

The present work was submitted to the LuFG Theory of Hybrid Systems

BACHELOR OF SCIENCE THESIS

**ABSTRACTION TECHNIQUES IN
EXPLORATION-GUIDED ALGORITHMS
FOR REAL ALGEBRA**

Jan Bienmüller

Communicated by

Prof. Dr. Dr. h. c. Erika Ábrahám

Examiners:

Prof. Dr. Dr. h. c. Erika Ábrahám

Prof. Dr. rer. nat. habil Eva Zerz

Additional Advisors:

Valentin Promies

Jasper Nalbach

Aachen, 31.07.2025

Abstract

This thesis explores abstraction techniques to accelerate the satisfiability checking of nonlinear real algebra formulas. In the abstraction process, parts of the input formula φ are replaced with fresh variables, resulting in an overapproximation $\hat{\varphi}$. Since $\hat{\varphi}$ is considerably less complex than φ , its satisfiability can be checked much faster. We use an extended version of the complete *Cylindrical Algebraic Coverings* method. If $\hat{\varphi}$ is recognized as unsatisfiable, the unsatisfiability of φ can be concluded immediately. Otherwise, the substitutions are reversed step by step. We first focus on the substitution of individual multiplications and examine several optimizations. Then we consider two additional strategies that substitute entire polynomials at once.

Contents

1	Introduction	7
2	Preliminaries	9
2.1	Satisfiability Modulo NRA Solving	9
2.2	Cylindrical Algebraic Decomposition	10
2.3	Cylindrical Algebraic Coverings	12
2.4	Incremental Reduction to LRA with EU F	15
3	Abstraction Techniques	17
3.1	Abstraction in Combination with an NRA Solver	17
3.2	Substitutions	20
3.3	Adapting the Variable Ordering	21
4	Experimental Setup and First Evaluation	23
4.1	Codebase	23
4.2	Benchmarking	24
4.3	First Evaluation	24
5	Possible Improvements	25
5.1	Delayed Activation of Substitution Constraints	25
5.2	Avoiding Cycles	26
5.3	Maintaining Outer Structure	29
5.4	Evaluation of Alternative Variable Orderings	30
5.5	Further Abstraction Techniques	33
6	Conclusion	39
6.1	Summary	39
6.2	Discussion and Future Work	39
	Bibliography	41

Chapter 1

Introduction

This thesis deals with efficiently deciding the satisfiability of formulas of nonlinear real algebra (NRA) using abstraction techniques.

The problem lies within the field of satisfiability modulo theories (SMT). The objective of SMT is to determine whether a first-order logic formula is satisfiable under a given background theory. Assuming NRA as the background theory, the formulas are Boolean connectives of constraints between multivariate polynomials. These polynomials have rational coefficients and real-valued variables. Within the formulas, we may also have quantification of variables.

We were inspired by the work of Cimatti et al. in [CGI⁺17]. They propose an approach in which all nonlinear multiplications of an input formula are replaced with uninterpreted functions. Linear multiplications with constants remain unaltered. If the resulting overapproximation is unsatisfiable, then the unsatisfiability of the input formula can be declared immediately. Otherwise, a check is performed to see whether the model found for the overapproximation can be transferred to a model for the input formula. If this fails, we iteratively add linear axioms to further restrict the models of the uninterpreted functions.

In their experiments, the method achieved very promising results for unsatisfiable input formulas. It operates entirely within linear real algebra (LRA) with uninterpreted functions (EUF). This has the advantage of avoiding expensive reasoning over NRA. However, the method has the major disadvantage of incompleteness, as it does not terminate for all inputs.

In this thesis, we combine the idea of abstraction with a complete NRA solver. As in [CGI⁺17], we also abstract the input formula. Yet instead of using an LRA with EUF solver, we use an extended version of the *Cylindrical Algebraic Coverings* method [NK24]. This removes the restriction of only being able to add linear axioms. We are no longer dependent on refining the overapproximation exclusively through linear axioms, but can undo entire substitutions. At the latest after the last substitution has been undone, we obtain a formula that is satisfiability-equivalent to the input formula. Thus, our method is complete. Although we sacrifice the advantage of foregoing NRA reasoning, we show that abstraction can benefit complete NRA solvers significantly.

We first implement the substitution of individual multiplications, analogous to [CGI⁺17]. In doing so, we discuss important implementation details, as well as a series of optimizations. Following this, we extend the approach to methods that substitute more complex polynomials in one step.

Chapter 2 introduces the necessary background. In chapter 3, we describe our idea and initial implementation of the abstraction. Chapter 4 discusses the codebase, benchmarking, and the results of a first evaluation. Then, in chapter 5, we present and evaluate a series of possible optimizations. Finally, chapter 6 summarizes the results and concludes with a discussion on future work.

Chapter 2

Preliminaries

Our contributions follow from chapter 3 onward. In this chapter, we introduce the basics that are essential for understanding them.

2.1 Satisfiability Modulo NRA Solving

Nonlinear real algebra (NRA) is a theory of first-order logic (presentation adapted from [Ábr24], [Gro24]). It has the signature

$$\sigma_{\text{NRA}} = \underbrace{\mathbb{Q}}_{\text{(rational) constants}} \cup \underbrace{\{+, \cdot\}}_{\text{binary function symbols}} \cup \underbrace{\{<, \leq, =, \geq, >\}}_{\text{binary relation symbols}}.$$

If quantifiers $(\forall, \exists)$ are omitted, we speak of quantifier-free NRA (QFNRA). All variables are taken from a fixed, countably infinite set Var . For a formula φ , we write $\text{var}(\varphi) \subset \text{Var}$ for the set of variables that occur in φ .

Example 2.1.1.

$$(x + y^2 < 3) \wedge (2 \cdot x - y = 0) \quad \text{as well as} \quad (x^2 + y^2 \leq 1) \vee (x > 1)$$

are quantifier-free σ_{NRA} -formulas. $x, y \in \text{Var}$ are variables and $1, 2, 3 \in \mathbb{Q}$ are constants.

In the following, we abbreviate σ_{NRA} -formulas as NRA formulas and quantifier-free σ_{NRA} -formulas as QFNRA formulas.

Definition 2.1.1 (σ -structure). A σ -structure is a pair $\mathfrak{A} = (A, \mathbf{a})$. It consists of

- a nonempty set A , which is called the domain, and
- a mapping $\mathbf{a}$, that assigns to each k -ary function symbol $f \in \sigma$ a function $\mathbf{a}(f): A^k \rightarrow A$ and to each k -ary relation symbol $R \in \sigma$ a relation $\mathbf{a}(R) \subseteq A^k$. Constants are viewed as 0-ary functions.

For NRA, we have the σ_{NRA} -structure $\mathfrak{A}_{\text{NRA}}$. It gives NRA formulas a semantics. The domain of $\mathfrak{A}_{\text{NRA}}$ is the set of real numbers $\mathbb{R}$. This means that variables can take on values in $\mathbb{R}$. Each constant symbol $q \in \mathbb{Q}$ is interpreted as the corresponding real

number. The binary function and relation symbols of the signature are interpreted according to their usual meaning on $\mathbb{R}$.

For a variable assignment $\beta: \text{Var} \rightarrow \mathbb{R}$ we write $(\mathfrak{A}_{\text{NRA}}, \beta) \models \varphi$ iff the NRA formula φ evaluates to **true** under the assignment. For the satisfiability of φ , only the assignment of the variables $\text{var}(\varphi)$ is relevant. We denote the restriction of β to these variables by $\beta|_{\text{var}(\varphi)}$.

Satisfiability modulo NRA solving deals with the following question: Given an NRA formula φ , decide whether φ is satisfiable (**SAT**) or unsatisfiable (**UNSAT**). If φ is **SAT**, then we also want to output a satisfying assignment $\beta|_{\text{var}(\varphi)}$.

Zero-Comparison Form Predicates of NRA allow polynomials on both sides of a comparison operator. An example is the valid NRA formula

$$\varphi' := x^{12} > 9 \wedge y^2 = x - 1.$$

However, each predicate can trivially be converted into a comparison with 0. For φ' , we can obtain the equivalent formula

$$\varphi := -x^{12} + 9 < 0 \wedge y^2 - x + 1 = 0.$$

Without loss of generality, we assume that all predicates of an input formula φ are of the form

$$p \tau 0, \quad \tau \in \{<, \leq, =\}, \quad p \in \mathbb{Z}[x_1, \dots, x_n],$$

where p is a multivariate polynomial with integer coefficients. We refer to predicates of this form as *constraints*.

Furthermore we denote the set of all polynomials p occurring in an NRA formula φ as $\mathcal{P}(\varphi)$.

2.2 Cylindrical Algebraic Decomposition

The Cylindrical Algebraic Decomposition (CAD), together with related approaches based on the same idea, is the most relevant complete method for solving NRA problems [Kre20].

2.2.1 Basic Idea

Let φ be a QFNRA formula that shall be evaluated and $\text{var}(\varphi) = x_1, \dots, x_n$. In the following, we identify an assignment $\beta|_{\text{var}(\varphi)}: \text{var}(\varphi) \rightarrow \mathbb{R}$ with a point in the space $\mathbb{R}^n$.

The idea of CAD is to partition the space $\mathbb{R}^n$ into equivalence classes with respect to the satisfaction of φ . A minimal partitioning into equivalence classes would consist of two sets: One set contains the elements from $\mathbb{R}^n$ under which φ evaluates to **true**, the other set contains those elements under which φ evaluates to **false**.

In practice, however, it is unnecessarily costly to calculate such an optimal partitioning [Kre20]. Instead, CAD divides $\mathbb{R}^n$ into finitely many, but significantly more connected subsets. In the following, we refer to connected subsets as *regions*. The regions are formed in such a way that every region is $\mathcal{P}(\varphi)$ -sign-invariant. This means that each polynomial from $\mathcal{P}(\varphi)$ retains its sign in the region. In other words, every polynomial either evaluates to 0, to values > 0 , or to values < 0 in the entire region. Sign-invariance of $\mathcal{P}(\varphi)$ implies truth-invariance of φ , since no predicate changes its truth value within a sign-invariant region.

2.2.2 Algorithmic Structure

The CAD algorithm is divided into two phases: The projection phase and the lifting phase [CJ12]. Initially, a fixed variable ordering must be defined. In the following, we work under the variable ordering $x_1 \prec x_2 \prec \dots \prec x_n$.

Projection Phase

A single projection step eliminates the variable x_k (as x_k is the largest variable according to our fixed ordering) of a set of polynomials $\mathcal{P} \subseteq \mathbb{Z}[x_1, \dots, x_k]$. We denote the corresponding operator as

$$\text{proj}: 2^{\mathbb{Z}[x_1, \dots, x_k]} \rightarrow 2^{\mathbb{Z}[x_1, \dots, x_{k-1}]}.$$

$2^{\mathbb{Z}[x_1, \dots, x_k]}$ denotes the power set of $\mathbb{Z}[x_1, \dots, x_k]$. The proj operator cannot be just any function, but has to ensure that $\mathcal{P}$ is delineable¹ over all $\text{proj}(\mathcal{P})$ -sign-invariant regions. However, understanding delineability is not necessary for this thesis.

