More is Less: Adding Polynomials for Faster Explanations in NLSAT

Valentin Promies[1 [0000−0002−3086−9976]]✉, Jasper Nalbach[1 [0000−0002−2641−1380]], Erika Ábrahám[1 [0000−0002−5647−6134]], and Paul Wagner[1]

RWTH Aachen University, Aachen, Germany
{promies, nalbach, abraham}@cs.rwth-aachen.de

Abstract. To check the satisfiability of (non-linear) real arithmetic formulas, modern satisfiability modulo theories (SMT) solving algorithms like NLSAT depend heavily on *single cell construction*, the task of generalizing a sample point to a connected subset (cell) of $\mathbb{R}^n$, that contains the sample and over which a given set of polynomials is sign-invariant. In this paper, we propose to speed up the computation and simplify the representation of the resulting cell by dynamically extending the considered set of polynomials with further linear polynomials. While this increases the total number of (smaller) cells generated throughout the algorithm, our experiments show that it can pay off when using suitable heuristics due to the interaction with Boolean reasoning.

Keywords: Single Cell Construction · Cylindrical Algebraic Decomposition · Real Algebra · SMT Solving

1 Introduction

Satisfiability checking deals with the problem of deciding whether a first-order logic formula admits a solution. *Satisfiability modulo theories (SMT)* solvers use specialized algorithms to tackle this problem for different theories. While the targeted problems are generally hard (NP-complete for propositional logic, and even undecidable for integer arithmetic), modern SMT solvers are highly efficient and widely used as integrated engines, e.g. for automated deduction [5,17].

In this paper, we focus on the *quantifier-free* fragment of *non-linear real arithmetic (NRA)*, denoted as *QF-NRA*, whose formulas are Boolean combinations of polynomial constraints with rational coefficients and real-valued variables. The *cylindrical algebraic decomposition (CAD)* method [11], which is in general a quantifier elimination procedure for NRA, was the first tractable technique for solving the satisfiability problem for QF-NRA. A CAD partitions the search space of the variables into a finite number of *cells*, such that all polynomials in the input formula are sign-invariant - and thus the input formula is truth-invariant - within each CAD cell. Consequently, we can decide the satisfiability problem by checking one point from each cell. Despite major improvements by, e.g., McCallum [23,24], Lazard [21], and Brown [8], CAD still scales poorly, often due to expensive *resultant* computations.

Inspired by CAD, one of the most successful SMT-solving techniques for QF-NRA is *NLSAT* [19], an instance of the *model constructing satisfiability calculus (MCSAT)* [25] by Jovanic and de Moura. NLSAT extends DPLL+CDCL-style propositional reasoning with a dual approach for the theory. The propositional part consists of deciding truth values for constraints, Boolean propagation, and Boolean conflict resolution. Dually, the theory part decides real values for theory variables, accompanied by theory propagation to assure that theory assignments evaluate constraints *consistently* with the Boolean assignment, and theory conflict resolution. Steps from both parts are interleaved, maintaining consistent partial assignments to guide each other towards a solution.

If all possible values for an unassigned theory variable would contradict the Boolean assignment of some constraint, then the current theory assignment cannot be extended consistently. This situation is resolved using an *explanation* backend, which generalizes the conflict's reason to a lemma (called explanation) that not only excludes the current conflicting assignment but also further similar situations from the future search. For example, given the formula $(x_1^2 + x_2^2 - 1 < 0 \ \vee \ x_2 > 0)$, NLSAT could decide the first constraint to be true, and then assign $x_1 = 2$. Now, no value of x_2 would satisfy $x_1^2 + x_2^2 - 1 < 0$, as it simplifies to $x_2^2 < -3$ under $x_1 = 2$. However, if $x_1^2 + x_2^2 - 1 < 0$ holds, then any value $x_1 > 1$ would require x_2^2 to be negative. Thus we can generalize the value $x_1 = 2$ to $x_1 > 1$ by learning the lemma $(\neg(x_1^2 + x_2^2 - 1 < 0) \ \vee \ x_1 \leq 1)$.

Explanations should be efficiently computable and generalize as strongly as possible. Note that the learnt clause may contain literals not present in the input formula (like $x_1 < 1$ above), thus the generalization technique is crucial for completeness. NLSAT uses CAD techniques to generalize a sample point to a single cell around it. However, NLSAT computes cells *locally* w.r.t. a sample and a subset of the constraints, which offers potential for simpler computations and larger cells. While improvements of Brown and Košta [9], Li and Xia [22], and Nalbach et al. [28] use this potential and avoid certain resultant computations, single cell construction remains a major factor for the running time.

Contributions. The cost of computing a resultant of two polynomials depends on their degree. If one of the polynomials is linear, the resultant is generally cheap to compute. In this paper, we dynamically insert linear polynomials during the cell construction from [28], effectively under-approximating the bounds on the cell. This reduces the effort of the construction by replacing expensive resultant computations with simpler ones. It also affects the quality of the cell for NLSAT: The representation is simpler, but the cell covers less of the search space. This paper follows up on the extended abstract [29], in which we briefly introduced this idea. In particular, our contributions include the following:

- We generalize the ideas from [29] and provide a clear algorithmic formulation.
- We elaborate on the reasons for potential non-termination of NLSAT using the modified single cell construction by providing an example, and adapt the method to guarantee termination.
- We explore several variants of our method, including different ways of constructing additional polynomials and criteria for when to insert them.

– We provide an extensive experimental evaluation of these variants, using our implementation in the SMT solver SMT-RAT [12,30].

Structure. In Section 2 we recall some background, including the levelwise single cell construction from [28], which we adapt for under-approximation in Section 3. We explore variants of our approach in Section 4 and discuss experiments in Section 5. In Section 6 we conclude with an outlook on future work.

Related Work. Some incomplete but fast explanation backends for NLSAT are not based on CAD, but on Fourier-Motzkin variable elimination [18], virtual substitution [2], or interval constraint propagation [20].

The general idea of using linear approximations or abstractions for QF-NRA has been explored before. For example, the `ksmt` calculus [6] transforms each formula into a set of linear arithmetic clauses and a set of non-linear constraints and then incrementally constructs a model, mainly performing linear reasoning. Partial assignments which falsify a non-linear constraint are generalized to conjunctions of linear constraints, using local linear approximations of the non-linear functions. Incremental linearization [10] computes a linear abstraction of the input formula by replacing all multiplications with uninterpreted functions, possibly allowing to derive unsatisfiability by purely linear reasoning. The abstraction is refined incrementally by adding linear arithmetic axioms for the individual multiplications. Neither of these methods is complete, though `ksmt` was shown to be δ-complete.

2 Preliminaries

We assume that the reader has some basic knowledge about multivariate polynomials, logic and SMT solving. For an introduction, we refer to, e.g., [4,13].

Let $\mathbb{N}$, $\mathbb{Q}$, and $\mathbb{R}$ be the sets of natural (incl. 0), rational, respectively real numbers. For $k \in \mathbb{N}$, let $[k] := \{1, \dots, k\}$; for $r \in \mathbb{R}^k$ and $i \in [k]$ let r_i be the ith entry in r, $r_{[i]} := (r_1, \dots, r_i)$, and $r_{[0]} = ()$. For the extent of this paper, we fix some $n \in \mathbb{N} \setminus \{0\}$ and ordered real-valued variables $x_1 \prec \dots \prec x_n$.

Polynomials. For $i \in [n]$, let $\mathbb{Q}[x_1, \dots, x_i]$ be the set of all polynomials in $x_1, \dots, x_i$ with rational coefficients (for $i = 0$, this is $\mathbb{Q}$). We can write any $p \in \mathbb{Q}[x_1, \dots, x_i]$ as a univariate polynomial $p = c_d x_i^d + c_{d-1} x_i^{d-1} + \dots + c_1 x_i + c_0$ in x_i with either $d = 0$ or $c_d \neq 0$, with *degree* $\deg_{x_i}[p] := d$, *coefficients* $\operatorname{coeff}_{x_i}[p] := \{c_0, \dots, c_d\} \subset \mathbb{Q}[x_1, \dots, x_{i-1}]$, and *leading coefficient* $\operatorname{ldcf}_{x_i}[p] := c_d$. Given $r \in \mathbb{R}^i$, we write $p(r)$ for the evaluation $p(r_1, \dots, r_i) \in \mathbb{R}$. Given $r \in \mathbb{R}^{i-1}$ and $r' \in \mathbb{R}$, let $p(r, x_i) \in \mathbb{R}[x_i]$ result from p by substituting $r_1, \dots, r_{j-1}$ for $x_1, \dots, x_{j-1}$, and we write $p(r, r')$ for $p(r_1, \dots, r_{j-1}, r')$.

