Subtropical Satisfiability for SMT Solving*

Jasper Nalbach[0000−0002−2641−1380] and Erika Ábrahám[0000−0002−5647−6134]

RWTH Aachen University, Germany

Abstract. A wide range of problems from aerospace engineering and other application areas can be encoded logically and solved using satisfiability modulo theories (SMT) tools, which themselves use dedicated decision procedures for the underlying theories.
Subtropical satisfiability is such a decision procedure for the theory of real arithmetic. Though incomplete, it is a very efficient algorithm and has a high potential for SMT solving. However, yet it has been seldomly used in this context. In this paper we elaborate on possibilities for the efficient usage of subtropical satisfiability in SMT solving.

Keywords: Satisfiability checking, real arithmetic, subtropical satisfiability

1 Introduction

Quantifier-free non-linear real arithmetic (QFNRA) is an expressive but still decidable first-order theory, whose formulas are Boolean combinations of constraints that compare polynomials to zero. Though the complexity of the satisfiability problem for QFNRA is known to be singly exponential, the only complete decision procedure—named the *cylindrical algebraic decomposition (CAD)* method [9]—that is applied in practice has a doubly exponential complexity.

The complexity can be reduced if we are ready to pay the price of giving up completeness. An incomplete but highly efficient method is the *subtropical real root finding* algorithm of Sturm [22] for checking the satisfiability of *one* multivariate *equation*. This method was later extended by Fontaine et al. [14] for the incomplete check of *conjunctions* of multivariate *in*equations for satisfiability.

Both [22] and [14] encode a sufficient condition for satisfying the original problem in *linear* real arithmetic, which can be solved in practice much more efficiently e.g. via a *satisfiability modulo theories (SMT)* solver. SMT solving [5,16] is a technology for checking the satisfiability of quantifier-free first-order logic formulas over different theories. Most SMT solvers are based on the CDCL(T) framework and combine a SAT solver [12,11,18] with one or more theory solver(s). The SAT solver checks whether the Boolean structure of a formula can be satisfied if a set of theory constraints can be assumed to be true, and consults the theory solver(s) to check the feasibility of these *sets (conjunctions)* of theory constraints.

* Jasper Nalbach was supported by the DFG RTG 2236 *UnRAVeL*.

This paper is devoted to the usage of the above mentioned subtropical methods [22,14] in SMT solving. The subtropical methods either detect satisfiability or return unknown. However, the computations are typically very fast, such that solutions bring a huge benefit by avoiding the heavy CAD machinery, and otherwise we can still fall back to the CAD method with only little effort wasted. Surprisingly, we are aware of just two SMT solvers—veriT [7] and SMT-RAT [10]—that use the subtropical methods for theory solving.

Our contributions are the following:

1. It is known that any QFNRA formula can be transformed to a satisfiability-equivalent equation (see e.g. [17]), whose satisfiability can be (incompletely) checked by subtropical real-root finding [22]. For SMT solving, this approach could serve as preprocessing, which tries to solve an input QFNRA formula and supersede the actual SMT call. However, we are not aware of any implementation, thus the practical relevance is unclear. We provide an implementation, attach it as a preprocessor to different SMT solvers and evaluate their efficiency.

2. Our second contribution is another preprocessing technique that is based on the subtropical method [14] for sets of inequations. We suggest a simple but elegant extension to QFNRA formulas with an arbitrary Boolean structure. This method can also be employed as a preprocessing algorithm independently of the internals of the SMT solver which uses it, thus the embedding requires low effort. We provide an implementation and evaluate it again in combination with different SMT solvers.

3. To put the above results in context, we also employ subtropical satisfiability as a theory solver in our CDCL(T)-based SMT solver named SMT-RAT. This allows us to compare the usefulness of subtropical satisfiability as a theory module in CDCL(T) versus using it as a preprocessor.

Outline. After introducing some preliminaries in Section 2, we present our novel subtropical extension in Section 3. We report on experimental results in Section 4 and conclude the paper in Section 5.

2 Preliminaries

Let $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{R}$ and $\mathbb{R}_{>0}$ denote the sets of natural (including 0), integer, real resp. positive real numbers. Assume $d \in \mathbb{N} \setminus \{0\}$ and let $x = (x_1, \ldots, x_d)$ be variables. The transpose of a vector v is denoted by v^T.

Polynomials. A *monomial* m over x is a product $\prod_{i=1}^{d} x_i^{e_i}$ with $e_1, \ldots, e_d \in \mathbb{N}$; we call $\sum_{i=1}^{d} e_i$ the *degree* of m. Note that the monomial of degree 0 is the constant 1. A *term* over x with coefficient domain $\mathbb{Z}$ is a product $c \cdot m$ with $c \in \mathbb{Z}$ and m a monomial over x. A *polynomial* over x with coefficient domain $\mathbb{Z}$ is a sum $\sum_{i=1}^{k} c_i \cdot m_i$ where $k \in \mathbb{N} \setminus \{0\}$ and $c_i \cdot m_i$ are terms over x with coefficient domain $\mathbb{Z}$, such that their monomials are pairwise different. We write $\mathbb{Z}[x]$ for the set of all polynomials over x with coefficient domain $\mathbb{Z}$. A polynomial is *linear* if its monomials are all of degree at most 1.

Constraints. A *(polynomial) constraint* over x with coefficient domain $\mathbb{Z}$ has the form $p \sim 0$ with *defining polynomial* $p \in \mathbb{Z}[x]$ and *relation* $\sim \in \{=, \neq, <, >, \leq, \geq\}$. *Equations* are constraints of the form $p = 0$; *disequations* are constraints shaped $p \neq 0$; *weak inequations* are constraints formed as $p \leq 0$ or $p \geq 0$, *strict inequations* are constraints built as $p < 0$ or $p > 0$; finally, *inequations* are either weak or strict inequations. We use $p(x)$ to explicitly refer to the variables in p, and for $v = (v_1, \ldots, v_d) \in \mathbb{R}^d$ we write $p(v)$ for the value to which p evaluates when we substitute v_i for x_i for $i = 1, \ldots, d$. A *solution* for $p \sim 0$ is any $v \in \mathbb{R}^d$ such that $p(v) \sim 0$ evaluates to true. The solution set of a linear equation (weak inequation) is called a *hyperplane* (*half-space*), whose *normal vector* $n = (n_1, \ldots, n_d)$ is the vector of the coefficients of $x = (x_1, \ldots, x_d)$ in the defining polynomial.

Formulas. Quantifier-free non-linear real arithmetic (QFNRA) formulas are Boolean combinations of constraints. A *quantifier-free linear real arithmetic (QFLRA)* formula is a QFNRA formula whose defining polynomials are all linear. Let φ and ψ be QFNRA formulas and a be a constraint in φ, then $\varphi[\psi/a]$ denotes the formula φ where each occurrence of a is substituted by ψ.

