Borralleras, C.; Lucas Alba, S.; Oliveras ., A.; Rodriguez-Carbonell, E.; Rubio ., A. (2012). SAT modulo linear arithmetic for Solving Polynomial Constraints. Journal of Automated Reasoning. 48(1):107-131. doi:10.1007/s10817-010-9196-8.

jar manuscript No.
(will be inserted by the editor)

SAT Modulo Linear Arithmetic for Solving Polynomial Constraints

Cristina Borralleras · Salvador Lucas · Albert Oliveras · Enric Rodríguez-Carbonell · Albert Rubio

Received: date / Accepted: date

Abstract Polynomial constraint solving plays a prominent role in several areas of hardware and software analysis and verification, e.g., termination proving, program invariant generation and hybrid system verification, to name a few. In this paper we propose a new method for solving non-linear constraints based on encoding the problem into an SMT problem considering only linear arithmetic. Unlike other existing methods, our method focuses on proving satisfiability of the constraints rather than on proving unsatisfiability, which is more relevant in several applications as we illustrate with several examples. Nevertheless, we also present new techniques based on the analysis of unsatisfiable cores that allow one to efficiently prove unsatisfiability too for a broad class of problems. The power of our approach is demonstrated by means of extensive experiments comparing our prototype with state-of-the-art tools on benchmarks taken both from the academic and the industrial world.

Keywords Non-linear arithmetic · constraint solving · polynomial constraints · SAT modulo theories · termination · system verification

1 Introduction

Polynomial constraints are ubiquitous in many areas of system analysis and verification. They arise, for instance, when synthesizing program invariants [10,41], as well as analyzing reachability of hybrid systems [31,40]. Another application is the generation of measures for proving termination of symbolic programs as well as rewrite systems (see e.g. [12,23,33,37]). In all these cases, it is paramount to have efficient automatic tools that, given a polynomial constraint with integer or real unknowns, either return a solution or notify that the constraint is unsatisfiable.

Unfortunately, the polynomial constraint solving problem over the integers is undecidable. The situation is not much better when considering the reals since, although

This work has been partially supported by the EU (FEDER) and the Spanish MEC/MICINN, under grants TIN 2007-68093-C02-01 and TIN 2007-68093-C02-02.

Universitat de Vic, Spain · Universitat Politècnica de València, Spain · Universitat Politècnica de Catalunya, Barcelona, Spain · Universitat Politècnica de Catalunya, Barcelona, Spain · Universitat Politècnica de Catalunya, Barcelona, Spain

the problem is decidable as it was shown for the *first-order theory of real closed fields* by Tarski, using the related algorithms in practice is unfeasible due to their complexity (see [7] for a recent account).

Therefore, all methods used in practice for both integer or real solution domains are incomplete. There are two approaches, namely focusing on proving satisfiability or focusing on proving unsatisfiability. In general, the decision on the approach is guided by the kind of problem in hand. For instance, as will be seen later on, in the applications to termination or invariant generation one is more interested in proving satisfiability, i.e. finding solutions, whereas when verifying properties of e.g. hybrid systems one is more concerned with proving unsatisfiability.

As methods focusing on unsatisfiability we have, among others, [36,38,44]. On the other hand, current techniques focusing on satisfiability encode the problem into SAT [28,19,20] or into CSP [34]. These methods, and especially the ones using SAT, are very successfully applied in rewriting-based termination proving tools, as they outperform the previously existing solvers [12]. Following the success of the translation into SAT, it is reasonable to consider whether there is a better target language than propositional logic to keep as much as possible the arithmetic structure of the source language. Thus, in this paper we propose a new method for solving non-linear constraints based on encoding the problem into an SMT problem over linear arithmetic.

The basic idea is to linearize the non-linear monomials we have in our constraints by applying a case analysis on the possible values that some of the variables in the monomial can take. For example, if we have the constraint $x = y \cdot z \ \wedge \ 0 \leq y \leq 2$, we can equivalently replace the non-linear equation $x = y \cdot z$ by the conjunction of the following three linear clauses:

$$\begin{array}{lcl} y = 0 & \longrightarrow & x = 0 \\ y = 1 & \longrightarrow & x = z \\ y = 2 & \longrightarrow & x = 2z \end{array}$$

On the other hand, if neither y nor z is both lower and upper bounded then we have to introduce new bounds and hence we lose completeness.

The resulting constraint is solved using *SAT modulo linear (integer or real) arithmetic* [16], i.e. satisfiability of quantifier-free boolean combinations of linear equalities, inequalities and disequalities. An interesting feature of this approach is that, in contrast to SAT translations [19,20], by having linear arithmetic built into the language, negative values can be handled without additional codification effort. Other recent techniques that incorporate arithmetic natively are similar in this sense (such as [45], which is based on bit-vector arithmetic). Another remarkable characteristic of our method is that, although it targets satisfiability, if enough variables are bounded it also allows one to reason about unsatisfiability, as pointed out in the example above.

Let us mention that the idea of linearization has also been considered for (non-linear) pseudo-boolean constraints in the literature.[1] However, this is a simpler case of linearization as it coincides with polynomial constraints over the interval domain $[0, 1]$, where products of variables are always in $[0, 1]$ as well.[2]

This paper builds on our earlier work [9] and, although for the sake of completeness part of our results are reviewed for the rationals, we will focus here on finding solutions

[1] See `http://www.cril.univ-artois.fr/PB07/coding.html`, the webpage of the Pseudo-Boolean Evaluation 2007.

[2] In this paper by $[\mathcal{L}, \mathcal{U}]$ we will represent the set of *integers* between $\mathcal{L}$ and $\mathcal{U}$, both included.

over the integers, since it is in this context where we have obtained new results that show the strength of the approach. More specifically, here we:

- extend our previous results to arbitrary boolean combinations (not just conjunctions) of polynomial atoms;
- describe several implementation issues that are crucial for having a good performance in practice; and
- present a method for iterating the process of adding new bounds in a clever way, based on the analysis of unsatisfiable cores. The benefits of this technique are twofold: First, it allows us to increase the size of the domains incrementally on demand. Second, as a by-product, although our method is aimed at satisfiability, it also allows us to prove unsatisfiability in many more cases than in [9]. Although unsatisfiable cores are already used similarly e.g. in finite model finding [42], up to our knowledge their application in the particular context of linearizing non-linear arithmetic constraints is new.
  All in all, in contrast to the approaches based on SAT, we can incrementally consider larger and larger bounds in an efficient way, thus getting better performance results in number of handled problems and, in general, in execution time, and moreover we are able to prove unsatisfiability.

Our method has been implemented inside Barcelogic [8].[3] To show its power, we:

- report a comparison with different tools that can handle the problems in the QF_NIA division of SMT-LIB [5], some of which come from industrial applications (namely, from sequential equivalence checking of arithmetic circuits). It is worth mentioning that Barcelogic won the division QF_NIA of SMT-COMP [4] in its first edition of 2009.
- report a comparison of our solver with the one used inside the termination tool AProVE [24], based on a SAT translation. This comparison is really fair since we have been provided with a version of the system that can use different polynomial constraint solvers as a black box. This has given us the opportunity to compare our solver not only with the built-in solver of AProVE, but also with the tool HySAT [17] (note that none of them accepts SMT format).
- revise the running example in [13] to show how our solver can be used to generate invariants. Note again that for this application, as for the previous one of termination, using an approach focusing on proving satisfiability is more convenient since the hope is to find solutions, which provide invariants or termination proofs respectively.

The paper is structured as follows. In Section 2 the notion of pure non-linear constraint, which is key in the presentation of the approach, is introduced. Next, Section 3 describes how, under certain conditions, given a (pure non-linear) constraint with integer or rational variables an equisatisfiable linear formula can be produced. Section 4 is devoted to implementation issues such as how variables should be chosen for linearizing the constraint, or how variables with large domains can be dealt with. The incremental unsatisfiability-guided approach to linearization is explained in Section 5. Sections 6 and 7 illustrate and experimentally demonstrate the potential of our technique in several application areas, namely SMT solving, termination proving and invariant synthesis. Finally in Section 8 we give the conclusions and sketch ideas for future work.

[3] Barcelogic-NIA is available at `http://www.lsi.upc.edu/~albert/barcelogic-NIA`.

2 Pure Non-linear Constraints

The idea of the method is that, given an arbitrary polynomial constraint, an equisatisfiable *pure non-linear constraint*, to be defined in this section, can be obtained by first *normalizing* the original constraint and then introducing auxiliary variables. In Section 3 we will see how for such a constraint an equisatisfiable linearization can be computed, which can then be tested for satisfiability with an SMT solver.

Let us first provide some formal definitions of the problems we are considering.

Definition 1 (Polynomial atom, constraint, clause, CNF) A *polynomial atom* is built from relational operators $\geq$, $>$ and $=$ over arithmetic expressions built from applying the arithmetic plus, minus and times operators over variables and numbers. A *polynomial constraint* (or *formula*) is a boolean combination of polynomial atoms. A *clause* is a polynomial constraint consisting of a disjunction of polynomial atoms or negations of polynomial atoms. A polynomial constraint is in *CNF* if it is a conjunction of clauses.

In the following, variables appearing in polynomial constraints are assumed to be existentially quantified. Thus, the problem that is considered in this paper is, given a polynomial constraint, find a *solution* for it, i.e., an assignment to variables such that the constraint is satisfied, or determine that none exists.