Real Roots. Let $p \in \mathbb{Q}[x_1, \dots, x_i]$. A *(real) root* of p is a point $r \in \mathbb{R}^i$ so that $p(r) = 0$; the *variety* of p is the set of its roots. The roots of univariate polynomials build the set of *(real) algebraic numbers* $\bar{\mathbb{Q}} := \{r \in \mathbb{R} \mid \exists q \in \mathbb{Q}[x].q(r) = 0\}$.

Given $r \in \bar{\mathbb{Q}}^{i-1}$, one can compute $\text{realRoots}(p, r) := \{r_j \in \mathbb{R} \mid p(r, r_j) = 0\}$, i.e. the *roots of* p *over* r. If $\text{realRoots}(p, r) = \mathbb{R}$, then we say that p is *nullified over* r. Otherwise, $\text{realRoots}(p, r)$ is a finite set of algebraic numbers.

Let $j \in \mathbb{N}$. An *indexed root expression* $\text{root}_{x_i}[p, j] : \mathbb{R}^{i-1} \to \mathbb{R} \cup \{\bot\}$ maps each $r \in \mathbb{R}^{i-1}$ to the j-th root of p over r if it exists, and to $\bot$ otherwise:

$$\text{root}_{x_i}[p, j](r) := \begin{cases} \bot & \text{if realRoots}(p, r) = \mathbb{R} \text{ or } j > |\text{realRoots}(p, r)|, \text{ and else} \\ z_j & \text{where realRoots}(p, r) = \{z_1, \ldots, z_k\}, z_1 < \ldots < z_k. \end{cases}$$

We refer to the polynomial p of an indexed root expression $\xi = \text{root}_{x_i}[p, j]$ by $\xi.p$, and we say that the level of ξ is i. The set of indexed root expressions of level i is $\text{IRE}(i)$. Given $P \subseteq \mathbb{Q}[x_1, \ldots, x_i]$ and $r \in \bar{\mathbb{Q}}^{i-1}$, one can compute the *indexed root expressions defined over* r: $\text{irExp}(P, r) := \{\xi \in \text{IRE}(i) \mid \xi.p \in P \text{ and } \xi(r) \neq \bot\}$.

The *resultant* of two polynomials $p, q \in \mathbb{Q}[x_1, \ldots, x_i]$ w.r.t. x_i is a polynomial $\text{res}_{x_i}[p, q] \in \mathbb{Q}[x_1, \ldots, x_{i-1}]$, such that for all $r \in \mathbb{R}^{i-1}$ it holds: if there is $r' \in \mathbb{R}$ with $p(r, r') = 0 = q(r, r')$, then $\text{res}_{x_i}[p, q](r) = 0$. The *discriminant* of p is $\text{disc}_{x_i}[p] := \text{res}_{x_i}[p, p']$, where p' is the derivative of p w.r.t x_i.

Formulas. A formula of the quantifier-free fragment of (non-linear) real arithmetic (QF-NRA) is a Boolean combination of (polynomial) *constraints* of the form $p \sim 0$, with $p \in \mathbb{Q}[x_1, \ldots, x_n]$ and $\sim \in \{<, >, =, \leq, \geq, \neq\}$. An *extended constraint* has the form $x_i \sim \xi$, where $\xi \in \text{IRE}(i)$ is an indexed root expression.

Cells. A *cell* is a non-empty connected set $S \subseteq \mathbb{R}^i$ for some $i \in [n]$. We call S (semi-)*algebraic* if it is the solution set of a conjunction of constraints and extended constraints. We call p *sign-invariant* over S, if the sign of $p(r)$ is the same for all points $r \in S$ (i.e. $\forall r \in S.\ p(r) \sim 0$ for a fixed $\sim \in \{<, =, >\}$). We call S sign-invariant for $P \subset \mathbb{Q}[x_1, \ldots, x_i]$, if all $p \in P$ are sign-invariant over S.

2.1 Levelwise Single Cell Construction

Given a constraint set C and an assignment $s \in \mathbb{R}^i$, if all extensions of s evaluate some constraints from C to false, then we say that s is *inconsistent* with C. In NLSAT, if the theory assignment is inconsistent with the constraints C defined to be true by the Boolean assignment, then we generalize s to a cell $S \subseteq \mathbb{R}^i$, whose points are all inconsistent with C. To do so, we derive a set $P \subset \mathbb{Q}[x_1, \ldots, x_i]$ of projection polynomials, such that $s \in S$ and the sign-invariance of P over S assures that all points in S are inconsistent with C. The learned explanation is then $(\neg C \vee \neg \varphi_S)$, where φ_S is a conjunction of extended constraints defining S.

Definition 1. *Given $i \in [n]$, $P \subset \mathbb{Q}[x_1, \ldots, x_i]$ and $s \in \mathbb{R}^i$, the problem of* single cell construction *(SCC) is to compute a description of an algebraic cell $S \subseteq \mathbb{R}^i$ so that $s \in S$ and all $p \in P$ are sign-invariant over S.*

We now recall the *levelwise* SCC approach [28] by Nalbach et al., which is summarized in Algorithm 1, and which we will modify in Section 3.

Algorithm 1: `levelwise-scc`(P, s)

Input : A finite $P \subseteq \mathbb{Q}[x_1, \ldots, x_i]$, and $s \in \mathbb{R}^i$ (with $i \in [n]$).
Output: A description $(I_1 \wedge \ldots \wedge I_i)$ of a sign-invariant cell for P containing s

1 **for** $j = i, \ldots, 1$ **do**
2 $\quad P_j := P \cap (\mathbb{Q}[x_1, \ldots, x_j] \setminus \mathbb{Q}[x_1, \ldots, x_{j-1}])$
3 $\quad$ **if** realRoots$(p, s_{[j-1]}) = \mathbb{R}$ for some $p \in P_j$ **then return** *FAIL*
4 $\quad \{\xi_1, \ldots, \xi_k\} :=$ irExp$(P_j, s_{[j-1]})$ s.t. $\xi_1(s_{[j-1]}) \leq \ldots \leq \xi_k(s_{[j-1]})$
5 $\quad$ **if** $k = 0$ **then** $I_j := (true)$
6 $\quad$ **else if** $s_j = \xi_\ell(s_{[j-1]})$ for some ℓ **then** $I_j := (x_j = \xi_\ell)$
7 $\quad$ **else if** $s_j > \xi_k(s_{[j-1]})$ **then** $I_j := (\xi_k < x_j)$
8 $\quad$ **else if** $s_j < \xi_1(s_{[j-1]})$ **then** $I_j := (x_j < \xi_1)$
9 $\quad$ **else** $I_j := (\xi_\ell < x_j \wedge x_j < \xi_{\ell+1})$ for the ℓ with $\xi_\ell(s_{[j-1]}) < s_j < \xi_{\ell+1}(s_{[j-1]})$
10 $\quad$ **add** to P discriminants and coefficients ensuring delineability
11 $\quad$ **add** to P resultants ensuring sign-invariance
12 **return** $(I_1 \wedge \ldots \wedge I_i)$

Let $i \in [n]$, $s \in \mathbb{R}^i$, $P \subset \mathbb{Q}[x_1, \ldots, x_i]$, and for $j \in [i]$ let P_j be the polynomials from P with largest variable x_j (i.e. those containing x_j, but not $x_{j+1}, \ldots, x_i$). For each dimension $j = i, \ldots, 1$, the algorithm determines a *symbolic interval* I_j of the form $(x_j = \xi)$, $(x_j < \xi)$, $(x_j > \xi)$, or $(\xi < x_j \wedge x_j < \xi')$ for some $\xi, \xi' \in \text{IRE}(j)$, bounding the value of x_j w.r.t. the lower variables $x_1, \ldots, x_{j-1}$. For all $r \in \mathbb{R}^{j-1}$ with $\xi(r) \neq \bot \neq \xi'(r)$, I_j defines a *concrete interval* $I_j(r) \subseteq \mathbb{R}$ which is $\{\xi(r)\}$, $(-\infty, \xi(r))$, $(\xi(r), \infty)$, or $(\xi(r), \xi'(r))$, respectively.