Polytopes. A set $P \subset \mathbb{R}^d$ is *convex* if $v_1 + \lambda(v_2 - v_1) \in P$ for all $v_1, v_2 \in P$ and all $\lambda \in [0, 1] \subseteq \mathbb{R}$. The *convex hull* of a set $V \subset \mathbb{R}^d$ is the smallest convex set $P \subseteq \mathbb{R}^d$ with $V \subseteq P$. *Polytopes* are convex hulls of finite subsets of $\mathbb{R}^d$. A point $v \in \mathbb{R}^d$ is a *vertex* of a polytope $P \subseteq \mathbb{R}^d$ if there exists a linear polynomial $p = c_0 + \sum_{i=1}^{d} c_i \cdot x_i \in \mathbb{Z}[x_1, \ldots, x_d]$ with $p(v) \geq 0$ and $p(u) < 0$ for all $u \in P \setminus \{v\}$; we call v the *vertex of P with respect to the normal vector* $(c_1, \ldots, c_d)$ and refer to $-c_0$ as the *bias*.

Frame and Newton polytope. Let $p = \sum_{i=1}^{k} c_i \cdot \prod_{j=1}^{d} x_j^{e_{i,j}} \in \mathbb{Z}[x]$. We define

- the *frame of* p as $\mathit{frame}(p) = \{(e_{i,1}, \ldots, e_{i,d}) \mid i \in \{1, \ldots, k\} \wedge c_i \neq 0\}$;
- the *positive frame* of p as $\mathit{frame}_+(p) = \{(e_{i,1}, \ldots, e_{i,d}) \in \mathit{frame}(p) \mid c_i > 0\}$;
- the *negative frame* of p as $\mathit{frame}_-(p) = \{(e_{i,1}, \ldots, e_{i,d}) \in \mathit{frame}(p) \mid c_i < 0\}$;
- the *Newton polytope of* p as the convex hull of $\mathit{frame}(p)$.

2.1 SMT solving

Satisfiability modulo theories (SMT) is a technique for checking the satisfiability of quantifier-free first-order logic formulas over different theories. Most SMT solvers implement the CDCL(T)-based framework [5], where a SAT solver [12,11,18] tries to satisfy the Boolean structure of the problem and consults theory solver(s) regarding the consistency of certain theory constraint *sets*.

For QFLRA, an adaptation of the *simplex* method named *general simplex* [13] can be employed as a theory solver. For QFNRA, the only complete decision procedure used in practice is the *cylindrical algebraic decomposition* method [9] and algorithms derived from it, such as the *cylindrical algebraic coverings* method [3] and *NLSAT* [15] in combination with the *single-cell construction*

algorithm [8] (the latter is not based on CDCL(T)). Due to its doubly exponential complexity, it might be advantageous to supplement the CAD by methods that are incomplete but oftentimes faster, e.g. by *interval constraint propagation* [6] or the *virtual substitution* method [23].

2.2 Subtropical satisfiability for a single inequation

The subtropical satisfiability method as introduced in [22,14] provides an incomplete but efficient method for finding solutions for a constraint $p > 0$ with $p \in \mathbb{Z}[x]$. Note that any solution to $p > 0$ is also a solution to $p \geq 0$, and that $p < 0$ and $p \leq 0$ are equivalent to $-p > 0$ respectively $-p \geq 0$, such that the method can be applied to all forms of inequations.

We first recall the sufficient condition from [22,14] for positive solutions.

Theorem 1. *[14, Lemma 2] Assume $k \in \mathbb{N} \setminus \{0\}$, $p = \sum_{i=1}^{k} c_i \cdot \prod_{j=1}^{d} x_j^{e_{i,j}} \in \mathbb{Z}[x] \setminus \{0\}$, and $i' \in \{1, \ldots, k\}$ such that $(e_{i',1}, \ldots, e_{i',d}) \in frame(p)$ is a vertex of the Newton polytope of p with respect to some $n = (n_1, \ldots, n_d) \in \mathbb{R}^d$. Then there exists $a_0 \in \mathbb{R}_{>0}$ such that for all $a \geq a_0$ it holds:*

1. $|c_{i'} \cdot \prod_{j=1}^{d} (a^{n_j})^{e_{i',j}}| > |\sum_{i \in \{1,\ldots,k\} \setminus \{i'\}} c_i \cdot \prod_{j=1}^{d} (a^{n_j})^{e_{i,j}}|$,
2. $sgn(p(a^{n_1}, \ldots, a^{n_d})) = sgn(c_{i'})$.

Assume a non-empty polytope $P \subseteq \mathbb{R}^d$ and a hyperplane with normal vector $(n_1, \ldots, n_d) \in \mathbb{R}^d$ that separates a vertex e of P from the rest of P (see the solid line hyperplane in Fig. 1). Now, assume any polynomial $p = \sum_{i=1}^{k} c_i m_i \in \mathbb{Z}[x]$ with Newton polytope P. Then p has a term, say $c_{i'} m_{i'}$, with the exponent vector e. If the coefficient $c_{i'}$ is positive (negative) then we can make $c_{i'} m_{i'}$ larger (smaller) than the sum of all the other terms of p by the point $(a^{n_1}, \ldots, a^{n_d})$ for a large enough value $a \in \mathbb{R}$.

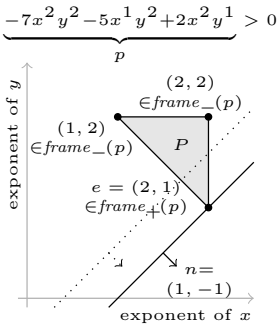


Fig. 1: Solving $p > 0$.

More concretely, a positive solution for $p > 0$, $p = \sum_{i=1}^{k} c_i \cdot \prod_{j=1}^{d} x_j^{e_{i,j}} \in \mathbb{Z}[x]$ can be obtained by:

1. Check whether there exists $e \in frame_{+}(p)$ that is a vertex of the Newton polytope of p with respect to some $n = (n_1, \ldots, n_d) \in \mathbb{R}^d$.
2. If no then return unknown.
3. Otherwise let $a \in \mathbb{R}_{>0}$ and $c \in \mathbb{R}$ with $c > 1$.
4. If $p[a^{n_1}/x_1] \ldots [a^{n_d}/x_d] > 0$ then return $(a^{n_1}, \ldots, a^{n_d})$ as solution.
5. Otherwise update a to $c \cdot a$ and go to 4.