Definition 2 (Monomial) A *monomial* is an expression $v_1^{p_1} \cdots v_m^{p_m}$ where $m > 0$, v_i are variables, $p_i > 0$ for all $i \in \{1 \dots m\}$ and $v_i \neq v_j$ for all $i, j \in \{1 \dots m\}$, $i \neq j$. The monomial is *linear* if $m = 1$ and $p_1 = 1$.

Note that any polynomial constraint F can be rewritten using the distributive law into a polynomial constraint N such that all polynomials occurring in N are expressed as a sum of (products of constants by) monomials, which is said to be in *normal form*. This process is called *normalization*. A polynomial constraint C in normal form is said to be *linear* if all monomials occurring in C are linear.

Example 1 If we want to normalize the constraint

$$2 \cdot (x + y) \leq 0$$

we can apply the distributive law and get $2x + 2y \leq 0$, which is linear. Similarly, if we want to normalize

$$x \cdot (y + y \cdot z) \geq 0$$

we can also apply the distributive law and get

$$x \cdot y + x \cdot y \cdot z \geq 0$$

In this case the above constraint is not linear.

Definition 3 (Pure non-linear constraint) A *pure non-linear constraint* is a formula of the form

$$L \wedge \left(\bigwedge_i y_i = M_i \right)$$

where L is a linear constraint, y_i are variables and all M_i are non-linear monomials.

Lemma 1 *Any polynomial constraint is equisatisfiable to a pure non-linear constraint.*

Proof Given a polynomial constraint F, proceed as follows:

1. Compute N a normal form of F applying the distributive law.
2. Compute C the formula obtained after replacing every non-linear monomial M_i in N by a fresh variable y_i, and finally adding $\bigwedge_i y_i = M_i$.

Then C is a pure non-linear constraint which is equisatisfiable to F. □

As usual, in the following, by *non-linear arithmetic* we mean polynomial arithmetic not restricted to the linear case.

2.1 An Alternative for Generating Short Pure Non-linear Constraints

When one computes an equisatisfiable pure non-linear constraint for a given polynomial formula, normalization using the distributive law as outlined above can be too expensive due to the potential exponential growth in size. To prevent that, we can use the same idea as in Tseitin's algorithm for computing an equisatisfiable conjunctive normal form of a propositional formula.

The idea is that instead of applying the distributive law, we can introduce a new fresh variable replacing the sum. This way we obtain an equisatisfiable formula in an efficient way. Let us show it in an example.

Example 2 Another way to normalize the constraint

$$x \cdot (y + y \cdot z) \geq 0$$

is to introduce a fresh variable v and obtain:

$$x \cdot v \geq 0 \ \wedge \ v = y + y \cdot z$$

which is in normal form.

This transformation is trivially equisatisfiable and can always be used. However in our context, by adding a new variable without bounds, linearization, as will be shown in the next section, may fail. Due to this, we only use this technique in two cases.

- If the rest of the variables in the monomial are bounded. In the example, if x is lower and upper bounded.
- If we can calculate bounds for the new fresh variable and the size of the resulting domain is not too large, since otherwise the transformation can be too expensive as well. In the example, if y and z are lower and upper bounded and their domains are small enough.

3 From Pure Non-linear to Linear Constraints

As said, now we will transform a given pure non-linear constraint into a problem in SAT modulo linear arithmetic, for which fast solving techniques exist.

In the following, we assume that we have a fresh set of variables containing a variable x_M for every monomial M of the form $v_1^{p_1} \cdots v_m^{p_m}$ that can be built out of the variables $v_1, \ldots, v_m$.

Now, we describe two transformations from pure non-linear to linear constraints. We consider two kinds of domains: integer intervals and finite sets of rational numbers.

3.1 Integer Intervals

First, we consider solutions over the integers. Given a polynomial constraint F, by $\mathcal{L}_x$ we denote the lower bound of the variable x in F if it exists and by $\mathcal{U}_x$ we denote the upper bound of the variable x in F if it exists. If both bounds exist then we can assume without loss of generality that $\mathcal{U}_x \geq \mathcal{L}_x$, since otherwise the constraint is trivially unsatisfiable.

As we will see, we need enough variables with lower and upper bounds. If this is not the case, we have to add some bounds and then, an unsatisfiability answer means that the problem is unsatisfiable with the added bounds. In Section 5, it is shown how to guide the introduction of bounds and sometimes still give correct unsatisfiability answers.

Example 3 Consider the pure non-linear constraint

$$2x_{a^3b} - 5x_{cd^2e} \geq 0 \ \wedge \ x_{a^3b} = a^3 \cdot b \ \wedge \ x_{cd^2e} = c \cdot d^2 \cdot e,$$

with $\mathcal{L}_a = \mathcal{L}_c = \mathcal{L}_d = 0$ and $\mathcal{U}_a = \mathcal{U}_c = \mathcal{U}_d = 2$. Then, the translation can be done by adding variable x_{d^2e}, which represents $d^2 \cdot e$.

Linearizing, we obtain the following equisatisfiable constraint:

$$\begin{array}{lll} 2x_{a^3b} - 5x_{cd^2e} \geq 0 & & \\ a = 0 \rightarrow x_{a^3b} = 0 & c = 0 \rightarrow x_{cd^2e} = 0 & d = 0 \rightarrow x_{d^2e} = 0 \\ a = 1 \rightarrow x_{a^3b} = b & c = 1 \rightarrow x_{cd^2e} = x_{d^2e} & d = 1 \rightarrow x_{d^2e} = e \\ a = 2 \rightarrow x_{a^3b} = 8b & c = 2 \rightarrow x_{cd^2e} = 2x_{d^2e} & d = 2 \rightarrow x_{d^2e} = 4e \end{array}$$

together with the bounds

$$0 \leq a \leq 2 \qquad 0 \leq c \leq 2 \qquad 0 \leq d \leq 2$$

In the following definition, the rules Linearization 1 and 2 remove a non-linear equality by adding new linear formulas but without introducing new intermediate variables. In these two rules, making a case analysis on one initial variable is enough to linearize. Finally, in rule Linearization 3 one variable of the monomial of some non-linear equality is removed by adding a case analysis and a new equality with a monomial containing one less variable is obtained.

Definition 4 (Linearization rules) Let C be a pure non-linear constraint. The transformation rules are the following:

Linearization 1:

$C \wedge x = v^p \Longrightarrow C \wedge \bigwedge_{\alpha=\mathcal{L}_v}^{\mathcal{U}_v} (v = \alpha \rightarrow x = \alpha^p)$ if $p > 1$ and $\mathcal{L}_v$ and $\mathcal{U}_v$ exist

Linearization 2:

$C \wedge x = v^p \cdot w \Longrightarrow C \wedge \bigwedge_{\alpha=\mathcal{L}_v}^{\mathcal{U}_v} (v = \alpha \rightarrow x = \alpha^p \cdot w)$ if $\mathcal{L}_v$ and $\mathcal{U}_v$ exist

Linearization 3:

$C \wedge x = v^p \cdot M \Longrightarrow C \wedge \bigwedge_{\alpha=\mathcal{L}_v}^{\mathcal{U}_v} (v = \alpha \rightarrow x = \alpha^p \cdot x_M) \wedge x_M = M$ if M is not linear and v does not occur in M and $\mathcal{L}_v$ and $\mathcal{U}_v$ exist

In the rest of this section, by normal form we mean normal forms with respect to Linearization 1, 2 and 3 (as opposed to the notion of normal form given in Section 2):

Lemma 2 *Given a pure non-linear constraint F, a normal form can be computed in a finite number of steps.*

Proof Considering the non-linear part of F, each application of these rules eliminates a non-linear monomial and either introduces another one with one less variable (rule 3), or no other non-linear monomial is introduced (rules 1 and 2). Hence these rules can only be applied a finite number of times. □

We will now show that the left and the right hand sides of every rule are equisatisfiable.

Lemma 3 *If a pure non-linear constraint F is transformed into F' by one application of Linearization 1, 2 or 3, the following statements hold:*

- Soundness: *If F' is satisfiable, then F is satisfiable.*
- Completeness: *If F is satisfiable, then F' is satisfiable.*

Proof We only detail the proof for Linearization 3, since the other two rules are simpler cases. For the sake of simplicity we use the notation given in the definition of the rule.

For soundness, let σ be a solution for the right hand side, i.e. we have that $\sigma(C)$, $\bigwedge_{\alpha=\mathcal{L}_v}^{\mathcal{U}_v}(\sigma(v) = \alpha \rightarrow \sigma(x) = \alpha^p \cdot \sigma(x_M))$ and $\sigma(x_M) = \sigma(M)$ hold. We need to show that the left hand side is satisfiable. Since we have $\sigma(C)$ by assumption, it is enough to show $\sigma(x) = (\sigma(v))^p \cdot \sigma(M)$ holds. Now, since $\mathcal{L}_v \leq \sigma(v) \leq \mathcal{U}_v$ (as these conditions are part of C), we have that $\bigwedge_{\alpha=\mathcal{L}_v}^{\mathcal{U}_v}(\sigma(v) = \alpha \rightarrow \sigma(x) = \alpha^p \cdot \sigma(x_M))$ implies $\sigma(x) = (\sigma(v))^p \cdot \sigma(x_M)$ and hence, $\sigma(x_M) = \sigma(M)$ allows us to conclude.