In the projection phase, the proj operator is applied repeatedly. After $n - 1$ projection steps, we are left with only univariate polynomials in $\mathbb{Z}[x_1]$ and move on to the next phase.

Lifting Phase

The lifting phase generates sample points for all regions of the final partition of $\mathbb{R}^n$. In each step, the partial sample points are extended by one dimension until we reach dimension n :

1. In $\mathbb{R}^1$, we consider the set of polynomials $\text{proj}^{n-1}(\mathcal{P})$ and choose
 - (a) each real zero of any of these polynomials,
 - (b) always one point between two neighboring zeros,
 - (c) one point to the left of the smallest and one to the right of the largest zero.

These partial one-dimensional sample points represent all $\text{proj}^{n-1}(\mathcal{P})$ -sign-invariant regions.

2. For every partial one-dimensional sample point a , we consider the cylinder $\{a\} \times \mathbb{R}$. We now take all polynomials from $\text{proj}^{n-2}(\mathcal{P})$, insert the sample point a and isolate the zeros. Let $\beta_1, \dots, \beta_n$ be the zeros. As in step 1, we choose a representative for each sign-invariant region and obtain a set of partial sample points in $\mathbb{R}^2$:

$$\{a\} \times \{\beta_1, \dots, \beta_m, r_0, \dots, r_m\}, \quad r_0 < \beta_1 < r_1 < \dots < \beta_m < r_m,$$

We do this for all one-dimensional sample points a and receive our set of two-dimensional sample points as the union of these sets.

1

Definition 2.2.1 (Delineability). *A set of polynomials $\mathcal{P} \subseteq \mathbb{Z}[x_1, \dots, x_k]$, $k \geq 2$ is delineable over a region $S \subseteq \mathbb{R}^{k-1}$ if the number of collective distinct roots*

$$|R_{\mathcal{P}}(s)| = |\{x_k \in \mathbb{R} \mid \exists f \in \mathcal{P} : f(s, x_k) = 0\}|$$

is the same for all $s \in S$ [DNSU23]. This means that the zeros of the polynomials neither intersect nor disappear.

3. Step 2 is repeated analogously for higher dimensions until the original dimension n is reached. At the end, each region of the CAD is represented by a sample point.

The combination of projection and lifting results in a sign-invariant CAD of $\mathbb{R}^n$. The number of regions is finite. By evaluating the input formula φ at all sample points, its satisfiability can be determined for each region and thus for the entire $\mathbb{R}^n$.

Combination with a SAT Solver

It is often inefficient to compute a CAD over all polynomials occurring in φ . Instead, CAD is combined with a SAT solver. The SAT solver takes care of the Boolean structure of φ and only forwards to CAD a subset of the predicates from φ whose conjoint satisfiability shall be checked.

2.3 Cylindrical Algebraic Coverings

In this thesis, we work with an extended version of the *Cylindrical Algebraic Coverings* method (CAIC), which itself is a modification of the classical CAD method. A detailed description of the original CAIC method can be found in [ÁDEK21]; the extended version we rely on is described in [NK24]. In the remainder of this thesis, “CAIC” denotes this extended version. We now explain the core aspects from [ÁDEK21] and [NK24] that are essential for understanding this bachelor’s thesis.

As already described, CAD is typically used in combination with a SAT solver. CAIC, however, does not require a separate SAT solver as it processes the Boolean structure itself. In contrast to CAD, projection and lifting are not performed in two separate phases, but in a single conflict-driven search process looking for a satisfying sample.

As with CAD, we assume a fixed variable ordering $x_1 \prec x_2 \prec \dots \prec x_n$ specified at the beginning. CAIC recursively tries to construct a (partial) sample point. To do this, one variable is assigned after the other (the smallest of the ordering first).

Let φ be a QFNRA formula with n variables. CAIC recursively constructs a partial sample point $\mathbf{s}^{(i)} = (s_1, \dots, s_i) \in \mathbb{R}^i$, $i \in \{0, \dots, n\}$.

In each step, a further variable is assigned a value. Let x_i be the next unassigned variable. We choose any value s_i that is not yet excluded for x_i (excluded intervals will be explained in a moment). After the assignment, we evaluate

$$\varphi(\mathbf{s}^{(i-1)} \times s_i) \in \{\text{true}, \text{false}, \text{multivariate}\}$$

with $\mathbf{s}^{(i-1)} \times s_i := (s_1, \dots, s_{i-1}, s_i) \in \mathbb{R}^i$. In the cases **true** or **multivariate**, we continue with the next variable. **multivariate** means that φ cannot yet be evaluated to a truth value under the current partial sample point. If φ evaluates to **false**, an implicant is computed.

Definition 2.3.1 (Implicant). *An implicant ψ is a quantifier-free formula that is simpler than the original formula φ and describes the reason for evaluating to false. The requirements for ψ are the following:*

1. ψ only contains polynomials from φ and only variables from $\mathbf{s}^{(i-1)} \times s_i$.
2. Under the partial sample point $\mathbf{s}^{(i-1)} \times s_i$, ψ is satisfied: $\psi(\mathbf{s}^{(i-1)} \times s_i) = \text{true}$.

3. $\psi \Rightarrow \varphi$ if $\varphi(\mathbf{s}^{(i-1)} \times s_i) \equiv \text{true}$ respectively $\psi \Rightarrow \neg\varphi$ if $\varphi(\mathbf{s}^{(i-1)} \times s_i) \equiv \text{false}$

The first case of the 3rd condition ($\psi \Rightarrow \varphi$ if $\varphi(\mathbf{s}^{(i-1)} \times s_i) \equiv \text{true}$) is required, since in the presence of quantifiers, sometimes reasons for $\varphi(\mathbf{s}^{(i-1)} \times s_i) = \text{true}$ must also be captured through implicants. However, in the quantifier-free case, which we are currently describing, this does not occur.

The real zeros of the calculated implicant ψ under the partial sample $\mathbf{s}^{(i-1)}$ bound an interval $I \subseteq \mathbb{R}$. The nearest zeros that enclose the value s_i are selected as interval limits. If no smaller or larger zero exists, $-\infty$ resp. $+\infty$ are chosen. The interval I generalizes the value s_i : Under the partial sample $\mathbf{s}^{(i-1)}$, each assignment of x_i to a value in I leads to an evaluation of φ to **false**. The interval I is therefore excluded for the reassignment of x_i under $\mathbf{s}^{(i-1)}$.

If these intervals cover the entire real line, this means that the partial sample point $\mathbf{s}^{(i-1)}$ cannot be extended to a satisfying sample. We then perform a projection step, as known from CAD, to generate an excluded interval in the lower dimension that rules out $\mathbf{s}^{(i-1)}$. For the projection, we use the polynomials whose zeros describe the excluded intervals. Beforehand, however, we calculate a covering of the real axis to exclude redundant intervals.

Stopping Criterion If a sample point of full dimension $\mathbf{s}^{(n)} \in \mathbb{R}$ is found and $\varphi(\mathbf{s}^{(n)}) = \text{true}$, then CAIC returns the answer **SAT** and the sample point $\mathbf{s}^{(n)}$ as a satisfying assignment. If the entire real line was excluded in the first dimension, the answer is **UNSAT**.

Projection Operator CAIC uses McCallum's projection operator. Only a reduced variant is used, as not all of the possible cross resultants are required. McCallum's operator is not complete in all cases, but such cases are recognized and **UNKNOWN** is returned.

Let $\mathcal{P} = \{p_1, \dots, p_n\} \subset \mathbb{Z}[x_1, \dots, x_i]$ be the set of polynomials to be projected. The projection step eliminates the variable x_i . Polynomials that do not contain x_i are returned unmodified. For each remaining polynomial $p \in \mathcal{P}$ and each remaining pair of polynomials $p, q \in \mathcal{P}$, a projection step generates three types of polynomials in $\mathbb{Z}[x_1, \dots, x_{i-1}]$:

Discriminant: $\text{disc}_{x_i}(p)$ evaluates to 0 (referred to as vanishing) at the points where p has a multiple zero in x_i .

Required coefficients: $\text{reqCoeffs}_{x_i}(p)$ does not necessarily yield a single polynomial, but rather a set of polynomials. The leading coefficient (LD) of p concerning x_i is always included. The vanishing of the LD indicates a possible asymptote of p . If the LD evaluates to 0 under the current partial sample point $\mathbf{s}^{(i-1)}$, the subsequent coefficients are added successively until one of them is not 0 under $\mathbf{s}^{(i-1)}$.

Resultant: $\text{res}_{x_i}(p, q)$ vanishes where p and q have a common zero in x_i . In McCallum's original operator, $\text{res}_{x_i}(p, q)$ is calculated for all pairs of polynomials. In the reduced variant, we only need those resultants that ensure that the intervals of the covering of x_i continue to overlap and that the interval boundaries actually originate from the nearest roots.

As already described, the projection is invoked when we determine by a covering of x_i that we cannot further extend a partial sample point $\mathbf{s}^{(i-1)} = (s_1, \dots, s_{i-1})$. We insert $\mathbf{s}^{(i-2)} = (s_1, \dots, s_{i-2})$ into all polynomials that result from the projection step and determine the two nearest zeroes that enclose s_{i-1} . The open interval between these zeros is excluded for the reassignment of x_{i-1} . If an interval boundary is missing, $-\infty$ or $+\infty$ is selected. The discriminants, required coefficients, and resultants ensure that the previously calculated covering for x_i remains valid under the generalization of the value s_{i-1} to the described interval. CAIC then continues with trying to reassign x_{i-1} .

Example 2.3.1 (Procedure of CAIC). *As input we consider the QFNRA formula*

$$\varphi := \underbrace{x^2 + y^2 - 1 = 0}_{c_1} \wedge \underbrace{x + y - 2 = 0}_{c_2}$$

and choose the variable ordering $x \prec y$.

1. Assignment of x : First, x is assigned 0. We get the partial sample point $\mathbf{s} = (0)$. $\varphi(\mathbf{s}) = \text{multivariate}$, so we continue with the next variable.

2. Assignment of y : We assign y with 0 and get $\mathbf{s} = (0, 0)$. Now we have $\varphi(\mathbf{s}) = \text{false}$. Both $\psi_1 := x^2 + y^2 - 1 \neq 0$ and $\psi_2 := x + y - 2 \neq 0$ would be valid implicants. We choose ψ_2 . Under the sample $\mathbf{s} = (0)$, the polynomial in ψ_2 has a zero at $y = 2$. We exclude the interval $(-\infty, 2)$ for y under $\mathbf{s} = (0)$.

