Seminar Report

Non-Linear Arithmetic

SAT Modulo Linear Arithmetic for Solving Polynomial Constraints

Simon Legner
`Simon.Legner@student.uibk.ac.at`

January 27, 2012

Supervisor: Dr. Harald Zankl

Abstract

We discuss a transformation procedure for solving non-linear arithmetic with existing SMT-solvers for linear arithmetic. This report is based on an article by (Borralleras et al., 2012). To eliminate non-linear variable multiplications, a case analysis is applied on the range of bounded variables. We also discuss an approach for dealing with unbounded variables. For both approaches, we prove their correctness.

Contents

1 Introduction

Motivation. Satisfiability modulo theory (SMT) enhances propositional logic with a fragment of first-order logic, e.g., equality logic, (non-)linear arithmetic over $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$ or $\mathbb{R}$, bit-vectors, arrays, etc.

In the context of this report, we deal with non-linear arithmetic. This logic is capable of handling polynomial constraints, which play an important role in several areas of hardware and software verification. For instance, polynomial constraints arise in termination proofs of term rewrite systems [FGM$^+$07, CMTU05], or in hybrid system verification [LPY99, SSM08]. However, non-linear arithmetic over the naturals (Peano arithmetic) is undecidable, and non-linear arithmetic over the reals is practically unfeasible due to the complexity of its decision procedures [Tar51, Col75].

In contrast, linear arithmetic over the naturals (Pressburger arithmetic) is decidable. Despite the fact that linear programming is NP-complete, effective solvers exist. In this report we discuss a transformation from non-linear arithmetic to linear arithmetic. From theory, we know in advance that this transformation has to be incomplete, i.e., some problems cannot be solved. Nevertheless, experiments reveal that this approach is useful for practical examples.

Organization of this Report. This report is structured as follows. In Section 2 we fix preliminaries on the problem context. We present the transformation in Section 3, including correctness statements as well as comprehensible examples. In Section 4 we formulate an abstract problem to increase efficiency of an implementation, and in Section 5 we sketch possible extensions. Section 6 reports on experimental results, and we conclude in Section 7.

2 Preliminaries

In this section we will briefly summarize some basic concepts of (non-)linear arithmetic. In the sequel, we define the syntax, fix some conventions, and mention underlying decision procedures.

Arithmetic. In the sequel, we deal with (non-)linear arithmetic, i.e., boolean constraints built from arithmetic expressions. Variables and numeric constants form the smallest expressions. Larger expressions are built when inductively combining them via arithmetic operations $(+, -, \cdot)$. To yield boolean atoms, expressions are compared to each other using standard relations $(=, >)$. For boolean constraints, the standard boolean connectives $(\wedge, \vee, \neg)$ are defined. Figure 1 shows the complete arithmetic grammar.

Convention. We use a, b, c, d, e, u, v, w, x, y, z (possibly indexed) as variables. We denote the variables of a constraint C by $\mathcal{V}\mathsf{ar}(C)$.

Additionally, we use $x \to y$ as abbreviation for $\neg x \vee y$, $x \neq y$ as abbreviation for $\neg(x = y)$, $x \geqslant y$ as abbreviation for $(x = y) \vee (x > y)$, $x < y$ as abbreviation for $y > x$, and $x < y < z$ as abbreviation for $(x < y) \wedge$

$$\begin{aligned}
\mathit{constr} &::= \mathit{atom} \mid (\mathit{constr} \land \mathit{constr}) \mid (\mathit{constr} \lor \mathit{constr}) \mid (\neg\, \mathit{constr}) \\
\mathit{atom} &::= (\mathit{expr} > \mathit{expr}) \mid (\mathit{expr} = \mathit{expr}) \\
\mathit{expr} &::= \mathit{variable} \mid \mathit{constant} \mid \\
&\qquad (\mathit{expr} + \mathit{expr}) \mid (\mathit{expr} - \mathit{expr}) \mid (\mathit{expr} \cdot \mathit{expr})
\end{aligned}$$

Figure 1: Grammar for valid arithmetic constraints.

$(y < z)$. We consider exponentiation with positive naturals as exponents as a sequence of multiplications. For brevity, we sometimes use the juxtaposition xy for $x \cdot y$.

To simplify notation, we consider $+$, $-$, $\cdot$, $\land$, $\lor$ to be left associative. We stick to the following precedence on connectives and spare parenthesis. Here, "$\sqsupset$" stands for *binds stronger than.*

$$\{\cdot\} \sqsupset \{+,-\} \sqsupset \{=,\neq,\leqslant,\geqslant,<,>\} \sqsupset \{\neg\} \sqsupset \{\lor,\land\} \sqsupset \{\rightarrow\}$$

Example 2.1. A correct arithmetic constraint w.r.t. the above conventions is $(0 \leqslant y \leqslant 2) \land (x = y^2 z)$, which represents $(((y = 0) \lor (y > 0)) \land ((y = 2) \lor (2 > y))) \land (x = ((y \cdot y) \cdot z))$ w.r.t. the grammar of Figure 1.