The final cell described by $I_1 \wedge \ldots \wedge I_i$ is *locally cylindrical*, i.e., I_1 defines a concrete interval $S_1 \subseteq \mathbb{R}$, and for $j = 2, \ldots, i$, the root expressions in I_j are defined everywhere over S_{j-1}, and they specify the cell

$$S_j = \{(r, r') \in \mathbb{R}^j \mid r \in S_{j-1} \wedge r' \in I_j(r)\}.$$

To determine I_j, we assign $x_1, \ldots, x_{j-1}$ to the underlying sample $s_{[j-1]}$, and compute realRoots$(p, s_{[j-1]})$ for all $p \in P_j$. These roots witness the indexed root expressions in Line 4. The greatest root below (or equal to) s_j and the smallest root above (or equal to) s_j provide the interval boundaries (if they do not exist, $-\infty$ and ∞ are used). Thus, the polynomials in P_j are sign-invariant over $\{s_{[j-1]}\} \times I_j(s_{[j-1]})$.

The idea is now that the to-be-constructed underlying cell $S_{j-1} \subseteq \mathbb{R}^{j-1}$ will be a neighbourhood around $s_{[j-1]}$ over which the root expressions in I_j define total continuous functions such that S_j is a sign-invariant cell for P_j containing $s_{[j]}$. To obtain an underlying cell with the desired properties, the concepts of delineability and order-invariance (a strengthening of sign-invariance) are used:

Definition 2 (Delineability [11]). *Let $j \in [n-1]$ and $S \subseteq \mathbb{R}^j$ be a cell. A non-zero polynomial $p \in \mathbb{Q}[x_1, \ldots, x_{j+1}]$ is* delineable over S *if there exist $k \geq 0$ continuous functions $\theta_1, \ldots, \theta_k : S \to \mathbb{R}$ and constants $m_1, \ldots m_k \in \mathbb{N}$ such that for all $r \in S$ holds $\theta_1(r) < \ldots < \theta_k(r)$,* realRoots$(p, r) = \{\theta_1(r), \ldots, \theta_k(r)\}$*, and for all $\ell = 1, \ldots, k$ the multiplicity of the root $\theta_\ell(r)$ in $p(r, x_{j+1})$ is m_ℓ.*

Definition 3 (Order-invariance [23]). *The* order of $p \in \mathbb{Q}[x_1, \ldots, x_j]$ at $r \in \mathbb{R}^j$*, denoted* $ord(p, r)$*, is the minimum* k *so that some partial derivative of* p *of total order* k *does not evaluate to* 0 *at* r *(or* ∞*, if all evaluate to* 0*). We call* p order-invariant *on* $R \subseteq \mathbb{R}^j$ *if* $ord(p, r) = ord(p, r')$ *for all* $r, r' \in R$*.*

The indexed root expressions of p determined in Line 4 witness the θ functions. That these are well-defined continuous functions is assured by the delineability of p. The method uses the fact that p is delineable over the underlying cell S_{j-1} if $\mathrm{disc}_{x_j}[p]$ is order-invariant on S_{j-1} and $\mathrm{ldcf}_{x_j}[p]$ is sign-invariant over S_{j-1}; thus it adds these polynomials to P and ensures their properties on the next level, thereby restricting I_{j-1} and the levels below.

The method still has to ensure that no root function crosses the cell boundaries (the root expressions in I_j) over S_{j-1}, because this would imply a sign change of some polynomial within S_j. For this purpose, we use that for any two $\xi, \xi' \in \mathrm{irExp}(P_j, s_{[j-1]})$ and $\sim \in \{<, =\}$ with $\xi(s_{[j-1]}) \sim \xi'(s_{[j-1]})$, it holds: If $\mathrm{res}_{x_j}[\xi.p, \xi'.p]$ is order-invariant on S_{j-1}, then $\xi(r) \sim \xi'(r)$ for all $r \in S_{j-1}$.

Since only intersections of roots with the cell boundaries are relevant, it suffices to maintain a *partial* ordering of the root functions, ensured by certain resultants (Line 11). For example, if $\xi_1, \ldots, \xi_k$ are as in Algorithm 1 and $I_j = (\xi_\ell < x_j \wedge x_j < \xi_{\ell+1})$, then we could add $\{\mathrm{res}_{x_j}[\xi_{\ell'}.p, \xi_\ell.p] \mid \ell' < \ell\} \cup \{\mathrm{res}_{x_j}[\xi_{\ell+1}.p, \xi_{\ell'}.p] \mid \ell + 1 < \ell'\}$, ensuring that $\xi_1, \ldots, \xi_{\ell-1}$ stay below (or equal to) ξ_ℓ and $\xi_{\ell+2}, \ldots, \xi_k$ stay above (or equal to) $\xi_{\ell+1}$. By exploiting transitivity, other partial orderings and thus other sets of resultants are also viable; this is a heuristic decision. However, the resultant $\mathrm{res}_{x_i}[\xi_\ell.p, \xi_{\ell+1}.p]$ of the bounds is always added to ensure connectedness of S_j.

Note that this method fails if any of the encountered polynomials is nullified on the underlying sample, because then delineability cannot be ensured in the same way. The method can detect this and return "FAIL", and a different, complete approach is used instead. To further ensure that no polynomial $p \in P_j$ is nullified over any other point in S_{j-1}, some $c \in \mathrm{coeff}_{x_j}[p]$ with $c(s_{[j-1]}) \neq 0$ is also added to P in Line 10. After adding all required polynomials to P, if $j > 1$, then the method proceeds with the construction of I_{j-1} in the same way.

Example 1. Figure 1 illustrates an example with a given sample $s \in \mathbb{R}^2$ and polynomials $P = \{p_1, p_2, p_3\} \subset \mathbb{Q}[x_1, x_2]$. The line labelled with p_1 indicates the variety of p_1 i.e. those points $r \in \mathbb{R}^2$ with $p_1(r) = 0$, and similarly for p_2, p_3.

We start at level 2, where $P_2 = P$. At $x_1 = s_1$, there is one root of p_2 below s_2 and one root of each polynomial above s_2. The roots closest to s_2 define the symbolic interval $I_2 := (\xi_1 < x_2 \wedge x_2 < \xi_2)$ (Figure 1a). To ensure correctness of this interval for all values of x_1 in the underlying cell (to be computed at level 1), the discriminants and leading coefficients of p_1, p_2, p_3 are added to P (dash-dotted lines in Figure 1b). Moreover, adding $\mathrm{res}_{x_2}[p_3, p_1]$ and $\mathrm{res}_{x_2}[p_3, p_2]$ (dashed lines in Figure 1b) ensures that none of the root functions cross the upper interval bound defined by p_1 over I_1. Note that the crossing of ξ_3 and ξ_4 is irrelevant, and the corresponding resultant of p_1 and p_2 is thus avoided. On level 1, we isolate the roots of these polynomials and use the closest to s_1 as interval boundaries, resulting in the shaded cell (Figure 1c).

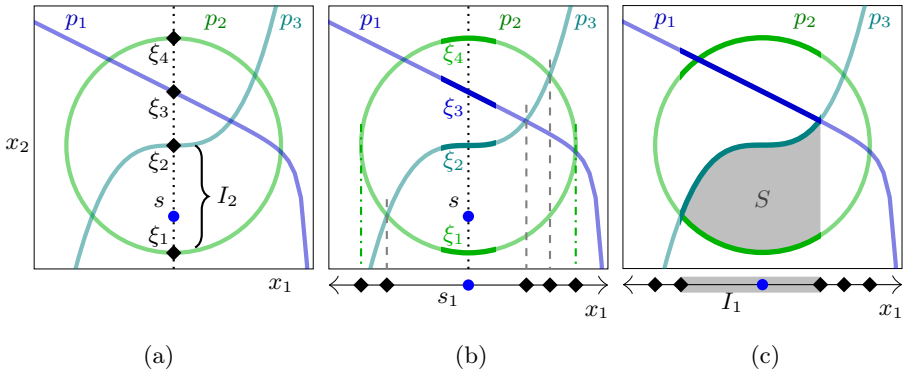

Fig. 1: Illustration of the levelwise construction described in Example 1.

