Incomplete SMT Techniques for Solving Non-Linear Formulas over the Integers

CRISTINA BORRALLERAS, Universitat de Vic - Universitat Central de Catalunya, Spain
DANIEL LARRAZ, The University of Iowa, USA
ENRIC RODRÍGUEZ-CARBONELL, Universitat Politècnica de Catalunya, Spain
ALBERT OLIVERAS, Universitat Politècnica de Catalunya, Spain
ALBERT RUBIO, Universidad Complutense de Madrid, Spain

We present new methods for solving the Satisfiability Modulo Theories problem over the theory of Quantifier-Free Non-linear Integer Arithmetic, SMT(QF-NIA), which consists in deciding the satisfiability of ground formulas with integer polynomial constraints. Following previous work, we propose to solve SMT(QF-NIA) instances by reducing them to linear arithmetic: non-linear monomials are linearized by abstracting them with fresh variables and by performing case splitting on integer variables with finite domain. For variables that do not have a finite domain, we can artificially introduce one by imposing a lower and an upper bound, and iteratively enlarge it until a solution is found (or the procedure times out).

The key for the success of the approach is to determine, at each iteration, which domains have to be enlarged. Previously, unsatisfiable cores were used to identify the domains to be changed, but no clue was obtained as to how large the new domains should be. Here we explain two novel ways to guide this process by analyzing solutions to optimization problems: (i) to minimize the number of violated artificial domain bounds, solved via a Max-SMT solver, and (ii) to minimize the distance with respect to the artificial domains, solved via an Optimization Modulo Theories (OMT) solver. Using this SMT-based optimization technology allows smoothly extending the method to also solve Max-SMT problems over non-linear integer arithmetic. Finally we leverage the resulting Max-SMT(QF-NIA) techniques to solve ∃∀ formulas in a fragment of quantified non-linear arithmetic that appears commonly in verification and synthesis applications.

CCS Concepts: • **Mathematics of computing → Solvers**; • **Theory of computation → Logic and verification**; *Automated reasoning*; • **Computing methodologies → Equation and inequality solving algorithms**; *Theorem proving algorithms.*

Additional Key Words and Phrases: non-linear arithmetic, satisfiability modulo theories

ACM Reference Format:
Cristina Borralleras, Daniel Larraz, Enric Rodríguez-Carbonell, Albert Oliveras, and Albert Rubio. YYYY. Incomplete SMT Techniques for Solving Non-Linear Formulas over the Integers. *ACM Trans. Comput. Logic* V, N, Article A (January YYYY), 37 pages. https://doi.org/10.1145/0000000.0000000

Authors' addresses: Cristina Borralleras, Universitat de Vic - Universitat Central de Catalunya, Facultat de Ciències i Tecnologia, C/ Laura 13, Vic, 08500, Spain, cristina.borralleras@uvic.cat; Daniel Larraz, The University of Iowa, Computer Science, 14 MacLean Hall, Iowa City, Iowa, 52242-1419, USA, daniel-larraz@uiowa.edu; Enric Rodríguez-Carbonell, Universitat Politècnica de Catalunya, Departament de Ciències de la Computació, C/ Jordi Girona 1, Barcelona, 08034, Spain, erodri@cs.upc.edu; Albert Oliveras, Universitat Politècnica de Catalunya, Departament de Ciències de la Computació, C/ Jordi Girona 1, Barcelona, 08034, Spain, oliveras@cs.upc.edu; Albert Rubio, Universidad Complutense de Madrid, Departamento de Sistemas Informáticos y Computación, C/ Profesor José García Santesmases, 9, Madrid, 28040, Spain, alberu04@ucm.es.

1529-3785/YYYY/1-ARTA $15.00
https://doi.org/10.1145/0000000.0000000

1 INTRODUCTION

Polynomial constraints are pervasive in computer science. They appear naturally in countless areas, ranging from the analysis, verification and synthesis of software and hybrid systems [20, 64–66] to, e.g., game theory [9]. In all these cases, it is crucial to have efficient automatic solvers that, given a formula involving polynomial constraints with integer or real variables, either return a solution to the formula or report that there is none.

Therefore, it is of no surprise that solving this sort of non-linear formulas has attracted wide attention over the years. A milestone result of Tarski's [72] is a constructive proof that the problem is decidable for the first-order theory of real closed fields, in particular for the real numbers. Unfortunately the algorithm in the proof has non-elementary complexity, i.e., its cost cannot be bounded by any finite tower of exponentials, and is thus essentially useless from a practical point of view. For this reason, for solving polynomial constraints in $\mathbb{R}$, computer algebra has traditionally relied on the more workable approach of cylindrical algebraic decomposition (CAD) [5, 19]. Still, its applicability is hampered by its doubly exponential complexity, and alternative techniques like *virtual substitution* [55, 75, 76] have appeared.

Due to the interest of the problem, further research has been carried out spurred by the irruption of propositional satisfiability (SAT) solvers and their extensions [10, 60]. Thus, several techniques have emerged in the last decade which leverage the efficiency and automation of this new technology. E.g., for solving polynomial constraints in $\mathbb{R}$, interval constraint propagation has been integrated with SAT and satisfiability modulo theories (SMT) engines [34, 39, 46]. Other works preprocess non-linear formulas before passing them to an off-the-shelf SMT solver for quantifier-free linear real arithmetic [37], or focus on particular kinds of constraints like convex constraints [61]. In the implementation of many of these approaches computations are performed with floating-point arithmetic. In order to address the ever-present concern that numerical errors can result in incorrect answers, the framework of δ-complete decision procedures has been proposed [38, 40]. In another line of research, as opposed to numerically-driven approaches, symbolic techniques from algebraic geometry such as the aforementioned CAD [43], Gröbner bases [44, 63], Handelman's representations [56] or virtual substitution [23] have been successfully adapted to SAT and SMT. As a result, several libraries and toolboxes have been made publicly available for the development of symbolically-driven solvers [24, 25, 29].

On the other hand, when variables have to take integer values, even the problem of solving a single polynomial equation is undecidable (Hilbert's 10th problem, [22]). Despite this theoretical limitation, and following a similar direction to the real case, several incomplete methods that exploit the progress in SAT and SMT have been proposed for dealing with integer polynomial constraints. The common idea of these approaches is to reduce instances of this kind of formulas into problems of a simpler language that can be straightforwardly handled by existing SAT/SMT systems, e.g., propositional logic [36], linear bit-vector arithmetic [77] or linear integer arithmetic [15]. All these techniques are oriented towards satisfiability, which makes them convenient in applications where finding solutions is more relevant than proving that none exists (e.g., in verification when generating ranking functions [49], invariants [51] or other inductive properties [16, 48]).