Note that Theorem 1 assures termination. Step 1 is executed by encoding the existence of the desired normal vector n as a QFLRA formula $\mathcal{ST}_{p>0}(n)$, which can then be solved by any QFLRA-solver. Most QFLRA solvers use adaptions of the simplex method, for which weak inequations are more advantageous. Therefore, the encoding uses another characterization for e being a vertex of a

polytope P: Its definition requiring the *existence of a half-space that contains the whole polytope P but e is the only point from P that lies exactly on its hyperplane* is equivalent to requiring the *existence of a half-space that excludes e but contains all other vertices of a polytope P* (see the hyperplane with the dotted line in Fig. 1).

$$\mathcal{ST}_{p>0}(n) := \exists b.\ \bigvee_{e \in frame_+(p)} \left(n^T \cdot e > b \wedge \bigwedge_{u \in frame(p) \setminus \{e\}} n^T \cdot u \leq b \right) .$$

This property can be encoded even more efficiently as suggested in [14]:

$$\mathcal{ST}_{p>0}(n) := \exists b.\ \left(\bigvee_{e \in frame_+(p)} n^T \cdot e > b \right) \wedge \left(\bigwedge_{u \in frame_-(p)} n^T \cdot u \leq b \right) .$$

With the above encoding we can only find positive solutions from $\mathbb{R}^d_{>0}$. However, the encoding can be extended to the general case based on the observation that we could substitute in p any subset of the variables x_i by $-x_i$: If we find a positive solution for this modified problem, then we get a solution for the original problem by exchanging the positive values v_i for $-x_i$ by the negative values $-v_i$ for x_i. To encode all possible such *sign changes*, we introduce for every variable x_i, $i = 1, \ldots, d$ a Boolean variable neg_i which encodes whether we search for a negative ($neg_i = 1$) or a positive ($neg_i = 0$) value for x_i. The exponent vector of a term $c \cdot x_1^{e_1} \ldots x_d^{e_d}$ of p with $c \neq 0$ is in the positive frame iff ($c > 0 \leftrightarrow |\{x_i \mid \exists k \in \mathbb{N}.\ e_i = 2k + 1 \wedge neg_i = 1\}|$ is even). For details on the encoding, we refer to [14].

The encodings above for a constraint of the form $p > 0$ can be generalized to constraints of all types but equations. To do so, for any polynomial p we define

$$\begin{aligned}
&\mathcal{ST}_{p \geq 0}(n, neg) := \mathcal{ST}_{p>0}(n, neg) \\
&\mathcal{ST}_{p \leq 0}(n, neg) := \mathcal{ST}_{p<0}(n, neg) := \mathcal{ST}_{-p>0}(n, neg) \\
&\mathcal{ST}_{p \neq 0}(n, neg) := (\mathcal{ST}_{p>0}(n, neg) \oplus \mathcal{ST}_{p<0}(n, neg))
\end{aligned}$$

where $\oplus$ denotes the exclusive-or operator. Given the fact that from each solution of $\mathcal{ST}_{p>0}(n, neg)$ we can derive positive values for p, it is easy to see that the satisfiability of $\mathcal{ST}_c(n, neg)$ implies the satisfiability of c.

Example 1. The encodings of $x_1 + x_1 \cdot x_2^3 < 0$ and $x_2 + x_1^3 > 0$ are

$$\begin{aligned}
\mathcal{ST}_{x_1 + x_1 \cdot x_2^3 < 0}(n, neg) =& \exists b_1.\ ((\neg neg_1 \wedge n_1 > b_1) \vee (\neg(neg_1 \oplus neg_2) \wedge n_1 + 3n_2 > b_1)) \\
&\wedge (neg_1 \rightarrow n_1 \leq b_1) \wedge ((neg_1 \oplus neg_2) \rightarrow n_1 + 3n_2 \leq b_1) \\
\mathcal{ST}_{x_2 + x_1^3 > 0}(n, neg) =& \exists b_2.\ ((\neg neg_2 \wedge n_2 > b_2) \vee (\neg neg_1 \wedge 3n_1 > b_2)) \\
&\wedge (neg_2 \rightarrow n_2 \leq b_2) \wedge (neg_1 \rightarrow 3n_1 \leq b_2) .
\end{aligned}$$

The above approach strictly separates a positive frame point from all negative ones. However, there are cases where a weak separation suffices (i.e. the separated positive frame point may lie on the hyperplane). The work presented in [20] defines an encoding considering these cases, but this encoding seems to be larger and without much computational advantage.

2.3 Subtropical satisfiability for a single equation

To find a solution for a single equation $p = 0$, $p \in \mathbb{Z}[x]$, the method from [22,14] tries to identify two points $v_-, v_+ \in \mathbb{R}^d$ such that $p(v_-) < 0$ and $p(v_+) > 0$ and use the intermediate value theorem to construct a $v_0 \in \mathbb{R}^d$ with $p(v_0) = 0$:

1. Choose $v_- = (1, \ldots, 1)$.
2. If $p(v_-) = 0$ then return v_- as a solution to $p = 0$.
3. If $p(v_-) > 0$ then we set p to $-p$. (Note that now $p(v_-) < 0$.)
4. Apply the method from Section 2.2 to find a $v_+ \in \mathbb{R}^d$ with $p(v_+) > 0$.
5. If unsuccessful then return unknown.
6. Let $p^* : [0,1] \to \mathbb{R}$, $t \mapsto p(v_- + t \cdot (v_+ - v_-))$. Since p^* is continuous, $p^*(0) < 0$ and $p^*(1) > 0$, we know that $p^*(t_0) = 0$ at some $t_0 \in (0,1) \subseteq \mathbb{R}$. We can use real root isolation techniques based on Descartes' rule of signs or Sturm sequences to find such a t_0, which yields $p(v_0) = 0$ for $v_0 = v_- + t_0(v_+ - v_-)$.

2.4 Subtropical satisfiability for conjunctions of inequations

For finding solutions for a conjunction of inequations $\varphi = p_1 \sim_1 0 \wedge \ldots \wedge p_\ell \sim_\ell 0$ with $\sim_i \in \{<, >, \leq, \geq, \neq\}$ for $i = 1, \ldots, \ell$, Fontaine et al. [14] propose to apply the method from Section 2.2 to simultaneously separate a suitable frame point for each involved polynomial through hyperplanes but with the same normal vector and assure to agree on a common sign change, resulting in the encoding

$$\mathcal{ST}_\varphi(n, neg) := \mathcal{ST}_{p_1 \sim_1 0}(n, neg) \wedge \ldots \wedge \mathcal{ST}_{p_\ell \sim_\ell 0}(n, neg) \ .$$

We solve $\mathcal{ST}_\varphi(n, neg)$ with a linear arithmetic solver. If $\mathcal{ST}_\varphi(n, neg)$ is satisfiable then from each solution μ of $\mathcal{ST}_\varphi(n, neg)$ we can derive a solution for φ by multiplying the value of some $a \in \mathbb{R}_{>0}$ with a factor $c \in \mathbb{R}$, $c > 1$ until all inequations are satisfied by the values $(-1)^{\mu(neg_i)} \cdot a^{\mu(n_i)}$ for x_i, $i = 1, \ldots, d$.