3. Reassignment of y : We assign y with 3. $\varphi(\mathbf{s}) = \text{false}$ and we again get the implicant $\psi := x + y - 2 \neq 0$ with a zero of its polynomial at $y = 2$. Now we exclude the interval $(2, \infty)$ for y under $\mathbf{s} = (0)$.

4. Last possible value for y : The last remaining value that we can assign to y is the value 2. We have our partial sample point $\mathbf{s} = (0, 2)$ and again we obtain $\varphi(\mathbf{s}) = \text{false}$. Now we use $\psi : x^2 + y^2 - 1 \neq 0$ as implicant. Under the sample $\mathbf{s} = (0)$, the polynomial in ψ has zeros at $y = \pm 1$. We exclude the interval $(1, \infty)$.

In total,

$$(-\infty, 2) \cup (2, \infty) \cup (1, \infty) = \mathbb{R},$$

thus the entire y -axis is now covered.

5. Projection onto the x -axis: We observe that the intervals $(-\infty, 2)$ and $(1, \infty)$ are already sufficient for a covering. We now project the corresponding polynomials onto the x -axis:

$$\begin{aligned} \text{reqCoeffs}_y(x + y - 2) &= \{1\}, & \text{reqCoeffs}_y(x^2 + y^2 - 1) &= \{1\}, \\ \text{disc}_y(x + y - 2) &= 1, & \text{disc}_y(x^2 + y^2 - 1) &= 4 - 4x^2, \\ \text{res}_y(x + y - 2, x^2 + y^2 - 1) &= x^2 - 4x + 3. \end{aligned}$$

The resulting polynomials have zeros at $x = -1$ and $x = 1$ under the partial sample $\mathbf{s} = (0)$. We therefore exclude the interval $(-1, 1)$ for x .

6. Further steps: We choose a value for x outside of $(-1, 1)$, such as $x = 2$. The variable y can then again be freely assigned, as the intervals excluded in steps 2 - 4 only applied under the partial sample $\mathbf{s} = (0)$. However, we will run into a covering for y again and later cover the entire x -axis with excluded intervals. At the end, CAIC returns UNSAT.

2.4 Incremental Reduction to LRA with EUF

The motivation for this bachelor's thesis originates from the paper [CGI⁺17]. The method described there replaces all multiplications between two variables with uninterpreted functions (EUF). The resulting overapproximation is a formula of linear real algebra (LRA) with EUF. If necessary, the resulting formula is gradually refined by adding linear axioms. The method avoids NRA reasoning and is referred to by the authors as “Incremental Reduction to LRA with EUF”. In the following, we describe it in more detail.

Abstraction Step Let φ be an NRA formula that shall be checked for satisfiability. Each multiplication $x \cdot y$ occurring in φ between two variables $x, y \in \text{Var}(\varphi)$ is replaced by the uninterpreted function $\text{fmul}(x, y)$. Multiplications with constants remain untouched. We refer to the resulting LRA with EUF formula as $\hat{\varphi}$. It is an overapproximation of φ .

If $\hat{\varphi}$ is satisfiable, then φ does not necessarily have to be satisfiable. However, if $\hat{\varphi}$ is unsatisfiable, then φ is also unsatisfiable, as every model $\mu \models \varphi$ could be trivially turned into a model $\hat{\mu} \models \hat{\varphi}$ by adding assignments for the fmul functions: $\hat{\mu}(\text{fmul}(x, y)) = \mu(x) \cdot \mu(y)$.

Iterative Algorithm Initially, let $\Gamma = \emptyset$.

1. **Abstraction:** As described above, create an LRA with EUF formula $\hat{\varphi}$, which is a linear abstraction of the input formula φ . Also, add simple axioms (will be explained later) to Γ .
2. **Solve:** Check the satisfiability of $\hat{\varphi} \wedge \bigwedge \Gamma$ with an LRA with EUF solver.
If the result is UNSAT: Then φ is also unsatisfiable and we return UNSAT.
If the result is SAT: Let $\hat{\mu}$ be a model for $\hat{\varphi} \wedge \bigwedge \Gamma$. Try to find a model μ for φ using $\text{get-NRA-model}(\hat{\varphi}, \hat{\mu})$.
 - If this succeeds, then return SAT and μ .
 - Otherwise, proceed to step 3: Refinement
3. **Refinement:** Create a finite set of axioms Γ' such that $\hat{\mu} \not\models \Gamma'$. The constraints (axioms) are constructed in such a way that they further constrain the behavior of the fmul functions. Then set $\Gamma := \Gamma \cup \Gamma'$.
4. **Iteration:** Repeat steps 2 and 3 until SAT or UNSAT are returned or a resource limit is reached.

Axioms The axioms used in step 3 to further restrict the possible models of $\hat{\varphi} \wedge \bigwedge \Gamma$ are tangent lemmas and monotonicity lemmas.

Tangent lemmas: They represent linear bounds for the assignment of individual $\text{fmul}(x, y)$ functions. These are derived from tangents to the surface generated by the multiplication function.

Monotonicity lemmas: They guarantee that for two fmul instances $\text{fmul}(x, y)$ and $\text{fmul}(w, z)$, the inequalities $|x| \leq |w|$ and $|y| \leq |z|$ together imply that $|\text{fmul}(x, y)| \leq |\text{fmul}(w, z)|$.

Further Simple Axioms In addition, the following simple axioms are added once in step 1 to Γ in order to impose certain properties of multiplication on the fmul functions:

Zero: The product is zero iff at least one factor is zero. It is positive when both factors have the same sign and negative when they have different signs.

Sign: Flipping the sign of both factors leaves the product unchanged. Flipping the sign of exactly one factor changes the sign of the product.

Commutativity: Multiplication is commutative.

Now it remains to explain how $\text{get-NRA-model}(\hat{\varphi}, \hat{\mu})$ works. The simplest variant would be a naive check whether for all $\text{fmul}(x, y)$ functions the equation $\hat{\mu}(\text{fmul}(x, y)) = \hat{\mu}(x) \cdot \hat{\mu}(y)$ holds. If this is the case, then $\hat{\mu}$ is also a model for φ . However, the chance of this is almost zero, as the real numbers are dense and infinite. Instead, we derive the formula

$$\begin{aligned} \hat{\psi}^* &:= \hat{\psi} \wedge \bigwedge_{\text{fmul}(x,y) \in Fmuls} \left((x = \hat{\mu}[x] \wedge \text{fmul}(x,y) = \hat{\mu}(x) \cdot y) \right. \\ &\quad \left. \vee (y = \hat{\mu}(y) \wedge \text{fmul}(x,y) = \hat{\mu}(y) \cdot x) \right) \\ \text{with } \hat{\psi} &:= \bigwedge_{\hat{\alpha}_i \in \text{atoms}(\hat{\varphi}) \text{ s.t. } \hat{\mu} \models \hat{\alpha}_i} \hat{\alpha}_i \wedge \bigwedge_{\hat{\alpha}_i \in \text{atoms}(\hat{\varphi}) \text{ s.t. } \hat{\mu} \not\models \hat{\alpha}_i} \neg \hat{\alpha}_i \end{aligned}$$

(directly taken from [CGI⁺17]), where $Fmuls$ denotes the set of all uninterpreted functions occurring in $\hat{\varphi}$. The formula $\hat{\psi}^*$ describes that for each $\text{fmul}(x, y)$, one of the two variables retains its value from $\hat{\mu}$, but the other variable can take on a new value. An LRA with EUF solver checks $\hat{\psi}^*$. If it returns a model, it is obviously also a model for the original formula φ .

Strengths and Limitations The major advantage of Incremental Reduction to LRA with EUF is that there is no need for an NRA solver. Compared to NRA Reasoning, LRA Reasoning is significantly cheaper in practice. Incremental reduction is expected to perform particularly well on unsatisfiable NRA formulas that contain many linear constraints or when the relations between variables can be approximated by linear constraints that already prevent satisfaction. The authors could substantiate this with benchmarks performed on NRA transition systems: Incremental reduction was very successful for UNSAT (safe) instances.

However, in the case of satisfiable NRA formulas, the method can be inefficient, because each replaced multiplication must ultimately obtain the exact product value. This may require many refinement cycles. In addition, the algorithm does not terminate at all if all models contain irrational numbers (e.g. $\sqrt{2}$), as an LRA with EUF solver only finds rational models. For the SAT (unsafe) instances, it was therefore inferior to the classical NRA methods, but still very competitive.

Chapter 3

Abstraction Techniques

3.1 Abstraction in Combination with an NRA Solver

In section 2.4 we presented Incremental Reduction to LRA with EUF proposed by Cimatti et al. [CGI⁺17], where every multiplication between two variables is replaced by an uninterpreted function and the resulting linear formula is iteratively refined with linear axioms.

In this bachelor's thesis, we follow the same basic idea of abstraction, but take a different approach: Instead of using an LRA with EUF solver, we use a complete NRA solver as the backend. This eliminates the restriction to linear axioms. Thus, we can restore the substituted multiplications directly instead of approximating their behavior via linear axioms.

Originally, the plan was to use a complete, incrementally working NRA solver as the backend. After an initial satisfiability check, such a solver can add additional constraints to the same formula without restarting the calculation, but by reusing information that has already been obtained. However, since we do not have access to such an NRA solver, we had to adapt this concept. For a better understanding, we will nevertheless first outline the original idea. In section 3.1.2 we will then explain the solution that was actually implemented.

3.1.1 Original Idea

Abstraction Step Let φ be an NRA formula whose satisfiability shall be checked. The linear overapproximation $\hat{\varphi}$ is generated from φ as follows: Each multiplication between two variables $x \cdot y$ in φ is replaced by a fresh variable new_i . We use fresh variables and not uninterpreted functions so as not to have to rely on the solver being able to handle uninterpreted functions. The substitutions are recorded in a stack (last-in-first-out), as we need to keep track of the order for later reversing the substitutions. The set Γ is initially empty.

Iterative Reversing

Definition 3.1.1 (Substitution Constraint). *We denote with a substitution constraint the predicate*

$$new_i = x \cdot y,$$

that connects a variable new_i , introduced during the abstraction, with the multiplication that it replaced.

After each call of the NRA solver on $\widehat{\varphi} \wedge \bigwedge \Gamma$ we continue as follows:

- If the result is UNSAT : Then φ is also unsatisfiable.
- If the result is SAT and the stack is empty : The found model also satisfies φ .
- If the result is SAT and the stack is not empty : We take the topmost substitution, form the corresponding substitution constraint, add it to Γ , and call the NRA solver again.

Algorithm 1 summarizes the procedure.

Algorithm 1 Incremental Reversing of Substitutions

Input: NRA formula φ
Output: SAT with model or UNSAT

```

1:  $(\widehat{\varphi}, stack) \leftarrow \text{initial\_abstraction}(\varphi)$ 
2:  $\Gamma \leftarrow \emptyset$ 
3: while true do
4:    $(res, \widehat{\mu}) \leftarrow \text{nra\_solve}(\widehat{\varphi} \wedge \bigwedge \Gamma)$ 
5:   if  $res = \text{UNSAT}$  then
6:     return UNSAT
7:   else if  $stack$  is empty then
8:     return (SAT,  $\widehat{\mu}$ )
9:   else
10:     $last\_substitution \leftarrow stack.pop()$ 
11:     $\Gamma \leftarrow \Gamma \cup \{\text{substitution\_constraint}(last\_substitution)\}$ 
12:  end if
13: end while

```

In this way, only as much non-linearity is generated as necessary for the final decision on satisfiability.