In this article[1] we build upon our previous method [15] for deciding satisfiability modulo the theory of quantifier-free non-linear integer arithmetic (SMT(QF-NIA)), i.e., the satisfiability of first-order quantifier-free formulas where atoms are *polynomial* inequalities over integer variables. In that work, the problem is reduced to that of satisfiability modulo the theory of quantifier-free linear integer arithmetic (SMT(QF-LIA)), i.e., the satisfiability of first-order quantifier-free formulas where atoms are *linear* inequalities over integer variables. More specifically, in [15] non-linear

[1]This is the extended version of the conference paper presented at SAT '14 [50].

monomials are linearized by abstracting them with fresh variables and by performing case splitting on integer variables with finite domain. In the case in which variables do not have finite domains, artificial ones are introduced by imposing a lower and an upper bound. While the underlying SMT(QF-LIA) solver cannot find a solution (and the time limit has not been exceeded yet), *domain enlargement* is applied: some domains are made larger by weakening the bounds. To guide which bounds have to be changed from one iteration to the following one, unsatisfiable cores are employed: at least one of the artificial bounds that appear in the unsatisfiable core should be weaker. Unfortunately, although unsatisfiable cores indicate which bounds should be weakened, they provide no hint on how large the new domains have to be made. This is of paramount importance, since the size of the new linearized formula (and therefore the time needed to determine its satisfiability) can increase significantly depending on the number of new cases that must be added.

A way to circumvent this difficulty could be to find alternative techniques to the unsatisfiable cores which, when a solution with the current domains cannot be found, provide more complete information for the domain enlargement. In this paper we propose such alternative techniques. The key idea is that an assignment of numerical values to variables that is "closest" to being a true solution (according to some metric) can be used as a reference as regards to how one should enlarge the domains. Thus, the models generated by the SMT(QF-LIA) engine are put in use in the search of solutions of the original non-linear problem, with a similar spirit to [26] for combining theories or to the model-constructing satisfiability calculus of [28].

However, in our case we are particularly interested in *minimal models*, namely those that minimize a cost function that measures how far assignments are from being a true solution to the non-linear problem. Minimal models have long been studied in the case of propositional logic [7, 8, 70]. In SMT, significant advancements have been achieved towards solving the optimization problems of *Maximum Satisfiability Modulo Theories* (Max-SMT, [18, 59]) and *Optimization Modulo Theories* (OMT, [62, 68]). Thanks to this research, several SMT systems are currently offering optimization functionalities ([12, 54, 69]).

In a nutshell, in this work we develop new strategies for domain enlargement to improve how SMT(QF-NIA) is solved, and then we leverage these enhancements to logics closer to applications in program analysis, verification and synthesis. The resulting techniques are incomplete, and so unsatisfiable instances cannot be detected when complex non-linear reasoning is needed. The goal is instead to find solutions efficiently for satisfiable formulas. To be precise we make the following contributions:

(1) In the context of solving SMT(QF-NIA), we present different heuristics for guiding the domain enlargement step by means of the analysis of minimal models. More specifically, we consider two different cost functions:

- the number of violated artificial domain bounds (leading to Max-SMT problems);
- the distance with respect to the artificial domains (leading to OMT problems).

We evaluate these model-guided heuristics experimentally with an exhaustive benchmark set and compare them with other techniques for solving SMT(QF-NIA). The results of this evaluation show the potential of the method.

(2) Based on the results of the aforementioned experiments, we extend our best approach for SMT(QF-NIA) to handle problems in Max-SMT(QF-NIA).

(3) Finally we apply our Max-SMT(QF-NIA) techniques to solve SMT and Max-SMT problems in the following fragment of quantified non-linear arithmetic: $\exists\forall$ formulas where $\exists$ variables

are of integer type and ∀ variables are of real type, and non-linear monomials cannot contain the product of two real variables. Formulas of this kind appear commonly in verification and synthesis applications [32], for example in control and priority synthesis [17], reverse engineering of hardware [41] and program synthesis [73].

The paper is structured as follows. Section 2 reviews basic background on SMT, Max-SMT and OMT, and also on our previous approach in [15]. In Section 3 two different heuristics for guiding the domain enlargement step are presented, together with experiments and several possible variants. Then Section 4 proposes an extension of our techniques from SMT(QF-NIA) to Max-SMT(QF-NIA). In turn, in Section 5 our Max-SMT(QF-NIA) approach is applied to solving Max-SMT problems with ∃∀ formulas. Finally, Section 6 summarizes the conclusions of this work and sketches lines for future research.

2 PRELIMINARIES

2.1 Polynomials, SMT, Max-SMT and OMT

A *monomial* is an expression of the form $v_1^{p_1} \cdots v_m^{p_m}$ 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 $i, j \in \{1 \dots m\}$, $i \neq j$. A monomial is *linear* if $m = 1$ and $p_1 = 1$.

A *polynomial* is a linear combination of monomials, i.e., an arithmetic expression of the form $\sum \lambda_i m_i$ where the λ_i are coefficients and the m_i are monomials. In this paper, coefficients will be integer numbers. A polynomial is *linear* if all its monomials are linear.

A *polynomial inequality* is built by applying relational operators $\geq$ and $\leq$ to polynomials. A *linear inequality* is a polynomial inequality in which the polynomials at both sides are linear.

Let $\mathcal{P}$ be a fixed finite set of *propositional variables*. If $p \in \mathcal{P}$, then p and $\neg p$ are *literals*. The *negation* of a literal l, written $\neg l$, denotes $\neg p$ if l is p, and p if l is $\neg p$. A *clause* is a disjunction of literals $l_1 \vee \cdots \vee l_n$. A *propositional formula* (in conjunctive normal form, CNF) is a conjunction of clauses $C_1 \wedge \cdots \wedge C_n$. Given a propositional formula, an assignment of Boolean values to variables that satisfies the formula is a *model* of the formula. A formula is *satisfiable* if it has a model, and *unsatisfiable* otherwise. The problem of, given a propositional formula, to determine whether it is satisfiable or not is called the *propositional satisfiability* (abbreviated SAT) problem.

The *satisfiability modulo theories (SMT)* problem is a generalization of SAT. In SMT, one has to decide the satisfiability of a given (usually, quantifier-free) first-order formula with respect to a background theory. In this setting, a model (which we may also refer to as a *solution*) is an assignment of values from the theory to variables that satisfies the formula. Examples of theories are *quantifier-free linear integer arithmetic (QF-LIA)*, where atoms are linear inequalities over integer variables, and the more general *quantifier-free non-linear integer arithmetic (QF-NIA)*, where atoms are polynomial inequalities over integer variables. Unless otherwise stated, in this paper we will assume that variables are all of integer type.

Another generalization of SAT is *Max-SAT* [53], which extends the problem by asking for more information when the formula turns out to be unsatisfiable: namely, the Max-SAT problem consists in, given a formula F, to find an assignment such that the number of satisfied clauses in F is maximized, or equivalently, that the number of falsified clauses is minimized. This problem can in turn be generalized in a number of ways. For example, in *weighted Max-SAT* each clause of F has a *weight* (a positive natural or real number), and then the goal is to find the assignment such that the *cost*, i.e., the sum of the weights of the falsified clauses, is minimized. Yet a further extension of Max-SAT is the *partial weighted Max-SAT* problem, where clauses in F are either weighted clauses as explained above, called *soft clauses* in this setting, or clauses without weights, called *hard clauses*. In this case, the problem consists in finding the model of the hard clauses such that the sum of the weights of the falsified soft clauses is minimized. Equivalently, hard clauses can also be seen as soft clauses with infinite weight.