2.5 Transformation of a QFNRA formula into a single equation

Assume a QFNRA formula φ in negation normal form (i.e. only constraints are allowed to be negated). The transformation proposed in [21] introduces for every constraint c in φ a fresh (real-valued) variable y_c to generate an equisatisfiable formula $Tr(\varphi)$ as follows, using a transformer subfunction tr and the negation operator from Figure 2 for $\sim \in \{<, >, \leq, \geq, =, \neq\}$:

$\sim$	$<$	$>$	$\leq$	$\geq$	$=$	$\neq$
$\not\sim$	$\geq$	$\leq$	$>$	$<$	$\neq$	$=$

Fig. 2: Predicate negation

$$
\begin{array}{ll}
Tr(p = 0) := p = 0 & Tr(\bigwedge_{i=1}^n \varphi_i) := tr(\bigwedge_{i=1}^n Tr(\varphi_i)) \\
Tr(p \geq 0) := p - (y_{p\geq 0})^2 = 0 & Tr(\bigvee_{i=1}^n \varphi_i) := tr(\bigvee_{i=1}^n Tr(\varphi_i)) \\
Tr(p > 0) := (y_{p>0})^2 \cdot p - 1 = 0 & Tr(\neg(p \sim 0)) := Tr(p \not\sim 0) \\
Tr(p \leq 0) := p + (y_{p\leq 0})^2 = 0 & tr(\bigvee_{i=1}^n p_i = 0) := \prod_{i=1}^n p_i = 0 \\
Tr(p < 0) := (y_{p<0})^2 \cdot p + 1 = 0 & tr(\bigwedge_{i=1}^n p_i = 0) := \sum_{i=1}^n (p_i)^2 = 0 \\
Tr(p \neq 0) := y_{p\neq 0} \cdot p + 1 = 0 &
\end{array}
$$

In this paper, we build on the work done in the thesis [17] which considered this transformation for solving Boolean combinations of constraints using subtropical real root finding.

3 Subtropical satisfiability for QFNRA formulas

Subtropical satisfiability is restricted to finding solutions for either a single equation or a conjunction of inequations. This is already sufficient for its embedding within a CDCL(T)-based SMT solver as a fast but incomplete theory solver backend, applicable only to a single equation or a set of inequations, and possibly returning unknown even in those cases. Using it in an MCSAT-based SMT solver would be possible to derive consistent extensions of partial assignments, even though we are not aware of any solver exploiting this possibility. However, in both cases, a relatively difficult individual adaption of the SMT solver is required to implement and embed the subtropical method.

In the following, we aim to increase the scope of the subtropical method by making it *applicable to general QFNRA formulas*. In Section 3.1 we first revisit and slightly adapt the method from Section 2.5 to transform any QFNRA formula into a single equation, which can be solved with the subtropical root finding from [22,14]. Then in Section 3.2 we propose two slightly different novel subtropical encodings for general QFNRA formulas. Similarly to the original method, we define a transformation to QFLRA, whose formulas can be checked by a linear SMT solver. Our method aims to serve as an incomplete but efficient *preprocessing check* for satisfiability before the main solver is called; as such, our C++ implementation could be relatively easily adapted as a preprocessor for other solvers, as it has no interaction with the main SMT algorithm.

3.1 Transforming a QFNRA formula into a single equation

With the method from Section 2.5, we can transform any QFNRA formula φ to a single equisatisfiable equation $Tr(\varphi)$ and then use the subtropical root finding algorithm as described in Section 2.3 for finding a solution for the equation $Tr(\varphi)$, which will also be a solution for φ. However, this transformation has one weakness: the transformation of a conjunction $\bigwedge_{i=1}^{n} p_i = 0$ results in a sum-of-square polynomial $\sum_{i=1}^{n}(p_i)^2 = 0$; a disjunction of conjunctions $\bigvee_{i=1}^{n}\bigwedge_{j=1}^{n_i} p_{i,j} = 0$ results in a product of sum-of-squares polynomials $\prod_{i=1}^{n}\sum_{i=1}^{n_i}(p_{i,j})^2 = 0$, which is itself a sum-of-squares. In both cases, the resulting sum-of-squares polynomials are non-negative at all points, thus the subtropical real root finding (which needs a positive as well as a negative value for a polynomial for finding a real root for it) will fail. We are not aware of any alternative for the transformation of conjunctions that makes the subtropical method applicable. To avoid computational effort in these cases, we alter the transformation by setting

$$Tr(\bigwedge_{i=1}^{n} \varphi_i) \;:=\; \mathit{false}\,.$$

3.2 Generalizing the subtropical encoding to QFNRA formulas

In this work we propose an alternative approach, which generalizes the idea of Section 2.4 from conjunctions of in- and disequations to arbitrary Boolean combinations of constraints. The generalization is straight-forward and simple to implement, and still surprisingly efficient.

Remember that the method in Section 2.4 takes a conjunction of constraints $\varphi = p_1 \sim_1 0 \wedge \ldots \wedge p_\ell \sim_\ell 0$ with $\sim_i \in \{<, >, \leq, \geq, \neq\}$ as input and encodes by

$$\mathcal{ST}_\varphi(n, neg) := \mathcal{ST}_{p_1 \sim_1 0}(n, neg) \wedge \ldots \wedge \mathcal{ST}_{p_\ell \sim_\ell 0}(n, neg)$$

the existence of a separating hyperplane for each in-/disequation but requiring a shared normal vector n for all separating hyperplanes and a shared sign change vector *neg*. If the formula $\mathcal{ST}_\varphi(n, neg)$ is satisfiable, then we get values for n_i and neg_i from which we can construct a solution for φ by setting x_i to the value of $neg_i \cdot a^{n_i}$ for $i = 1, \ldots, d$ with a large enough value for a.

We generalize this idea to arbitrary QFNRA formulas φ. First we eliminate all negations in φ by bringing the formula to negation normal form and applying negation to the predicates as in Figure 2. We assume in the following that φ contains *no negation*.