3 Adding Polynomials to Avoid Expensive Resultants

The running time of SCC is dominated by discriminant and resultant computations. Given $p, q \in \mathbb{Q}[x_1, \ldots, x_n]$ with $d_j = \deg_{x_j}[p]$, $e_j = \deg_{x_j}[q]$ for all $j \in [n]$, the resultant of p and q requires $\mathcal{O}(d_n e_n)$ polynomial multiplications and, in the worst case, its degree w.r.t. any x_j is $d_n e_j + d_j e_n$. Given $P \subset \mathbb{Q}[x_1, \ldots, x_n]$ with maximal degree d in any variable, the degree and time complexity of resultants during SCC grows doubly exponential in worst-case (d^{2^n}), as resultants of resultants are computed iteratively. The levelwise method already mitigates the effort for computing resultants, e.g. it avoids involving polynomials of high degree. However, some cannot be avoided, e.g. polynomials defining the bounds of an interval are always included in some resultant computations.

Our approach is as follows: If a high-degree polynomial p would define a bound of a symbolic interval I_j, then we add a new *linear* polynomial $p_* = (x_j - c)$ to P, whose root $c \in \mathbb{Q}$ lies *strictly* between that bound and the sample s_j. Using this as a new, under-approximating bound for I_j allows replacing expensive resultants of p and some $q \in P$ by resultants of p_* with q, which are simply computed by substituting c for x_j in q, and their degree is bounded by the one of q. The choice of c ensures that (1) the resulting cell still contains the sample, (2) all other roots remain outside the cell, and (3) the underlying levels still generalize to some larger cell. Towards the latter, c should not be equal to any polynomial's root, as the cell then would only generalize to a section $I_{j-1} = (x_{j-1} = \xi)$ on the level below as the resultant of that polynomial with p_* would have a root at $s_{[j-1]}$.

Example 2. In Example 1 (depicted in Figure 2a), the levelwise method cannot avoid the resultants $\mathrm{res}_{x_2}[p_2, p_3]$ and $\mathrm{res}_{x_2}[p_3, p_1]$, which are expensive if p_1, p_2 and p_2 have high degree. Adding a linear polynomial p_* with a root ξ_* between s and ξ_2, lets us use ξ_* as upper bound of I_2, and it suffices to compute only the cheap resultants of p_* with p_1, p_2, p_3 (like shown in Figure 2b). Figure 2c shows the resulting cell S' (pink, shaded) and the original cell S (hatched).

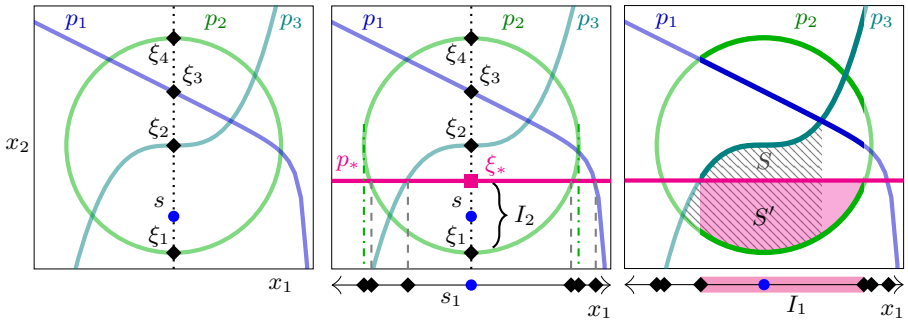

Fig. 2: Approximated cell using a new linear polynomial

Importantly, p_1, p_2 and p_3 are sign-invariant over S'. Note that we might also under-approximate the lower bound of I_2, or the bounds of I_1, leading to an even smaller cell, but also reducing computations.

We generalize this approach and modify `levelwise-scc`, so that on each level, it can dynamically extend the working set P with arbitrary polynomials, resulting in our new method `apx-scc` shown in Algorithm 2, which adds Line 4. The method `apx-criteria` decides whether adding new polynomials is beneficial, by checking e.g. whether the symbolic interval I_j would be defined by a polynomial with high degree. If the criteria are fulfilled, `apx-polys` computes a set of new polynomials (called *auxiliary* polynomials) that are added to P. The auxiliary polynomials have roots at favourable positions, admitting an easier set of resultants to be computed. We discuss the different approaches for implementing `apx-criteria` and `apx-polys` in Section 4.

Adding auxiliary polynomials makes the maximal possible sign-invariant cell around the given sample point smaller, hence we compute some kind of *under-approximation*. However, as shown in Figure 2c, the cell computed by `apx-scc` is

Algorithm 2: `apx-scc`(P, s)

Input : A finite $P \subseteq \mathbb{Q}[x_1, \ldots, x_i]$ and $s \in \mathbb{R}^i$ (with $i \in [n]$).
Output: A description $(I_1, \ldots, I_i)$ of a sign-invariant cell for P containing s

1 **for** $j = i, \ldots, 1$ **do**
2 $P_j := P \cap (\mathbb{Q}[x_1, \ldots, x_j] \setminus \mathbb{Q}[x_1, \ldots, x_{j-1}])$
3 **if** realRoots$(p, s_{[j-1]}) = \mathbb{R}$ for some $p \in P_j$ **then return** *FAIL*
4 **if** `apx-criteria`$(P_j, s_{[j]})$ **then** $P := P \cup$ `apx-polys`$(P_j, s_{[j]})$
5 **compute** I_j as before (Lines 2-9 in Algorithm 1)
6 **add** to P polynomials ensuring delineability and sign-invariance (Lines 10-11 in Algorithm 1)
7 **return** $(I_1, \ldots, I_i)$

not necessarily a subset of the cell computed by `levelwise-scc`, as strengthening the bounds of I_j might allow weakening some bound of I_{j-1}. In any case, both the original and the approximated cell (S and S') are subsets of the maximal sign-invariant cell S_{max} for P. While $S = S_{max}$ may hold, it always holds $S' \subsetneq S_{max}$.

Our modification has two main benefits for the usage in NLSAT: (1) Avoiding expensive resultant computations means that explanations can be computed much faster; and (2) during later computations, NLSAT needs to isolate the roots of the cell boundaries over further sample points, for checking whether a given sample lies in the excluded cell - the effort for these computations may be drastically reduced by polynomials of lower degree (or even degree 1). On the other hand, the under-approximated cells may lead to more cells generated throughout the search, and even lead to non-termination, as we will see in Section 3.1.

It is important to note that our modification does not eliminate the strong degree growth entirely, because the discriminants needed for delineability are also resultants. Moreover, for sections $I_j = (x_j = \xi)$ we cannot compute meaningful approximations, forcing us to fall back to the default method.

Theorem 1 (Correctness). *Let $P \subset \mathbb{Q}[x_1, \ldots, x_i]$ be finite and $s \in \mathbb{R}^i$. If `apx-scc`(P, s) yields the cell $S \subseteq \mathbb{R}^i$, then $s \in S$ and P is sign-invariant over S.*

Proof. The idea is that the original method could produce the same cell, when given an appropriately modified input. For each $j \in [i]$, let $Q_j \subset \mathbb{Q}[x_1, \ldots, x_j]$ be the set of polynomials added by `apx-scc` on level j, and let $Q := Q_1 \cup \ldots \cup Q_i$. Consider the cell $S' \subseteq \mathbb{R}^i$ computed by `levelwise-scc`$(P \cup Q, s)$. As that method is correct, $s \in S'$ and $P \cup Q$ is sign-invariant over S'. By definition of sign-invariance, this implies that P is sign-invariant over S'.

We show $S = S'$ to complete the proof: The polynomials in Q_j do not impact the computations of the levels $j+1, \ldots, i$, because $Q_j \cap P_k = \emptyset$ for each $k \neq j$. Thus, both `levelwise-scc` and `apx-scc` compute each level j only based on $P \cup \bigcup_{k=j}^{i} Q_k$. As their computations do not differ (apart from adding Q_j), they compute exactly the same intervals and projections. □

3.1 Incompleteness

Like `levelwise-scc`, our approach fails in the case of nullification and is thus incomplete as a stand-alone procedure. However, one can detect nullification and resort to a complete construction for that cell. More critically, the termination of NLSAT is no longer guaranteed when using `apx-scc` for explanations.