Monomial. A *monomial* is an arithmetic expression only consisting of multiplications. We write M (or $M_1, \ldots$) for monomials, and we typically denote the multiplications using the shorter exponent notation. A monomial is *linear* if it consists only of a single factor (e.g., $M_1 = x$), or it is a single variable multiplied with a constant (e.g., $M_2 = 5x$). Whenever a monomial contains a multiplication of two or more variables, it is *non-linear* (e.g., $M_3 = x^2$ or $M_4 = xy$). We say that a constraint is linear if all of its monomials are linear.

A variable v is *bounded* if the constraint contains conjuncts specifying a lower and an upper bound of v (e.g., $(0 < v) \land (v < 24)$).

Underlying Domain. Arithmetic constraints may be interpreted over various domains, including $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$ and $\mathbb{R}$. In the following, we will mainly deal with the integers ($\mathbb{Z}$). In Section 5 we sketch how fractions can be incorporated in this approach.

Existential Quantification. In the context of this work, we are interested in existentially quantified formulas, i.e., a formula is virtually prefixed with existential quantifiers for each occurring variable. When solving such a formula, we are interested in finding a satisfying assignment for the variables, or, stating that no solution exists if the formula is unsatisfiable.

Example 2.2. The formula from Example 2.1 is satisfiable as indicated by the model $y = 1$ and $x = z = 2$.

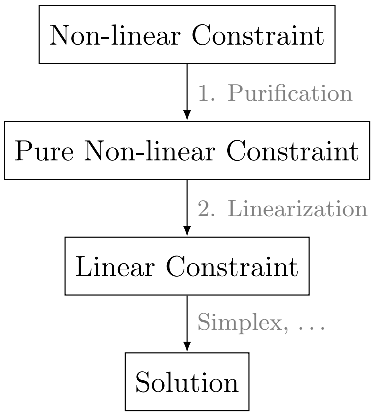

Figure 2: Visualization of transformation procedure.

3 Transformation

The main contribution of [BLO+12] is a transformation procedure from non-linear arithmetic to linear arithmetic. In this section we will review this transformation. If sufficiently many variables are bounded, the transformation is sound and complete, i.e., the original constraint is satisfiable if and only if the transformed constraint is (cf. Section 3.4). In the other case, auxiliary bounds can be added, but then the transformation might produce unsatisfiable formulas although the input was satisfiable (cf. Section 3.5).

Example 3.1 (Introductory Example)**.** We start with an introductory example to sketch the idea of the transformation. Consider the non-linear constraint $(0 \leqslant y \leqslant 2) \wedge (x = y^2 \cdot z)$. Note that the non-linearity is due to the monomial $y^2 \cdot z$. As y is bounded from below and above, we can get rid of the non-linearity in the following way: $(0 \leqslant y \leqslant 2) \wedge (y = 0 \to x = 0) \wedge (y = 1 \to x = z) \wedge (y = 2 \to x = 4 \cdot z)$. We applied a case analysis on all possible values of y.

Two-phase Transformation. In the sequel, we present the two phases of the transformation: *Purification* basically separates the linear part from the non-linear part of a constraint. *Linearization* applies the case analysis on bounded variables as sketched above. If successful, the result of this transformation is a linear constraint which can be solved with simplex-based SMT-solvers. This transformation is visualized in Figure 2.

3.1 Purification

The purification phase separates the non-linear part from the linear part of a constraint. We define this intermediate result formally in terms of a pure non-linear constraint.

Pure Non-linear Constraint A pure non-linear constraint is a constraint of the shape $L \wedge \bigwedge_i (y_i = M_i)$ where L is a linear constraint, y_i are variables, and

M_i are non-linear monomials. As the non-linear part is restricted to this form, pattern matching in the second phase is simplified.

Example 3.2. For instance, $(0 \leqslant y \leqslant 2) \wedge (x = y^2 \cdot z)$ is a pure non-linear constraint. In contrast, $(0 \leqslant y \leqslant 2) \wedge (x < y^2 \cdot z)$ and $x \cdot (y + y \cdot z) \geqslant 0$ are not purely non-linear.

Every non-linear constraint can be transformed into an equisatisfiable pure non-linear constraint: Firstly, apply the distributive law (multiplication distributes over addition/subtraction). Secondly, replace every non-linear monomial M_i by a fresh variable y_i, and add a side constraint $y_i = M_i$.

Example 3.3. We consider the non-pure non-linear constraint $x \cdot (y + y \cdot z) \geqslant 0$. Applying the distributive law yields $x \cdot y + x \cdot y \cdot z \geqslant 0$. We introduce fresh variables for the non-linear monomials $x \cdot y$ and $x \cdot y \cdot z$, and obtain the pure non-linear constraint $(y_1 + y_2 \geqslant 0) \wedge (y_1 = x \cdot y) \wedge (y_2 = x \cdot y \cdot z)$. The first conjunct corresponds to L in the above definition, and the conjuncts two and three correspond to $y_i = M_i$.