We use the same encoding $\mathcal{ST}_c(n, neg)$ as before for in- and disequations c, but apply the formula's Boolean structure to these encodings. As the formula might also contain equations, which we cannot handle in combination with other constraints, we extend the previous encoding $\mathcal{ST}$ to

$$\hat{\mathcal{ST}}_{p \sim 0}(n, neg) := \begin{cases} \mathcal{ST}_{p \sim 0}(n, neg) & \text{if } \sim \in \{<, >, \leq, \geq, \neq\} \\ \mathit{false} & \text{otherwise} \end{cases}$$

Note that even though we neglect the possibility of satisfying the formula by fulfilling equations, the following encodings might still lead to a subtropical solution when a solution can be found by satisfying in- or disequations. Let in the following $c_1, \ldots, c_\ell$ be all the different constraints that occur in φ. We follow two approaches for the encoding.

1. Direct substitution of constraints. Our first approach generates an encoding $\mathcal{ST}_\varphi^{\text{direct}}(n, neg)$ by replacing each constraint c_i in the formula φ directly by $\hat{\mathcal{ST}}_{c_i}(n, neg)$. This way, we preserve the Boolean structure of the formula and assure that each satisfying solution of $\mathcal{ST}_\varphi^{\text{direct}}(n, neg)$ encodes separating hyperplanes with a common normal vector and a common sign change vector for a set of constraints, whose satisfaction implies the satisfaction of the Boolean structure of φ:

$$\mathcal{ST}_\varphi^{\text{direct}}(n, neg) := \varphi[\hat{\mathcal{ST}}_{c_1}(n, neg)/c_1] \ldots [\hat{\mathcal{ST}}_{c_\ell}(n, neg)/c_\ell].$$

2. Encoding via auxiliary variables. In our second approach we separately encode the Boolean structure by building φ's Boolean skeleton and combine this with the encodings of the sufficient conditions for the satisfaction of constraints. The

motivation behind this variant is to enable the encoding of more knowledge about the relations between the formula's constraints.

In this encoding, we first fix some monomial ordering and normalize all constraints in φ such that the leading coefficients of their defining polynomials become 1 (through divisions by suitable constants), and replace each disequation $p \neq 0$ by $(p < 0 \vee p > 0)$. Note that these transformations might change the number of different constraints. Next we replace all occurrences of all constraints $c_1, \ldots, c_k$ in the formula, c_i having the form $p_i \sim_i 0$ with $\sim_i \in \{<, \leq, =, \geq, >\}$, by fresh Boolean variables $a = (a_1, \ldots, a_k)$. Remember that we assumed the input formula to contain no negation, such that after this transformation no proposition is negated. The encoding is defined as

$$\mathcal{ST}^{\text{aux}}_{\varphi}(n, b, a) := \varphi[a_1/c_1] \ldots [a_k/c_k] \wedge \bigwedge_{i \in \{1,\ldots,k\}} (a_i \rightarrow \hat{\mathcal{ST}}_{c_i}(n, neg))$$
$$\wedge \bigwedge_{\substack{i,j \in \{1,\ldots,k\} \\ i \neq j,\ p_i = p_j, \\ \sim_i \in \{<,\leq\},\ \sim_j \in \{>,\geq\}}} (\neg a_i \vee \neg a_j)$$

While this encoding shares the subtropical encoding idea with the previous one, there are two main differences: 1) The first encoding would correspond to an iff "$\leftrightarrow$" for defining the meaning of the propositions a_i, which is weakened here to an implication. 2) The second row encodes additional knowledge about constraints with identical defining polynomials: each pair of constraints putting zero as a lower respectively upper bound on the same polynomial are considered to be conflicting. Note that, even though such a pair of weak bounds might be simultaneously satisfied if the polynomial evaluates to 0, since the subtropical method is unable to handle equations, we neglect this possibility in our sufficient condition.

Number of auxiliary variables. For the first encoding, we need d real variables $n = (n_1, \ldots, n_d)$ to encode the shared normal vector of the separating hyperplanes, d Boolean variables $neg = (neg_1, \ldots, neg_d)$ to encode the sign changes, and ℓ real variables $d = (d_1, \ldots, d_\ell)$ to encode the offsets of the hyperplanes for the constraints. Thus in total we introduce $2d+\ell$ variables for the whole formula.

For the second encoding, in addition we introduce at most 2ℓ Boolean variables for the abstraction of constraints and at most double the number of offset variables (note that the elimination of $\neq$ might double the number of constraints). This gives us in total $2d + 4\ell$ variables for the whole formula.

Example 2. The second encoding for $x_1 + x_1 \cdot x_2^3 < 0 \vee x_2 + x_1^3 > 0$ yields: $\exists a_1.\, \exists a_2.\, \big(a_1 \vee a_2\big) \wedge \big(a_1 \rightarrow \mathcal{ST}_{x_1 + x_1 \cdot x_2^3 < 0}(n, neg)\big) \wedge \big(a_2 \rightarrow \mathcal{ST}_{x_2 + x_1^3 > 0}(n, neg)\big)$ where a_1 and a_2 are the abstraction literals of the two constraints, and the encodings of neg_1, neg_2, n_1, n_2, b_1, and b_2 are given in Example 1.

4 Experimental results

In order to evaluate their practical usefulness, we implemented the previously presented algorithms in our `SMT-RAT` framework [10].

For the experiments, we employ the complete *QFNRA* [1] benchmark set from *SMT-LIB* [2]. This covers 12134 benchmarks (5209 known to be satisfiable, 5029 known to be unsatisfiable, remaining with unknown status). When comparing different approaches, we also consider a *virtual best (VB)* solver, which is computed by taking the best solver for each benchmark (i.e. for each benchmark we take the best solver for it: we prefer solved instances over unknowns over timeouts over memouts, and take the shortest running time). Furthermore, we apply standard preprocessing on all benchmarks.

For the execution we used an *Intel® Xeon® Platinum 8160 2.1GHz* processor with a memory limit of *4GB* per run and a timeout of *two minutes*. For QFNRA, this timeout suffices to cover almost all benchmarks that can be solved (as indicated also in the results below); further, the relatively short timeout is justified as we aim to apply the subtropical methods for fast incomplete satisfiability checks to supplement a complete solver.

The implementation which generated the following results is available at `https://doi.org/10.5281/zenodo.7509171`.

4.1 Pure subtropical solvers

Solvers. We first use the presented ideas for solving a QFNRA formula φ four different ways:

`Equation` We transform φ into a single equation as in Sec. 3.1 and solve the equation with subtropical root finding from Sec. 2.2. We use SMT-RAT for the involved QFLRA checks.

`Formula` We generate the first encoding $\mathcal{ST}_{\varphi}^{\text{direct}}$ as in Sec. 3.2 and invoke SMT-RAT for solving the result.

`FormulaAlt` We generate the second encoding $\mathcal{ST}_{\varphi}^{\text{aux}}$ as in Section 3.2 and invoke SMT-RAT for solving the result.

`Incremental` We apply to φ SMT-RAT with CDCL(T) and the subtropical satisfiability method for conjunctions from Sec. 2.4 as the only theory solver. If the input contains an equation or a disequation then the theory solver returns *unknown*.