For completeness, let σ be a solution for the left hand side, i.e. $\sigma(C)$ and $\sigma(x) = (\sigma(v))^p \cdot \sigma(M)$ hold. We need to show that the right hand side is satisfiable. Since by assumption we have $\sigma(C)$, it suffices to show that σ extended with $\sigma(x_M) = \sigma(M)$ satisfies $\bigwedge_{\alpha=\mathcal{L}_v}^{\mathcal{U}_v}(v = \alpha \rightarrow x = \alpha^p \cdot x_M)$ and $\sigma(x_M) = \sigma(M)$. The latter trivially holds. As for the former, note that if $\sigma(v) < \mathcal{L}_v$ or $\sigma(v) > \mathcal{U}_v$ we are done, otherwise we need $\sigma(x) = (\sigma(v))^p \cdot \sigma(x_M)$, which holds since $\sigma(x) = (\sigma(v))^p \cdot \sigma(M)$ and $\sigma(x_M) = \sigma(M)$. Note that, if x_M already exists in C, then $x_M = M$ occurs in C, and therefore σ already fulfils $\sigma(x_M) = \sigma(M)$ and there is no need to extend σ. □

Thus we get the following result:

Lemma 4 *Any normal form G of a pure non-linear constraint F is equisatisfiable. Moreover, either G is linear or there is a non-linear monomial in G in which no variable has both a lower and an upper bound.*

Proof The first claim follows from Lemma 3 by induction on the number of rule applications. The second claim is proved by analyzing the rules. □

As a consequence of Lemma 4, if we find a linear normal form then our transformation provides a sound and complete method for deciding non-linear constraints over the integers. Otherwise, we have to add some upper or lower bounds in order to reach a linear normal form, which causes incompleteness since we can only trust a satisfiability answer. In Section 5, it is shown how we can also sometimes give correct unsatisfiability answers.

Finally, the last result in this section shows what is the effect of the transformation on the size of the formulas:

Lemma 5 *Let $L \wedge N$ be a pure non-linear constraint, where L and N are its linear and non-linear parts respectively. If F is a linear normal form of $L \wedge N$, then F is of the form $L \wedge G$, where G is a conjunction of at most $O(V \cdot K)$ binary clauses, V is the number of occurrences of variables in the non-linear monomials of N and K is the maximum cardinality of the domains of the variables involved in the linearization.*

Proof Linearization rules preserve the linear part of the original formula. Moreover, every time a variable in a non-linear monomial is chosen for linearization, at most K binary clauses are added. Finally, following the reasoning in the proof of Lemma 2, the number of rule applications is at most the number of occurrences of variables in the non-linear monomials of N. □

3.2 Rationals as Integers

In this case, given $\mathcal{L}$,$\mathcal{U}, D \in \mathbb{Z}$ such that $\mathcal{L} \leq \mathcal{U}$ and $D \neq 0$, we consider the set of rational numbers $\{\frac{n}{D} \mid \mathcal{L} \leq n \leq \mathcal{U}\}$ which are obtained by fixing the denominator D and bounding the numerator. For instance taking $D = 4$ and the numerator in $[0, 16]$ we consider all rational numbers in the set $\{\frac{0}{4}, \frac{1}{4}, \ldots, \frac{15}{4}, \frac{16}{4}\}$. We denote this domain by $[\mathcal{L},\mathcal{U}]/D$.

Then, we simply replace any variable x by $\frac{x'}{D}$ for some fresh variable x' and eliminate denominators from the resulting constraint by multiplying as many times as needed by D. As a result we obtain a constraint over the integers to be solved in the integer interval domain of the numerator. It is straightforward to show completeness of the transformation.

This translation turns out to be reasonably effective. The performance is similar to the integer interval case, but depending on the size of D it works worse due to the fact that the involved integer numbers become much larger.

We have also considered more general domains like the rational numbers expressed by $k + \frac{n}{D}$ with $0 \leq n < D$ for a bounded k and a fixed D that can also be transformed into constraints over bounded integers. For this kind of domains, our experiments revealed a bad trade-off between the gain in the expressiveness of the domain and the performance of the SMT solver on the resulting constraints.

Similarly, we have studied domains of rational values of the form $\frac{n}{d}$ with a bounded numerator n and a bounded denominator d (which cannot be zero). In this case, in order to solve the problem over the integers, every variable x is replaced by $\frac{n_x}{d_x}$ where n_x and d_x are fresh integer variables. Due to the increase of the complexity of the monomials after the elimination of denominators, there is an explosion in the number of intermediate variables needed to linearize the constraint, which finally results in a very poor performance of the linear arithmetic solver.

These kinds of domains were also considered in the context of SAT translations in [20], where similar conclusions were drawn.

3.3 Finite Domains

Following a similar idea as for integer intervals we can handle variables with a domain represented by a finite set of (rational) values. In this case, for every variable v that is used to linearize the formula we have a set S_v of domain values. Then the only

difference with respect to the approach in Section 3.1 is that the domain of the variables is described by a disjunction of equality literals:

$$\bigvee_{\alpha \in S_v} v = \alpha$$

and the three **Linearization** rules perform a case analysis on all the values in the domain set of the form $\bigwedge_{\alpha \in S_v}^{\mathcal{U}_v}(v = \alpha \rightarrow x = \ldots)$ (instead of a case analysis as in Definition 4 of the form $\bigwedge_{\alpha = \mathcal{L}_v}^{\mathcal{U}_v}(v = \alpha \rightarrow x = \ldots)$). The following example illustrates the transformation.

Example 4 Consider the pure non-linear constraint:

$$3x_{abc} - 4x_{cd} + 2a \geq 0 \ \wedge \ x_{abc} = a \cdot b \cdot c \ \wedge \ x_{cd} = c \cdot d$$

with S_a, S_b and S_c being $\{0, \frac{1}{2}, 1, 2\}$. We have the following equisatisfiable linear constraint:

$$\begin{array}{lll} 3x_{abc} - 4x_{cd} + 2a \geq 0 & & \\ a = 0 \rightarrow x_{abc} = 0 & b = 0 \rightarrow x_{bc} = 0 & c = 0 \rightarrow x_{cd} = 0 \\ a = \frac{1}{2} \rightarrow 2x_{abc} = x_{bc} & b = \frac{1}{2} \rightarrow 2x_{bc} = c & c = \frac{1}{2} \rightarrow 2x_{cd} = d \\ a = 1 \rightarrow x_{abc} = x_{bc} & b = 1 \rightarrow x_{bc} = c & c = 1 \rightarrow x_{cd} = d \\ a = 2 \rightarrow x_{abc} = 2x_{bc} & b = 2 \rightarrow x_{bc} = 2c & c = 2 \rightarrow x_{cd} = 2d \end{array}$$

together with the finite domain description

$$(a = 0 \vee a = \tfrac{1}{2} \vee a = 1 \vee a = 2) \quad (b = 0 \vee b = \tfrac{1}{2} \vee b = 1 \vee b = 2)$$
$$(c = 0 \vee c = \tfrac{1}{2} \vee c = 1 \vee c = 2)$$

In the experimental part of this paper we have made very limited use of this kind of domains. However, these are quite useful in termination tools, as shown by our experimental results in [9].

4 Implementation Issues

In this section, we present some implementation decisions we have taken when implementing the general transformation rules given in previous sections that are relevant for the performance of the SMT solver on the final formula.

4.1 Choosing Variables for Linearization

In every step of our transformation, some variable corresponding to a non-linear monomial (e.g. x_{ab^2c}) is treated by choosing one of the original variables in its expression (in this case a, b or c) and fixing the value of the variable according to the different values of the chosen original variable (for instance, a) and the appropriate intermediate variable (in this case, x_{b^2c}), if necessary.

Both decisions, namely which non-linear expression we handle first and which original variable we take, have an important impact on the solving process. The number of intermediate variables and thus the number of clauses in the final formula are highly

dependent on these decisions. Not surprisingly, in general, the performance is improved when the number of intermediate variables is reduced. A similar notion of intermediate variables (only representing products of two variables), called product and square variables, and heuristics for choosing them are also considered in [12].

Let us now formalize the problem of finding a *minimal* (wrt. cardinality) set of intermediate variables for linearizing the initial constraint. In the rest of this section, we assume that all variables have both lower and upper bounds.

Definition 5 (Closed set) A *closed set of non-linear monomials* S for a given initial set S_0 of non-linear monomials is a set fulfilling $S_0 \subseteq S$ and for every $M \in S$ either

- M is v^k, or
- M is $v^k \cdot w^{k'}$ with $k = 1$ or $k' = 1$, or
- M is $v^k \cdot M'$ and $M' \in S$

Notice that, in the third item of Definition 5, v might occur in M'.

Lemma 6 *Let S_0 be the set of monomials of a pure non-linear constraint C. If S is a closed set of S_0 then we can linearize C using the set of variables $\{x_M \mid M \in S\}$.*

Example 5 Let S_0 be the set of non-linear monomials $\{ ab^2c^3d,\ a^2b^2,\ bde \}$. A closed set of non-linear monomials required to linearize S_0 could be $S = \{ ab^2c^3d,\ a^2b^2,\ bde,\ ab^2d,\ ab^2,\ de \}$.

As said, we are interested in finding a minimal closed set S for S_0. The decision version of this problem, i.e., deciding whether there is a closed set for S_0 of size at most K (hereafter the *Closed Set* problem), can be shown to be NP-complete:

Lemma 7 *Closed Set is NP-complete.*

Proof Let us show that, given an initial set S_0 of non-linear monomials, deciding whether there is a closed set for S_0 of size at most K is NP-complete.