Similar to a potential exponential growth of propositional formulas during a transformation to conjunctive normal form (CNF), arithmetic constraints may become exponentially larger when applying the distributive law. To avoid this issue, one can adapt the idea of the Tseitin transformation [Tse70]: Instead of multiplying out, introduce a fresh variable for the addition/subtraction expression.

Example 3.4. We show the alternative approach on the previous example. In the constraint $x \cdot (y + y \cdot z) \geqslant 0$, we replace the sum $y + y \cdot z$ by a fresh variable v, and add a side constraint, which results in $(x \cdot v \geqslant 0) \wedge (v = y + y \cdot z)$. This constraint contains two non-linear monomials $x \cdot v$ and $y \cdot z$ which are — as explained before — replaced by fresh variables y_1 and y_2, respectively. This yields the pure non-linear constraint $(y_1 \geqslant 0) \wedge (v = y + y_2) \wedge (y_1 = x \cdot v) \wedge (y_2 = y \cdot z)$.

As the following phase (linearization) requires bounded variables, the Tseitin-like transformation is not useful if it gives rise to non-linear monomials without bounds. Thus, this transformation is only applied if all but the fresh variable in the monomial are bounded (x in the example), or if bounds for the fresh variable can be computed (in the example, the bounds for v can be computed if y and z are bounded).

The result of the purification phase is a pure non-linear constraint, which is required as input for the seconds phase.

3.2 Linearization

The linearization phase transforms a pure non-linear constraint into a linear constraint, which can then be solved with effective SMT-solvers like simplex-based algorithms.

We show the linearization steps in terms of inference rules that perform a one-step transformation of a pure non-linear constraint. We denote the lower (upper) bounds of a variable v by $\mathcal{L}_v$ ($\mathcal{U}_v$). All rules have in common that they replace a bounded variable v by a case analysis on the possible values of v. Note that we deal with the domain $\mathbb{Z}$ here.

$$\frac{C \wedge x = v^p}{C \wedge \bigwedge_{\alpha=\mathcal{L}_v}^{\mathcal{U}_v} (v = \alpha \rightarrow x = \alpha^p)} \quad \text{if } \mathcal{L}_v \leqslant v \leqslant \mathcal{U}_v \text{ and } p > 1 \tag{1}$$

$$\frac{C \wedge x = v^p \cdot w}{C \wedge \bigwedge_{\alpha=\mathcal{L}_v}^{\mathcal{U}_v} (v = \alpha \rightarrow x = \alpha^p \cdot w)} \quad \text{if } \mathcal{L}_v \leqslant v \leqslant \mathcal{U}_v \tag{2}$$

The first two rules replace a non-linear equality by a number of linear implications. Note that α^p are constants that are computed at transformation time.

$$\frac{C \wedge x = v^p \cdot M}{C \wedge \bigwedge_{\alpha=\mathcal{L}_v}^{\mathcal{U}_v} (v = \alpha \rightarrow x = \alpha^p \cdot x_M) \wedge x_M = M} \quad \begin{array}{l}\text{if } \mathcal{L}_v \leqslant v \leqslant \mathcal{U}_v \text{ and} \\ M \text{ not linear}\end{array} \tag{3}$$

The third rule transforms arbitrarily large monomials by replacing one variable by a case analysis. For efficiency reasons, a fresh variable x_M is introduced for the remaining non-linear monomial M outside of the added implications. For that reason, M needs to be transformed only once.

Example 3.5. We review the introductory example (Example 3.1) to familiarize the reader with the notation of the linearization rules. The constraint of interest is $(0 \leqslant y^2 \leqslant 2) \wedge (x = y \cdot z)$. From the first conjunct we derive the lower bound $\mathcal{L}_y = 0$ and the upper bound $\mathcal{U}_y = 2$ of variable y. Rule (2) is applicable by matching x, v, p, w in the rule with x, y, 2, z in the constraint, respectively. Applying the rule yields $(0 \leqslant y \leqslant 2) \wedge \bigwedge_{\alpha=0}^{2} (y = \alpha \rightarrow x = \alpha^2 \cdot z)$ which expands to the linear constraint $(0 \leqslant y \leqslant 2) \wedge (y = 0 \rightarrow x = 0) \wedge (y = 1 \rightarrow x = z) \wedge (y = 2 \rightarrow x = 4 \cdot z)$.

Before showing some correctness results, we illustrate the two-phase transformation on a larger example.