The problem of *Max-SMT* merges Max-SAT and SMT, and is defined from SMT analogously to how Max-SAT is derived from SAT. Namely, the *Max-SMT* problem consists in, given a set of pairs $\{[C_1, \omega_1], \ldots, [C_m, \omega_m]\}$, where each C_i is a clause and ω_i is its weight (a positive number or infinity), to find an assignment that minimizes the sum of the weights of the falsified clauses in the background theory. As in SMT, in this context we are interested in assignments of values from the theory to variables.

Finally, the problem of *Optimization Modulo Theories (OMT)* is similar to Max-SMT in that they are both optimization problems, rather than decision problems. It consists in, given a formula F involving a particular numerical variable called *cost*, to find the model of F such that the value assigned to *cost* is minimized. Note that this framework allows one to express a wide variety of optimization problems (maximization, piecewise linear objective functions, etc.).

2.2 Solving SMT(QF-NIA) with Unsatisfiable Cores

In [15], we proposed a method for solving SMT(QF-NIA) problems based on encoding them into SMT(QF-LIA). The basic idea is to linearize each non-linear monomial in the formula by applying a case analysis on the possible values of some of its variables. For example, if the monomial x^2yz appears in the input QF-NIA formula and x must satisfy $0 \le x \le 2$, we can introduce a fresh variable v_{x^2yz}, replace the occurrences of x^2yz by v_{x^2yz} and add to the clause set the following three *case splitting clauses*:

$$\begin{array}{rcl} x = 0 & \longrightarrow & v_{x^2yz} = 0\,, \\ x = 1 & \longrightarrow & v_{x^2yz} = yz\,, \\ x = 2 & \longrightarrow & v_{x^2yz} = 4yz\,. \end{array}$$

In turn, new non-linear monomials may appear, e.g., yz in this example. All non-linear monomials are handled in the same way until a formula in QF-LIA is obtained, for which efficient decision procedures exist [30, 33, 42].

Note that, in order to linearize a non-linear monomial, there must be at least one variable in it which is both lower and upper bounded. When this property does not hold, new *artificial* domains can be introduced for the variables that require them (for example, for unbounded variables one may take $\{-1, 0, 1\}$). In principle, this implies that the procedure is no longer complete, since a linearized formula with artificial bounds may be unsatisfiable while the original QF-NIA formula is actually satisfiable. A way to overcome this problem is to proceed iteratively: variables start with bounds that make the size of their domains small, and then the domains are enlarged on demand if necessary, i.e., if the formula turns out to be unsatisfiable. The decision of which bounds to change is heuristically taken based on the analysis of an *unsatisfiable core* (an unsatisfiable subset of the clause set) that is obtained when the solver reports unsatisfiability. There exist many techniques in the literature for computing unsatisfiable cores (see, e.g., [2] for a sample of them). In [15] we employed the well-known simple and effective approach of [78], consisting in writing a trace on disk and extracting a resolution refutation, whose leaves form an unsatisfiable core. Note that the method tells *which* bounds should be weakened, but does not provide any guidance in regard to *how large* the change on the bounds should be. This is critical, as the size of the formula in the next iteration (and so the time required to determine its satisfiability) can grow significantly depending on the number of new case splitting clauses that have to be added. Therefore, in lack of a better strategy, a typical heuristic is to decrement or increment the bound (for lower bounds and for upper bounds, respectively) by a constant value.

ALGORITHM 1: Algorithm for solving SMT(QF-NIA) with unsatisfiable cores

```
status solve_SMT_QF_NIA_cores(formula F0) { // returns whether F0 is satisfiable
  B = artificial_bounds(F0);                 // B are artificial bounds enough to linearize F0
  F = linearize(F0, B);
  while (not timed_out()) {
    ⟨ST, UC⟩ = solve_SMT_QF_LIA(F, B);       // unsatisfiable core UC computed here for simplicity
    if (ST == SAT) return SAT;               // is F ∧ B satisfiable?
    else if (B ∩ UC == ∅) return UNSAT;
    else {
      B′ = new_domains_cores(B, UC);         // at least one in the intersection is weakened
      F = update(F, B, B′);                  // add case splitting clauses
      B = B′;
    }
  }
  return UNKNOWN;
}
```

ALGORITHM 2: Procedure artificial_bounds

```
set artificial_bounds(formula F0) {          // returns the artificial bounds for linearization
  S = choose_linearization_variables(F0);    // choose enough variables to linearize F0
  B = ∅;                                     // set of artificial bounds
  for (V in S) {
    if (lower_bound(V, F0) == ⊥)             // cannot find lower bound of V in F0
      B = B ∪ { V ≥ L};                      // for a parameter L, e.g. L = −1
    if (upper_bound(V, F0) == ⊥)             // cannot find upper bound of V in F0
      B = B ∪ { V ≤ U};                      // for a parameter U, e.g. U = 1
  }
  return B;
}
```

Procedure solve_SMT_QF_NIA_cores in Algorithm 1 describes more formally the overall algorithm from [15] for solving SMT(QF-NIA). [2] First, the required artificial bounds are computed (procedure artificial_bounds, with pseudo-code in Algorithm 2). Then the linearized formula (procedure linearize, with pseudo-code in Algorithm 3) together with the artificial bounds are passed to an SMT(QF-LIA) solver (procedure solve_SMT_QF_LIA), which tests if their conjunction is satisfiable [3]. If the solver returns SAT, we are done. If the solver returns UNSAT, then an unsatisfiable core is also computed. If this core does not contain any of the artificial bounds, then the original non-linear formula must be unsatisfiable, and we are done too. Otherwise, at least one of the artificial bounds appearing in the core must be chosen to be weakened (procedure new_domains_cores, with pseudo-code in Algorithm 4). Once the domains are enlarged and the appropriate case splitting clauses are added (procedure update, with pseudo-code in Algorithm 5), the new linearized

[2]To avoid obfuscating the description of the algorithms with excessive details, pseudo-code in this paper uses self-explanatory generic types: **status** for the enumeration type with values SAT, UNSAT, UNKNOWN, **set** for sets of objects (formulas, bounds, etc.), **formula** for formulas in QF-NIA, QF-LIA, etc., **map** for assignments, and **number** for numbers. Tuples are represented with angle brackets ⟨ ⟩.

[3]Note that, in this formulation, the linearization consists of the clauses of the original formula after replacing non-linear monomials by fresh variables, together with the case splitting clauses. On the other hand, it does *not* include the artificial bounds, which for the sake of presentation are kept as independent objects.