3.1.2 Actual Implementation

As described above, a separate module was supposed to perform the abstraction and gradually forward the constraints to a Boolean SAT Solver in combination with a complete NRA Solver as backend. However, we do not have access to an implementation of a complete, incrementally working NRA solver. The subsequent appending of further constraints would have required a restart each time, which is very inefficient.

That is why we instead use CAIC (section 2.3), which does not rely on an external SAT solver for the Boolean structure. We also start by creating a linear abstraction $\widehat{\varphi}$ of the input formula φ . Instead of gradually adding one substitution constraint at a time to the initially empty set Γ , we collect all substitution constraints in Γ right at the beginning. CAIC then immediately receives the formula $\widehat{\varphi} \wedge \bigwedge \Gamma$. This means that CAIC works on the conjunction of the abstraction with all substitution constraints from the very beginning.

We exploit the fact that CAIC only evaluates a predicate when all variables that occur in the predicate have been assigned a value. In the following, we call a predicate (and by analogy also a formula) active once all its variables are assigned.

We adapt the variable ordering to ensure that (as far as possible) only the linear abstraction becomes active first, and then the substitution constraints become active one after the other. In this way, CAIC acts as if the constraints were added step by step without the formula having to be altered during the search.

We implemented the abstraction logic directly into the module that realizes the CAIC method. More about the codebase can be read in section 4.1.

To make the procedure more tangible, we present a sample run in Example 3.1.1.

Example 3.1.1 (Abstraction using CAIC). *We consider the unsatisfiable NRA formula*

$$\psi := x^{32}y^{16} + x^8y^{16}z^8 - 1 < 0 \quad \wedge \quad x^{32}y^{16} + 2x^{20}y^{16}z^4 + x^8y^{16}z^8 - 9 = 0.$$

Without abstraction, CAIC generates polynomials with a sotd¹ of above 400 during the projection steps, even with an optimal variable ordering.

Abstraction *We replace multiplications step by step with fresh variables. A possible sequence of substitutions is*

- | | |
|--------------------------------------|---|
| 1. $x^2 \mapsto new_0$ | 9. $new_1 \cdot new_7 \mapsto new_8$ |
| 2. $new_0^2 \mapsto new_1$ | 10. $new_1^2 \mapsto new_9$ |
| 3. $y^2 \mapsto new_2$ | 11. $new_9^2 \mapsto new_{10}$ |
| 4. $new_2^2 \mapsto new_3$ | 12. $new_4 \cdot new_{10} \mapsto new_{11}$ |
| 5. $new_3^2 \mapsto new_4$ | 13. $new_8^2 \mapsto new_{12}$ |
| 6. $z^2 \mapsto new_5$ | 14. $new_{11}^2 \mapsto new_{13}$ |
| 7. $new_5^2 \mapsto new_6$ | 15. $new_8 \cdot new_{11} \mapsto new_{14}$ |
| 8. $new_4 \cdot new_6 \mapsto new_7$ | |

which leads to the linear abstraction

$$\hat{\varphi} := new_{13} + new_{12} - 1 < 0 \quad \wedge \quad new_{13} + 2 new_{14} + new_{12} - 9 = 0.$$

CAIC now operates on the conjunction $\hat{\varphi} \wedge \bigwedge \Gamma$, where Γ already contains all substitution constraints.

Variable ordering *We choose the following variable ordering (details on the variable ordering follow in section 3.3):*

$$new_{13} \prec new_{12} \prec new_{14} \prec new_{11} \prec new_8 \prec new_{10} \prec new_4 \prec new_9 \prec new_1 \prec new_7 \\ \prec new_6 \prec new_5 \prec z \prec new_3 \prec new_2 \prec new_0 \prec x.$$

¹Sum of total degrees (sotd): Sum of the total degrees of all monomials with non-zero coefficients. The total degree of a monomial is the sum of its exponents.

Search for a satisfying sample After the variables new_{13} , new_{12} and new_{14} are assigned, the linear abstraction $\hat{\varphi}$ is active. $\hat{\varphi}$ is not unsatisfiable on its own, so *CALC* needs to assign further variables. After the subsequent assignment of new_{11} and new_8 , $\hat{\varphi}$ as well as the substitution constraints $new_8 \cdot new_{11} = new_{14}$, $new_{11}^2 = new_{13}$ and $new_8^2 = new_{12}$ are active. These together are already contradictory, and *CALC* returns UNSAT.

All other substitution constraints remain inactive, so that the necessary projection steps only produce polynomials of $sotd \leq 6$ - a drastic decrease compared to over 400 using *CALC* without abstraction.

3.2 Substitutions

To calculate the substitutions, we begin by implementing the two most natural strategies. Those have also been used in earlier work, including [Spe23]. Let φ be the input formula. Both strategies work directly on the polynomial set $\mathcal{P}(\varphi)$.

3.2.1 Highest Exponent First (HEF)

The highest exponent first strategy operates as follows:

1. As long as a variable with exponent ≥ 2 exists:
 - Select the variable x with the highest exponent,
 - Replace the product $x \cdot x$ with a fresh variable new_i .
2. As long as a multiplication between two variables $x \cdot y$ exists:
 - Select the most frequent product $x \cdot y$ and replace it with a fresh variable new_i .

Example

$$\varphi := \underbrace{\underbrace{x \cdot x}_{new_0} \cdot \underbrace{x \cdot x}_{new_0}}_{new_2} \cdot \underbrace{x \cdot y}_{new_4} + \underbrace{x \cdot \underbrace{y \cdot y}_{new_1}}_{new_3} = 0 \wedge x < 0$$

new_5

We obtain the linear abstraction $\hat{\varphi} := new_5 + new_3 = 0 \wedge x < 0$. The strategy ensures that the highest occurring exponents become smaller as fast as possible.

3.2.2 Most Frequent First (MFF)

The most frequent first strategy skips phase 1 of HEF and immediately always replaces the current most frequent product of two variables, also considering squares of variables.

Example

$$\varphi := \underbrace{x \cdot x \cdot x}_{new_0} \cdot \underbrace{x \cdot x}_{new_0} \cdot \underbrace{x \cdot y}_{new_1} + \underbrace{x \cdot y}_{new_1} \cdot y = 0 \wedge x < 0$$

$$\underbrace{\hspace{10em}}_{new_2} \quad \underbrace{\hspace{10em}}_{new_3}$$

$$\underbrace{\hspace{10em}}_{new_4}$$

We obtain the linear abstraction $\hat{\varphi} := new_4 + new_3 = 0 \wedge x < 0$. This strategy ensures that the *sotd* decreases as quickly as possible and that only as few substitutions are performed as necessary. In the example, we need one substitution less than with HEF.

We note that neither heuristic achieves the abstraction shown in example 3.1.1. Its substitution sequence was optimized by hand and cannot be calculated using a simple heuristic.

3.3 Adapting the Variable Ordering

As already explained, the linear abstraction $\hat{\varphi}$ should be activated first (whenever possible). Only then should the substitution constraints become active, in reverse order of their creation, i.e., from the latest substitution to the earliest.

After we have passed the formula $\hat{\varphi} \wedge \bigwedge \Gamma$ to *CAIC*, the method first determines a variable ordering on its own. During a large part of this work, the implementation of *CAIC* used the heuristic *GreedyMaxUnivariate*. This heuristic always selects next the variable that most frequently occurs as the sole remaining variable in the constraints of the input formula. If no such variable exists, the first remaining variable is chosen. Towards the end of our research, the implementation switched to *FeatureBasedPickering* (independently of this bachelor's thesis). *FeatureBasedPickering* [PdRAEC24] uses three metrics that have been identified by explainable AI. It always selects next the remaining variable with the lowest sum of its highest exponents across all polynomials. If there is a tie, the smallest mean value of the mean exponents of this variable per polynomial decides, and in case of another tie, the smallest total sum of all exponents of this variable decides.

All runtime measurements were repeated with this new heuristic to obtain comparable results.

Adaption From the ordering $<_{CAIC}$ selected by *CAIC*, we now construct an adapted ordering $<_{adapted}$ that implements the activation behavior described above.

For this purpose, let $latest_subst(x)$ be the index of the last substitution, in which x is a factor of the substituted product (for variables that do not occur in any substitution, this value can be arbitrary).

For two variables lhs and rhs , we define

$$lhs <_{adapted} rhs \iff \begin{cases} lhs \in \text{var}(\hat{\varphi}) \wedge rhs \notin \text{var}(\hat{\varphi}) \text{ or} \\ lhs, rhs \in \text{var}(\hat{\varphi}) \wedge lhs <_{CAIC} rhs \text{ or} \\ lhs, rhs \notin \text{var}(\hat{\varphi}) \wedge latest_subst(lhs) > latest_subst(rhs) \text{ or} \\ lhs, rhs \notin \text{var}(\hat{\varphi}) \wedge latest_subst(lhs) = latest_subst(rhs) \wedge lhs <_{CAIC} rhs. \end{cases}$$

Intuitive explanation: Variables from the abstraction $\hat{\varphi}$ are assigned first. Within these variables, the ordering determined by *CAIC* decides. For the remaining variables, the last substitution in which they were part of the replaced product is the

deciding factor. The later the substitution, the earlier the variable is assigned. In the event of a tie, the ordering determined by *CAIC* decides.

It is obvious that $<_{\text{adapted}}$ is a total order and therefore a valid variable ordering.

Despite adapting the variable ordering, it is not possible to completely prevent substitution constraints of earlier substitutions from becoming active earlier than or at the same time as constraints from later substitutions. This happens when all variables from an earlier substitution also occur in later substitutions. For the same reason, it can happen that substitution constraints become active at the same time as or even before $\hat{\varphi}$.

However, the following upper limit applies: If n variables are already assigned, a maximum of $n - 1$ substitution constraints can be active. The reason is straightforward:

- Each substitution introduces exactly one fresh variable new_i .
- The product to be substituted contains at least one existing variable.
- A constraint only becomes active when both new_i and the variables of the replaced product have been assigned.

After n assignments, a maximum of $n - 1$ substitution constraints can therefore be active. Even if several constraints become active at once instead of sequentially, their total number is always limited linearly by the number of variables already assigned.