3.3 Example

Example 3.6. Let

$$B = (0 \leqslant a \leqslant 2) \wedge (0 \leqslant c \leqslant 2) \wedge (0 \leqslant d \leqslant 2)$$

and consider the non-linear constraint

$$F = B \wedge (2a^3b - 5cd^2e \geqslant 0).$$

We now show a transformation of F to a linear arithmetic constraint. Purification introduces fresh variables for the non-linear monomials a^3b and cd^2e, yielding the pure non-linear constraint

$$F' = B \wedge (2y_1 - 5y_2 \geqslant 0) \wedge (y_1 = a^3b) \wedge (y_2 = cd^2e).$$

We now perform the linearization. Firstly, using linearization rule (2), $y_1 = a^3b$ is transformed into

$$\bigwedge_{\alpha=0}^{2}(a = \alpha \to y_1 = \alpha^3 b).$$

Secondly, rule (3) transforms $y_2 = cd^2e$ into[1]

$$\bigwedge_{\gamma=0}^{2}(c = \gamma \to y_2 = \gamma y_3) \wedge y_3 = d^2e.$$

Finally, with an application of rule (2), $y_3 = d^2e$ is linearized to

$$\bigwedge_{\delta=0}^{2}(d = \delta \to y_3 = \delta^2 e).$$

The result of this transformation is the linear arithmetic constraint

$$\begin{aligned} G = B \wedge (2y_1 - 5y_2 \geqslant 0) \wedge \left(\bigwedge_{\alpha=0}^{2}(a = \alpha \to y_1 = \alpha^3 b)\right) \wedge \\ \left(\bigwedge_{\gamma=0}^{2}(c = \gamma \to y_2 = \gamma y_3)\right) \wedge \left(\bigwedge_{\delta=0}^{2}(d = \delta \to y_3 = \delta^2 e)\right). \end{aligned}$$

To illustrate the linearity of G, we fully write out the constraint by unfolding the conjunction operators:

$$\begin{aligned} G = {} & (0 \leqslant a \leqslant 2) \wedge (0 \leqslant c \leqslant 2) \wedge (0 \leqslant d \leqslant 2) \wedge (2y_1 - 5y_2 \geqslant 0) \wedge \\ & (a = 0 \to y_1 = 0) \wedge (a = 1 \to y_1 = b) \wedge (a = 2 \to y_1 = 8b) \wedge \\ & (c = 0 \to y_2 = 0) \wedge (c = 1 \to y_2 = y_3) \wedge (c = 2 \to y_2 = 2y_3) \wedge \\ & (d = 0 \to y_3 = 0) \wedge (d = 1 \to y_3 = e) \wedge (d = 2 \to y_3 = 4e) \end{aligned}$$

3.4 Correctness

In this section we will address theorems proving the correctness of the presented transformation. We will focus on the following questions:

- ⋆ Does the transformation produce correct results?
- ⋆ Does the transformation terminate?
- ⋆ How much larger do the constraints get during transformation?
- ⋆ Does the transformation always produce a linear constraint?

Lemma 3.7 (Correctness). *Let F be a pure non-linear constraint and G the result of the transformation, which is obtained by exhaustively applying the transformation rules* (1)–(3) *on F. The constraints F and G are equisatisfiable, i.e., F is satisfiable if and only if G is satisfiable.*

Proof Sketch. One has to prove that each of the linearization rules retains satisfiability. This is done by showing that a satisfying assignment of the constraint before the application of the rule can be transformed into a satisfying assignment of the constraint after the application of the rule, and vice versa. It is not difficult to see that this holds for all three rules. As we only give the proof idea, we refer the reader to [BLO+12] for details. □

[1]We remark that in this step also e.g. d^2 could be linearized. How this choice affects the resulting formula is discussed in Section 4.

Lemma 3.8 (Termination)**.** *The transformation procedure is guaranteed to terminate on all arithmetic constraints.*

Proof Sketch. This is a result from the fact that each rule either reduces the number of non-linear monomials or reduces the size of one non-linear monomial. Rules (1) and (2) remove one non-linear equality and rule (3) reduces one non-linear monomial at least by one multiplication. □

Lemma 3.9 (Complexity)**.** *Let the pure non-linear constraint F be of the form $L \wedge N$ where L (N) is the (non-)linear part of F. Let the linearization produce a constraint G of the form $L \wedge H$. Then the number of conjuncts in H is $\mathcal{O}(|N| \cdot M \cdot K)$ where $|N|$ equals the number of conjuncts in N, M is the maximal number of multiplications in one constraint, and $K = \max_i\{\mathcal{U}_i - \mathcal{L}_i\}$.*

Proof Sketch. Firstly, note that the linearization preserves the linear part L. Every application of the transformation rules adds at most K conjuncts, and every rule reduces either the number of non-linear monomials or the size of a non-linear monomial. □

Lemma 3.10 (Result)**.** *The outcome of the linearization, i.e., the constraint G, is either linear or for every a non-linear monomial no variable is bounded, i.e., every variable lacks lower or upper bound.*

Proof Sketch. This follows from the fact that the case analysis of the transformation rules is exhaustive with respect to the shape of the non-linear monomial. Thus, a further applicability of a rule may only be prohibited when the side constraint concerning the bounds of a variable is not fulfilled. □

We remark that Lemma 3.10 refines [BLO$^+$12, Lemma 4], which states that *there is* a non-linear monomial of which no variable is bounded.