ALGORITHM 3: Procedure linearize

```
formula linearize(formula F₀, set B) {                  // returns the linearization of F₀
  N = nonlinear_monomials(F₀);
  F = F₀;
  while (N ≠ ∅) {
    let Q in N;                                          // non-linear monomial to be linearized next
    V_Q = fresh_variable();
    F = replace(Q, F, V_Q);                              // replace all occurrences of Q as a monomial in F by V_Q
    C = ∅;                                               // clauses of the case splitting
    V = linearization_variable(Q);                       // choose a finite domain variable in Q to linearize
    for (K in [lower_bound(V, F₀ ∪ B), upper_bound(V, F₀ ∪ B)])
      C = C ∪ { V = K → V_Q = evaluate(Q, V, K)};
    F = F ∪ C;
    N = N  {Q} ∪ nonlinear_monomials(C);                 // new non-linear monomials may be introduced
  }
  return F;
}
```

ALGORITHM 4: Procedure new_domains_cores

```
set new_domains_cores(set B, set UC) {                   // returns the new set of artificial bounds
  let S ⊆ B ∩ UC such that S ≠ ∅;
  B′ = B;
  for (V ≥ L in S) B′ = B′  {V ≥ L} ∪ {V ≥ L′};          // e.g. L′ = L − K_L for a parameter K_L > 0
  for (V ≤ U in S) B′ = B′  {V ≤ U} ∪ {V ≤ U′};          // e.g. U′ = U + K_U for a parameter K_U > 0
  return B′;
}
```

ALGORITHM 5: Procedure update

```
formula update(formula F, set B, set B′) { // adds cases when weakening the bounds from B to B′
  F′ = F;
  for (V such that V ≥ L in B and V ≥ L′ in B′ and L ≠ L′)
    for (K in [L′, L − 1])
       for (Q such that V == linearization_variable(Q))     // V was used to linearize monomial Q
          F′ = F′ ∪ { V = K → V_Q = evaluate(Q, V, K)};     // V_Q is the variable standing for Q

  for (V such that V ≤ U in B and V ≤ U′ in B′ and U ≠ U′)
    for (K in [U + 1, U′])
       for (Q such that V == linearization_variable(Q))
          F′ = F′ ∪ { V = K → V_Q = evaluate(Q, V, K)};

  return F′;
}
```

formula is tested for satisfiability again, and the process is repeated (typically, while a predetermined time limit is not exceeded).

In our implementation (see Section 3.3), procedure new_domains_cores changes all artificial bounds in the core, that is, $S = B \cap UC$. As regards how to enlarge the domains, a good strategy turns out to do so slowly at the beginning, and then be more aggressive after some point. Namely, the values of K_L and K_U depend on the size of the domain: if it is equal to or less than 30, then $K_L = K_U = 1$; otherwise, $K_L = K_U = 30$. We refer the reader to [15] for further details.

Example 2.1. Let F_0 be the formula

$$tx + y \ \geq \ 4 \ \ \land \ \ t^2w^2 + t^2 + x^2 + y^2 + w^2 \ \leq \ 13,$$

where variables t, x, y, w are integer. Let us also assume that we introduce the following artificial bounds so as to linearize: $B \equiv -1 \leq t, x, y, w \leq 1$. Now a linearization F of F_0 could be for example:

$$\begin{array}{l}
v_{tx} + y \geq 4 \ \land \ v_{t^2w^2} + v_{t^2} + v_{x^2} + v_{y^2} + v_{w^2} \leq 13 \ \land \\
\begin{array}{llllllllll}
(t = -1 & \rightarrow & v_{tx} = -x) & \land & (t = -1 & \rightarrow & v_{t^2w^2} = v_{w^2}) & \land \\
(t = 0 & \rightarrow & v_{tx} = 0) & \land & (t = 0 & \rightarrow & v_{t^2w^2} = 0) & \land \\
(t = 1 & \rightarrow & v_{tx} = x) & \land & (t = 1 & \rightarrow & v_{t^2w^2} = v_{w^2}) & \land \\
\\
(t = -1 & \rightarrow & v_{t^2} = 1) & \land & (x = -1 & \rightarrow & v_{x^2} = 1) & \land \\
(t = 0 & \rightarrow & v_{t^2} = 0) & \land & (x = 0 & \rightarrow & v_{x^2} = 0) & \land \\
(t = 1 & \rightarrow & v_{t^2} = 1) & \land & (x = 1 & \rightarrow & v_{x^2} = 1) & \land \\
\\
(y = -1 & \rightarrow & v_{y^2} = 1) & \land & (w = -1 & \rightarrow & v_{w^2} = 1) & \\
(y = 0 & \rightarrow & v_{y^2} = 0) & \land & (w = 0 & \rightarrow & v_{w^2} = 0) & \\
(y = 1 & \rightarrow & v_{y^2} = 1) & \land & (w = 1 & \rightarrow & v_{w^2} = 1) &
\end{array}
\end{array}$$

where $v_{tx}, v_{t^2w^2}, v_{t^2}, v_{x^2}, v_{y^2}, v_{w^2}$ are fresh integer variables standing for the non-linear monomials in the respective subscripts.

In this case the formula $F \land B$ turns out to be unsatisfiable. For instance, the SMT(QF-LIA) solver could produce the following unsatisfiable core:

$$\begin{array}{lllllllll}
\{v_{tx} + y \geq 4, \\
\ \ t = -1 & \rightarrow & v_{tx} = -x, & y & \leq & 1, \\
\ \ t = 0 & \rightarrow & v_{tx} = 0, & -1 & \leq & t, & -1 & \leq & x, \\
\ \ t = 1 & \rightarrow & v_{tx} = x, & t & \leq & 1, & x & \leq & 1\}
\end{array}$$

Intuitively, if $|t|, |x|, y \leq 1$, then it cannot be the case that $tx + y \geq 4$. At this stage, one has to weaken at least one of the artificial bounds in the core, for example $x \leq 1$. Notice that, on the other hand, the core does not provide any help in regard to deciding the new upper bound for x. If, e.g., we chose that it were $x \leq 4$, then $x \leq 4$ would replace $x \leq 1$ in the set of artificial bounds B, and the following clauses would be added to the linearization F:

$$\begin{array}{lll}
x = 2 & \rightarrow & v_{x^2} = 4 \\
x = 3 & \rightarrow & v_{x^2} = 9 \\
x = 4 & \rightarrow & v_{x^2} = 16
\end{array}$$

In the next iteration one could already find solutions to the non-linear formula F_0, for instance, $t = v_{t^2} = w = v_{w^2} = v_{t^2w^2} = y = v_{y^2} = 1$, $x = v_{tx} = 3$, and $v_{x^2} = 9$. ∎

3 SOLVING SMT(QF-NIA) WITH MINIMAL MODELS

Taking into account the limitations of the method based on cores when domains have to be extended, in this section we present a model-guided approach to perform this step. Namely, we propose to replace the satisfiability check in linear arithmetic with an optimization call: Among all models of the linearization, even those that violate the artificial bounds, the linear solver will look for the one that is closest to being a solution to the original non-linear formula. Then this model will be used as a reference for weakening the bounds.

This is the key idea of the procedure solve_SMT_QF_NIA_min_models for solving SMT(QF-NIA) shown in Algorithm 6 (cf. Algorithm 1; note all subprocedures except for optimize_QF_LIA and new_domains_min_models are the same). Now the SMT(QF-LIA) black box (procedure optimize_QF_LIA) does not just decide satisfiability, but finds the minimal model of its input formula F according to a certain cost function. If this model does not satisfy the original non-linear formula, it can be employed as a hint in the domain enlargement (procedure new_domains_min_models, with pseudo-code in Algorithm 7) as follows. Since the non-linear formula is not satisfied, it must be the case that some of the artificial bounds are not respected by the minimal model. By gathering these bounds, a set of candidates to be weakened is obtained, as in the approach of Section 2.2. However, and most importantly, unlike with unsatisfiable cores now for each of these bounds a new value can be guessed too: one just needs to take the corresponding variable and enlarge its domain so that the value assigned in the minimal model is included. For example, let V be a variable whose artificial upper bound $V \leq U$ is falsified in the minimal model, and let U' be the value assigned to V in that model (hence, $U < U'$). Then $V \leq U'$ becomes the new upper bound for V. A similar construction applies for lower bounds.