Benchmarks. We observed that both `Formula` and `FormulaAlt` could conclude unsatisfiability on 2614 benchmarks, either during the transformation of the formula to negation normal form or due to detecting unsatisfiability of the Boolean structure during preprocessing. Therefore, in this subsection we decided to omit trivial benchmarks from the benchmark set to make the differences between the approaches better visible. The omitted benchmarks are those which - after standard preprocessing is performed - can be solved either by calling a SAT solver on the Boolean abstraction, or with a CDCL(T)-based SMT solver with

Table 1: Results for the pure subtropical solvers on 3580 benchmarks with a timeout of 120 seconds. Each cell contains the number of benchmarks in the given category; where meaningful, mean running times in seconds are given in parentheses.

	`Equation`	`Formula`	`FormulaAlt`	`Incremental`	`VB`
solved/sat	16 (0.01)	1399(0.17)	1398(0.12)	1399(0.13)	1399(0.11)
unknown	2232 (1.84)	782(5.58)	793(5.83)	272(1.05)	852(4.89)
timeout	990	1057	1046	1566	995
memout	342	342	343	343	334

basic conflict detection (i.e. normalizing the input constraints and checking for contradictory relation symbols).

This reduces the benchmark set from 12134 to 3580 benchmarks (1735 of them are known to be satisfiable, 499 of them are known to be unsatisfiable, and for the remaining benchmarks the status is unknown).

Results. The results are listed in Table 1. The `Equation` transformation finds a solution only in a few cases. This was expected as it fails on conjunctions; in the benchmark set, conjunctions on the top-level are common.

`Formula` solves 1399 of the satisfiable benchmarks while `FormulaAlt` solves one large benchmark less. Furthermore, `Formula` returns unknown on 782 benchmarks, while `FormulaAlt` returns unknown on 11 benchmarks more (one of which is unsatisfiable and the others have unknown status and are thus rather big). However, `FormulaAlt` has fewer timeouts. Thus this encoding might pay off in the sense that if it does not find a solution then it terminates earlier.

`Formula` and `FormulaAlt` do timeout on unsolved benchmarks less often than `Incremental` but return *unknown* instead. As also indicated in Figure 3, `Incremental` is slightly faster than `Formula` on about 15 benchmarks, but slower on one benchmark. However, due to the relatively low number of benchmarks, it is hard to draw a clear conclusion.

The mean running times in seconds, given in parentheses in Table 1, clearly show that finding solutions is very fast. Determining unknown, i.e. that the encoding is unsatisfiable, takes a bit more time, but at this point we note that SMT-RAT is tuned for QFNRA and is not very competitive on QFLRA. Table 3 on page 14 contains running times as for `Formula` but using Z3 (column `ForZ3`) resp. cvc5 (column `ForCvc5`) instead of SMT-RAT; there, both sat and unknown results are computed in 0.03-0.04 seconds (seconds) in average, with just a few timeouts and without any memouts.

4.2 Combining subtropical methods with a complete procedure

Solvers. Next we combine `Formula`, `FormulaAlt` and `Incremental` with the complete *cylindrical algebraic covering (CAlC)* algorithm as a theory solver to decide the satisfiability of sets of polynomial constraints. We consider `CAlC`, which is a CDCL(T)-based solver with the CAlC algorithm as a complete theory

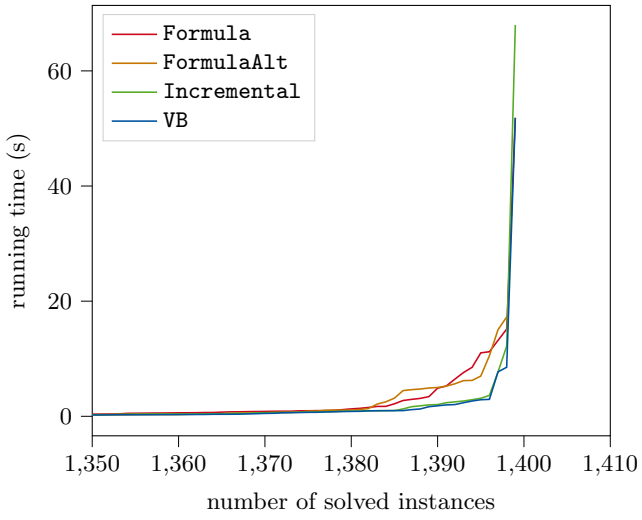


Fig. 3: Performance profile for the plain variants. A point on a line can be understood as follows: the horizontal axis denotes the number of benchmarks which can be solved (i.e. satisfiability can be concluded) if the timeout is set to the value on the vertical axis.

solver backend (implemented in SMT-RAT). The solvers `F+CAlC` and `FA+CAlC` run `Formula` respectively `FormulaAlt` and if the result is inconclusive then they invoke `CAlC`. `I+CAlC` is like `Incremental` but for each theory call, if the subtropical theory solver fails then the complete CAlC theory solver backend in invoked. Furthermore, we consider the combinations `F+I+CAlC` and `FA+I+CAlC`, running run `Formula` respectively `FormulaAlt` first and if they cannot solve the problem then invoking `I+CAlC`. In addition, we again list the virtual best (`VB`) results. All variants employ standard preprocessing.

Results for satisfiable benchmarks. Results for all 12134 SMT-LIB benchmarks are shown in Table 2. The subtropical methods do gain some slight improvements: the `I+CAlC` variant solves 16 satisfiable benchmarks more than `CAlC`, `F+CAlC` and `FA+CAlC` solve 19 respectively 18 satisfiable benchmarks more than `CAlC`. The combinations `F+I+CAlC` and `FA+I+CAlC` solve 21 and 20 satisfiable benchmarks more. Note that `I+CAlC` and `F+CAlC` do not solve the same set of benchmarks because `VB` solves the most satisfiable benchmarks.

Results for unsatisfiable benchmarks. The number of solved unsatisfiable benchmarks (and thus the total number of solved benchmarks) needs to be interpreted carefully: Some benchmarks are detected to be unsatisfiable by the transformation to negation normal form (as done in the implementations of `Formula` and `FormulaAlt`) and relatively simple conflict checks based on normalizing of the constraints and comparing for contradictory relation symbols (as implemented

Table 2: Results for the combination with the `CAlC` backend. Each cell contains the number of benchmarks in the given category; where meaningful, mean running times in seconds are given in parentheses.