Example 3.3.1 (Adapted Variable Ordering).

$$\hat{\varphi} := \textcolor{red}{new}_5 + \textcolor{red}{new}_3 = 0 \wedge \textcolor{red}{x} < 0$$

Substitutions

1. $x^2 = new_0$
2. $y^2 = new_1$
3. $\textcolor{blue}{new}_0^2 = new_2$
4. $\textcolor{orange}{x} \cdot \textcolor{orange}{new}_1 = new_3$
5. $\textcolor{green}{x} \cdot \textcolor{green}{y} = new_4$
6. $\textcolor{brown}{new}_2 \cdot \textcolor{brown}{new}_4 = new_5$

VarOrder FeatureBasedPickering (CAIC)

$$\begin{aligned} x < y < new_0 < new_1 < new_2 \\ < new_3 < new_4 < new_5 \end{aligned}$$

Adapted Variable Ordering

$$\begin{aligned} \textcolor{red}{x} < \textcolor{red}{new}_3 < \textcolor{red}{new}_5 < \textcolor{brown}{new}_2 \\ < \textcolor{brown}{new}_4 < \textcolor{green}{y} < \textcolor{orange}{new}_1 < \textcolor{blue}{new}_0 \end{aligned}$$

Chapter 4

Experimental Setup and First Evaluation

Having introduced our abstraction method, we proceed with an initial evaluation. However, this is by no means the end of this thesis: In chapter 5, we present a variety of optimizations and their corresponding evaluations. To ensure that both the initial evaluation and the subsequent ones are transparent and comprehensible, we first describe our codebase and the benchmark set on which they are performed.

4.1 Codebase

We use an existing implementation of CAIC in the form of the `COVERINGNGMODULE` (CNG). CNG is a module of SMT-RAT [CKJ⁺15], an SMT solver developed by the Theory of Hybrid Systems group at RWTH Aachen University.

The abstraction techniques were integrated directly into this module, in a new class called `Abstraction`. The source code written for this bachelor's thesis is fully commented and available in the GitLab repository of SMT-RAT¹. It can be found in the branch `jan-bienmueller/bachelor-thesis`. The relevant files `Abstraction.cpp` and `Abstraction.h` are located in the directory `src/smrat-coveringng`. The abstraction can be activated and configured via the file `src/smrat-modules/CoveringNGModule/CoveringNGSettings.h`. Every evaluation shown in this thesis can be reproduced by an appropriate configuration. Do not forget to select the strategy `CoveringNG/Default` in `ccmake`.

The integration of the class `Abstraction` into the CNG is deliberately minimally invasive. Only in two places do we use a `constexpr` variable in order to check whether abstraction is activated, and if so, to perform the corresponding action:

- **Immediately after receiving the input formula**
Invoking `substitute()`. The function returns the conjunction $\widehat{\varphi} \wedge \bigwedge \Gamma$ of abstraction and substitution constraints.
- **Right after calculating the variable ordering**
We invoke `adaptVariableOrdering()` to adapt the ordering. A method of the derived constraints optimization (section 5.2.2) is also executed if necessary.

¹<https://git.rwth-aachen.de/ths/smt/smrat>

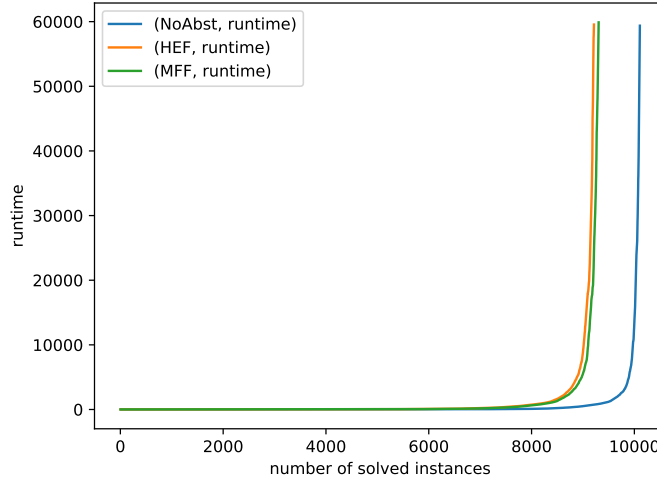


Figure 4.1: Performance plot of first evaluation

4.2 Benchmarking

To evaluate the implemented methods, we use the entire QF_NRA benchmark set that is in the non-incremental part of the SMT-LIB benchmark library [PSB⁺24].

The set comprises 12,154 QFRNA formulas. They stem from areas such as automatic theorem proving, program verification, bounded model checking, geometric problems, and economic reasoning.

All benchmarks are performed on the Claix-2023 cluster at RWTH Aachen University. We use nodes with Intel Xeon 8468 Sapphire Rapids CPUs. Each instance is given a time limit of 60 seconds and 4GB of main memory. The runs are coordinated with BENCHMARK [PN25], a tool for automated benchmarking. The plots shown in this thesis are also generated with BENCHMARK.

Since the rest of our thesis relies on continuous benchmarking, we decided to present the results directly in the respective sections and not collect them at the end of the thesis. This allows one to immediately understand and grasp the effects of individual optimizations.

4.3 First Evaluation

We perform a first benchmark run to measure the impact of the presented linear abstraction on the performance of CAIC. The following configurations were compared:

noAbst: “Vanilla CAIC”, i.e., CAIC without any abstraction.

HEF: CAIC with HEF abstraction (3.2.1) and adapted variable ordering (3.3).

MFF: CAIC with MFF abstraction (3.2.2) and adapted variable ordering (3.3).

Figure 4.1 shows that MFF performs better than HEF; therefore, we stay with MFF in the following. Nevertheless, MFF solves significantly fewer instances within the time limit than noAbst. That is why we look at possible improvements in the next chapter.

Chapter 5

Possible Improvements

5.1 Delayed Activation of Substitution Constraints

As already mentioned, it can happen that individual substitution constraints become active at the same time as or even before the abstraction $\widehat{\varphi}$. Although at most $n - 1$ substitution constraints can already be active when n variables have been assigned, prematurely activated constraints lead to additional computation if $\widehat{\varphi}$ is already unsatisfiable on its own. We therefore implement a variant that forces all substitution constraints to become active only after $\widehat{\varphi}$.

Delay Mechanism We iterate over all substitutions, from the earliest (lowest) to the latest (highest). Let $\alpha : x \cdot y \mapsto new_k$ be the currently considered substitution. If $x, y, new_k \in \text{var}(\widehat{\varphi})$, then we add a further substitution $new_k \mapsto new_i$, which replaces the variable new_k in $\widehat{\varphi}$ with a fresh variable new_i .

The substitution constraint associated with α can still only become active after new_k has been assigned a value. However, the abstraction $\widehat{\varphi}$ now becomes active before new_k is assigned, as new_k has been replaced by new_i and new_i is assigned earlier by the adapted variable ordering. Example 5.1.1 shows the mechanism in action.

Example 5.1.1 (Delayed Activation).

$$\widehat{\varphi} := new_2 + new_1 + x = 0, \quad \Gamma := \{new_1 = x^2, new_2 = new_1^2\}.$$

The linear abstraction $\widehat{\varphi}$ and the two substitution constraints would become active at the same time - after new_1, new_2, x are assigned. Through the additional substitution $new_1 \mapsto new_3$, we get

$$\widehat{\varphi} := new_2 + new_3 + x = 0, \quad \Gamma := \Gamma \cup \{new_3 = new_1\}.$$

The adapted variable ordering is

$$new_3 \prec x \prec new_2 \prec new_1.$$

This guarantees that once (after assigning new_3, x , and new_2) only $\widehat{\varphi}$ is active. The constraints from Γ become active later.

Results The delay mechanism did not result in a runtime advantage in the performed benchmark. In some cases, the performance even got worse because the overhead caused by the additional variables and constraints outweighs the benefits. We therefore discard the delay mechanism.

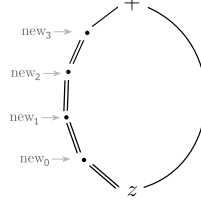


Figure 5.1: Original polynomial φ from example 5.2.1 together with the substitutions visualized as a multigraph.

5.2 Avoiding Cycles

We illustrate the next problem with a small example.

Example 5.2.1. *Let*

$$\varphi := z^{16} + z < 0.$$

The MMF strategy successively substitutes

$$z^2 \mapsto \text{new}_0, \text{new}_0^2 \mapsto \text{new}_1, \text{new}_1^2 \mapsto \text{new}_2, \text{new}_2^2 \mapsto \text{new}_3,$$

so that

$$\hat{\varphi} := \text{new}_3 + z < 0.$$

CAIC therefore receives the formula

$$\hat{\varphi} \wedge \bigwedge \Gamma \quad \text{with} \quad \Gamma = \{z^2 = \text{new}_0, \text{new}_0^2 = \text{new}_1, \text{new}_1^2 = \text{new}_2, \text{new}_2^2 = \text{new}_3\}.$$

According to section 3.3, the adapted variable ordering is

$$z \prec \text{new}_3 \prec \text{new}_2 \prec \text{new}_1 \prec \text{new}_0.$$

Search for a Satisfying Sample After assigning a value to the variable z , the proper value of new_3 is functionally fixed to z^{16} . The problem, though, is that all substitution constraints must be active in order to uncover the possible contradiction $\text{new}_3 \neq z^{16}$. However, since all constraints in Γ are still inactive, CAIC (while considering $\hat{\varphi}$) must guess a value for new_3 at this point. Only after new_2 , then new_1 , and then new_0 are assigned, all constraints become active and reveal the contradiction $\text{new}_3 \neq z^{16}$.

Several projection steps take place, which finally lead to an interval being excluded for new_3 and to CAIC guessing a new value for new_3 , which is again highly unlikely to equal z^{16} .

In this example, the polynomials generated when running CAIC with MFF abstraction remained relatively small, with a maximum `sotd` of 28. However, it occurred 7 times that new_3 was assigned, then all following variables new_2 , new_1 , and new_0 were assigned, and as a late consequence of the actual contradiction $\text{new}_3 \neq z^{16}$, we had to project all the way back to new_3 .

Figure 5.1 illustrates that all substitution variables are already determined functionally after z has been assigned.

Assigning the fresh variables last is not an option. In example 5.2.1, the variable ordering $z \prec new_0 \prec new_1 \prec new_2 \prec new_3$ would prevent us from first guessing unsuitable values for fresh variables and thereby running into conflicts much later. However, it would have a serious disadvantage: $\widehat{\varphi}$ would then contain the last assigned variable new_3 and would only be active at the very end of the search. This would result in the loss of exactly the effect that we want to gain through the abstraction - that we first consider the abstraction $\widehat{\varphi}$ and then the substitution constraints become active incrementally.

We implemented two different variants to tackle the described problem.

5.2.1 Variant 1: Acyclicity

In Figure 5.1, we have depicted the original polynomial from φ as a multigraph and written the fresh variables of the substitutions to the corresponding nodes. You can see that the fresh variables are parts of a large cycle. If variables lie on a cycle, then they are related and conflicts may result from earlier assignments later in the process.

To avoid this, we only allow a substitution $t_1 \cdot \dots \cdot t_k \mapsto new_i$ if no factor t_i then occurs in any monomial of the formula.