As non-linear constraints as result of the transformation would prevent using linear solvers, we show a way to also linearize those in the next section by sacrificing equisatisfiability of the transformed formula.

3.5 Artificial Bounds and Unsatisfiability

As indicated in the previous section, linearization may fail in the sense that no linear constraint is generated. This is due to a lack of bounds of variables. To overcome this problem, and hence to cope with such cases, one can introduce "*artificial bounds*" for unbounded variables.

More formally, a pure non-linear constraint F is enhanced by artificial bounds for unbounded variables (denoted by B), yielding a constraint $B \wedge F$. Applying the linearization results in a linear constraint G, which is known to be equisatisfiable to $B \wedge F$ by Lemma 3.7.

Satisfiability. Adding bounds changes the constraint. Thus, one is interested in how the satisfiability of G (or $B \wedge F$) relates to the satisfiability of the original constraint F. As the bounds are a restriction of the original constraint, satisfiability of $B \wedge F$ implies satisfiability of F.

Concerning unsatisfiability of G, the relation is not clear as the it may either be due to the unsatisfiability of F or due to too restrictive bounds B. To resolve this situation if G is unsatisfiable, we inspect the *unsatisfiable core* $\mathcal{C}_G$ of G, i.e., an unsatisfiable subset of the conjuncts of a constraint G.

Lemma 3.11. *If G is unsatisfiable and if the artificial bounds B do not contain a variable of $\mathcal{C}_G$, then we conclude unsatisfiability of F. In formal notation, this statement reads as follows:* $\neg\mathsf{SAT}(G) \wedge \mathcal{V}\mathsf{ar}(\mathcal{C}_G) \cap \mathcal{V}\mathsf{ar}(B) = \varnothing \Longrightarrow \neg\mathsf{SAT}(F)$.

Proof Sketch. This is a corollary of Lemma 11 in [BLO+12], which states that for a given unsatisfiable core $\mathcal{C}_G$ of G, there exists an unsatisfiable core $\mathcal{C}_{B\wedge F}$ of the constraint $B \wedge F$, such that $\mathcal{V}\mathsf{ar}(\mathcal{C}_{B\wedge F}) \cap \mathcal{V}\mathsf{ar}(B) = \mathcal{V}\mathsf{ar}(\mathcal{C}_G) \cap \mathcal{V}\mathsf{ar}(B)$. By assumption, $\mathcal{V}\mathsf{ar}(\mathcal{C}_G) \cap \mathcal{V}\mathsf{ar}(B) = \varnothing$, and therefore $\mathcal{V}\mathsf{ar}(\mathcal{C}_{B\wedge F}) \cap \mathcal{V}\mathsf{ar}(B) = \varnothing$. Hence, $\mathcal{V}\mathsf{ar}(\mathcal{C}_{B\wedge F}) \subseteq \mathcal{V}\mathsf{ar}(F)$ and F is unsatisfiable. □

Given the unsatisfiability of G, if we cannot conclude unsatisfiability of F, then the unsatisfiable core $\mathcal{C}_G$ indicates which bounds of B might be too strict. After relaxing the bounds to B', the process can be iterated, i.e., performing linearization on $B \wedge F'$ again, yielding G', and checking for satisfiability of G'.

This approach relies on the ability of the used linear arithmetic SMT-solver to extract unsatisfiable cores. The list of solvers providing this feature includes Yices [DdM06] and MathSAT [BCF+08].

We illustrate the unsatisfiability approach on several examples.

Example 3.12 (Derive Unsatisfiability)**.** Consider the constraint

$$F = (a^2 > 3) \wedge (b^2 < 3) \wedge (2 \leqslant b \leqslant 4).$$

This constraint is obviously unsatisfiable since b is at least 2, but b^2 should be less than 3. Purification yields

$$F' = (y_1 > 3) \wedge (y_2 < 3) \wedge (2 \leqslant b \leqslant 4) \wedge (y_1 = a^2) \wedge (y_2 = b^2).$$

But actually the bounds of a are irrelevant since unsatisfiability is purely due to the constraints involving b. To see this, we perform the transformation: To linearize the constraint F', we add artificial bounds B. Let some heuristics produce $B = (0 \leqslant a \leqslant 2)$ as bounds. Linearization transforms $B \wedge F'$ into the linear constraint G.

$$\begin{aligned} G = (y_1 > 3) \wedge (y_2 < 3) \wedge (0 \leqslant a \leqslant 2) \wedge (2 \leqslant b \leqslant 4) \wedge \\ \textstyle\bigwedge_{\alpha=0}^{2}(a = \alpha \to y_1 = \alpha^2) \wedge \bigwedge_{\beta=2}^{4}(b = \beta \to y_2 = \beta^2) \end{aligned}$$

An SMT-solver derives unsatisfiability of G together with an unsatisfiable core $\mathcal{C}_G$ with $\mathcal{V}\mathsf{ar}(\mathcal{C}_G) = \{b, y_2\}$. As $\mathcal{V}\mathsf{ar}(B) \cap \mathcal{V}\mathsf{ar}(\mathcal{C}_G) = \{a\} \cap \{b, y_2\} = \varnothing$, we derive unsatisfiability of F.