The intuition behind this approach is that the cost function should measure how far assignments are from being a solution to the original non-linear formula. Formally, the function must be non-negative and have the property that the models of the linearized formula with cost 0 are those that satisfy all artificial bounds:

THEOREM 3.1. *Let F_0 be an arbitrary formula in QF-NIA, and F be any linearization of F_0 in QF-LIA obtained using the procedure linearize with artificial bounds B.*

A function cost that takes as input the models of F is admissible *if:*

(1) *$cost(M) \geq 0$ for any model M of F;*
(2) *$cost(M) = 0$ if and only if $M \models B$.*

If the cost functions in procedure solve_SMT_QF_NIA_min_models are admissible then the procedure is correct. That is, given a formula F_0 in QF-NIA:

(1) *if solve_SMT_QF_NIA_min_models (F_0) returns* SAT *then formula F_0 is satisfiable; and*
(2) *if solve_SMT_QF_NIA_min_models (F_0) returns* UNSAT *then formula F_0 is unsatisfiable.*

PROOF. (1) Let us assume that solve_SMT_QF_NIA_min_models (F_0) returns SAT. Then there is a set of artificial bounds B such that F, the linearization of F_0 using B, satisfies the following: optimize_QF_LIA (F, B) returns a model M of F such that $cost(M) = 0$. As *cost* is admissible we have that $M \models B$. But since F is a linearization of F_0 with artificial bounds B, we have that all additional variables standing for non-linear monomials have values in M that are consistent with the theory. Hence, we conclude that $M \models F_0$.

(2) Let us assume that solve_SMT_QF_NIA_min_models (F_0) returns UNSAT. Then there is a set of artificial bounds B such that F, the linearization of F_0 using B, satisfies that optimize_QF_LIA (F, B) returns UNSAT. By the specification of optimize_QF_LIA, this means that F is unsatisfiable. But since only case splitting clauses are added in the linearization, any model of F_0 can be extended to a model of F. By reversing the implication we conclude that F_0 must be unsatisfiable.

□

Under the assumption that cost functions are admissible, note that, if at some iteration in procedure solve_SMT_QF_NIA_min_models there are models of the linearization with null cost (hence

ALGORITHM 6: Algorithm for solving SMT(QF-NIA) with minimal models

```
status solve_SMT_QF_NIA_min_models(formula F0){  // returns whether F0 is satisfiable
  B = artificial_bounds(F0);                      // B are artificial bounds enough to linearize F0
  F = linearize(F0, B);
  while (not timed_out()) {

    // If ST == UNSAT then F is UNSAT
    // If ST == SAT then M is a model of F minimizing function cost below among all models of F
    ⟨ST, M⟩ = optimize_QF_LIA(F, B);

    if  (ST == UNSAT)      return UNSAT;
    else if  (cost(M) == 0) return SAT;
    else {
      B′ = new_domains_min_models(B, M);
      F = update(F, B, B′);                       // add case splitting clauses
      B = B′;
  } }
  return UNKNOWN;
}
```

ALGORITHM 7: Procedure new_domains_min_models

```
set new_domains_min_models(set B, map M) {            // returns the new set of artificial bounds
  let S ⊆ {b | b ∈ B, M ⊭ b} such that S ≠ ∅;          // choose among bounds violated by the model
  B′ = B;
  for (V ≥ L in S)  B′ = B′  {V ≥ L} ∪ {V ≥ M(V)};     // L > M(V) as M ⊭ V ≥ L
  for (V ≤ U in S)  B′ = B′  {V ≤ U} ∪ {V ≤ M(V)};     // U < M(V) as M ⊭ V ≤ U
  return B′;
}
```

satisfying the artificial bounds and the original non-linear formula), then the search is over: optimize_QF_LIA will return such a model, as it minimizes a non-negative cost function.

In what follows we propose two different admissible (classes of) cost functions: the *number* of violated artificial bounds (Section 3.1), and the *distance* with respect to the artificial domains (Section 3.2). In both cases, to complete the implementation of solve_SMT_QF_NIA_min_models the only procedure that needs to be defined is optimize_QF_LIA, as procedure new_domains_min_models is independent of the cost function (see Algorithm 7).

3.1 A Max-SMT(QF-LIA) Approach to Domain Enlargement

As sketched out above, a possibility is to define the cost of an assignment as the number of violated artificial bounds. A natural way of implementing this is to transform the original non-linear formula into a linearized weighted formula and use a Max-SMT(QF-LIA) tool. In this setting, the clauses of the linearization are hard, while the artificial bounds are considered to be soft (e.g., with weight 1 if we literally count the number of violated bounds). Procedure optimize_QF_LIA_Max_SMT is described formally in Algorithm 8. It is worth highlighting that not only is the underlying Max-SMT(QF-LIA) solver required to report the optimum value of the cost function, but it must also produce an assignment in the theory for which this optimum value is

ALGORITHM 8: Procedure optimize_QF_LIA_Max_SMT based on Max-SMT(QF-LIA)

⟨ **status**, **map** ⟩ optimize_QF_LIA_Max_SMT(**formula** F, **set** B) {
  $F' = F$;
  for ($V \geq L$ **in** B)
    $F' = F' \cup \{[V \geq L, 1]\}$;   // added as a soft clause, e.g. with weight 1
  for ($V \leq U$ **in** B)
    $F' = F' \cup \{[V \leq U, 1]\}$;   // added as a soft clause, e.g. with weight 1

  return solve_Max_SMT_QF_LIA(F'); // call to Max-SMT solver
}

attained (so that it can be used in the domain enlargement). A direct and effective way of accomplishing this task is by performing branch-and-bound on top of an SMT(QF-LIA) solver, as done in [59][4].

The next lemma justifies, together with Theorem 3.1, that solve_SMT_QF_NIA_min_models, when instantiated with optimize_QF_LIA_Max_SMT, is correct:

LEMMA 3.2. *Let F_0 be an arbitrary formula in QF-NIA, and F be any linearization of F_0 in QF-LIA obtained using the procedure linearize with artificial bounds B.*

The function cost that takes as an input a model M of F and returns the number of bounds from B that are not satisfied by M is admissible.

PROOF. It is clear that the function is non-negative. Moreover $cost(M) = 0$ if and only if all bounds in B are satisfied, i.e., $M \models B$. □

Regarding the weights of the soft clauses, as can be observed from the proof of Lemma 3.2, it is not necessary to have unit weights. One may use different values, provided they are positive, and then the cost function corresponds to a weighted sum. Moreover, note that weights can be different from one iteration of the loop of solve_SMT_QF_NIA_min_models to the next one.

Example 3.3. Let us consider the same formula as in Example 2.1:

$$tx + y \geq 4 \;\wedge\; t^2w^2 + t^2 + x^2 + y^2 + w^2 \leq 13.$$

Recall that, in this case, the artificial bounds are $-1 \leq t, x, y, w \leq 1$. We obtain the weighted formula consisting of the clauses of F (as defined in Example 2.1) as hard clauses, and

$$\begin{array}{ccccccccc} [-1 \leq t, 1] & \wedge & [-1 \leq x, 1] & \wedge & [-1 \leq y, 1] & \wedge & [-1 \leq w, 1] & \wedge \\ [t \leq 1, 1] & \wedge & [x \leq 1, 1] & \wedge & [y \leq 1, 1] & \wedge & [w \leq 1, 1] & \end{array}$$

as soft clauses (written following the format [clause, weight]).