Substitution Algorithm

1. **Power Substitution** For each variable x we consider all its exponents. If the set of exponents has a prime factor $p > 1$ as a common divisor, we replace x^p with a fresh variable. We consider variables with larger exponents first.
2. **Product Substitution** If no further power substitution is possible, we check whether two different variables always occur as a common product. If this is the case, we replace the product with a fresh variable.

Examples

1. $\varphi := x^{15} + x^{30} < 0$
A substitution $x^2 \mapsto new_0$ would not be allowed because x would still remain in the formula. Instead we substitute

$$x^3 \mapsto new_0, \quad new_0^5 \mapsto new_1,$$

which results in $\widehat{\varphi} := new_1 + new_1^2 < 0$.

2. $\varphi := x y z + x y z v < 0$
Stepwise substitutions $x y \mapsto new_0$, $new_0 z \mapsto new_1$ result in $\widehat{\varphi} := new_1 + new_1 v < 0$. We see that this variant does not necessarily result in a completely linear abstraction.
3. $\varphi := x^4 z y + x^2 < 0$
The only admissible substitution is $x^2 \mapsto new_0$, which results in $\widehat{\varphi} := new_0^2 z y + new_0 < 0$.

Results The idea turns out to be too restrictive:

- In our benchmark ($\approx 12,000$ instances), substitutions are only possible for 416 instances. No single instance allows more than four substitutions.
- The performance plots of CAIC without abstraction and the acyclic variant are practically identical. In total, the acyclic variant is able to solve one instance more.

Due to its limited applicability, we discard this approach.

5.2.2 Variant 2: Derivation of Further Constraints

In example 5.2.1 discussed above, the set Γ consists of the four substitution constraints

$$x^2 = new_0, \quad new_0^2 = new_1, \quad new_1^2 = new_2 \quad \text{and} \quad new_2^2 = new_3.$$

However, since these constraints are only active altogether after all variables have been assigned, the conflict $new_3 \neq z^{16}$ is only discovered very late. If an additional constraint $new_3 = z^{16}$ had been included in the set Γ from the very beginning, new_3 would immediately receive the correct value, and the later conflict would not arise at all.

The idea behind this variant is therefore to add further constraints to Γ right at the beginning. We only add constraints that can be derived purely from the existing substitution constraints.

Generation of additional constraints. Each substitution constraint is of the form

$$a_0^{e_0} \cdot \dots \cdot a_n^{e_n} = new_k \quad \text{with} \quad a_0, \dots, a_n \in Var, \quad e_0, \dots, e_n \in \mathbb{N}_{>0}.$$

If one of the variables $a_0, \dots, a_n$ is itself a fresh variable, we can substitute it with its definition and obtain a new equation. If the substituted definitions also contain fresh variables, these can also be replaced recursively. We go through all possible combinations of such replacements and add all resulting equations to the set Γ . This can be efficiently implemented using dynamic programming.

We add the derived constraints to Γ after the variable ordering has been determined. This means that activating this optimization does not influence the variable ordering. We also experimented with adding the derived constraints to Γ before determining the variable ordering, but there was no significant difference in performance.

Result for the current example 5.2.1. We obtain the additional constraints

$$new_1 = x^4, \quad new_2 = x^8, \quad new_2 = new_0^4, \quad new_3 = x^{16}, \quad new_3 = new_0^8, \quad new_3 = new_1^4,$$

and therefore in particular the discussed constraint $new_3 = x^{16}$.

Effect of the derived constraints. The additional constraints do not interfere with the advantages of abstraction as they only become active if the value of a variable is already uniquely determined by previous assignments. In all best-case examples of abstraction presented in this paper, the abstraction can still demonstrate its advantages. Figure 5.2 shows that MFF performs better with the derived constraints than without them.

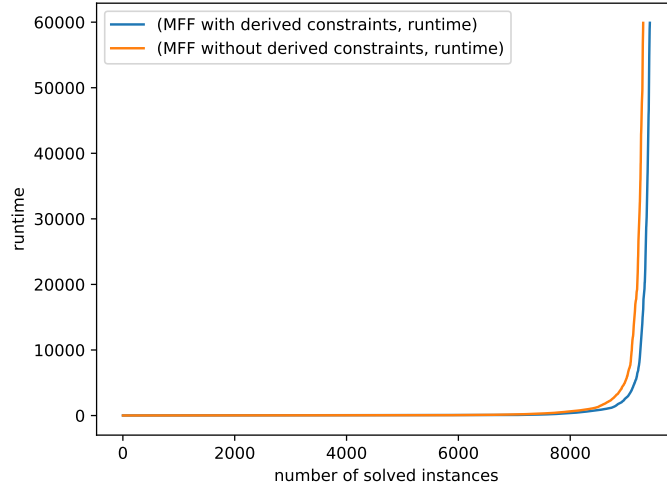


Figure 5.2: Evaluation of deriving further constraints

5.3 Maintaining Outer Structure

Substitutions show their particular strength in the case of unsatisfiable formulas in which monomials of high degree hide a low-dimensional structure. However, if this structure is destroyed too early in the course of substitutions, CAIC must first reactivate all subsequent substitution constraints before the structure becomes visible again. The following example 5.3.1 illustrates this problem.

Example 5.3.1. *Let*

$$\varphi := (a^{32}b^2c^8)^2 + (c^8b^8d)^2 - 1 = 0 \wedge (a^{32}b^2c^8)^3 + (c^8b^8d)^3 - 2 = 0.$$

At first glance, this NRA formula looks like a perfect example for abstraction. Here, an unsatisfiable two-dimensional problem is hidden in a four-dimensional formula. If one were to abstract the inner terms $(a^{32}b^2c^8)$ and (c^8b^8d) each with a fresh variable, the formula is still unsatisfiable.

Problem with HEF and MFF. *The expectation for the substitutions would be that the inner terms are substituted first, and the outer exponents (the 2 and the 3) are only considered last. This way, the outer structure of the problem would be restored after just a few activated substitution constraints, and the unsatisfiability would be quickly identified.*

We have formulated this example using the HEF heuristic. However, the result would be similar using MFF.

HEF (section 3.2.2) leads to the following substitutions (excerpt):

- | | |
|---------------------------------|----------------------------|
| 1. $a^2 \mapsto new_0$ | 4. $c^2 \mapsto new_3$ |
| 2. $new_0^2 \mapsto new_1$ | 5. $new_1^2 \mapsto new_4$ |
| 3. $b^2 \mapsto new_2$ | 6. $new_2^2 \mapsto new_5$ |
| ... (total of 24 substitutions) | |

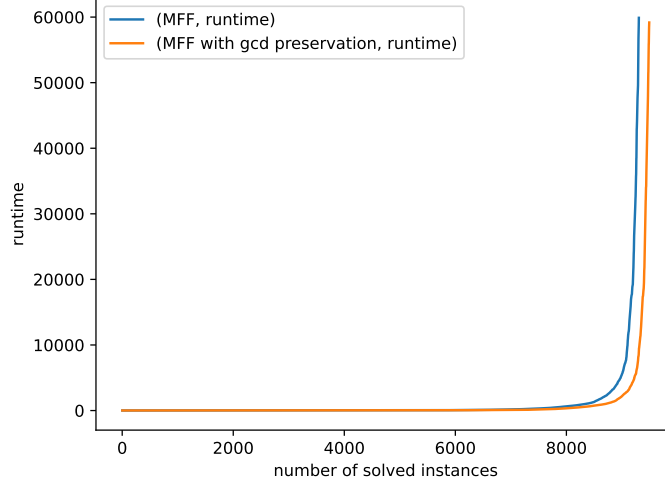


Figure 5.3: Evaluation of gcd preservation

The outer exponents are still intact after substitution 5. But already after substitution 6, two of the outer exponents got destroyed. *CALC* must therefore activate $24 - 5 = 19$ substitution constraints before the conflict becomes visible. This example takes significantly longer with abstraction than without.

Preservation of the greatest common divisor. We adapt the substitution strategy as follows: For each monomial $m = \prod_i x_i^{e_i}$ of the input formula we calculate $g := \gcd(e_1, \dots, e_k)$. We write $m = (\prod_i x_i^{e_i/g})^g$ and leave the outer exponent g untouched when substituting. Only the inner part $\prod_i x_i^{e_i/g}$ is substituted. This preserves the outer exponents and thus the core structure of the formula.

Example 5.3.1 continued. For $a^{64}b^4c^{16}$ we have $g = \gcd(64, 4, 16) = 4$. We therefore keep the outer exponent 4 and only abstract $a^{16}b^4c^4$. We proceed in the same way for the other monomials.

After all substitutions, we obtain the unsatisfiable abstraction

$$\hat{\varphi} := new_{13}^4 + new_{11}^2 - 1 = 0 \wedge new_{13}^6 + new_{11}^3 - 2 = 0.$$

Results Figure 5.3 compares MFF with gcd preservation to normal MFF. Preserving the gcds leads to a considerable increase in performance, and the number of solved instances increases significantly.

We also experimented with substituting entire “inner monomials” with a fresh variable in one step. However, this was slower overall.

5.4 Evaluation of Alternative Variable Orderings

The adaptation of the variable ordering described in section 3.3 was used in all previous evaluations. This adaptation most closely reflects the incremental nature of our approach: First, only the variables from the abstraction $\hat{\varphi}$ are assigned, then

Variable ordering	Implicant polynomials from $\hat{\varphi}$ (%)
Variables in abstraction first	47
Variables in abstraction last	69
Fresh variables first	47
Fresh variables last	74
Not adapted ordering	69

Table 5.1: Influence of different variable orderings on the implicants

the substitution constraints become active in reverse order of origination. Although this procedure is the most intuitive, it cannot be ruled out that alternative variable orderings could be more efficient. In this section, we therefore examine four further variants.

Procedure We create a performance plot for each variable ordering. In addition, we record how many polynomials of the implicants come from the abstraction $\hat{\varphi}$ and how many come from a substitution constraint $\in \Gamma$.

An implicant ψ , as introduced in section 2.3, is a quantifier-free subformula that describes the reason for the current truth value and contains only polynomials from the formula $(\hat{\varphi} \wedge \bigwedge \Gamma)$ passed to CAIC.

Evaluated Variable Orderings

- 1. Variables in abstraction first:** Basic ordering from section 3.3. All variables from $\hat{\varphi}$ are assigned first; the remaining variables are assigned in reverse substitution order.
- 2. Variables in abstraction last:** All variables from $\hat{\varphi}$ are assigned last; the remaining variables are assigned in substitution order. It is not exactly the opposite of 1, because in the case of a tie we still use the relation $<_{\text{CAIC}}$ - and not its opposite - to break the tie.
- 3. Fresh variables first:** All fresh variables $new_{n-1} \prec \dots \prec new_0$ are assigned before the original variables.
- 4. Fresh variables last:** All fresh variables $new_0 \prec \dots \prec new_{n-1}$ are assigned after the original variables.
- 5. Not adapted ordering:** No adjustment to the variable ordering computed by CAIC (*FeatureBasedPickering*).