Example 3 (Non-Termination). We continue Example 2. After excluding the approximated cell from the search, NLSAT chooses a new value for y. This value can lie between the auxiliary cell boundary and the root of p_3, leading to the sample s' as shown in Figure 3a. Since s' and s are in the same maximal sign-invariant cell for p_1, p_2, p_3, it leads to a conflict with the same constraints, and thus `apx-scc` is called with the same polynomials. When computing the new explanation, another auxiliary boundary is introduced between s' and p_3, and

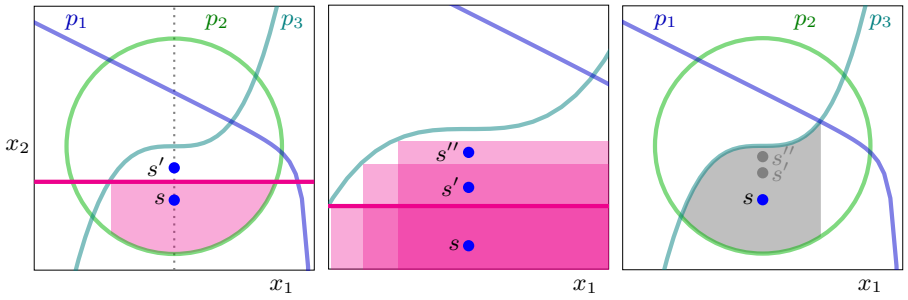

Fig. 3: Non-termination of NLSAT with under-approximating cells, as described in Example 3. Figure (b) zooms in on the area between p_3 and the approximation.

this behaviour repeats, leading to a sample s'' as in Figure 3b. This can repeat indefinitely, and NLSAT will run into the same conflict over and over, without ever covering the entire search space. This behaviour cannot occur with the original construction, as is illustrated in Figure 3c.

Note that NLSAT does not always run into this situation. It might also choose samples that are further away, thus escaping the conflict. Thanks to the Boolean structure of the formula, the remaining search space might still be covered by approximated cells for other conflicts, which involve different sets of polynomials.

We can also express this with formal terms from [19]: the termination of NLSAT relies on the fact that a *finite basis* explanation function is used. That is, for every input formula φ, there is a *finite* set $B(\varphi) \subset \mathbb{Q}[x_1, \ldots, x_n]$ such that in all possible runs of NLSAT, all explanations use only polynomials from B. Since there are runs of NLSAT for which our explanation produces *infinitely* many different literals, `apx-scc` does not yield a *finite basis* explanation function.

Retaining Completeness. If we ensure that in every run of NLSAT, we add only finitely many auxiliary polynomials, then termination is guaranteed again. The reasoning is that there will be a point after which the output of `apx-scc` is always equal to the original levelwise construction `levelwise-scc`. Since this is a finite-basis explanation function, termination will be guaranteed.

Lemma 1 (Termination). *A run of NLSAT using* `apx-scc` *terminates iff* `apx-criteria`$(P_j, s_{[j]})$ *returns* `true` *only finitely many times during that run.*

To make use of this lemma, we provide additional information to `apx-criteria`, like the number n_{cells} of so far approximated cells. Then, `apx-criteria` could return `false` whenever n_{cells} exceeds some fixed threshold, fulfilling the condition from Lemma 1 and thus implying termination.

However, the optimal threshold will vary depending on the input formula. A more flexible approach is to gradually strengthen the criteria as the number of

approximated cells increases. For example, we only add a polynomial if a cell boundary is defined by some $p \in P$ with $\deg_{x_p}[j] \geq c \cdot n_{cells} + d$, where $c, d \in \mathbb{Q}_{\geq 0}$.

This avoids the behaviour from Example 3, since $c \cdot n_{cells} + d$ eventually exceeds the degree of the involved polynomials, but other cells with more expensive resultants are still approximated. Importantly, this only ensures termination because the added polynomials are linear; otherwise, the degrees of polynomials derived in the construction could grow indefinitely, always fulfilling the criterion.

The guarantee may also be given by other criteria, e.g. using individual counters for the involved polynomials.

4 Variants

We now present several instantiations for `apx-polys` and `apx-criteria`. The first three methods approximate a cell boundary: Given the sample $s_{[j]}$ and some $p \in P$, whose root ξ would be a bound of I_j, we construct some p_* with a root $c \in \mathbb{Q}$ between s_j and $b := \xi(s_{[j-1]})$.

Simple Approach. At the beginning of Section 3, we already introduced the idea of adding polynomials $x_j - c$ defining a constant bound c on x_j. We now elaborate on the choice of c. Although choosing c close to b restricts I_j less, it may shrink the underlying cell depending on the shape of p's variety. This can be observed in Figure 3: the closer the approximate bound is to the actual bound, the smaller becomes I_1 (the interval for x_1). It is thus not immediately clear what to choose. More importantly, this approach can produce numbers $c = {}^{num}/_{den}$ with large bit size $log(num)+log(den)$, causing significant overhead in operations like substituting c into high-degree polynomials. Therefore, we choose c with a minimal bit size using a method based on the *Stern-Brocot tree* [7,31].

The following two approaches try to provide better approximations of the cell boundary, hoping to increase the cell's quality for NLSAT.

Taylor. We want to construct p_* so that its gradient at its root $(s_{[j-1]}, c)$ is equal to the gradient of p at its root $r := (s_{[j-1]}, b)$:

$$p_* \quad = \quad \frac{\partial p}{\partial x_j}(r) \cdot (x_j - c) + \sum\nolimits_{k \in [j-1]} \Big(\frac{\partial p}{\partial x_k}(r) \cdot (x_k - s_k) \Big)$$

This is a slight modification of the *first-order Taylor expansion* of p at r, the difference being that the constant term $p(r)$ is left out as it is always zero, and that $(x_j - c)$ is used instead of $(x_j - r_j)$ in the first term. This ensures that p_* has its root at $(s_{[j-1]}, c)$ instead of r. Now, clearly

$$p_*(s_{[j-1]}, c) = 0 \text{ and } \frac{\partial p_*}{\partial x_k}(s_{[j-1]}, c) = \frac{\partial p}{\partial x_k}(r) \text{ for all } k \in [j].$$

The idea is that the root functions of p and p_* will behave similarly around $s_{[j-1]}$, as illustrated in Figure 4a. The dashed line is the root of the tangent to p_3 at (s_{j-1}, b), which is then shifted to pass through c.

Unfortunately, if some sample coordinate s_k is *irrational*, then (1) we cannot use the term $(x_k - s_k)$, since we require *rational* coefficients, and (2) the gradients $\partial p / \partial x_k(r)$ are harder to compute and might also be irrational. We tackle (1) by omitting the summand corresponding to x_k for the k where s_k is irrational, and (2) by finding rational approximations of the gradients: Each (irrational) algebraic number can be isolated using an open interval containing a single root of its defining polynomial; this interval can be refined arbitrarily by bisection. For $k \in [j]$, let $g_k \approx \partial p / \partial x_k(r)$ be a rational approximation, then we get

$$p_* \quad = \quad g_j \cdot (x_j - c) + \sum_{k \in [j-1] \text{ s.t. } s_k \in \mathbb{Q}} \Big(g_k \cdot (x_k - s_k) \Big)$$

Both mitigations harm the quality of the approximation, as some gradients are only approximately equal, or not counted in at all.

Piecewise Linear. Instead of approximating the boundary at a single point, we can use piecewise linear interpolation. For this purpose, we determine an interval $D \subseteq \mathbb{R}$ around s_{j-1} so that p is delineable over

$$\{s_{[j-2]}\} \times D = \{(s_{[j-2]}, s') \in \mathbb{R}^{j-1} \mid s' \in D\} \subseteq \mathbb{R}^{j-1},$$

i.e. $\operatorname{disc}_{x_j}[p]$ and certain coefficients of p are sign-invariant over that set. We now know that ξ is a total continuous function over $\{s_{[j-2]}\} \times D$, which is needed for deriving a meaningful interpolation.