Example 3.13 (Relax Bounds)**.** We modify the constraint from the previous example to illustrate how the unsatisfiable core can be used to relax artificial bounds. Consider

$$F = (a^2 > 10) \wedge (b < 3) \wedge (2 \leqslant b \leqslant 4),$$

which is purified to

$$F' = (y_1 > 10) \wedge (b < 3) \wedge (2 \leqslant b \leqslant 4) \wedge (y_1 = a^2).$$

Let $B = (0 \leqslant a \leqslant 2)$ as in the previous example. The linearized constraint

$$G = (0 \leqslant a \leqslant 2) \wedge (y_1 > 10) \wedge (b < 3) \wedge (2 \leqslant b \leqslant 4) \wedge \textstyle\bigwedge_{\alpha=0}^{2} (a = \alpha \rightarrow y_1 = \alpha^2)$$

is unsatisfiable and $\mathcal{V}\mathsf{ar}(\mathcal{C}_G) = \{a, y_1\}$. As $\mathcal{V}\mathsf{ar}(B) \cap \mathcal{V}\mathsf{ar}(\mathcal{C}_G) = \{a\} \cap \{a, y_1\} \neq \varnothing$, we cannot derive unsatisfiability of F. Instead, by inspecting $\mathcal{C}_G$, we let the heuristic relax the artificial bounds to $B' = (-2 \leqslant a \leqslant 4)$. This gives rise to a different linear constraint

$$G' = (-2 \leqslant a \leqslant 4) \wedge (y_1 > 10) \wedge (b < 3) \wedge (2 \leqslant b \leqslant 4) \wedge \textstyle\bigwedge_{\alpha=-2}^{4} (a = \alpha \rightarrow y_1 = \alpha^2),$$

which turns out to be satisfiable. Therefore, we conclude satisfiability of F.

Example 3.14 (Loop)**.** The constraint

$$F = (a^2 < 0) \wedge (b < 3) \wedge (2 \leqslant b \leqslant 4)$$

is unsatisfiable, but it cannot be solved with the presented approach: Any artificial bounds B for a needed for linearization will not satisfy the constraint and the unsatisfiable core will always contain the variable a. Therefore, the presented approach loops and does not solve this constraint. Hence in contrast to [BLO$^+$12], we suggest to inspect the unsatisfiable core itself and not its variables.

Summary of this Section. In this section we reviewed the transformation process presented in [BLO$^+$12], i.e., the two phases purification and linearization, and proved their correctness. Ultimately, we recalled a way to linearize constraints where not sufficiently many variables are bounded.

4 Implementation Aspects

When applying the linearization rules from Section 3.2, one has to make many choices concerning the variable v. For rule (3), these choices directly influence the left-over monomial. For an efficient implementation, it is important to meet these decisions smartly in order to reduce the number of (intermediate) monomials, and, thus, get a small linear constraint, which is typically preferred by the underlying linear solver.

In this section we will show that the corresponding decision problem is NP-complete. Thus, one has to rely on good heuristics instead of following the optimal way.

Closed Set. A set of monomials S is said to be a *closed set* if two conditions are met: Firstly, a given initial set S_0 is a subset of S ($S_0 \subseteq S$). Secondly, for every monomial M in the set S, either $M = v^p$, or $M = v^p \cdot w$, or $M = v^p \cdot M' \wedge M' \in S$ holds.

If we take the set of non-linear monomials of the constraint of interest for S_0, the closed set S guides the application of linearization rules. Non-linear monomials of the shape v^p and $v^p \cdot w$ can be linearized in one step corresponding to the rules (1) and (2). The rule (3) handles non-linear monomials of the form $v^p \cdot M'$ where M' is non-linear. The definition of a closed set S ensures, that one finds in S a decomposition M' of the monomial of this form. Thus, this decomposition M' determines how rule (3) is applied.

Example 4.1. Consider $S_0 = \{ab^2c^3d, a^2b^2, bde\}$, a set of non-linear monomials. Then $S = \{ab^2c^3d, a^2b^2, bde, ab^2c^3, ab^2, a^2, bd\}$ is a closed set for S_0. To see this, one has to verify the conditions from the definition: Firstly, $S_0 \subseteq S$ holds. Secondly, every element of S has to fulfil one of the three conditions. For $ab^2c^3d \in S$, ab^2c^3 is in S; for $a^2b^2 \in S$, a^2 is in S; for $bde \in S$, bd is in S; for $ab^2c^3 \in S$, ab^2 is in S; and finally, $ab^2, bd \in S$ are of the shape $v^p \cdot w$.