In this case minimal solutions have cost 1: at least one of the artificial bounds has to be violated so as to satisfy $v_{tx} + y \geq 4$. For instance, the Max-SMT(QF-LIA) solver could return the assignment: $t = v_{t^2} = 1, x = v_{tx} = 4$ and $w = v_{w^2} = v_{t^2w^2} = y = v_{y^2} = v_{x^2} = 0$, where the only soft clause that is violated is $[x \leq 1, 1]$. Note that, as $x = 4$ is not covered by the case splitting clauses for v_{x^2}, the values of v_{x^2} and x are unrelated. Now the new upper bound for x would become $x \leq 4$ (so the

[4]Other approaches could also employed for solving Max-SMT(QF-LIA); for example, one could iteratively obtain unsatisfiable cores and add indicator variables and cardinality or pseudo-Boolean constraints to the instance until a SAT answer is obtained [1, 35, 58]. Nevertheless, here we opted for branch-and-bound for its simplicity and because it can be easily adapted to meet the requirements for solving Max-SMT(QF-NIA); see Section 4.

soft clause $[x \leq 1, 1]$ would be replaced by $[x \leq 4, 1]$), and similarly to Example 2.1, the following hard clauses would be added:

$$\begin{array}{lcl} x = 2 & \longrightarrow & v_{x^2} = 4 \\ x = 3 & \longrightarrow & v_{x^2} = 9 \\ x = 4 & \longrightarrow & v_{x^2} = 16 \end{array}$$

As seen in Example 2.1, in the next iteration there are solutions with cost 0, e.g., $t = v_{t^2} = w = v_{w^2} = v_{t^2w^2} = y = v_{y^2} = 1$, $x = v_{tx} = 3$ and $v_{x^2} = 9$. ■

One of the disadvantages of this approach is that potentially the Max-SMT(QF-LIA) solver could return models with arbitrarily large numerical values: note that what the cost function takes into account is just whether a bound is violated or not, but not by how much. For instance, in Example 2.1, it could have been the case that the Max-SMT(QF-LIA) solver returned $w = y = 0$, $t = 1$, $x = 10^5$, $v_{x^2} = 0$, etc. Since the model is used for extending the domains, a large number would involve adding a prohibitive number of case splitting clauses, and at the next iteration the Max-SMT(QF-LIA) solver would not able to handle the formula with a reasonable amount of resources. However, having said that, as far as we have been able to experiment, this kind of behaviour is rarely observed in our implementation; see Section 3.3 for more details. On the other hand, the cost function in Section 3.2 below does not suffer from this drawback.

3.2 An OMT(QF-LIA) Approach to Domain Enlargement

Another possibility of cost function for models of the linearization is to measure the distance with respect to the artificial domains. This can be cast as a problem in OMT(QF-LIA) as follows.

Given a non-linear formula F_0, let us consider a linearization F obtained after applying procedure linearize with artificial bounds B. Now, let vars(B) be the set of variables V for which an artificial domain $[L_V, U_V] \in B$ is added for the linearization. Formally, the cost function is $\sum_{V \in \text{vars}(B)} \delta(V, [L_V, U_V])$, where $\delta(z, [L, U])$ is the *distance* of z with respect to $[L, U]$:

$$\delta(z, [L, U]) = \begin{cases} L - z & \text{if } z < L \\ 0 & \text{if } L \leq z \leq U \\ z - U & \text{if } z > U \end{cases}$$

Note that, in the definition of the cost function, one can safely also include bounds which are not artificial but derived from the non-linear formula: the contribution to the cost of these is null, since they are part of the original formula and therefore must always be respected.

The approach is implemented in the procedure optimize_QF_LIA_OMT shown in Algorithm 9. In this procedure, an OMT(QF-LIA) solver is called (procedure solve_OMT_QF_LIA). Such a system can be built upon an existing SMT(QF-LIA) solver by adding an optimization simplex phase II [67] when the SAT engine reaches a leaf of the search space. For the OMT(QF-LIA) solver to handle the cost function, the problem requires the following reformulation. Let *cost* be the variable that the solver minimizes. For each variable $V \in$ vars(B) with domain $[L_V, U_V]$, let us introduce once and for all two extra integer variables l_V and u_V (meaning the distance with respect to the lower and to the upper bound of the domain of V, respectively) and the *auxiliary constraints* $l_V \geq 0$, $l_V \geq L_V - V$, $u_V \geq 0$, $u_V \geq V - U_V$. Then the cost function is determined by the equation $cost = \sum_{V \in \text{vars}(B)} (l_V + u_V)$, which is added to the formula together with the aforementioned auxiliary constraints.

The following result claims that the proposed cost function is admissible. Hence, by virtue of Theorem 3.1, if procedure optimize_QF_LIA_OMT is implemented as in Algorithm 9, then procedure solve_SMT_QF_NIA_min_models is sound:

ALGORITHM 9: Procedure optimize_QF_LIA_OMT based on OMT(QF-LIA)

```
⟨ status, map ⟩ optimize_QF_LIA_OMT(formula F, set B) {

  F′ = F;
  E = 0;                                   // expression for the cost function

  for (V ≥ L in B) {
    l_V = fresh_variable();
    F′ = F′ ∪ {l_V ≥ 0, l_V ≥ L − V};
    E = E + l_V;
  }

  for (V ≤ U in B) {
    u_V = fresh_variable();
    F′ = F′ ∪ {u_V ≥ 0, u_V ≥ V − U};
    E = E + u_V;
  }

  F′ = F′ ∪ {cost = E};                    // cost is the variable to be minimized

  return solve_OMT_QF_LIA(⟨cost, F′⟩);     // call to OMT solver
}
```

Lemma 3.4. *Let F_0 be an arbitrary formula in QF-NIA, and F be any linearization of F_0 in QF-LIA obtained using the procedure linearize with artificial bounds B.*

The function cost that takes as an input a model of F and returns its distance to the artificial domains:

$$\sum_{V \in \mathrm{vars}(B)} \delta(V, [L_V, U_V])$$

is admissible.

Proof. The proof is analogous to that of Lemma 3.2. □

Intuitively the proposed cost function corresponds to the *number of new cases* that will have to be added in the next iteration of the loop in solve_SMT_QF_NIA_min_models. However, it is also possible to consider slightly different cost functions: for instance, one could count the *number of new clauses* that will have to be added. For this purpose, it is only necessary to multiply variables l_V, u_V in the equation that defines *cost* by the number of monomials that were linearized by case splitting on V. In general, similarly to Section 3.1, one may have a template of cost function of the form $cost = \sum_{V \in \mathrm{vars}(B)} (\alpha_V\, l_V + \beta_V\, u_V)$, where $\alpha_V, \beta_V > 0$ for all $V \in \mathrm{vars}(B)$. Further, again these coefficients may be changed from one iteration to the next one.