We then choose $k \in \mathbb{N}$ support points $d_1 < \ldots < d_k$ from D such that $s_{j-1} \in \{d_1, \ldots, d_k\}$, and for each $\ell \in [k]$ we compute the value $\xi((s_{[j-2]}, d_\ell))$ of the root function at that support point (in practice, we only isolate it in a rational interval). Each of those values is under-approximated by choosing a value $d'_\ell \in \mathbb{Q}$ "close" to the boundary $\xi((s_{[j-2]}, d_\ell))$ such that $(s_{[j-2]}, d_\ell, d'_\ell) \in \mathbb{R}^j$ is inside the cell. We require that one support point d_ℓ is equal to s_{j-1}, because this guarantees that $(s_{[j-2]}, d_\ell, d'_\ell)$ (for this particular ℓ) is between the bound and the sample point.

The approximate bound then consists of $k-1$ pieces, which connect the points $\{(s_{[j-2]}, d_\ell, d'_\ell) \in \mathbb{R}^j \mid \ell \in [k]\}$ and which are defined by the roots of

$$p_*^{(\ell)} := (d_{\ell+1} - d_\ell)(x_j - d'_\ell) - (d'_{\ell+1} - d'_\ell)(x_{j-1} - d_\ell), \quad \ell \in [k-1].$$

That is, p_* is a piecewise function so that for $r \in \mathbb{R}^j$ holds $p_*(r) = p_*^{(\ell)}(r)$ if $d_\ell \leq r \leq d_{\ell+1}$ (or $\ell = k$ if $d_k \leq r$). We can encode this function by a minimum over maximum of linear functions as described in [33,32], and thus derive an QF-NRA formula. Further, we can adapt the subsequent cell construction to handle such compound interval bounds by using the techniques described in [26]: We compute the resultant of each $p_*^{(\ell)}$ with the polynomials below or above the interval and filter out roots of the resultants witnessing spurious intersections. This approach is illustrated in Figure 4b where the intersections with the dashed lines are filtered out.

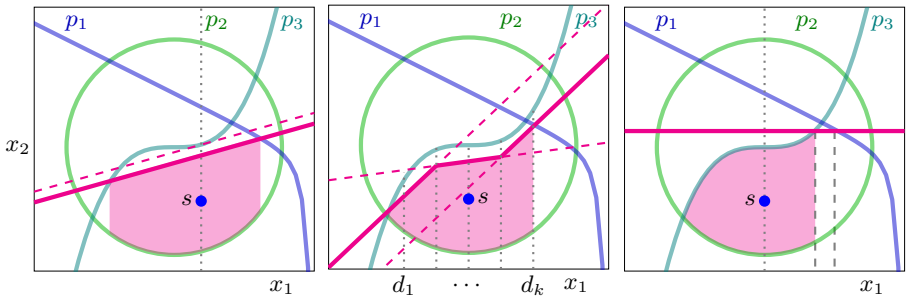

Fig. 4: Variants of the modified construction: (a) using Taylor expansion, (b) using piecewise linear bounds, and (c) adding roots outside the cell.

There are two cases where we cannot apply this approach (and thus need to apply e.g. the simple approach): (1) Like the Taylor approach, if s_{j-1} is irrational, we cannot construct p_*, because $s_{j-1} = d_\ell$ for some $\ell \in [k]$, and thus we cannot use the term $x_{j-1} - d_\ell$. (2) If $D = \{s_{j-1}\}$, i.e. D contains only one point, we cannot choose more than one support point. However, at least two points are needed to construct one piece of the approximate boundary. This case happens e.g. when the discriminant of p (the original boundary-defining polynomial) has a root ξ_{disc} at $s_{[j-1]}$, which means that the next level will collapse to a section $I_{j-1} = (x_{j-1} = \xi_{disc})$ anyway and the behaviour of p_* around the sample does not matter as much.

Roots Outside the Cell. We could also introduce polynomials p_* with a root function ξ_* between the cell boundary ξ and some other root ξ' outside the cell. Instead of $\mathrm{res}_{x_j}[\xi.p, \xi'.p]$, we compute $\mathrm{res}_{x_j}[\xi.p, p_*]$ and $\mathrm{res}_{x_j}[p_*, \xi'.p]$ and then use transitivity, which may simplify the projection. While this has the advantage of leaving the top level I_i unchanged, the underlying cell may still be restricted by the new resultants. This is illustrated in Figure 4c.

Approximation Criteria. In addition to varying the way *how* auxiliary polynomials are computed, we can also adapt *when* they are computed, by using different instantiations of `apx-criteria`.

As our goal is to avoid expensive resultant computations, we are interested in quantities which influence the resultant complexity on the current level. A simple (yet effective in practice) criterion depends on the degree of the boundary-defining polynomial $p \in \mathbb{Q}[x_1, \ldots, x_i]$: the bound is approximated if $\deg_{x_i}[p]$ exceeds a fixed threshold. This has some limitations: (1) It can happen that `levelwise-scc` would not compute any (expensive) resultants with p, but this criterion would still advise inserting auxiliary polynomials. While this defeats the purpose of avoiding expensive resultants, it is not completely useless as it

still simplifies the cell description. (2) Different thresholds will be optimal for different kinds of input problems, and it is not easy to guess a good one a priori.

Addressing the first issue, one can additionally check whether a resultant of p with another nonlinear polynomial would be computed. However, this did not improve the performance in our preliminary experiments. The second issue is (partially) addressed with the dynamic termination criterion from Section 3.1, where the threshold for the degree grows with the number of approximated cells.

Instead of the degree $\deg_{x_i}[p]$, we might also consider similar measures, e.g. the *sum-of-total-degrees* of p's monomials, which has been used for CAD projection orderings [15].

Transfer to CAlC. Similar to NLSAT, the *cylindrical algebraic covering (CAlC)* method [1,27] tries to extend a partial assignment $s \in \mathbb{R}^i$ to a full model. It uses the same theoretical framework as the levelwise construction to derive symbolic intervals to be excluded from the search. In particular, it also ensures that certain root functions ξ, ξ' do not intersect by making $\text{res}_{x_{i+1}}[\xi.p, \xi'.p]$ order-invariant. We can apply our technique and dynamically introduce a polynomial with a root between ξ, ξ', replacing an expensive resultant by two simpler ones, but shrinking the underlying cell. This will introduce similar issues regarding termination, which can be solved in similar ways as presented above for SCC.

5 Experiments

We implemented several variants of our approach in the SMT-RAT solver [12,30], allowing us to use its existing implementations of the levelwise SCC and NLSAT. SMT-RAT also uses explanation backends based on the Fourier-Motzkin variable elimination (FM) [18], interval constraint propagation (ICP) [20], and virtual substitution (VS) [2], which are fast, but may fail to provide an explanation, especially for polynomials of degree 3 or higher. The backends are called sequentially (FM, ICP, VS, SCC), so that the single cell construction is only needed when all other backends fail. We compare the following variants for SCC:

`Baseline`: The original levelwise SCC.

`Simple-j`: For `j` $\in \{3, 4, 5, 6\}$, cell bounds defined by polynomials of degree `j` or higher are approximated by simple polynomials of the form $x_j - c$. Termination is ensured by limiting the number of approximated cells to 50.

`Simple-*`: a virtual best portfolio of the `Simple-j` variants.

`Dynamic`: Uses the dynamic termination criterion from Section 3.1. A cell bound is approximated by a simple polynomial of the form $x_j - c$, if the defining polynomial has degree higher than or equal to $1/5 \cdot n_{cells} + 3$, where n_{cells} is the number of so far approximated cells.

`Taylor`: Like `Dynamic`, but using Taylor approximations as in Section 4.

`PWL-j`: For `j` $\in \{2, 4, 6\}$, like `Dynamic`, but using piecewise linear approximations with `j` pieces, as presented in Section 4.

`PWL-*`: a virtual best portfolio of the `PWL-j` variants.

`Outside`: If the lower (resp. upper) cell bound and another root function below (resp. above) that bound are defined by two non-linear polynomials, one of which fulfils the dynamic criterion, then we add a polynomial (of the form $x_j - c$) whose root lies outside the cell, between the bound and the other root, as explained in Section 4.

Moreover, we also compare to the state-of-the-art `cvc5` [14] solver (version 1.2.1), which uses incremental linearization and cylindrical algebraic coverings.