It is easy to see that the problem is in NP, because a non-deterministic algorithm needs only to guess a set of size at most K and check in polynomial time that each element M in the set is v_i^k or $v_i^{k_i} v_j^{k_j}$ with either $k_i = 1$ or $k_j = 1$, or there is another element M' in the set such that $M = v_i^k \cdot M'$.

In order to prove that Closed Set is NP-hard we transform Vertex Cover [22], a well-known NP-complete problem, to Closed Set. Let the graph $G = (V, A)$ and a positive integer $K \leq |V|$ be an arbitrary instance of Vertex Cover. Note that, since self-edges (v, v) can only be covered by v, it can be assumed without loss of generality that G does not contain self-edges. We construct in polynomial time an instance of Closed Set (S_0, K') as follows: let S_0 be $\{a_i b_i a_j b_j \mid v_i, v_j \in V \text{ and } (v_i, v_j) \in A\}$ and $K' = 2|A| + K$. In the rest of the proof it is shown that we have a vertex cover of G of size at most K if and only if we have a closed set for S_0 of size at most K':

$\Rightarrow$) Let W be a vertex cover of size at most K. We define the set of monomials S as

$$\begin{aligned} S_0 &\cup \{a_i b_i \mid v_i \in W\} \\ &\cup \{a_i b_i a_j \mid (v_i, v_j) \in A, v_i \in W\} \\ &\cup \{a_j b_j a_i \mid (v_i, v_j) \in A, v_i \notin W\} \end{aligned}$$

Let us prove that S is a closed set for S_0 of size at most K'. It is clear that $S_0 \subseteq S$ and that the size of S is at most $K' = 2|A| + K$. Moreover it is closed: for each $a_i b_i a_j b_j \in S_0$, since $(v_i, v_j) \in A$ we have

- if $v_i \in W$, then $a_i b_i a_j \in S$ and $a_i b_i \in S$;
- if $v_i \notin W$, then $a_j b_j a_i \in S$. Further, as W is a vertex cover, $v_i \notin W$ implies $v_j \in W$. Hence $a_j b_j \in S$.

$\Leftarrow$) Let S be a closed set for S_0 of size at most K'. Note that for each $(v_i, v_j) \in A$, as $a_i b_i a_j b_j \in S_0$ and S is a closed set for S_0, at least one of the following holds:

- $b_i a_j b_j \in S$.
- $a_i a_j b_j \in S$.
- $a_i b_i b_j \in S$.
- $a_i b_i a_j \in S$.

But, again as S is closed, for each of these monomials in three variables there is at least a corresponding monomial $M \in S$ in two variables. We say that such an M *covers* (v_i, v_j).

Now we define the set of vertices W as follows. For each monomial in two variables M:

- if M is of the form $a_i b_i$, then we include v_i in W.
- otherwise M covers at most one edge; if it covers (v_i, v_j), then we include v_i in W.

By construction W is a vertex cover. Further, note that as S includes S_0 and S_0 has size $|A|$, and for each $(v_i, v_j) \in A$ we have a different monomial in three variables, S has at most K monomials in two variables. But the size of W is at most the number of monomials in two variables, and therefore at most K. □

Thus finding a minimal set of monomials can be, in general, too expensive as a subproblem of our transformation algorithm. For this reason, we have implemented a greedy algorithm that provides an approximation to this minimal solution.

Our experiments have shown that applying a reduction of intermediate variables produces a linear constraint that is, in general, easier to be checked by the SMT solver. However, this impact is more important when considering integer interval domains than when considering domains with rationals which are expressed by a set of particular elements. In any case, further analysis on the many different ways to implement efficient algorithms approximating the minimal solution is still necessary.

4.2 Handling Large Domains

In general we avoid choosing variables with a large domain when applying the linearization rules. However, there are many cases in which we have no choice. For instance, if the exponent of a variable is greater than one, then we must apply a linearization rule.

If there are few such variables and the size of the domain is not very large (about 300 values), we can still apply the rules directly, but otherwise the transformation may not work in practice. As an example, in the SMT-LIB set of benchmarks for QF_NIA there are problems with variables with a domain size of hundreds of millions.

Therefore, for this kind of variables we need an alternative solution. In our tool, we have solved this problem by introducing new variables with a smaller interval domain $[0, B-1]$ which are roughly used for describing the solution of the variable with a large domain in base B. This is done by repeatedly replacing the variable v with a large domain $[\mathcal{L}_v, \mathcal{U}_v]$ by $B \cdot v_R + v_B$ where $v_R \in [\lfloor \frac{\mathcal{L}_v}{B} \rfloor, \lfloor \frac{\mathcal{U}_v}{B} \rfloor]$ and $v_B \in [0, B-1]$. Note

that if v occurs with an exponent p then this replacement will be done p times. Our experiments show that 32 is the best candidate for B, although 16, 64 and 128 have all a very similar behaviour.

Let max be a positive integer defining the limit size for the domain of the variables that need a case analysis and let B be a positive integer with $2 \leq B \leq max$. Then Linearization 1, 2 and 3 of Definition 4 might be applied on a variable v if $\mathcal{U}_v - \mathcal{L}_v \leq max$. Otherwise, we repeatedly apply the following rules

Linearization 4:

$$\begin{array}{lll} C \wedge x = v \cdot z \Longrightarrow C \wedge \bigwedge_{\alpha=0}^{B-1}(v_B = \alpha \rightarrow x = B \cdot y + \alpha \cdot z) & & \text{if } \mathcal{L}_v \text{ and } \mathcal{U}_v \text{ exist} \\ \quad \wedge\ y = v_R \cdot z & & \text{and} \\ \quad \wedge\ v = B \cdot v_R + v_B & & \mathcal{U}_v - \mathcal{L}_v > max \\ \quad \wedge\ \lfloor \frac{\mathcal{L}_v}{B} \rfloor \leq v_R \leq \lfloor \frac{\mathcal{U}_v}{B} \rfloor & & \\ \quad \wedge\ 0 \leq v_B \leq B - 1 & & \end{array}$$

where y, v_R and v_B are fresh variables.

Linearization 5:

$$\begin{array}{lll} C \wedge x = v^p \cdot M \Longrightarrow C \wedge \bigwedge_{\alpha=0}^{B-1}(v_B = \alpha \rightarrow x = B \cdot y + \alpha \cdot z) & & \text{if } \mathcal{L}_v \text{ and } \mathcal{U}_v \text{ exist} \\ \quad \wedge\ y = v_R \cdot z & & \text{and} \\ \quad \wedge\ v = B \cdot v_R + v_B & & \mathcal{U}_v - \mathcal{L}_v > max \\ \quad \wedge\ z = v^{p-1} \cdot M & & \text{and} \\ \quad \wedge\ \lfloor \frac{\mathcal{L}_v}{B} \rfloor \leq v_R \leq \lfloor \frac{\mathcal{U}_v}{B} \rfloor & & p > 1 \text{ or} \\ \quad \wedge\ 0 \leq v_B \leq B - 1 & & M \text{ non-linear} \end{array}$$

where z, y, v_R and v_B are fresh variables.

Note that $v = B \cdot v_R + v_B \ \wedge\ \lfloor \frac{\mathcal{L}_v}{B} \rfloor \leq v_R \leq \lfloor \frac{\mathcal{U}_v}{B} \rfloor \ \wedge\ 0 \leq v_B \leq B - 1$ needs to be added only once for every variable v. Moreover, in order to simplify the presentation of Linearization 5 we consider that $p - 1$ can be 0 but, in this case, we know that M is non-linear and hence $z = M$ is still necessary.

In what follows, by normal form we mean normal forms with respect to Linearization 1, 2, 3, 4 and 5. The following lemma states that the linearization rules are terminating:

Lemma 8 *Given a pure non-linear constraint F, a normal form can be computed in a finite number of steps.*

Proof We can take as a measure the sum of the sizes of the domains of the bounded variables occurring in non-linear monomials multiplied by the corresponding exponent. It is straightforward to see that this measure decreases with the application of any of the five rules. □

Again, the left and the right hand sides of the new rules are equisatisfiable:

Lemma 9 *If a pure non-linear constraint F is transformed into F' by one application of Linearization 4 or 5, the following statements hold:*

- Soundness: *If F' is satisfiable, then F is satisfiable.*

– Completeness: *If F is satisfiable, then F' is satisfiable.*

Proof For the sake of simplicity we use the notation given in the definition of the rules.

We only show soundness for rule 5, as the proof for rule 4 is just a particular case. Assume we have a solution σ for the transformed constraint. Then we just have to prove that σ is a solution for $x = v^p \cdot M$. By construction of the transformed constraint we have that $\sigma(y) = \sigma(v_R) \cdot \sigma(z)$, $\sigma(v) = B \cdot \sigma(v_R) + \sigma(v_B)$ and $\sigma(z) = (\sigma(v))^{p-1} \cdot \sigma(M)$. Moreover, since $0 \leq \sigma(v_B) \leq B-1$, from the case analysis we have $\sigma(x) = B \cdot \sigma(y) + \sigma(v_B) \cdot \sigma(z)$. Hence we have $\sigma(x) = B \cdot \sigma(v_R) \cdot \sigma(z) + \sigma(v_B) \cdot \sigma(z)$ and thus, $\sigma(x) = (B \cdot \sigma(v_R) + \sigma(v_B)) \cdot \sigma(z) = \sigma(v) \cdot \sigma(z)$. Finally $\sigma(x) = \sigma(v) \cdot (\sigma(v))^{p-1} \cdot \sigma(M) = (\sigma(v))^p \cdot \sigma(M)$.