Example 3.5. Yet again let us take the same non-linear formula from Example 2.1:

$$tx + y \ \geq\ 4 \ \ \wedge\ \ t^2w^2 + t^2 + x^2 + y^2 + w^2 \ \leq\ 13.$$

Let us also recall the artificial bounds: $-1 \leq t, x, y, w \leq 1$. By using the linearization F as defined in Example 2.1, one can express the resulting OMT(QF-LIA) problem as follows:

$$\min\ \ \delta(t, [-1, 1]) + \delta(x, [-1, 1]) + \delta(y, [-1, 1]) + \delta(w, [-1, 1]) \quad \text{subject to } F,$$

or equivalently,

$$
\begin{array}{c}
\min\ \ cost \quad \text{subject to} \\
F \ \wedge \\
\begin{array}{ccccccccccccccc}
l_t & \geq & 0 & \wedge & l_t & \geq & -1-t & \wedge & u_t & \geq & 0 & \wedge & u_t & \geq & t-1 & \wedge \\
l_x & \geq & 0 & \wedge & l_x & \geq & -1-x & \wedge & u_x & \geq & 0 & \wedge & u_x & \geq & x-1 & \wedge \\
l_y & \geq & 0 & \wedge & l_y & \geq & -1-y & \wedge & u_y & \geq & 0 & \wedge & u_y & \geq & y-1 & \wedge \\
l_w & \geq & 0 & \wedge & l_w & \geq & -1-w & \wedge & u_w & \geq & 0 & \wedge & u_w & \geq & w-1 & \wedge
\end{array} \\
cost = l_t + u_t + l_x + u_x + l_y + u_y + l_w + u_w
\end{array}
$$

In this case, it can be seen that minimal solutions have cost 1. For example, the OMT(QF-LIA) solver could return the assignment: $x = v_{x^2} = 1$, $t = 2$, $v_{tx} = 4$ and $w = v_{w^2} = v_{t^2} = v_{t^2w^2} = y = v_{y^2} = 0$. Note that, as $t = 2$ is not covered by the case splitting clauses, the values of v_{tx}, $v_{t^2w^2}$ and v_{t^2} are unrelated to t. Now the new upper bound for t is $t \leq 2$, and clauses

$$
\begin{array}{lcl}
t = 2 & \longrightarrow & v_{tx} = 2x \\
t = 2 & \longrightarrow & v_{t^2w^2} = 4v_{w^2} \\
t = 2 & \longrightarrow & v_{t^2} = 4
\end{array}
$$

are added to the linearization.

At the next iteration there is still no solution with cost 0, so at least another further iteration is necessary before a true model of the non-linear formula can be found. ■

One of the drawbacks of this approach is that, as the previous example suggests, domains may be enlarged very slowly. This implies that, in cases where solutions have large numbers, many iterations are needed before one of them is discovered. See Section 3.3 below for more details on the performance of this method in practice.

3.3 Experimental Evaluation of Model-guided Approaches

Here we evaluate experimentally our approaches for SMT(QF-NIA) and compare them with other non-linear SMT solvers, namely those participating in the QF-NIA division of the 2016 edition of SMT-COMP (http://smtcomp.sourceforge.net). More in detail, we consider the following tools:

- AProVE-NIA [36] with its default configuration;
- CVC4 [3] version of 05-27-2016;
- ProB [47];
- SMT-RAT [24];
- yices-2 [31] version 2.4.2;
- raSAT [74], with two different versions: raSAT-0.3 and raSAT-0.4 exp;
- z3 [27] version 4.4.1;
- bcl-cores, our core-based algorithm [15];
- bcl-maxsmt, our Max-SMT-based algorithm from Section 3.1;
- bcl-omt, our OMT-based algorithm from Section 3.2.

All bcl-* solvers[5] share essentially the same underlying SAT engine and QF-LIA theory solver. Moreover, some strategies are also common:

[5] Available at http://www.lsi.upc.edu/~albert/tocl2017.tgz.

- procedure artificial_bounds uses a greedy algorithm for approximating the minimum set of variables that have to be introduced in the linearization (as shown in [15], computing a set with minimum size is NP-complete). For each of these variables we force the domain $[-1, 1]$, even if variables have true bounds (for ease of presentation, we will assume here that true bounds always contain $[-1, 1]$). This turns out to be useful in practice, as quite often satisfiable formulas have solutions with small coefficients. By forcing the domain $[-1, 1]$, unnecessary case splitting clauses are avoided and the size of the linearized formula is reduced.
- the first time a bound is chosen to be weakened is handled specially. Let us assume it is the first time that a lower bound (respectively, an upper bound) of V has to be weakened. By virtue of the remark above, the bound must be of the form $V \geq -1$ (respectively, $V \leq 1$). Now, if V has a true bound of the form $V \geq L$ (respectively, $V \leq U$), then the new bound is the true bound. Otherwise, if V does not have a true lower bound (respectively, upper bound), then the lower bound is decreased by one (respectively, the upper bound is increased by one). Again, this is useful to capture the cases in which there are solutions with small coefficients.
- from the second time on, domain enlargement of bcl-maxsmt and bcl-omt follows basically what is described in Section 3, except for a correction factor aimed at instances in which solutions have some large values. Namely, if $V \leq u$ has to be weakened and in the minimal model V is assigned value U, then the new upper bound is $U \cdot \lfloor (n/C) + 1 \rfloor$, where C is a parameter which currently has value 30, and n is the number of times the upper bound of V has been weakened. As regards bcl-cores, a similar expression is used in which the current bound u is used instead of U, since there is no notion of "best model". The analogous strategy is applied for lower bounds.

The experiments were carried out on the StarExec cluster [71], whose nodes are equipped with Intel Xeon 2.4GHz processors. The memory limit was set to 60 GB, the same as in the 2016 edition of SMT-COMP. As regards wall clock time, although in SMT-COMP jobs were limited to 2400 seconds, in our experiments the timeout was set to 1800 seconds, which is the maximum that StarExec allowed us.

Two different sources of benchmarks were considered in this evaluation. The first benchmark suite (henceforth referred to as Term) was already used in the conference version of this paper [50] and consists of 1934 instances generated by the constraint-based termination prover described in [49]. In these problems non-linear monomials are quadratic.

The other benchmarks are the examples of QF-NIA in the SMT-LIB [4], which are grouped into the following families:

- AProVE: 8829 instances
- calypto: 177 instances
- LassoRanker: 120 instances
- leipzig: 167 instances
- mcm: 186 instances
- UltimateAutomizer: 7 instances
- UltimateLassoRanker: 32 instances
- LCTES: 2 instances

Results are displayed in two tables (Tables 1 and 2) for the sake of presentation. Rows represent systems and distinguish between SAT and UNSAT outcomes. Columns correspond to benchmark families. For each family, the number of instances is indicated in parentheses. The cells either show the number of problems of a given family that were solved by a particular system with outcome

Table 1. Experimental evaluation of SMT(QF-NIA) solvers on benchmark families Term, AProVE, calypto and LassoRanker.