However, there is another closed set $S' = \{ab^2c^3d, a^2b^2, bde, ab^2c^3, ab^2, bd\}$ for S_0, which incorporates sharing of the linearization of ab^2c^3 and a^2b^2, namely using the monomial ab^2. This closed set gives rise to a smaller linear constraint.

The example indicates the usage of closed sets. How can one be sure to have the minimal closed set for a given initial set S_0? To this end, we define the following decision problem.

Closed Set Problem. Given a set of monomials and a natural number k, does there exist a closed set S for S_0 with at most k elements?

If this problem could be solved efficiently, the result could be used to practically ensure the smallest possible linear constraint. However, the closed set problem is NP-complete.

Proof Sketch. The closed set problem is in NP, as given S_0, S and k, one can verify the conditions of the closed set and the size limitation in quadratic time (polytime verification). The NP-hardness is shown in [BLO+12] by a reduction from the vertex cover problem. $\square$

Therefore, for an efficient implementation, [BLO+12] relies on a good heuristic for the linearization phase.

5 Extensions

In [BLO+12], two senses of extensions of the transformation are presented, firstly, to support different domains, and secondly, to efficiently handle large domains. We will review them shortly in the sequel.

Rationals. To support rational numbers, [BLO+12] motivated the usage of a fixed denominator d with reference to good experimental results. During linearization, case analysis is employed over the finite domain $\{\frac{x}{d} \mid \mathcal{L}_v \leqslant \frac{x}{d} \leqslant \mathcal{U}_v\}$ for the bounds $\mathcal{L}_v$, $\mathcal{U}_v$ of variable v.

Example 5.1. Let $d = 3$ and consider the constraint $(1 \leqslant v \leqslant 2) \wedge (x = v \cdot z)$. Linearization w.r.t. this rational domain gives rise to the linear constraint

$$(1 \leqslant v \leqslant 2) \wedge \textstyle\bigwedge_{\alpha \in \{1, \frac{4}{3}, \frac{5}{3}, 2\}} (v = \alpha \rightarrow x = \alpha \cdot z).$$

Additionally, alternative representations have been considered in [BLO+12], which turned out to be less efficient. On the one hand, a rational number could be represented as $a + \frac{b}{d}$ for a fixed denominator d, a bounded a and $0 \leqslant b < d$. On the other hand, a representation $\frac{n}{d}$ of bounded numerators n and bounded denominators d has been considered.

Finite Domains. In a similar way, finite domains can be handled. Consider a constraint $x = v^2 \cdot z$ where $v \in \{1, 3, 7, 13\}$. This is encoded as a non-linear constraint $(x = v^2 \cdot z) \wedge (v = 1 \vee v = 3 \vee v = 7 \vee v = 13)$. Linearization yields $(v = 1 \vee v = 3 \vee v = 7 \vee v = 13) \wedge \bigwedge_{\alpha \in \{1,3,7,13\}} (v = \alpha \rightarrow x = \alpha^2 \cdot z)$.

Large Domains. As the transformation rules from Section 3.2 exploit a case analysis on the domain of variables, this transformation gets less efficient as the domain size increases.

To overcome this problem, large domains exceeding a threshold (e.g., 1000) are split into a lower and an upper domain. To this end, two additional transformation rules are given in [BLO+12]. Here, we refrain from copying these rules, and, instead, illustrate the idea on an example.

Example 5.2. Consider the non-linear constraint $(0 \leqslant v < 1000) \wedge (x = v \cdot z)$. Applying the linearization rules from Section 3.2 would result in 1000 additional conjuncts. Instead of that, the large domain of v is split yielding the constraint $(0 \leqslant v < 1000) \wedge (x = v \cdot z) \wedge (v = 20 \cdot v' + v'') \wedge (0 \leqslant v' < 50) \wedge (0 \leqslant v'' < 20)$ where 20 corresponds to a fixed "split factor". Notice that this split could be repeated if the domain of v' still would exceed the threshold.

6 Experimental Results

The authors of [BLO+12] have implemented the presented transformation in Barcelogic [BNO+08]. In this section we will discuss some experimental results. We will mainly compare Barcelogic with MiniSmt [ZM10], a solver for (non-)linear arithmetic which performs bit-blasting to propositional formulas and utilizes SAT-solvers.

We refrain from copying numeric tables, and, instead, try to summarize some trends. To do so, we resort to characteristics of test benches that are worked out in [ZM10], which compares test benches to each other w.r.t. the number of variables, additions and multiplications in constraints.