Regarding completeness, we again only prove it for rule 5, as rule 4 is just a particular instance. Let us assume that we have a solution σ for the constraint $x = v^p \cdot M \wedge C$, i.e., $\sigma(x) = (\sigma(v))^p \cdot \sigma(M) \wedge \sigma(C)$. In order to prove that the transformed constraint has also a solution, we extend σ with values for v_R, v_B, y and z. Values for v_R and v_B fulfilling $\sigma(v) = B \cdot \sigma(v_R) + \sigma(v_B)$ can be obtained by dividing $\sigma(v)$ by B, which also ensures that $0 \leq \sigma(v_B) \leq B-1$ and $\lfloor \frac{\mathcal{L}_v}{B} \rfloor \leq \sigma(v_R) \leq \lfloor \frac{\mathcal{U}_v}{B} \rfloor$. The value for z is obtained directly fulfilling $\sigma(z) = (\sigma(v))^{p-1} \cdot \sigma(M)$, since v and all variables in M are variables of the original problem. Finally the value for y is taken as $\sigma(y) = \sigma(v_R) \cdot \sigma(z)$. Now it only remains to show that $\bigwedge_{\alpha=0}^{B-1} (\sigma(v_B) = \alpha \rightarrow \sigma(x) = B \cdot \sigma(y) + \alpha \cdot \sigma(z))$, which can be seen to hold by reversing the argument in the proof of soundness:

$$\sigma(x) = (\sigma(v))^p \cdot \sigma(M) = \sigma(v) \cdot \sigma(z) = (B \cdot \sigma(v_R) + \sigma(v_B)) \cdot \sigma(z) = B \cdot \sigma(y) + \sigma(v_B) \cdot \sigma(z)$$

□

The next lemma analyzes the increase in the size of the transformed formulas:

Lemma 10 *Let $L \wedge N$ be a pure non-linear constraint, where L and N are its linear and non-linear parts respectively. Let F be a linear normal form of $L \wedge N$ with respect to Linearization 1, 2, 3, 4 and 5. Then F is of the form $L \wedge G$, with G a conjunction of at most $O((B \cdot \log_B K + max) \cdot V)$ (unit or binary) clauses, where:*

– *B is the base for Linearization 4 and 5;*
– *K is the maximum cardinality of the domains of the variables involved in the linearization;*
– *max is the threshold size for a domain to be considered large; and*
– *V is the number of occurrences of variables in the non-linear monomials of N (counting multiplicities according to exponents).*

Proof Each time we apply Linearization 4 or 5 we add $O(B)$ (unit or binary) linear clauses. Linearization 4 can only be applied $\log_B K$ times, while Linearization 5 can be applied p times for each v^p occurring in a non-linear monomial in N, and each time adds a non-linear equation $y = v_R \cdot z$ that needs at most $\log_B K$ applications of Linearization 4. Thus the number of times we can apply rules 4 and 5 is in $O(\log_B K \cdot V)$, and hence the number of added clauses is in $O(B \cdot \log_B K \cdot V)$. Joining this result with Lemma 5, we have that the number of clauses of G is in $O((B \cdot \log_B K + max) \cdot V)$. □

5 Unsatisfiability and Learning

When some variable that needs to be linearized lacks an upper or lower bound, then we have to add bounds. This makes our method incomplete since then we cannot prove unsatisfiability any longer, as only "SAT" answers imply the satisfiability of the original constraint. In order to lose fewer "SAT" cases we can add bounds to make the domain as large as possible, but this is clearly a bad strategy in many cases, since we may easily produce a too hard problem, due to the size or the shape of the resulting transformation, which cannot be solved within a reasonable time limit.

An alternative idea is starting with bounds that make the domain size small and enlarge them if necessary. The way to decide which are the bounds to be changed is by analyzing an *unsatisfiable core* that we obtain when the solver answers unsatisfiable. We proceed as follows.

From now on we will assume that the given pure non-linear constraint is in CNF and represented as a set of clauses.

Definition 6 (Unsatisfiable core) Let C be an unsatisfiable constraint in CNF. An *unsatisfiable core* is an unsatisfiable subset of C.

Let $\mathcal{S}_\mathcal{L}$ be a set of lower bounding constraints and let $\mathcal{S}_\mathcal{U}$ be a set of upper bounding constraints, i.e. constraints of the form $b \leq x$ and $x \leq b$ respectively for some integer b and variable x.

Assume we are given a pure non-linear constraint F and two sets $\mathcal{S}_\mathcal{L}$ and $\mathcal{S}_\mathcal{U}$, such that $F \cup \mathcal{S}_\mathcal{L} \cup \mathcal{S}_\mathcal{U}$, denoted by C, can be linearized. Note that $\mathcal{S}_\mathcal{L}$ and $\mathcal{S}_\mathcal{U}$ will be included in the linearization. Moreover, in the following we also assume that all variables have small domains, so that only rules 1, 2 and 3 are applied in the linearization (variables with large domains can be treated a priori by exhaustively applying rules 4 and 5 until all variables have small domains).

Lemma 11 *Let C be a pure non-linear constraint, $\mathcal{S}_\mathcal{L} \subseteq C$ and $\mathcal{S}_\mathcal{U} \subseteq C$ be respectively the set of added lower and upper bounding constraints and D be the linearization of C.*

If U_D is an unsatisfiable core for D then there is an unsatisfiable core U_C for C such that $U_C \cap (\mathcal{S}_\mathcal{L} \cup \mathcal{S}_\mathcal{U}) = U_D \cap (\mathcal{S}_\mathcal{L} \cup \mathcal{S}_\mathcal{U})$.

Proof Let C be of the form $C' \cup N$, where C' is a linear constraint and N is the set of equations $y = M$ with M non-linear. Then, since rules 1, 2 and 3 only remove the non-linear part from the initial C, we have that D is of the form $C' \cup L$, where L is the set of clauses added by the linearization.

Let U_D be an unsatisfiable core for D. Let us define U_C as the set of clauses $(U_D \cap C') \cup N$. It is clear that $U_C \subseteq C$. Note also that satisfiability of $(U_D \cap C') \cup N$ implies satisfiability of $(U_D \cap C') \cup L$ (following the argument in the completeness proof of Lemma 3). Moreover, since $(U_D \cap C') \cup L \supseteq (U_D \cap C') \cup (U_D \cap L) = U_D$, we have unsatisfiability of U_D implies unsatisfiability of U_C. Hence, U_C is an unsatisfiable core for C. Finally, since $\mathcal{S}_\mathcal{L}$ and $\mathcal{S}_\mathcal{U}$ are in C' and $U_C \cap C' = U_D \cap C'$, we conclude $U_C \cap (\mathcal{S}_\mathcal{L} \cup \mathcal{S}_\mathcal{U}) = U_D \cap (\mathcal{S}_\mathcal{L} \cup \mathcal{S}_\mathcal{U})$. □

Now we have the following completeness result.

Lemma 12 *Let F be a pure non-linear constraint, and let D be the linearization of $F \cup \mathcal{S}_\mathcal{L} \cup \mathcal{S}_\mathcal{U}$. If D is unsatisfiable, U is an unsatisfiable core for D, and $U \cap (\mathcal{S}_\mathcal{L} \cup \mathcal{S}_\mathcal{U}) = \emptyset$, then F is unsatisfiable.*

Proof Let C be $F \cup \mathcal{S}_{\mathcal{L}} \cup \mathcal{S}_{\mathcal{U}}$. By Lemma 11 there is an unsatisfiable core U_C for C such that $U_C \cap (\mathcal{S}_{\mathcal{L}} \cup \mathcal{S}_{\mathcal{U}}) = U \cap (\mathcal{S}_{\mathcal{L}} \cup \mathcal{S}_{\mathcal{U}}) = \varnothing$. Hence $U_C \subseteq F$ and F is unsatisfiable. □

As a consequence, if we cannot conclude that the original constraint is unsatisfiable, then we know that either $U_{\mathcal{L}} := (U \cap \mathcal{S}_{\mathcal{L}}) \neq \varnothing$ or $U_{\mathcal{U}} := (U \cap \mathcal{S}_{\mathcal{U}}) \neq \varnothing$. But U is an unsatisfiable core for D. Therefore, we can use these two sets to choose which lower and upper bounding constraints we want to modify. Once we have enlarged the domains by modifying all or part of the constraints in $U_{\mathcal{L}}$ and $U_{\mathcal{U}}$, we can iterate the process.

Note also that if we replace $x \leq b$ (respectively $b \leq x$) by $x \leq b'$ (respectively $b' \leq x$) with $b' > b$ (respectively $b' < b$), then the new transformed constraint can be obtained by replacing the old added bounds by the new ones and adding the case analysis only for the values between b and b'.

Finally, let us mention that even if we have all the needed bounds for the linearization, it can be useful to apply the technique described in this section. For instance, in the problems of the `calypto` collection for QF_NIA in the SMT-LIB there are always enough bounded variables to be able to linearize (and hence all these problems are decidable). However, in general, the domains are quite large, and it turns out to be very useful to consider first the domain $[0, 1]$ for all the variables that need a case analysis. Then, in case of unsatisfiability, we replace the added bounds occurring in the unsatisfiable core by new ones (or directly the original ones) and start again.

6 Comparison with Existing Solvers

This section compares the performance of our implementation in Barcelogic [8] with that of other SMT solvers on SMT benchmarks for the theory of non-linear arithmetic. Namely, we have considered the tools Z3 [35], CVC3 [6], and minismt and minismtbv [45]. While the former two solvers implement techniques based on proving unsatisfiability, the latter two are based on proving satisfiability, like our approach.