	`CAlC`	`I+CAlC`	`F+CAlC`	`FA+CAlC`	`F+I+CAlC`	`FA+I+CAlC`	`VB`
sat	5055 (0.27)	5071 (0.27)	5074 (0.31)	5073 (0.29)	5076 (0.28)	5075 (0.26)	5076 (0.26)
unsat	4825 (1.40)	4837 (1.45)	4829 (1.51)	4830 (1.48)	4837 (1.44)	4841 (1.46)	4845 (1.48)
solved	9880 (0.82)	9908 (0.84)	9903 (0.90)	9903 (0.87)	9913 (0.85)	9916 (0.85)	9921 (0.85)
timeout	1854	1843	1843	1839	1837	1839	1840
memout	400	383	388	392	384	379	373

in `Incremental`); these implementation details lead to more solved unsatisfiable benchmarks in our experiments.

Subtropical as preprocessing vs subtropical as incremental theory solver. In general, the differences in solved benchmarks between the different variants are not big enough to draw any reliable conclusion. Still, we find that the `F+CAlC` and `FA+CAlC` variants are comparable to the incremental `I+CAlC` variant - which supports our idea to use subtropical as an efficient preprocessing method to complement other solvers.

4.3 Z3 and cvc5

As `Formula` and `FormulaAlt` can be employed as a preprocessing unit without any modification to an SMT solver, we can easily combine them with `Z3` [19] and `cvc5` [4]. In the following we focus on the `Formula` transformation. Out of the 12134 SMT-LIB benchmarks, `Formula` is applicable (i.e. the transformation does not directly simplify to false) to 4717 benchmarks (3170 satisfiable, 1439 unsatisfiable, and the others have unknown status) and not applicable to the remaining 7417.

We have run the solvers `Z3` and `cvc5` on the QFLRA transformation results of `Formula` (`ForZ3` resp. `ForCvc5`) as well as on the original QFNRA benchmarks (`Z3` resp. `Cvc5`), and combined the results into virtual solvers `ForZ3+Z3` resp. `ForCvc5+cvc5` by first applying `ForZ3` resp. `ForCvc5`, and only if the result is inconclusive then calling `Z3` resp. `Cvc5` on the original formula. In Table 3 we set the timeout for the subtropical solver `Formula` to 10 seconds, leaving at least 110 seconds solving time for the SMT solver if a call is needed. In total, the combination of both solvers has a timeout of 120 seconds. We ignore the time required for the subtropical transformation here, claiming that this time is negligible.

Effectiveness of the subtropical method. The subtropical method solves a good portion of the satisfiable benchmarks: `ForZ3` and `ForCvc5` solve 1357 resp. 1352 satisfiable benchmarks within 10 seconds timeout, with a mean running time

Table 3: Results for Z3 and cvc5 without subtropical preprocessing (`Z3`, `cvc5`), only the subtropical preprocessing employing Z3 and cvc5 for solving the encodings (`ForZ3`, `ForCvc5`) and their sequential combination (`ForZ3+Z3`, `ForCvc5+cvc5`). The last row in the header specifies the respective timeouts. Each cell contains the number of benchmarks in the given category; where meaningful, mean running times in seconds are given in parentheses. Not applicable means that the transformation directly simplifies to false.

	`Z3` to=120	`ForZ3` to=10	`ForZ3 +Z3` to=120	`cvc5` to=120	`ForCvc5` to=10	`ForCvc5 +cvc5` to=120
sat	5515(0.64)	1357(0.04)	5524(0.62)	5370(1.51)	1352(0.04)	5395(1.49)
unsat	5336(1.20)	-	5336(1.21)	5728(2.13)	-	5728(2.14)
solved	10851(0.92)	1357(0.04)	10860(0.91)	11098(1.82)	1352(0.04)	11123(1.82)
unknown	3(15.36)	3333(0.03)+ 7417 not appl.	3(15.36)	0	3324(0.04)+ 7417 not appl.	0
timeout	1280	27	1271	1036	41	1011

of just 0.04 seconds. Both have relatively few timeouts. The unknown results are presented as a sum of (i) the applicable but not solvable and (ii) the not applicable cases.

Effectiveness of subtropical as preprocessing to complete solvers. The results shown in Table 3 are similar to the previous findings that some additional benchmarks can be solved. The supplementation with the subtropical method solves 9 benchmarks which cannot be solved by `Z3`; in the case of `cvc5` 25 new benchmarks are solved. Although these gains are small in number, the gained instances are hard for the considered SMT solvers. Furthermore, we remind that we cannot generalize these statements beyond the given benchmark set as the applicability of the subtropical method heavily depends on the structure of its input. If the method is applicable, it can be very efficient: Figure 4 shows that the subtropical method solves some benchmarks on which the pure solvers fail even in the relatively short time of 10 seconds. Furthermore, Figure 4 also illustrates nicely that even if it fails, the subtropical method has no remarkable negative effect on the running time. Figure 5 shows that subtropical fails relatively quickly on unsatisfiable instances, while it may fail later on satisfiable ones.

5 Conclusion

In this paper we introduced and evaluated different variants of the subtropical satisfiability method for checking the satisfiability of QFNRA formulas. While some of the approaches need to be integrated into an SMT solver, other methods are suitable as preprocessing algorithms that can be implemented outside of an existing solver and are thus more generally applicable. The results demonstrate that our new approach for such a preprocessing enabled the solvers to solve some more instances which could not be solved before.

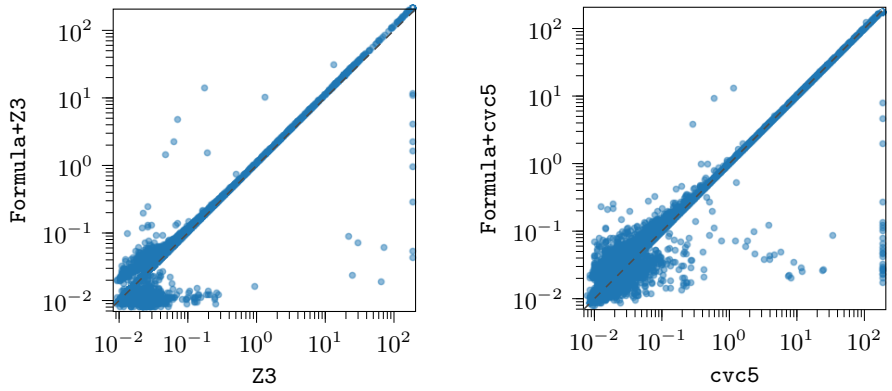
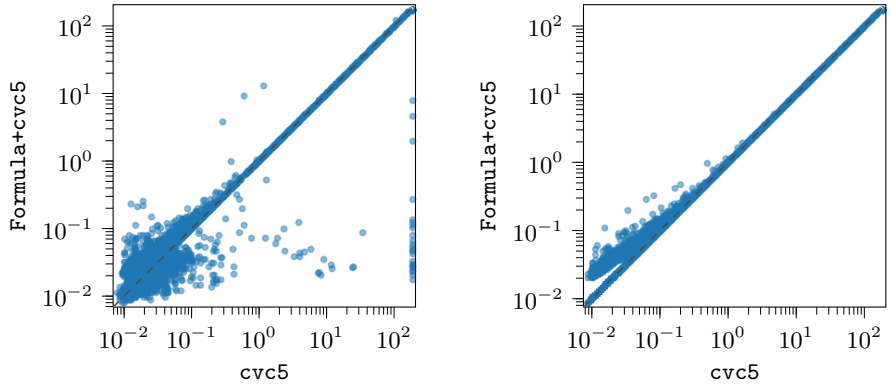


Fig. 4: Scatter plots for Z3 and cvc5, without versus with subtropical preprocessor.



