*

$$\mathrm{samp}_B(U) := \mathrm{samp}(U \cup P_{I_i}) \cap [\llcorner I_i, I_i \urcorner].$$

$\mathrm{samp}_B(U)$ *is called a set of* interval-based *(*CAD*)* samples.

Let $U \subseteq \mathbb{Z}[x_i]$. The important aspects of $\mathrm{samp}_B(U)$ are, first, the inclusion of the bounds as real roots in order to ensure that points between possible real roots of U and the bounds of I_i are constructed, and second, the intersection with the closed version of I_i to prune all samples outside the bounds of I_i but not the bounds themselves in order to guarantee that the bounds are available for the sample construction in higher dimensions.

In our implementation of $\mathrm{samp}_B(U)$, we use I_i as initial interval for the real root isolation due to Sturm's theorem rather than the Cauchy bounds so that only real roots inside I_i are found. We then apply the usual CAD sample construction and remove the outer samples at the left and right bounds.

Joining of the two adapted phases. The new operators from the Definitions 1 and 2 enable us to define the eager method. We depict its scheme in Fig. 5 referring to the scheme of the native CAD method in Fig. 3.

In the adapted projection phase, we use the adapted ith successive projection $P_i := \mathrm{prun}_B(\mathrm{proj}_{x_{i+1}}(P_{i+1} \cup P_{I_{i+1}}))$ for $0 < i < n$. In particular, we project the polynomials representing the bounds only when the corresponding variable is projected and do not keep the bound-representing polynomials themselves in the projections. Moreover, the interval-based polynomial pruning is applied after each projection and to the input set P.

$$\begin{tikzcd}
{} \arrow[r] & P_n := \mathrm{prun}_B(P) \arrow[d, "\mathrm{prun}_B \circ \mathrm{proj}_{x_n}"'] & Z_n := \bigcup_{z \in Z_{n-1}} \{z\} \times \mathrm{samp}_B(\{p(z) \mid p \in P_n\}) \\
& P_{n-1} := \mathrm{prun}_B(\mathrm{proj}_{x_n}(P_n \cup P_{I_n})) \arrow[d, "\mathrm{prun}_B \circ \mathrm{proj}_{x_{n-1}}"'] & \vdots \arrow[u, "\mathrm{samp}_B"'] \\
& \vdots \arrow[d, "\mathrm{prun}_B \circ \mathrm{proj}_{x_2}"'] & Z_2 := \bigcup_{z \in Z_1} \{z\} \times \mathrm{samp}_B(\{p(z) \mid p \in P_2\}) \arrow[u, "\mathrm{samp}_B"'] \\
& P_1 := \mathrm{prun}_B(\mathrm{proj}_{x_2}(P_2 \cup P_{I_2})) \arrow[r, bend right, "\mathrm{samp}_B"'] & Z_1 := \mathrm{samp}_B(P_1) \arrow[u, "\mathrm{samp}_B"']
\end{tikzcd}$$

Fig. 5: Eager method to solve (1).

The adapted construction phase, in turn, is working analogously to the native construction phase as shown in Fig. 3, except that the interval-based sample operator samp_B is used.

Both phases work hand in hand. Firstly, prun_B could cut off also the polynomials in P_B which are important to construct samples between the bounds and possible real roots inside B. This shortcoming is tackled by samp_B, as it always produces the bounds of B as roots. Moreover, the projection of the bound-representing polynomials together with other polynomials have to be generated. We force this by adding the respective polynomials from P_B for the projections.

7 Experimental results

We have embedded three versions of the CAD procedure of Section 6 into `iSAT`: `iSAT+CAD`$_1$ implementing the lazy method, `iSAT+CAD`$_2$ the eager method without interval-based polynomial pruning and `iSAT+CAD`$_3$ the eager method itself. We compared our approaches with two well-known solvers also based on CAD: `Z3` and `Redlog` (calling `rlcad`). Table 1 shows the performances of `iSAT` and our CAD implementation embedded in a DPLL-style SMT solver, `SMT+CAD`.

The tests cover the benchmark sets `hong` (Hon), `meti-tarski` (Met), `zankl` (Zan) and `keymaera` (Key) as described in [12]. In addition, we tested the set `Laser` (Las) containing unrolled bounded model checking instances which describe whether a laser can reach a target state traversing a labyrinth of reflecting obstacles.

Table 1 Results obtained on a 2.1 GHz AMD with a timeout of 200 seconds.