The benchmarks that have been used for this experimental evaluation have been taken from the QF_NIA division of SMT-LIB [5], which consists of the families `calypto` (produced by Calypto Design Systems, Inc. in the context of sequential equivalence checking of arithmetic circuits) and `leipzig` (coming from termination analysis with matrix interpretations). Besides, a family of problems `m2int` considered in [45] that arises in a similar application to the latter has also been included.

We have performed the experiments in Tables 1, 2, 3, 6 and 7 on a 2.4 GHz 2.9 GB Intel Core Duo with a 32-bit architecture. On the other hand, the experiments in Tables 4 and 5 have been kindly provided by Harald Zankl and have been carried out on a server equipped with 8 dual-core AMD Opteron 885 running at a clock rate of 2.6 GHz and 64 GB of main memory with a 64-bit architecture.

There is a table for every family of benchmarks we have considered: `calypto` (Table 1), `leipzig` (Table 2) and `m2int` (Table 3). The total number of instances for each family is indicated between parentheses in the corresponding table. Each of these tables has four columns, which for every solver show the number and total time in seconds of "SAT", "UNSAT" and "UNKNOWN" answers (first to third columns), and the number of timeouts (fourth column). We have set the timeout to 1200 seconds for those problems coming from the SMT-LIB and to 60 seconds for the problems in the family `m2int`, which is the timeout used in [45].

Tables 1, 2 and 3 clearly show the superiority of our solver, although CVC3 performs slightly better with unsatisfiable problems on the `leipzig` and `m2int` families, as could

calypto	SAT		UNSAT		UNKNOWN		TIMEOUT
(303)	Total	Time	Total	Time	Total	Time	1200
Barcelogic	110	14	193	191	0	0	0
CVC3	57	37	138	10	11	6288	97

Table 1 Experiments with family calypto

leipzig	SAT		UNSAT		UNKNOWN		TIMEOUT
(167)	Total	Time	Total	Time	Total	Time	1200
Barcelogic	162	1563	0	0	0	0	5
CVC3	51	23	1	0	34	6386	81

Table 2 Experiments with family leipzig

m2int	SAT		UNSAT		UNKNOWN		TIMEOUT
(1331)	Total	Time	Total	Time	Total	Time	60
Barcelogic	1186	1699	60	11	1	50	84
CVC3	92	45	84	187	427	2584	728

Table 3 Experiments with family m2int

be expected since it applies methods that focus on proving unsatisfiability. Note that, even though our method focuses on proving satisfiability, using Lemma 12, our tool can prove unsatisfiability of 60 problems of the m2int family, only 24 less than CVC3.

For the good results with the family calypto, the incremental approach explained in Section 5 played a very important role. With respect to the previous implementation used in [9], the difference is most significant precisely with unsatisfiable instances, which used to be the bottleneck in this family and can now be solved very fast. As for satisfiable instances, avoiding large domains in this fashion also has a beneficial effect and reduces the solving time moderately.

The results for Z3 have not been included in the tables in Tables 1, 2 and 3 as we have observed an unstable behavior for the problems in the QF_NIA division.

It is worth pointing out that, in the original m2int family, 60 of the benchmarks were actually empty. Since the behavior of an SMT solver on these degenerate instances is undefined, we have not considered these problems for the experiments in Table 3.

Finally, Tables 4 and 5 show the performance of minismt and minismtbv on our benchmark suite. In this case rows correspond to families of instances, and the time limit is 60 seconds.

Even though for the benchmarks in calypto and leipzig the time limit for the experiments with Barcelogic is 1200 seconds, our tool required more than 60 seconds to produce an answer only for one "UNSAT" problem in the calypto set (more specifically 62 seconds) and four "SAT" problems in the leipzig set. Although this is the same number of timeouts of minismt and minismtbv in the leipzig set, only two timeouts are common to the three tools.

Notice that, although the experiments were carried out on a more powerful machine, our results are better or equal in number of "SAT" answers, and except for the family leipzig, also in total solving time. Moreover, it has to be taken into account too that by construction these solvers cannot produce "UNSAT" answers, i.e., they are incomplete.

In the remaining of this section we report on a comparison of our solver with other existing solvers for non-linear arithmetic that cannot handle SMT-LIB format; namely,

	SAT		UNSAT		UNKNOWN		TIMEOUT
	Total	Time	Total	Time	Total	Time	60
`calypto`	109	1931	0	0	148	2167	46
`leipzig`	158	401	0	0	2	15	7
`m2int`	995	8563	0	0	122	1282	214

Table 4 Experiments with minismt

	SAT		UNSAT		UNKNOWN		TIMEOUT
	Total	Time	Total	Time	Total	Time	60
`calypto`	110	67	0	0	168	285	25
`leipzig`	158	309	0	0	2	1	7
`m2int`	1068	4373	0	0	123	104	140

Table 5 Experiments with minismtbv

a SAT-based solver implemented in the termination tool AProVE [24] and HySAT[4], a solver based on interval analysis.

Since the SAT-based solver was intended to handle problems generated by the AProVE system, in order to make a fair comparison, we have been provided with a parameterized version of AProVE that can use different solvers taken off-the-shelf by the user. In this way, we have been able to assess the performance and the impact of using our solver against others. The comparison with HySAT is also fair, since we send to this tool exactly the same constraint provided by AProVE adding only the declaration of the variables with the domain under consideration (for instance, `int [0,7] x`).

For our experiments we have considered the benchmarks included in the Termination Problems Data Base (TPDB; see `http://www.lri.fr/~marche/tpdb`), version 7.0, in the category of TRS. We have removed all examples that consider special kinds of rewriting (basically, rewriting modulo an equational theory and conditional, relative and context-sensitive rewriting), since they cannot be handled by the simplified version of AProVE we have been provided. Some of the benchmarks in the first part of this section (namely, families `leipzig` and `m2int`) also come from this application to termination.

The experiments have been performed with a time limit *for each termination problem* of 60 seconds (which is the same as in the Termination Competition; see `http://www.termination-portal.org/wiki/Termination_Competition`).

For each of the domains we have considered we provide a table with three columns, corresponding to the three solvers under evaluation: the original AProVE solver (SAT), HySAT and our implementation (SMT). For every solver the results (in number of problems and total running time in seconds) are split in three rows depending on whether the answer is YES (then we have a termination proof), MAYBE (we cannot prove termination) or TIMEOUT (we have exceeded the time limit; in this case, only the number of problems that timed out are shown).

In this particular application to rewriting-based termination, when considering integer domains it turns out that having small domains suffices in general. With upper bounds from 1 up to 7, all solvers improve on the number of positive answers, although

[4] HySAT has recently been replaced by its successor iSAT; see `http://isat.gforge.avacs.org`. Nevertheless, in this particular application the former performs better than the latter, so we have chosen HySAT for our experiments.

	DOMAIN [0,7]					
	SAT		HySAT		SMT	
	Total	Time	Total	Time	Total	Time
YES	781	1859	776	1274	787	1456
MAYBE	1656	5383	1528	5400	1670	3429
TIMEOUT	53		186		33	

	DOMAIN [0,15]					
	SAT		HySAT		SMT	
	Total	Time	Total	Time	Total	Time
YES	770	1998	774	1539	790	1384
MAYBE	1549	9834	1331	6659	1622	4455
TIMEOUT	171		385		78	

Table 6 Experiments with integers in AProVE

	DOMAIN [0,16]/4					
	SAT		HySAT		SMT	
	Total	Time	Total	Time	Total	Time
YES	922	3033	917	3223	946	2605
MAYBE	1273	9194	1090	6332	1415	6524
TIMEOUT	295		483		129	

Table 7 Experiments with rationals in AProVE

this improvement is very relevant from 1 to 2 and becomes almost negligible from 4 to 7. Moreover, all solvers have a similar behavior until 4, although HySAT has a worse behavior as the bound increases. In Table 6 we have reported the results for upper bound 7 and 15, as it shows that our solver can handle larger domains. In the reported results one can observe that our solver is the only one that keeps on getting more positive answers when considering the interval domain $[0, 15]$, while the others lose positive answers and increase the number of timeouts considerably. On the other hand, except for domain $[0, 1]$, our solver is faster than the SAT-based AProVE solver and faster in the overall runtime (without counting timeouts), starting very similar and increasing as the domain grows. This improvement is more significant if we take into account that there is an important part of the process that is common (namely the generation of the constraints) independently of the solver. Moreover, the number of problems for which our solver timed out is only bigger than or equal to that of the SAT-based one for the smallest two domains but is always the lowest from that point on and the difference grows as the domain is enlarged.

Note that the time limit of 60 seconds is for the whole proof of termination of a single problem, which may involve solving a huge number of constraints. Due to this severe time restriction, the learning techniques described in Section 5 do not have a good global impact.

Regarding rational domains, we can only compare the solvers on domains of rationals with fixed denominator and a bounded numerator (see Section 3.2). The reason is that this is the only kind of rational domains that the original solver of our version of AProVE can handle (more general domains are available in the full version of AProVE, but they turned out to have a poorer performance). In particular, we have considered the domain $[0, 16]/4$. When encoding this domain with integers, Table 7 shows that our solver has a better performance than the other solvers, although we have noticed that, when using rationals, the version of AProVE we have has sometimes an unstable

behavior producing different constraints from one execution to another on the same problem and using the same solver.