Results Figure 5.4 shows the performance plots of the five variants. Table 5.1 shows what proportion of the implicant polynomials originates from the abstraction $\hat{\varphi}$. The remaining part comes from substitution constraints.

It is notable that the variable ordering used so far *vars in abstraction first* has one of the lowest proportions of abstraction polynomials with 47%. This suggests that conflicts arise primarily between substitution constraints and that the abstraction $\hat{\varphi}$ is less frequently involved.

The ordering *fresh vars last* achieves the highest proportion at 74%. However, as already explained in section 5.2, with this ordering, the linear abstraction only

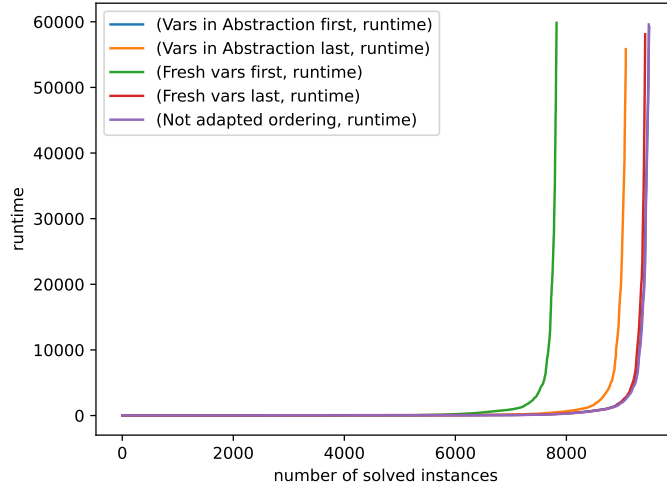
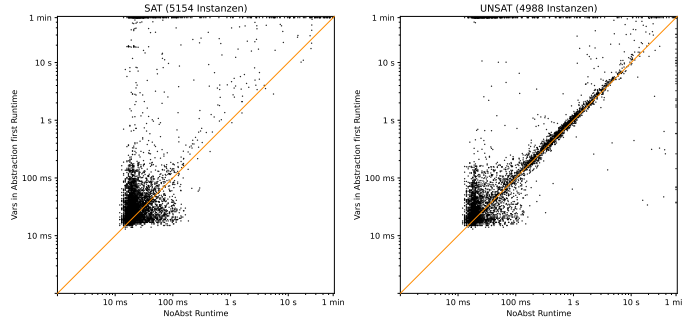


Figure 5.4: Comparison of different variable orderings

Figure 5.5: Comparison between CAIC without abstraction (NoAbst) and *vars in abstraction first*

becomes active last, after all substitution constraints are already active. This counteracts the actual benefit of abstraction, and the additional constraints merely cause overhead compared to CAIC without abstraction.

Our ordering *vars in abstraction first* delivers the best results in the benchmark: The corresponding blue curve in Figure 5.4 is almost identical to the purple curve of the *not adapted ordering*, but lies slightly to the right of it and solves eight more instances. Although this advantage is not significant, it affirms this choice of ordering, as it most closely resembles the basic idea of gradually undoing the substitutions. We therefore continue to use *vars in abstraction first* as our ordering.

Figure 5.5 shows a runtime comparison of the individual benchmark instances between *vars in abstraction first* and CAIC without abstraction (NoAbst). Each point corresponds to one formula of the benchmark. The instances are divided into two diagrams: The left one contains the satisfiable instances, the right one the unsatisfiable instances.

As expected, CAIC with abstraction consistently performs worse than CAIC without abstraction for the satisfiable instances. Abstraction also performs worse overall

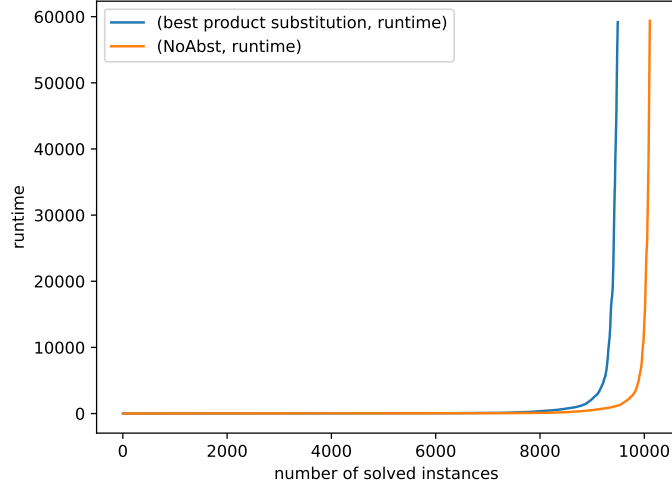


Figure 5.6: Comparison between *best product substitution* and CAIC without abstraction (NoAbst)

for the unsatisfiable instances. However, it is remarkable that there are some unsatisfiable instances where the abstraction unleashes its potential and provides a significant runtime advantage.

5.5 Further Abstraction Techniques

The methods looked at so far only replace products of variables (monomials) with fresh variables. The best configuration for product substitution, which is MFF (3.2.2) with derived constraints (5.2.2), preservation of the gcd (5.3), and *vars in abstraction first* variable ordering (3.3), is called *best product substitution* in the following.

Figure 5.6 shows that *best product substitution* - despite our optimizations - still clearly falls behind CAIC without abstraction (NoAbst). Therefore, in this chapter, we examine two alternative approaches that abstract not only monomials but entire polynomials in one substitution step.

5.5.1 Replacing Irreducible Factors (RIP)

The basic idea of this abstraction strategy is to determine all irreducible factors of all polynomials of the input formula and to replace each of these factors with a fresh variable.

Let φ be the input formula. For each polynomial $p \in \mathcal{P}(\varphi)$, we proceed as follows:

1. **Ignore constants.** If p is constant, it is kept unmodified.
2. **Factorize (over $\mathbb{Q}[x_1, \dots, x_n]$).** Decompose p over $\mathbb{Q}[x_1, \dots, x_n]$ into normalized, irreducible factors $f_1, \dots, f_m$ with multiplicities $e_1, \dots, e_m$:

$$p = c \cdot f_1^{e_1} f_2^{e_2} \dots f_m^{e_m}, \quad c \in \mathbb{Q}.$$

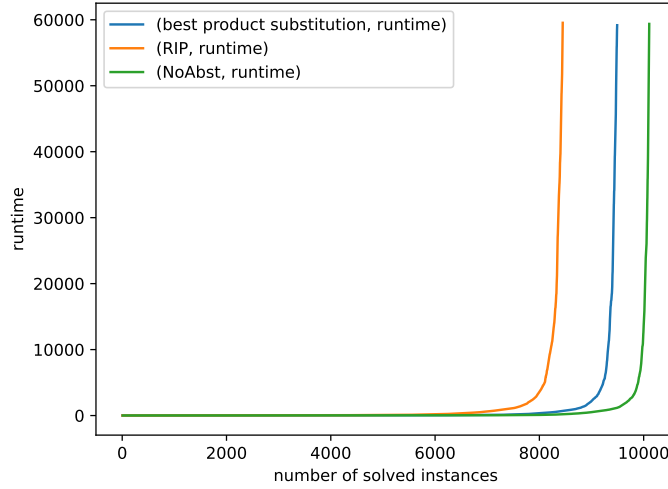


Figure 5.7: Evaluation of RIP

3. Substitute. Each irreducible factor f_i is substituted with a fresh variable new_i . If an identical factor occurs in several polynomials, the same fresh variable is used. These substitutions result in the abstraction $\hat{\varphi}$.

For each substitution $f_i \mapsto new_i$, the substitution constraint $f_i = new_i$ is included in the set Γ . Analogous to product substitution, the formula $\hat{\varphi} \wedge \bigwedge \Gamma$ is then passed to *CAIC* and the variable ordering is adapted using *vars in abstraction first*, as described in section 3.3.

RIP is aimed in particular at NRA formulas such as in example 5.5.1, in which several constraints share common factors.

Example 5.5.1.

$$\begin{aligned} \varphi := & - \underbrace{(x^6y + x^5 + y^{13} + x^7y^2)}_{new_0} \underbrace{(x^{27} + y^{12} + x^5y^2 + y^{11} + y^7)}_{new_1} \underbrace{(-x^2y^6 - x^8 - y^{10})}_{new_2} < 0 \\ & \wedge \underbrace{(x^6y + x^5 + y^{13} + x^7y^2)}_{new_0} \leq 0 \wedge \underbrace{(x^{27} + y^{12} + x^5y^2 + y^{11} + y^7)}_{new_1} \leq 0. \end{aligned}$$

The resulting abstraction is

$$\hat{\varphi} := -new_0 new_1 new_2 < 0 \wedge new_0 \leq 0 \wedge new_1 \leq 0.$$

The abstraction on its own is not yet unsatisfiable in this case. To determine unsatisfiability, the original variables must also be assigned so that the substitution constraint associated with new_2 becomes active and can be taken into account in the projection.

Nevertheless, RIP leads to a significantly better performance in this example: With RIP, the largest *sotd* of the polynomials occurring in the projection is 1023. Using *CAIC* without abstraction, the *sotd* could not be calculated because even 70 GB of swap memory was not enough.

Figure 5.7 compares the performance of *best product substitution*, RIP, and *CAIC* without abstraction (*noAbst*).

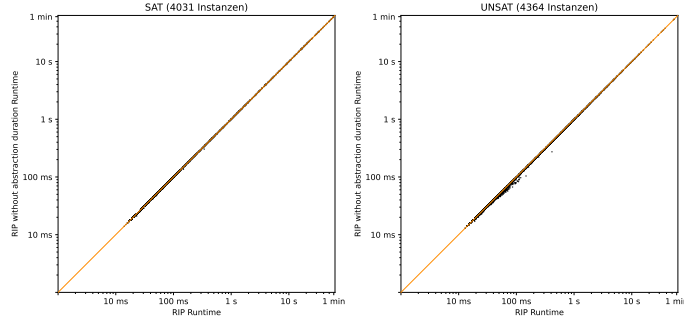


Figure 5.8: Evaluation of the abstraction duration. Each point corresponds to one formula from the benchmark set; most of the points are barely visible as they lie on the diagonal.

RIP lags far behind *best product substitution* overall. The main reason for this is that in most instances of the benchmark, there are hardly any reducible polynomials. Almost 3,000 of the approximately 12,000 formulas contain only irreducible polynomials. In such cases, RIP has to replace the entire original polynomials with fresh variables. Of course, this means that no structure of the input formula is exploited, and the substitutions introduced merely cause overhead.

The time required for performing the abstraction itself is very short and does not contribute to the poor performance. Figure 5.8 compares the standard RIP with a variant in which the abstraction time has been factored out. The points lie pretty much on the diagonal, which shows that the time needed for calculating the abstraction is negligible.