bench (#)	Hon (20)		Met (8276)		Las (381)		Zan (166)		Key (421)		*all* (9264)	
	#	*sec*	#	*sec*	#	*sec*	#	*sec*	#	*sec*	#	*sec*
`iSAT+CAD`$_3$	20	0.1	7634	18840.0	331	758.6	31	195.4	331	216.4	8347	20010.5
- sat	0	0.0	4967	15678.1	0	0.0	12	183.8	0	0.0	4979	15861.7
- unsat	20	0.3	2667	3161.9	331	758.6	19	11.6	331	216.4	3368	4148.8
`iSAT+CAD`$_2$	20	0.3	7321	29143.5	331	710.2	31	181.4	329	196.2	8032	30231.6
- sat	0	0.0	4667	26301.6	0	0.0	12	170.9	0	0.0	4679	26472.5
- unsat	20	0.3	2654	2841.9	331	710.2	19	10.5	329	196.2	3353	3759.1
`iSAT+CAD`$_1$	20	0.3	6732	23092.9	331	755.0	31	182.4	328	307.8	7442	24338.4
- sat	0	0.0	4125	20673.8	0	0.0	12	170.4	0	0.0	4137	20844.2
- unsat	20	0.3	2607	2419.0	331	755.0	19	12.0	328	307.8	3305	3494.1
`iSAT`	20	0.2	2614	285.4	331	719.1	19	10.2	307	2.6	3291	1017.5
- sat	0	0.0	123	2.1	0	0.0	0	0.0	0	0.0	123	2.1
- unsat	20	0.2	2491	283.4	331	719.1	19	10.2	307	2.6	3168	1015.4
`SMT+CAD`	2	0.2	5358	29494.8	17	4.9	15	1.3	307	581.3	5699	30082.5
- sat	0	0.0	3746	14151.8	0	0.0	11	0.9	0	0.0	3761	14152.7
- unsat	2	0.2	1612	15343.0	17	4.9	4	0.4	307	581.3	1942	15929.8
`Z3`	11	353.5	8257	593.1	237	2340.7	91	173.5	418	10.0	9014	3470.7
- sat	0	0.0	5350	253.0	7	240.0	61	113.5	0	0.0	5418	606.5
- unsat	11	353.5	2907	340.1	230	2100.6	30	60.0	418	10.0	3596	2864.2
`Redlog`	3	2.7	7403	28472.2	17	13.6	17	205.9	243	721.9	7683	29416.2
- sat	0	0.0	4969	14023.8	0	0.0	12	42.3	2	143.2	4983	14209.4
- unsat	3	2.7	2434	14448.3	17	13.6	5	163.5	241	578.7	2700	15206.8

8 Outlook

From our results we can conclude that extra input information such as bounds for the variables can aid the CAD search routine and improve its efficiency. Following that line, we want to connect the two procedures even tighter. `iSAT` is an SMT solver calling the CAD solver for consistency checks, just as a theory solver in the SMT framework. Empirical data from the field of SMT solving shows that a tremendous speed-up can be gained if the respective theory solver supports incremental adding and removing of constraints plus the generation of minimal reasons in case of a conflict and theory deductions in case of no conflict.

Moreover, our results show that, in contrast to a conflict-oriented approach such as in `Z3` [12], a solution-oriented implementation of the CAD method as in `SMT+CAD` is widely inefficient on the SMT benchmarks. However, its adaption and combination with an incomplete ICP solver in a prototypical implementation is already almost at eye level with highly developed SMT solvers such as `Z3`. We expect that a bound-using adaption of a complete NRA solving strategy not only composed of the CAD method would be even more beneficial.

Our CAD implementation uses arbitrary-precision arithmetic for computing with interval bounds. In contrast, `iSAT` is mainly implemented based on fast floating-point arithmetic. It is possible to use validated floating-point computations in the CAD method, too. Thus, we could improve the efficiency of our CAD solver and reduce the overhead of the CAD solver calls within `iSAT`.

Acknowledgments. We appreciate the very interesting and detailed comments by the anonymous reviewers.

References

1. Anai, H., Yokoyama, K.: Cylindrical algebraic decomposition via numerical computation with validated symbolic reconstruction. In: Algorithmic Algebra and Logic. pp. 25–30 (2005)
2. Basu, S., Pollack, R., Roy, M.: Algorithms in Real Algebraic Geometry. Springer (2010)
3. Benhamou, F., Granvilliers, L.: Continuous and Interval Constraints. In: Handbook of Constraint Programming, pp. 571–603. Foundations of Artificial Intelligence (2006)
4. Biere, A., Cimatti, A., Clarke, E.M., Strichman, O., Zhu, Y.: Bounded model checking. Advances in Computers 58, 118–149 (2003)
5. Biere, A., Heule, M.J.H., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press (2009)
6. Collins, G.E.: Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In: Brakhage, H. (ed.) Automata Theory and Formal Languages. LNCS, vol. 33, pp. 134–183. Springer (1975)
7. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem proving. Communications of the ACM 5, 394–397 (1962)
8. Eggers, A., Kruglov, E., Kupferschmid, S., Scheibler, K., Teige, T., Weidenbach, C.: Superposition modulo non-linear arithmetic. In: Tinelli, C., Sofronie-Stokkermans, V. (eds.) FroCos. LNCS, vol. 6989, pp. 119–134. Springer (2011)
9. 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)
10. Hong, H.: An improvement of the projection operator in cylindrical algebraic decomposition. In: ISSAC '90. pp. 261–264. ACM (1990)
11. Iwane, H., Yanami, H., Anai, H.: An effective implementation of a symbolic-numeric cylindrical algebraic decomposition for optimization problems. In: Proceedings of the 2011 International Workshop on Symbolic-Numeric Computation. pp. 168–177. ACM (2012)
12. Jovanović, D., de Moura, L.: Solving non-linear arithmetic. In: IJCAR'12. LNCS, Springer (2012)
13. Mishra, B.: Algorithmic Algebra. Texts and Monographs in Computer Science, Springer (1993)
14. Ratschan, S.: Approximate quantified constraint solving by cylindrical box decomposition. Reliable Computing 8(1), 21–42 (2002)
15. Silva, J.P.M., Sakallah, K.A.: Grasp - a new search algorithm for satisfiability. In: ICCAD. pp. 220–227 (1996)
16. Strzebonski, A.W.: Cylindrical algebraic decomposition using validated numerics. Journal of Symbolic Computation 41(9), 1021–1038 (2006)
17. Tseitin, G.S.: On the complexity of derivations in propositional calculus. In: Slisenko, A. (ed.) Studies in Constructive Mathematics and Mathematical Logics (1968)
18. Weispfenning, V.: The complexity of linear problems in fields. Journal of Symbolic Computation 5(1-2), 3–27 (1988)