It is worth mentioning that small finite rational domains, which were dealt with in [9] using the linearization rules outlined in Section 3.3 and which could not be handled by the AProVE original solver, improved the performance of the termination tools significantly and turned out to be the best choice for this particular application.

We have also performed experiments with an old version of another termination tool called MU-TERM [32]. This version is also parameterized by the polynomial constraint solver but, in this case, the original solver is based on CSP. Our experiments show that the performance of the tool is far better using our solver than using the CSP-based solver. These experiments can be found in [9].

Due to this, the current version of the tool MU-TERM-5.05 that participated in the Termination Competition 2009 (see `termcomp.uibk.ac.at/termcomp/`) has already incorporated our solver [1]. This tool ranked first in the *TRS Contextsensitive* category tied with AProVE-1.8 in number of solved problems, but using a smaller amount of time.

Finally, let us mention another tool called CORD [21], which, like HySAT, is based on interval analysis. Unfortunately, it is not publicly available and we have not been able to compare it with our implementation. On the other hand, the experiments in [21] involve real variables with infinite domains, and thus our techniques do not apply.

7 Application to Invariant Generation

Here we show how our solver can be used inside the so-called *constraint-based invariant generation* approach described in [10] and [39]. Let us outline this method using the running example of [10].

Example 6 Consider the program in Figure 1 (which has been taken from [13]) and its corresponding transition system, where l_0 and l_1 are program locations, being l_0 the initial one, and τ_0, τ_1 and τ_2 are transitions from one location to another.

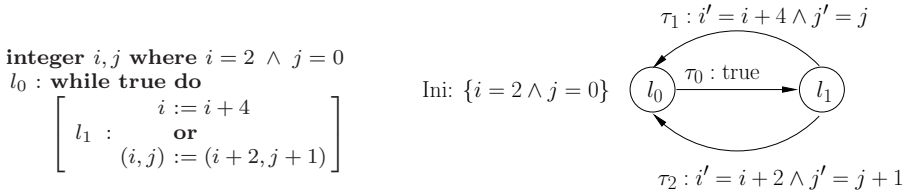

Fig. 1 Program and its corresponding transition system

The idea of the constraint-based invariant generation approach is as follows. First, for each location, a template formula expressed in terms of the program variables and unknown parameters is considered as a candidate invariant. Then it is forced that these candidates are inductive invariants: namely, one imposes *initiation* conditions (which ensure that initial states satisfy the invariant of the initial location) and *consecution* conditions (which ensure that if a state can be reached at a location and a transition from this to another location is followed, the resulting state will satisfy the invariant of the latter). These conditions are encoded conservatively so that a set of constraints

in the unknowns is obtained whose solutions correspond to inductive invariants of the program.

In practice, it is not needed to consider all locations but a cut-set. A *cut-set* of a program is a set of locations (called *cut-points*) such that all cyclic paths pass through a location in the set. For instance, note that in the example there are two cyclic paths $\pi_1 = (\tau_0, \tau_1)$ and $\pi_2 = (\tau_0, \tau_2)$, both of which pass through l_0; hence $\{l_0\}$ is a cut-set.

One of the possible classes of template formulas that can be considered in this method are *linear inequalities*. In this case, Farkas' Lemma is used to transform the initiation and consecution conditions into polynomial constraints [10].

For the simple program above, let us consider a template invariant of the form $c_1 i + c_2 j + d \leq 0$ at location l_0. The solutions of the polynomial constraints to be given next in the unknowns c_1, c_2 and d will provide invariants at that location. More specifically, the constraint encoding the initiation condition is

$$\exists\overline{\lambda} \begin{bmatrix} c_1 = \lambda_1 \ \wedge \ c_2 = \lambda_2 \ \wedge \\ d \ = -2\lambda_1 - \lambda_0 \ \wedge \ \lambda_0 \geq 0 \end{bmatrix},$$

the consecution constraints for π_1 are

$$(\exists\overline{\lambda}, \mu) \begin{bmatrix} 0 \ = \mu c_1 + \lambda_1 \ \wedge \\ 0 \ = \mu c_2 + \lambda_2 \ \wedge \\ c_1 = -\lambda_1 \ \wedge \ c_2 = -\lambda_2 \ \wedge \\ d \ = \mu d + 4\lambda_1 - \lambda_0 \ \wedge \\ \mu, \ \lambda_0 \geq 0 \end{bmatrix} \ \vee \ (\exists\overline{\lambda}, \mu) \begin{bmatrix} 0 \ = \mu c_1 + \lambda_1 \ \wedge \\ 0 \ = \mu c_2 + \lambda_2 \ \wedge \\ 0 \ = -\lambda_1 \ \wedge \ 0 = -\lambda_2 \ \wedge \\ 1 \ = \mu d + 4\lambda_1 - \lambda_0 \ \wedge \\ \mu, \ \lambda_0 \geq 0 \end{bmatrix}$$

and the consecution constraints for π_2 are

$$(\exists\overline{\lambda}, \mu) \begin{bmatrix} 0 \ = \mu c_1 + \lambda_1 \ \wedge \\ 0 \ = \mu c_2 + \lambda_2 \ \wedge \\ c_1 = -\lambda_1 \ \wedge \ c_2 = -\lambda_2 \ \wedge \\ d \ = \mu d + 2\lambda_1 + \lambda_2 - \lambda_0 \ \wedge \\ \mu, \ \lambda_0 \geq 0 \end{bmatrix} \ \vee \ (\exists\overline{\lambda}, \mu) \begin{bmatrix} 0 \ = \mu c_1 + \lambda_1 \ \wedge \\ 0 \ = \mu c_2 + \lambda_2 \ \wedge \\ 0 \ = -\lambda_1 \ \wedge \ 0 = -\lambda_2 \ \wedge \\ 1 \ = \mu d + 2\lambda_1 + \lambda_2 - \lambda_0 \ \wedge \\ \mu, \ \lambda_0 \geq 0 \end{bmatrix},$$

where $\exists\overline{\lambda}$ stands for $\exists\lambda_1 \, \exists\lambda_2 \, \exists\lambda_3$.

We can move the existential quantifiers out by renaming the quantified variables. Then we get a problem in QF_NIA that can be easily solved by Barcelogic obtaining a solution. We can iterate the process forbidding redundant solutions, obtaining automatically $j \geq 0$ and $i - 2j \geq 2$, which are all the invariant relations found in [13] and [10].

However, the application of polynomial constraint solving is not limited to the generation of linear invariants. For instance, we can handle similarly the polynomial constraints that need to be solved for generating *polynomial equality* invariants [41]. Moreover, in that paper, some complete encoding for ensuring consecution is discarded because of the complexity of the generated polynomial constraints. Thus, it could be the case that thanks to the improvement in polynomial constraint solvers some of these techniques can become useful now.

Similarly, our approach can also be applied for generating *polynomial inequality* invariants [30]. The techniques presented in [30] have had limited success so far because current tools for quantifier elimination in the reals (e.g., QEPCAD [29], REDLOG [14]) only work on very small problems. Our methods, in combination with Positivstellensatz [43] playing a similar role to Farkas' lemma for linear inequalities, could also be applied

for producing this kind of very expressive invariants and thus open the door to much more precise program analyses.

Last but not least, the previous applications were focused on *intraprocedural* analysis, i.e., ignoring other procedures and the contexts in which they call each other. In [25], it is shown how several problems of *interprocedural* program analysis can be reduced to polynomial constraints over the integers, which are solved in that paper by translating into SAT. Based on our results in the area of termination of rewriting systems, we foresee that the methods proposed here could yield a significant improvement on the efficiency of those interprocedural program analyzers.

8 Conclusions

We have proposed a simple method for solving non-linear polynomial constraints over finite domains of the integer and the rational numbers, which is based on translating the constraints into *SAT modulo linear (real or integer) arithmetic*.

Our method focuses on proving satisfiability, but we have shown that it can also be used to prove unsatisfiability. It has been implemented within the Barcelogic SMT-solver and has shown its power in a variety of examples coming from academia as well as from industry.

As shown in the different applications, the existence of new powerful solvers for non-linear arithmetic can be crucial for reconsidering methods that were discarded in the past because they generate non-linear constraints.

As future work, we want to extend the learning mechanism on bounds to the case of rationals where the finite domains cannot be expressed using bounds. Moreover, we want to study how to combine our method with the other existing approaches based on proving unsatisfiability [36,38,44].

Acknowledgments: we would like to thank Jürgen Giesl, Carsten Fuhs and Karsten Behrmann for providing us with a version of AProVE to be used in our experiments. We also thank Harald Zankl for providing us with the detailed data on experiments with minismt.

References

1. B. Alarcón, R. Gutiérrez, S. Lucas, R. Navarro-Marset. Proving Termination Properties with MU-TERM. In *Proceedings of the 13th International Conference on Algebraic Methodology and Software Technology, AMAST'10*, Lecture Notes in Computer Science. To appear.
2. T. Arts and J. Giesl. Termination of Term Rewriting Using Dependency Pairs. *Theoretical Computer Science*, 236(1-2):133-178, Elsevier, 2000.
3. F. Baader and T. Nipkow. Term Rewriting and All That. Cambridge University Press, 1998.
4. C. Barrett, M. Deters, A. Oliveras and A. Stump. The Satisfiability Modulo Theories Competition (SMT-COMP). `http://www.smtcomp.org`, 2009.
5. C. Barrett, S. Ranise, A. Stump and C. Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `http://www.smt-lib.org`, 2008.
6. C. Barrett and C. Tinelli. CVC3. In *Proceedings of the 19th International Conference on Computer Aided Verification, CAV'07*, Lecture Notes in Computer Science, volume 4590, pp. 298-302, Springer, 2007.
7. S. Basu, R. Pollack and M.-F. Roy. Algorithms in Real Algebraic Geometry. Springer, 2003.