The test bench `calypto` contains relatively small constraints over the domain $\mathbb{Z}$ with only 10 variables, 6 additions and 6 multiplications in average. The constraints often involve large constants. This setting is beneficial for the approach used in Barcelogic as it solves 110 instances in a total time of 14 seconds. In contrast, MiniSmt solves 109 instances in a total time of 1 931 seconds. The reason for this is that bit-blasting requires many bits to represent large numbers, and the addition over integers is more demanding than over naturals (as it involves two-complement representations). Barcelogic does not need a different handling of integers or larger constants. Similar results are revealed by `m2int` (1 186 vs. 995 instances, 1 699 vs. 8 563 seconds for Barcelogic vs. MiniSmt, respectively). The test bench `m2int` is originating from 2-dimensional matrix interpretations for termination proofs of term rewrite systems [EWZ06].

The situation is different for `leipzig`, a test bench over the domain $\mathbb{N}$ with different characteristics, namely, larger constraints containing 301 variables, 113 additions and 164 multiplications in average. Here, MiniSmt is significantly more efficient than Barcelogic as it solves 158 instances in 401 seconds, whereas Barcelogic solves 162 instances in 1 563 seconds.

7 Conclusion

We conclude this report by summing up the approach and giving an outlook for future work.

Summary. In this report we have reviewed an approach of Borralleras et al. [BLO$^+$12] to transform non-linear arithmetic to linear arithmetic in order to employ an external linear SMT-solver. The interest in this research topic arose from the field of automatic system verification.

The main idea was to perform case analysis on all possible values of a bounded variable to get rid of non-linear multiplications. If the constraint itself lacks sufficiently many bounds, we have to introduce artificial bounds in order to apply this transformation.

Future Work. Borralleras et al. propose to further inspect the case of rationals where finite domains cannot be expressed using bounds. Furthermore, they suggest to incorporate other ideas to prove unsatisfiability.

References

[BCF$^+$08] Roberto Bruttomesso, Alessandro Cimatti, Anders Franzén, Alberto Griggio, and Roberto Sebastiani. The MathSAT 4 SMT Solver. In *Computer Aided Verification*, volume 5123 of *Lecture Notes in Computer Science*, pages 299–303. Springer-Verlag, 2008.

[BLO$^+$12] Cristina Borralleras, Salvador Lucas, Albert Oliveras, Enric Rodríguez-Carbonell, and Albert Rubio. SAT modulo linear arithmetic for solving polynomial constraints. *Journal of Automated Reasoning*, 48:107–131, 2012.

[BNO$^+$08] Miquel Bofill, Robert Nieuwenhuis, Albert Oliveras, Enric Rodríguez-Carbonell, and Albert Rubio. The Barcelogic SMT solver. In *Computer Aided Verification*, volume 5123 of *Lecture Notes in Computer Science*, pages 294–298. Springer-Verlag, 2008.

[CMTU05] Evelyne Contejean, Claude Marché, Ana Paula Tomás, and Xavier Urbain. Mechanically proving termination using polynomial interpretations. *Journal of Automated Reasoning*, 34(4):325–363, 2005.

[Col75] George E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Automata Theory and Formal Languages*, pages 134–183, 1975.

[DdM06] B. Dutertre and L. de Moura. The Yices SMT solver. Tool paper at `http://yices.csl.sri.com/tool-paper.pdf`, August 2006.

[EWZ06] Jörg Endrullis, Johannes Waldmann, and Hans Zantema. Matrix interpretations for proving termination of term rewriting. In *Automated Reasoning*, volume 4130 of *Lecture Notes in Computer Science*, pages 574–588. Springer-Verlag, 2006.

[FGM$^+$07] Carsten Fuhs, Jürgen Giesl, Aart Middeldorp, Peter Schneider-Kamp, René Thiemann, and Harald Zankl. SAT solving for termination analysis with polynomial interpretations. In *Theory and Applications of Satisfiability Testing – SAT 2007*, volume 4501 of *Lecture Notes in Computer Science*, pages 340–354. Springer-Verlag, 2007.

[LPY99] Gerardo Lafferriere, George J. Pappas, and Sergio Yovine. A new class of decidable hybrid systems. In *Proceedings of the Second International Workshop on Hybrid Systems: Computation and Control*, HSCC '99, pages 137–151. Springer-Verlag, 1999.

[SSM08] Sriram Sankaranarayanan, Henny B. Sipma, and Zohar Manna. Constructing invariants for hybrid systems. *Formal Methods in System Design*, 32(1):25–55, 2008.

[Tar51] Alfred Tarski. A decision method for elementary algebra and geometry. *Bulletin of the American Mathematical Society*, 59, 1951.

[Tse70] G. S. Tseitin. On the complexity of derivation in propositional calculus. In *International Workshop on Automated Reasoning*, 1970.

[ZM10] Harald Zankl and Aart Middeldorp. Satisfiability of non-linear (ir)rational arithmetic. In *Proceedings of the 16th International Conference on Logic for Programming and Automated Reasoning*, volume 6355 of *Lecture Notes in Computer Science*, pages 481–500. Springer-Verlag, 2010.