5.5.2 Substituting Whole Polynomials (SWP)

The second alternative substitution strategy, substituting whole polynomials (SWP), is more flexible than RIP. It aims to divide smaller polynomials from larger polynomials and then substitute them.

Algorithm

1. **Initialization.** Let $\mathcal{P} = (p_0, \dots, p_{m-1})$ be a sorted list of the polynomials of the set $\mathcal{P}(\varphi)$. The list is sorted in ascending order by `sotd`. In case of a tie, the graded-lexicographical order decides. Thus, p_0 is the smallest element.
2. **Outer loop** ($i = 0, \dots, m - 1$)
 - (a) Initialize a flag `replaced := false`.
 - (b) **Inner loop** ($j = i + 1, \dots, m - 1$)
 - i. Perform the polynomial division with remainder $p_j = qp_i + r$ (over $\mathbb{Q}[x_1, \dots, x_n]$).
 - ii. **Replacement condition.** If `sotd( $qnew_k + r$ ) < sotd( $p_j$ )`, then substitute p_j with $qnew_k + r$. If not already done, introduce the fresh variable new_k for p_i and set `replaced := true`.

- iii. **Recursive division of the quotient.** As long as the further division of the quotient q by p_i lowers the sotd again, replace these occurrences accordingly.
- (c) **Self substitution.** If at least one substitution was performed in the inner loop ($replaced=true$), then also substitute p_i with new_k .

For each substitution $p_i \mapsto new_k$, we add the substitution constraint $p_i = new_k$ to the set Γ . CAIC finally receives the formula $\hat{\varphi} \wedge \bigwedge \Gamma$.

To make the strategy more tangible, consider example 5.5.2.

Example 5.5.2 (SWP). *Let*

$$\varphi := x^4y^2 + x^2y^3 + xy < 0 \wedge -x^3y - xy^2 < 0 \wedge -xy < 0.$$

Sorted polynomial list

$$\mathcal{P} = (p_0, p_1, p_2) = (-xy, -x^3y - xy^2, x^4y^2 + x^2y^3 + xy).$$

Outer loop $i = 0$ ($p_0 = -xy$)

$j = 1$: $p_1 = (y + x^2)p_0$. Since $\text{sotd}((y + x^2) new_0) = 4 < 7 = \text{sotd}(p_1)$, we perform the substitution $p_1 := (y + x^2) new_0$ and set $replaced=true$.

$j = 2$: $p_2 = q_1 p_0$ with $q_1 = -1 - x^3y - xy^2$. Because $\text{sotd}(q_1 new_0) = 10 < 13 = \text{sotd}(p_2)$, we could substitute p_2 with $q_1 new_0$. But because subsequent division of q_1 by p_0 yields $q_1 = (y + x^2)p_0 - 1$, causing the sotd to decrease again, we finally substitute

$$p_2 := ((y + x^2) new_0 - 1) new_0 = -new_0 + y new_0^2 + x^2 new_0^2.$$

Since $replaced=true$, we also substitute $p_0 := new_0$.

Updated polynomial list

$$\mathcal{P} = (p_0, p_1, p_2) = (new_0, y new_0 + x^2 new_0, -new_0 + y new_0^2 + x^2 new_0^2).$$

Outer loop $i = 1$ ($p_1 = y new_0 + x^2 new_0$)

$j = 2$: $p_2 = (new_0)p_1 - new_0$. Since $\text{sotd}(new_0 new_1 - new_0) = 3 < 8 = \text{sotd}(p_2)$, we substitute $p_2 := new_0 new_1 - new_0$ and in the following $p_1 := new_1$.

Final polynomial list

$$\mathcal{P} = (p_0, p_1, p_2) = (new_0, new_1, new_0 new_1 - new_0).$$

Abstraction We obtain the abstraction

$$\hat{\varphi} := new_0 new_1 - new_0 < 0 \wedge new_1 < 0 \wedge new_0 < 0$$

and the substitution constraints

$$\Gamma = \{new_0 = -xy, new_1 = y new_0 + x^2 new_0\}.$$

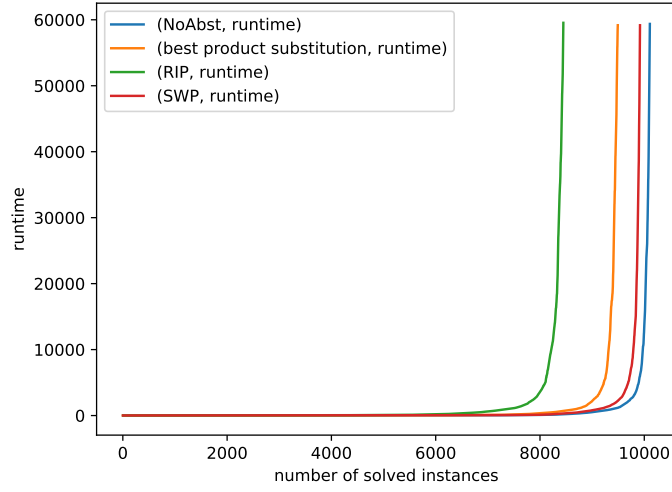


Figure 5.9: Evaluation of SWP

In this example 5.5.2, $\hat{\varphi}$ on its own is already unsatisfiable. The substitution constraints from Γ therefore do not even need to be activated.

RIP fails with this example, but SWP solves it effortlessly. The best-case example for RIP 5.5.1 is also mastered by SWP, as it generates the unsatisfiable abstraction $\hat{\varphi} := new_0 new_1 (-x^2 y^6 - x^8 - y^{10}) > 0 \wedge new_0 \leq 0 \wedge new_1 \leq 0$. If the multiplications between the irreducible factors in example 5.5.1 are replaced by additions, SWP can also handle this, in contrast to RIP.

As shown in Figure 5.9, SWP significantly outperforms RIP on the benchmark set and also performs better than *best product substitution*. Nevertheless, no substitution was possible with SWP in 4,667 of the $\approx 12,000$ instances of the benchmark. In the remaining instances, only 32.5 fresh variables were introduced on average. A large part of the good performance is therefore due to the fact that there have not been many substitutions. The performance of SWP is slightly behind that of CAIC without abstraction. As with RIP, the time needed for calculating the abstraction does not have a significant impact on the runtime.

Chapter 6

Conclusion

6.1 Summary

The paper by Cimatti et al. [CGI⁺17] discussed in section 2.4 presents the method of *Incremental Reduction to LRA with EUF* and achieves remarkable results. Motivated by this success, we attempted to eliminate its weakness of being incomplete by using a complete NRA solver.

Since we did not have a complete, incrementally working NRA solver available, we used the CAIC method [NK24]. Although CAIC itself does not work incrementally either, we can adapt the variable ordering to achieve a stepwise activation of the substitution constraints.

Inspired by [CGI⁺17], we started by substituting individual multiplications. Initial results were disappointing, but could be significantly improved by optimization. Nevertheless, the substitution of individual multiplications still lagged behind CAIC without abstraction.

In the following step, we no longer substituted individual products but entire polynomials. Here too, we were still unable to exceed the performance of CAIC without abstraction.

6.2 Discussion and Future Work

The use of CAIC with an adapted variable ordering has a major weakness: As long as a substitution constraint is not yet active, the corresponding fresh variable can take on any value in the model. Even a single such fresh variable is often enough to “defuse” an unsatisfiable problem and make it satisfiable.

We originally planned to add a selection of the linear axioms from chapter 2.4 to Γ right at the beginning, thereby ensuring essential properties of the multiplication. This would prevent the fresh variables from taking on arbitrary values before the activation of the corresponding substitution constraint. Additionally, we would like to add linear axioms that ensure functional congruence of the fresh variables so that they behave like uninterpreted functions.

Unfortunately, this is not possible with CAIC. The linear axioms should be active from the start, or at least before the substitution constraints. Meaningful linear axioms inevitably contain all the variables that also occur in the corresponding sub-

stitution constraint. In order to activate a linear axiom, it is necessary for those variables to already be assigned. However, this would immediately also activate the substitution constraint.

Therefore, a promising approach for future work is to use a complete NRA solver that actually works incrementally. This would make it possible that the linear axioms are active from the very beginning, while the nonlinear substitution constraints are added successively in a controlled manner. We expect that such an approach could reduce the performance losses and turn out to be more efficient than solving without abstraction.

Bibliography

- [Ábr24] Erika Ábrahám. Satisfiability checking, 2024. Lecture Notes, RWTH Aachen University, Winter Semester 2024/2025.
- [ÁDEK21] Erika Ábrahám, James H Davenport, Matthew England, and Gereon Kremer. 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.
- [CGI⁺17] Alessandro Cimatti, Alberto Griggio, Ahmed Irfan, Marco Roveri, and Roberto Sebastiani. Invariant checking of NRA transition systems via incremental reduction to LRA with EUF. In *Tools and Algorithms for the Construction and Analysis of Systems: 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part I* 23, pages 58–75. Springer, 2017.
- [CJ12] Bob F Caviness and Jeremy R Johnson. *Quantifier elimination and cylindrical algebraic decomposition*. Springer Science & Business Media, 2012.
- [CKJ⁺15] Florian Corzilius, Gereon Kremer, Sebastian Junges, Stefan Schupp, and Erika Ábrahám. Smt-rat: an open source c++ toolbox for strategic and parallel smt solving. In *International conference on theory and applications of satisfiability testing*, pages 360–368. Springer, 2015.
- [DNSU23] James Harold Davenport, Akshar Sajive Nair, Gregory Kumar Sankaran, and Ali Kemal Uncu. Lazard-style CAD and equational constraints. In *Proceedings of the 2023 International Symposium on Symbolic and Algebraic Computation*, pages 218–226, 2023.
- [Gro24] Martin Grohe. Mathematical logic, 2024. Lecture Notes, RWTH Aachen University, Summer Semester 2024.
- [Kre20] Gereon Kremer. *Cylindrical algebraic decomposition for nonlinear arithmetic problems*. PhD thesis, Dissertation, RWTH Aachen University, 2020.
- [NK24] Jasper Nalbach and Gereon Kremer. Extensions of the cylindrical algebraic covering method for quantifiers. *arXiv preprint arXiv:2411.03070*, 2024.

-
- [PdRAEC24] Lynn Pickering, Tereso del Río Almajano, Matthew England, and Kelly Cohen. Explainable ai insights for symbolic computation: a case study on selecting the variable ordering for cylindrical algebraic decomposition. *Journal of Symbolic Computation*, 123:102276, 2024.
- [PN25] Valentin Promies and Jasper Nalbach. Benchmax, 2025.
- [PSB⁺24] Mathias Preiner, Hans-Jörg Schurr, Clark Barrett, Pascal Fontaine, Aina Niemetz, and Cesare Tinelli. SMT-LIB release 2024 (non-incremental benchmarks), April 2024.
- [Spe23] Philippe Specht. Improving incremental linearization for satisfiability modulo non-linear real arithmetic checking, 2023.