8. M. Bofill, R. Nieuwenhuis, A. Oliveras, E. Rodríguez-Carbonell and A. Rubio. The Barcelogic SMT Solver. In *Proceedings of the 20th International Conference on Computer Aided Verification, CAV'08*, Lecture Notes in Computer Science, volume 5123, pp. 294-298, Springer, 2008.
9. C. Borralleras, S. Lucas, R. Navarro-Marset, E. Rodríguez-Carbonell and A. Rubio. Solving Non-linear Polynomial Arithmetic via SAT Modulo Linear Arithmetic. In *Proceedings of the 22nd International Conference on Automated Deduction, CADE-22*, Lecture Notes in Computer Science, volume 5663, pp. 294-305, Springer, 2009.
10. M.A. Colón, S. Sankaranarayanan and H.B. Sipma. Linear Invariant Generation Unsing Non-Linear Constraint Solving. In *Proceedings of 15th International Conference on Computer Aided Verification, CAV'03*, Lecture Notes in Computer Science, volume 2725, pp. 420-432, Springer, 2003.
11. E. Contejean, C. Marché, B. Monate and X. Urbain. Proving Termination of Rewriting with C*i*ME. In *Extended Abstracts of the 6th International Workshop on Termination, WST'03*, pp. 71-73, 2003.
12. E. Contejean, C. Marché, A.-P. Tomás and X. Urbain. Mechanically Proving Termination Using Polynomial Interpretations. Journal of Automated Reasoning, 34(4):325-363, Springer, 2006.
13. P. Cousot and N. Halbwachs. Automatic Discovery of Linear Restraints Among Variables of a Program. In *Proceedings of the 5th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, POPL'78*, pp. 84-96, ACM Press, 1978.
14. A. Doltzmann and T. Sturm. REDLOG: Computer Algebra meets Computer Logic. Technical Report, University of Passau, MIP-9603, Sep. 1996.
15. B. Dutertre and L. de Moura. The Yices SMT solver. System description report. `http://yices.csl.sri.com/`
16. B. Dutertre and L. de Moura. A Fast Linear-Arithmetic Solver for DPLL(T). In *Proceedings of the 18th International Conference on Computer Aided Verification, CAV'06*, Lecture Notes in Computer Science, volume 4144, pp. 81-94, Springer, 2006.
17. M. Fränzle, C. Herde, T. Teige, S. Ratschan and T. Schubert. Efficient Solving of Large Non-linear Arithmetic Constraint Systems with Complex Boolean Structure. *Journal on Satisfiability, Boolean Modeling and Computation* 1(3-4):209-236, 2007.
18. C. Fuhs, J. Giesl, A. Middeldorp, P. Schneider-Kamp, R. Thiemann and H. Zankl. Maximal Termination. In *Proceedings of the 19th International Conference on Rewriting Techniques and Applications, RTA'08*, Lecture Notes in Computer Science, volume 5117, pp. 110-125, Springer, 2008.
19. C. Fuhs, J. Giesl, A. Middeldorp, P. Schneider-Kamp, R. Thiemann and H. Zankl. SAT Solving for Termination Analysis with Polynomial Interpretations. In *Proceedings of the 10th International Conference on Theory and Applications of Satisfiability Testing, SAT'07*, Lecture Notes in Computer Science, volume 4501, pp. 340-354, Springer, 2007.
20. C. Fuhs, R. Navarro-Marset, C. Otto, J. Giesl, S. Lucas and P. Schneider-Kamp. Search Techniques for Rational Polynomial Orders. In *Proceedings of the 9th International Conference on Intelligent Computer Mathematics, AISC'08*, Lecture Notes in Computer Science, volume 5144, pp. 109-124, Springer, 2008.
21. M. K. Ganai and F. Ivančić. Efficient decision procedure for non-linear arithmetic constraints using CORDIC. In *Proceedings of the 9th International Conference on Formal Methods in Computer Aided Design, FMCAD'09*, pp. 61-68, IEEE, 2009.
22. M. R. Garey and D. S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., 1979.
23. J. Giesl. Generating Polynomial Orderings for Termination Proofs. In *Proceedings of the 6th International Conference on Rewriting Techniques and Applications, RTA'95*, Lecture Notes in Computer Science, volume 914, pp. 426-431, Springer, 1995.
24. J. Giesl, P. Schneider-Kamp and R. Thiemann. Automatic Termination Proofs in the Dependency Pair Framework. In *Proceedings of the 3rd International Joint Conference on Automated Reasoning, IJCAR'06*, Lecture Notes in Computer Science, volume 4130, pp. 281-286, Springer, 2006.
25. S. Gulwani, S. Srivastava and R. Venkatesan. Program Analysis as Constraint Solving. In *Proceedings of the ACM SIGPLAN 2008 Conference on Programming Language Design and Implementation, PLDI'08*, pp. 281-292, ACM Press, 2008.
26. S. Gulwani and A. Tiwari. Constraint-Based Approach for Analysis of Hybrid Systems. In *Proceedings of the 20th International Conference on Computer Aided Verification, CAV'08*, Lecture Notes in Computer Science, volume 5123, pp. 190-203, Springer, 2008.

27. N. Hirokawa and A. Middeldorp. Tyrolean Termination Tool: Techniques and Features. *Information and Computation*, 205(4):474-511, 2007.
28. D. Hofbauer and J. Waldmann. Termination of $\{aa \to bc, bb \to ac, cc \to ab\}$. *Information Processing Letters*, 98(4):156-158, 2006.
29. H. Hong et al. `http://www.usna.edu/Users/cs/qepcad/B/WhatisQEPCAD.html`
30. D. Kapur. Automatically Generating Loop Invariants Using Quantifier Elimination. In *Proceedings of the IMACS International Conference on Applications of Computer Algebra*, 2004.
31. G. Lafferriere, G.J. Pappas and S. Yovine. A New Class of Decidable Hybrid Systems. In *Proceedings of the 2nd International Workshop on Hybrid Systems: Computation and Control, HSCC'99*, Lecture Notes in Computer Science, volume 1569, pp. 137-151, Springer, 1999.
32. S. Lucas. MU-TERM: A Tool for Proving Termination of Context-Sensitive Rewriting. In *Proceedings of the 15th International Conference on Rewriting Techniques and Applications, RTA'04*, Lecture Notes in Computer Science, volume 3091, pp. 200-209, Springer, 2004. `http://zenon.dsic.upv.es/muterm`.
33. S. Lucas. Polynomials over the Reals in Proofs of Termination: from Theory to Practice. *RAIRO Theoretical Informatics and Applications*, 39(3):547-586, 2005.
34. S. Lucas. Practical Use of Polynomials over the Reals in Proofs of Termination. In *Proceedings of the 9th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, PPDP'07*, pp. 39-50, ACM Press, 2007.
35. L. de Moura and N. Björner. Z3: An Efficient SMT Solver. In *Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'08*, Lecture Notes in Computer Science, volume 4963, pp. 337-340, Springer, 2008.
36. L. de Moura and G. Olney Passmore. On Locally Minimal Nullstellensatz Proofs. Technical Report, Microsoft Research, MSR-TR-2009-90.pdf, 2009.
37. M.T. Nguyen and D. De Schreye. Polynomial Interpretations as a Basis for Termination Analysis of Logic Programs. In *Proceedings of the 21st International Conference on Logic Programming, ICLP'05*, Lecture Notes in Computer Science, volume 3668, pp. 311-325, Springer, 2005.
38. P.A. Parrilo. Semidefinite Programming Relaxations for Semialgebraic Problems. *Mathematical Programming*, 96(2):293–320, 2003.
39. S. Sankaranarayanan, H.B. Sipma and Z. Manna. Constraint-Based Linear-Relations Analysis. In *Proceedings of the 11th International Symposium on Static Analysis, SAS'04*, Lecture Notes in Computer Science, volume 3148, pp. 53-68, 2004.
40. S. Sankaranarayanan, H.B. Sipma and Z. Manna. Constructing Invariants for Hybrid Systems. Formal Methods in System Design, 32(1):25-55, 2008.
41. S. Sankaranarayanan, H.B. Sipma and Z. Manna. Non-linear Loop Invariant Generation Using Gröbner Bases. In *Proceedings of the 31st ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, POPL'04*, pp. 318-329, ACM Press, 2004.
42. I. Shlyakhter, R. Seater, D. Jackson, M. Sridharan and M. Taghdiri. Debugging Overconstrained Declarative Models Using Unsatisfiable Cores. In *Proceedings of the 18th IEEE International Conference on Automated Software Engineering, ASE'03*, pp. 94-105, IEEE Computer Society, 2003.
43. G. Stengle. A Nullstellensatz and a Positivstellensatz in Semialgebraic Geometry. Mathematische Annalen, 207(2):87-97, 1973.
44. A. Tiwari. An Algebraic Approach for the Unsatisfiability of Nonlinear Constraints. In *Proceedings of the 19th International Workshop on Computer Science Logic, CSL'05*, Lecture Notes in Computer Science, volume 3634, pp. 248-262, 2005.
45. H. Zankl and A. Middeldorp. Satisfiability of Non-Linear (Ir)rational Arithmetic. *Proceedings of the 16th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, LPAR-16*, 2010.