(a) Instances known to be satisfiable.

(b) Instances known to be unsatisfiable.

Fig. 5: Scatter plots for cvc5, without versus with subtropical preprocessor.

Though the number of additionally solved instances is relatively small, these problems are hard for mature SMT solvers like Z3 or cvc5. Furthermore, we need to keep in mind that the variety of real-algebraic benchmarks in SMT-LIB is still limited; the evaluation in [14] discusses that the benchmarks do not contain inequations with high degrees on which the subtropical method should be efficient.

Acknowledgements. We thank Ömer Sali and Gereon Kremer for the implementation of the subtropical method as a CDCL(T) theory solver in SMT-RAT, and Giang Lai for discussions.

References

1. Satisfiability modulo theories library for QF_NRA, https://clc-gitlab.cs.uiowa.edu:2443/SMT-LIB-benchmarks/QF_NRA
2. The Satisfiability Modulo Theories Library (SMT-LIB), https://www.SMT-LIB.org
3. Ábrahám, E., Davenport, J., England, M., Kremer, G.: Deciding the consistency of non-linear real arithmetic constraints with a conflict driven search using cylindrical algebraic coverings. Journal of Logical and Algebraic Methods in Programming **119**, 100633 (2021), https://doi.org/10.1016/j.jlamp.2020.100633
4. Barbosa, H., Barrett, C.W., Brain, M., Kremer, G., Lachnitt, H., Mann, M., Mohamed, A., Mohamed, M., Niemetz, A., Nötzli, A., Ozdemir, A., Preiner, M., Reynolds, A., Sheng, Y., Tinelli, C., Zohar, Y.: cvc5: A versatile and industrial-strength SMT solver. In: Proceedings of the 28th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2022). LNCS, vol. 13243, pp. 415–442. Springer (2022), https://doi.org/10.1007/978-3-030-99524-9_24
5. Barrett, C., Sebastiani, R., Seshia, S.A., Tinelli, C.: Satisfiability modulo theories. In: Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185, chap. 26, pp. 825–885. IOS Press (2009)
6. Benhamou, F., Granvilliers, L.: Continuous and interval constraints. Foundations of Artificial Intelligence **2**, 571–603 (2006), https://doi.org/10.1016/S1574-6526(06)80020-9
7. Bouton, T., de Oliveira, D.C.B., Déharbe, D., Fontaine, P.: veriT: An open, trustable and efficient SMT-solver. In: Proc. of CADE-22. LNCS, vol. 5663, pp. 151–156. Springer (2009)
8. Brown, C.W., Košta, M.: Constructing a single cell in cylindrical algebraic decomposition. Journal of Symbolic Computation **70**, 14–48 (2015), https://doi.org/10.1016/j.jsc.2014.09.024
9. Collins, G.E.: Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In: Proceedings of the 2nd GI Conference on Automata Theory and Formal Languages (1975). pp. 134–183. Springer (1975), https://doi.org/10.1007/3-540-07407-4_17
10. Corzilius, F., Kremer, G., Junges, S., Schupp, S., Ábrahám, E.: SMT-RAT: An open source C++ toolbox for strategic and parallel SMT solving. In: Proceedings of the 18th International Conference on Theory and Applications of Satisfiability Testing (SAT 2015). pp. 360–368. Springer (2015), https://doi.org/10.1007/978-3-319-24318-4_26
11. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Communications of the ACM **5**(7), 394–397 (1962)
12. Davis, M., Putnam, H.: A computing procedure for quantification theory. Journal of the ACM **7**(3), 201–215 (1960)
13. Dutertre, B., de Moura, L.: A fast linear-arithmetic solver for dpll(t). In: Proceedings of the 18th International Conference on Computer Aided Verification (CAV 2006). pp. 81–94. Springer (2006). https://doi.org/https://doi.org/10.1007/11817963_11
14. Fontaine, P., Ogawa, M., Sturm, T., Vu, X.T.: Subtropical satisfiability. In: International Symposium on Frontiers of Combining Systems (FroCoS 2017). pp. 189–206. Springer (2017), https://doi.org/10.1007/978-3-319-66167-4_11

15. Jovanovic, D., de Moura, L.: Solving non-linear arithmetic. In: Proceedings of the 6th International Joint Conference on Automated Reasoning (IJCAR 2012), LNCS, vol. 7364, pp. 339–354. Springer (2012), https://doi.org/10.1007/978-3-642-31365-3_27
16. Kroening, D., Strichman, O.: Decision Procedures - An Algorithmic Point of View. Springer (2008)
17. Lai, G.: Subtropical satisfiability for polynomial constraint sets (2022), https://ths.rwth-aachen.de/wp-content/uploads/sites/4/lai_bachelor.pdf
18. Marques-silva, J.P., Sakallah, K.A.: GRASP: A search algorithm for propositional satisfiability. IEEE Transactions on Computers **48**, 506–521 (1999)
19. de Moura, L., Bjørner, N.: Z3: An efficient smt solver. In: Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2008). pp. 337–340. LNCS, Springer (2008), https://doi.org/10.1007/978-3-540-78800-3_24
20. Sali, Ö.: Linearization Techniques for Nonlinear Arithmetic Problems in SMT. Master's thesis, RWTH Aachen University (2018), https://ths.rwth-aachen.de/wp-content/uploads/sites/4/teaching/theses/sali_master.pdf
21. Seidenberg, A.: A new decision method for elementary algebra. Annals of Mathematics **60**(2), 365–374 (1954), https://doi.org/10.2307/1969640
22. Sturm, T.: Subtropical real root finding. In: Proceedings of the 2015 ACM on International Symposium on Symbolic and Algebraic Computation (ISSAC 2015). pp. 347–354 (2015), https://doi.org/10.1145/2755996.2756677
23. Weispfenning, V.: Quantifier elimination for real algebra—the quadratic case and beyond. Applicable Algebra in Engineering, Communication and Computing **8**(2), 85–101 (1997), https://doi.org/10.1007/s002000050055