We used the QF-NRA benchmark set from SMT-LIB [3], but only consider the 1684 instances where SMT-RAT calls SCC at least once, since there is no difference between the variants on the other instances. The tests were conducted on identical Intel®Xeon®8468 Sapphire CPUs with 2.1 GHz per core, with a time limit of 1 minute and memory limit of 4GB per instance.

The results are summarized in Figure 5. While `cvc5` usually solves more instances than SMT-RAT on the entire QF-NRA set, already the `Baseline` solver outperforms `cvc5` on our restricted benchmark set.

Simple Approximations. Already the `Simple-j` variants solve around 45 instances more than `Baseline`. However, these variants excel (partly) on different instances: the virtual portfolio `Simple-*` solves at least 18 instances more than each `Simple-j` variant.

`Dynamic` almost matches this performance, solving 58 instances more than `Baseline`, and it is in fact the best (non-portfolio) variant in our tests. Interestingly, the differences in solved satisfiable instances and unsatisfiable instances are almost equal (30 more satisfiable, 28 more unsatisfiable). This suggests that our explanations can not only help NLSAT find a model more quickly, but the Boolean structure of the unsatisfiable problems often still allows to cover the search space with the under-approximated cells.

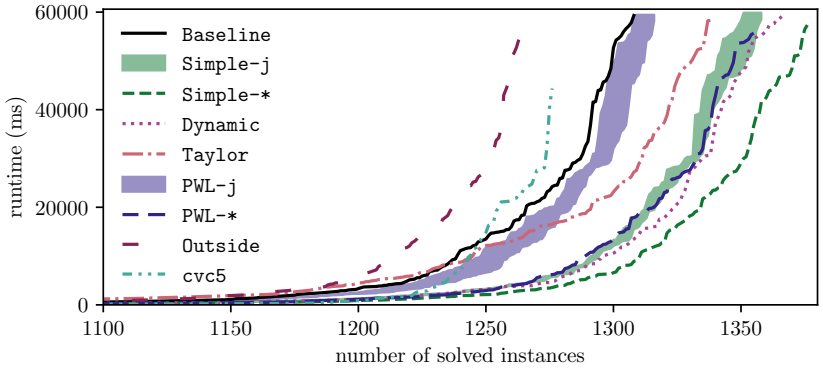

Fig. 5: Performance profile. The purple area covers all `PWL-j` variants, the green area covers all `Simple-j` variants.

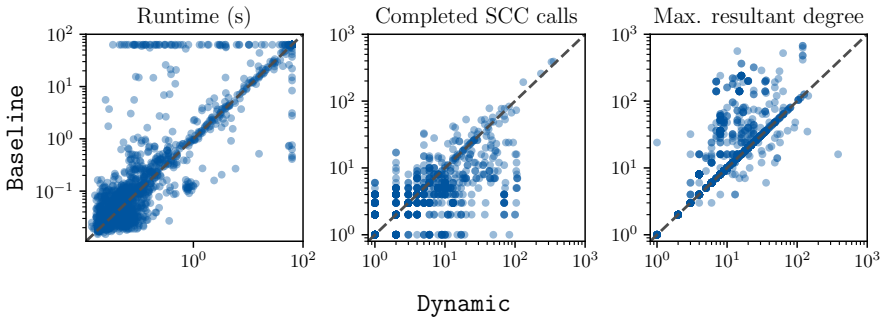

Fig. 6: Scatter plots comparing `Dynamic` with `Baseline`

Figure 6 shows a more detailed comparison between `Dynamic` and `Baseline`. Looking at the running times, `Dynamic` can solve many instances within one second, for which `Baseline` needs multiple seconds or even times out. On the other hand, there are very few instances on which `Dynamic` is significantly slower, even though the number of SCC calls is usually much higher. This is expected: the approximated cells can be computed faster, but they cover less of the search space. Moreover, our approach often significantly reduces the maximum degree of any computed resultant. However, we did not find a clear reduction of degrees of discriminants and coefficients.

Other Variants The `Taylor` and `PWL-j` variants performed slightly better than `Baseline`, but worse than the simple approximations. It seems that the induced overhead outweighs potential benefits of better approximations. In particular, our investigations often showed large bit sizes of the rational coefficients in the auxiliary polynomials and their resultants. Another reason could be that the cell representation is less convenient, which is supported by the underwhelming performance of `Outside`, which solves fewer instances than `Baseline`. While it does avoid some resultants like the other variants, it does not simplify the cell representation. Accordingly, this seems to be a significant factor.

Transfer to the CAlC Method. We also tested our modification in the context of the *cylindrical algebraic covering (CAlC)* method [27], which is implemented in SMT-RAT as well. Now considering all 12154 instances of the QF_NRA benchmark set, there is no significant difference between the baseline CAlC implementation (solving 9964 instances) and our modification (solving 9975 instances). Once again, this might indicate that NLSAT especially benefits from simpler cell descriptions (which are not used in CAlC) and from compensating smaller cells by Boolean reasoning.

A Note on Non-Termination. In Section 3.1, we showed that our approach can lead to non-termination with some sort of "looping" behaviour and presented ways to combat this. Naturally, it would be interesting to know how often this

occurs in practice. As it is quite hard to reliably detect this behaviour, we have no concrete data. However, when turning off the limit on approximated cells, the `Simple-j` variants time out significantly more often (solving around 40 instances less than with the limit), hinting at the importance of our termination criteria. However, there are other influences in practice: When sampling new values, SMT-RAT prioritizes simple numbers, e.g. integers over rational numbers, and thus even the variants without the hard bound might (temporarily) escape looping when the gap between the actual cell boundary and the approximation becomes small. Then it may still time out in other parts of the computation or get caught in another loop.

6 Conclusions

We modified the levelwise single cell construction for NLSAT and CAlC by dynamically inserting linear polynomials into the projection to avoid expensive high-degree polynomials in resultant computations and description of the resulting cell, at the cost of smaller cells - which are unterstood as "under-approximations". We introduced various variants to introduce such polynomials, as well as criteria to mitigate potential non-termination of NLSAT and CAlC.

In our experiments, our approach could significantly improve the running time of NLSAT. Interesingly, relatively simple under-aproximations performed best, while more complex approximations did not pay off, suggesting that the gains are mainly due to simpler root isolation in NLSAT.

There are several directions for further research: Firstly, we conjecture that more intricate variants of `apx-criteria` may further improve efficiency. This is not an easy task: In experiments not presented here, various other approaches based e.g. on the sum-of-total-degrees of the polynomials or involving the degrees of all polynomials on a level did not yield better results. Secondly, our approach can only reduce computational effort for resultants, but not yet for discriminants, which have an even greater impact. Thirdly, the basic modification of the CAlC method had little impact, which might be improved. Finally, we guarantee termination of NLSAT by resorting to the original levelwise construction at some point. Alternatively, we may consider δ-completeness instead, as done e.g. in [16,6], and cover cells up to some precision δ.

Data Availability. Our implementation, experimental results, and tools for reproducing them are available at https://doi.org/10.5281/zenodo.14916587.

Acknowledgments. J. Nalbach and V. Promies were supported by the DFG project SMT-ART (AB 461/9-1), and J. Nalbach and P. Wagner by the DFG RTG 2236 UnRAVeL. Computing resources were granted under the RWTH project rwth1560.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Ábrahám, E., Davenport, J.H., England, M., Kremer, G.: Deciding the consistency of non-linear real arithmetic constraints with a conflict driven search using cylindrical algebraic coverings. J. Log. Algebraic Methods Program. **119**, 100633 (2021). https://doi.org/10.1016/J.JLAMP.2020.100633
2. Ábrahám, E., Nalbach, J., Kremer, G.: Embedding the virtual substitution method in the model constructing satisfiability calculus framework. In: England, M., Ganesh, V. (eds.) Proc. of SC^2'17. CEUR Workshop Proceedings, vol. 1974. CEUR-WS.org (2017), https://ceur-ws.org/Vol-1974/EAb.pdf
3. Barrett, C., Stump, A., Tinelli, C.: The SMT-LIB standard – version 2.0. In: Proc. of SMT '10 (2010), http://theory.stanford.edu/~barrett/pubs/BST10.pdf
4. Barrett, C., Tinelli, C.: Satisfiability Modulo Theories, pp. 305–343. Springer (2018). https://doi.org/10.1007/978-3-319-10575-8_11
5. Blanchette, J.C., Böhme, S., Paulson, L.C.: Extending sledgehammer with SMT solvers. In: Bjørner, N.S., Sofronie-Stokkermans, V. (eds.) Proc. of CADE'11. LNCS, vol. 6803, pp. 116–130. Springer (2011). https://doi.org/10.1007/978-3-642-22438-6_11, https://doi.org/10.1007/978-3-642-22438-6_11
6. Brauße, F., Korovin, K., Korovina, M., Müller, N.: A cdcl-style calculus for solving non-linear constraints. In: Herzig, A., Popescu, A. (eds.) Frontiers of Combining Systems. pp. 131–148. Springer (2019)
7. Brocot, A.: Calcul des rouages par approximation: Nouvelle méthode. Revue chronométrique. Journal des horlogers, scientifique et pratique **3**, 186–194 (1861), https://gallica.bnf.fr/ark:/12148/bpt6k1661912
8. Brown, C.W.: Improved projection for cylindrical algebraic decomposition. J. Symb. Comput. **32**(5), 447–465 (2001). https://doi.org/10.1006/JSCO.2001.0463
9. Brown, C.W., Košta, M.: Constructing a single cell in cylindrical algebraic decomposition. J. Symb. Comput. **70**, 14–48 (2015). https://doi.org/10.1016/J.JSC.2014.09.024
10. Cimatti, A., Griggio, A., Irfan, A., Roveri, M., Sebastiani, R.: Incremental linearization for satisfiability and verification modulo nonlinear arithmetic and transcendental functions. ACM Trans. Comput. Log. **19**(3), 19:1–19:52 (2018). https://doi.org/10.1145/3230639, https://doi.org/10.1145/3230639
11. Collins, G.E.: Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In: Barkhage, H. (ed.) Automata Theory and Formal Languages, 2nd GI Conference, Kaiserslautern, May 20-23, 1975. LNCS, vol. 33, pp. 134–183. Springer (1975). https://doi.org/10.1007/3-540-07407-4_17
12. Corzilius, F., Kremer, G., Junges, S., Schupp, S., Ábrahám, E.: SMT-RAT: An open source C++ toolbox for strategic and parallel SMT solving. In: Proc. of SAT'15. LNCS, vol. 9340, pp. 360–368. Springer (2015). https://doi.org/10.1007/978-3-319-24318-4_26
13. Cox, D.A., Little, J., O'Shea, D.: Ideals, varieties, and algorithms - an introduction to computational algebraic geometry and commutative algebra (2. ed.). Undergraduate texts in mathematics, Springer (1997). https://doi.org/10.1007/978-3-662-41154-4
14. cvc5: An efficient open-source automatic theorem prover for satisfiability modulo theories (smt) problems., https://cvc5.github.io/
15. Dolzmann, A., Seidl, A., Sturm, T.: Efficient projection orders for CAD. In: Gutierrez, J. (ed.) Proc. of ISSAC'04. pp. 111–118. ACM (2004). https://doi.org/10.1145/1005285.1005303, https://doi.org/10.1145/1005285.1005303

16. Gao, S., Kong, S., Clarke, E.M.: dreal: An SMT solver for nonlinear theories over the reals. In: Bonacina, M.P. (ed.) Proc. of CADE'13. LNCS, vol. 7898, pp. 208–214. Springer (2013). https://doi.org/10.1007/978-3-642-38574-2_14
17. Giesl, J., Aschermann, C., Brockschmidt, M., Emmes, F., Frohn, F., Fuhs, C., Hensel, J., Otto, C., Plücker, M., Schneider-Kamp, P., Ströder, T., Swiderski, S., Thiemann, R.: Analyzing program termination and complexity automatically with aprove. J. Autom. Reason. **58**(1), 3–31 (2017). https://doi.org/10.1007/S10817-016-9388-Y, https://doi.org/10.1007/s10817-016-9388-y
18. Jovanovic, D., Barrett, C., de Moura, L.: The design and implementation of the model constructing satisfiability calculus. In: Proc. of FMCAD'13. pp. 173–180 (2013). https://doi.org/10.1109/FMCAD.2013.7027033
19. Jovanovic, D., de Moura, L.M.: Solving non-linear arithmetic. In: Proc. of IJCAR'12. LNCS, vol. 7364, pp. 339–354. Springer (2012). https://doi.org/10.1007/978-3-642-31365-3_27
20. Kremer, G.: Cylindrical algebraic decomposition for nonlinear arithmetic problems. Ph.D. thesis, RWTH Aachen University, Germany (2020), https://publications.rwth-aachen.de/record/792185
21. Lazard, D.: An Improved Projection for Cylindrical Algebraic Decomposition, pp. 467–476. Springer (1994). https://doi.org/10.1007/978-1-4612-2628-4_29
22. Li, H., Xia, B.: Solving satisfiability of polynomial formulas by sample-cell projection. CoRR **abs/2003.00409** (2020), https://arxiv.org/abs/2003.00409
23. McCallum, S.: An improved projection operation for cylindrical algebraic decomposition. In: Caviness, B.F. (ed.) Proc. of EUROCAL'85. LNCS, vol. 204, pp. 277–278. Springer (1985). https://doi.org/10.1007/3-540-15984-3_277
24. McCallum, S.: On projection in cad-based quantifier elimination with equational constraint. In: Geddes, K.O., Salvy, B., Dooley, S.S. (eds.) Proc. of ISSAC '99. pp. 145–149. ACM (1999). https://doi.org/10.1145/309831.309892
25. de Moura, L.M., Jovanovic, D.: A model-constructing satisfiability calculus. In: Giacobazzi, R., Berdine, J., Mastroeni, I. (eds.) Proc of VMCAI'13. LNCS, vol. 7737, pp. 1–12. Springer (2013). https://doi.org/10.1007/978-3-642-35873-9_1
26. Nalbach, J., Ábrahám, E.: Merging adjacent cells during single cell construction. In: Boulier, F., Mou, C., Sadykov, T.M., Vorozhtsov, E.V. (eds.) Proc. of CASC'24. LNCS, vol. 14938, pp. 252–272. Springer (2024). https://doi.org/10.1007/978-3-031-69070-9_15, https://doi.org/10.1007/978-3-031-69070-9_15
27. Nalbach, J., Kremer, G.: Extensions of the cylindrical algebraic covering method for quantifiers. CoRR **abs/2411.03070** (2024). https://doi.org/10.48550/ARXIV.2411.03070, https://doi.org/10.48550/arXiv.2411.03070
28. Nalbach, J., Ábrahám, E., Specht, P., Brown, C.W., Davenport, J.H., England, M.: Levelwise construction of a single cylindrical algebraic cell. Journal of Symbolic Computation **123**, 102288 (2024). https://doi.org/10.1016/j.jsc.2023.102288
29. Promies, V., Nalbach, J., Ábrahám, E.: Under-approximation of a single algebraic cell (extended abstract). In: Brown, C.W., Kaufmann, D., Nalon, C., Steen, A., Suda, M. (eds.) Proc. of IJCAR'24. CEUR Workshop Proceedings, vol. 3717, pp. 132–136. CEUR-WS.org (2024), https://ceur-ws.org/Vol-3717/short3.pdf
30. SMT-RAT, a toolbox for strategic and parallel satisfiability modulo theories solving, https://github.com/ths-rwth/smtrat
31. Stern, M.: Ueber eine zahlentheoretische Funktion. Journal für die reine und angewandte Mathematik (Crelles Journal) **1858**(55), 193–220 (Jan 1858). https://doi.org/10.1515/crll.1858.55.193

32. Xu, J., Van Den Boom, T.J.J., De Schutter, B., Luo, X.L.: Minimal Conjunctive Normal Expression of Continuous Piecewise Affine Functions. IEEE Transactions on Automatic Control **61**(5), 1340–1345 (May 2016). https://doi.org/10.1109/TAC.2015.2465212
33. Xu, J., van den Boom, T.J., De Schutter, B.: Irredundant lattice piecewise affine representations and their applications in explicit model predictive control. In: Decision and Control (CDC 2014). pp. 4416–4421. IEEE (Dec 2014). https://doi.org/10.1109/CDC.2014.7040078