		Term (1934)		AProVE (8829)		calypto (177)		LassoRanker (120)	
		# p.	time	# p.	time	# p.	time	# p.	time
AProVE-NIA	SAT	0	0.00	8,028	4,242.65	77	1,715.02	3	1.97
	UNSAT	0	0.00	0	0.00	0	0.00	0	0.00
CVC4	SAT	45	8,898.80	7,892	144,767.87	25	78.29	4	692.98
	UNSAT	0	0.00	10	0.18	35	0.71	71	1.26
ProB	SAT	0	0.00	7,415	19,715.86	41	85.52	3	3.07
	UNSAT	0	0.00	16	15.70	13	498.51	0	0.00
SMT-RAT	SAT	232	82,122.64	8,026	313.44	79	163.58	3	0.64
	UNSAT	15	1,377.74	221	7,654.78	89	663.89	21	12.59
yices-2	SAT	1,830	79,764.09	7,959	3,293.65	79	6.53	4	0.16
	UNSAT	69	940.15	**764**	**4,964.66**	**97**	**488.38**	97	875.44
raSAT-0.3	SAT	20	2,444.87	7,421	35,053.18	32	3,393.93	3	2.41
	UNSAT	0	0.00	320	554,482.86	47	30,232.16	43	75,603.23
raSAT-0.4 exp	SAT	36	5,161.97	7,745	50,695.06	31	954.16	3	1.54
	UNSAT	4	2,454.21	18	105.59	31	547.26	2	2.46
z3	SAT	194	77,397.16	8,023	14,790.21	79	943.03	4	13.16
	UNSAT	**70**	**3,459.77**	286	7,989.62	96	1,932.11	100	3,527.34
bcl-cores	SAT	1,857	4,396.09	**8,028**	**1,726.49**	80	6.20	4	0.09
	UNSAT	0	0.00	15	0.41	94	1,596.99	72	2.53
bcl-maxsmt	SAT	**1,857**	**811.54**	8,027	1,763.70	**80**	**5.74**	**4**	**0.08**
	UNSAT	67	31.33	202	51.50	97	994.17	**103**	**2.96**
bcl-omt	SAT	1,854	6,420.59	8,013	25,274.94	80	6.75	4	0.10
	UNSAT	67	34.99	203	36.18	97	1,327.95	103	3.59

SAT/UNSAT respectively (subcolumn "# p."), or the total time in seconds to process all problems of the family with that outcome (subcolumn "time"). The best solver for each family (for SAT and for UNSAT examples) is highlighted in bold face.

Due to lack of space, the results for family LCTES do not appear in the tables. This family consists of just two benchmarks, one of which was not solved by any system. The other instance was only solved by CVC4, which reported UNSAT in 0.5 seconds.

As the tables indicate, overall our techniques perform well on SAT instances, being the results particularly favourable for the Term family. This is natural: linearizing by case splitting is aimed at finding solutions quickly without having to pay the toll of heavy-weight non-linear reasoning. If satisfiable instances have solutions with small domains (which is often the case, for instance, when they come from our program analysis applications), our techniques usually work well. On the other hand, for families Aprove, leipzig and mcm the results are only comparable or slightly worse than those obtained with other tools[6]. One of the reasons could be that, at least for Aprove and leipzig, formulas have a very simple Boolean structure: they are essentially conjunctions of

[6]However, it must be remarked that we detected several inconsistencies between raSAT-0.3 and the rest of the solvers in the family mcm, which makes the results of this tool unreliable.

Table 2. Experimental evaluation of SMT(QF-NIA) solvers on benchmark families leipzig, mcm, UltimateAutomizer (UA) and UltimateLassoRanker (ULR).

		leipzig (167)		mcm (186)		UA (7)		ULR (32)	
		# p.	time	# p.	time	# p.	time	# p.	time
AProVE-NIA	SAT	161	1,459.27	0	0.00	0	0.00	6	5.02
	UNSAT	0	0.00	0	0.00	0	0.00	0	0.00
CVC4	SAT	**162**	**237.63**	**48**	**22,899.02**	0	0.00	6	3.76
	UNSAT	0	0.00	0	0.00	6	0.06	22	69.19
ProB	SAT	50	54.81	1	1,631.89	0	0.00	4	5.58
	UNSAT	0	0.00	0	0.00	1	1.02	1	1.34
SMT-RAT	SAT	160	2,827.37	21	2,516.21	0	0.00	6	0.86
	UNSAT	0	0.00	0	0.00	1	2.44	24	186.14
yices-2	SAT	92	715.04	11	5,816.44	0	0.00	**6**	**0.05**
	UNSAT	**1**	**0.01**	0	0.00	**7**	**0.02**	**26**	**11.07**
raSAT-0.3	SAT	32	15,758.07	2	1,787.57	0	0.00	2	5.88
	UNSAT	1	1,800.07	**99**	**178,204.54**	1	5.28	1	1,351.68
raSAT-0.4 exp	SAT	134	17,857.21	8	3,309.13	0	0.00	3	1.60
	UNSAT	0	0.00	0	0.00	1	8.08	1	1.50
z3	SAT	162	1,472.00	23	3,906.84	0	0.00	6	0.34
	UNSAT	0	0.00	7	7,127.61	7	0.54	26	45.20
bcl-cores	SAT	158	3,596.74	15	1,160.10	0	0.00	6	0.33
	UNSAT	0	0.00	0	0.00	1	0.06	24	32.87
bcl-maxsmt	SAT	153	4,978.91	17	1,004.25	0	0.00	6	0.31
	UNSAT	0	0.00	0	0.00	1	0.02	26	28.56
bcl-omt	SAT	148	7,351.45	19	2,937.99	0	0.00	6	0.34
	UNSAT	0	0.00	0	0.00	1	0.02	26	29.36

literals and few clauses (if any). For this particular kind of problems CAD-based techniques such as those implemented in yices-2 and z3, which are precisely targeted at conjunctions of non-linear literals, may be more adequate.

Regarding UNSAT instances, it can be seen that our approaches, while often competitive, can be outperformed by other tools in some families. Again, this is not surprising: linearizing may not be sufficient to detect unsatisfiability when deep non-linear reasoning is required. On the other hand, sometimes there may be a purely linear argument that proves that an instance is unsatisfiable. Our techniques can be effective in these situations, which may be relatively frequent depending on the application. This would be the case of families Term, calypto, LassoRanker and ULR.

Comparing our techniques among themselves, overall bcl-maxsmt tends to give the best results in terms of number of solved SAT and UNSAT instances and timings. For example, we can see that bcl-cores proves many fewer unsatisfiable instances than model-guided approaches. The reason is the following. Let F_0 be a formula in QF-NIA, and F be a linearization of F_0 computed with artificial bounds B. Let us assume that F is unsatisfiable. In this case, when the algorithm in bcl-cores tests the satisfiability of $F \wedge B$, it finds that it is unsatisfiable. Then, if we are lucky and an unsatisfiable core that only uses clauses from F is obtained, then it can be concluded that F_0 is unsatisfiable immediately. However, there may be other unsatisfiable cores of $F \wedge B$, which use

ALGORITHM 10: Procedure new_domains_min_models_non_inc

```
set new_domains_min_models_non_inc(set B, map M) {   // returns the new set of artificial bounds
  B' = {b | b ∈ B, M ⊨ b};
  W = {var(b) | b ∈ B, M ⊭ b};                       // variables whose domain is to be updated
  for (V in W)
    B' = B' ∪ {M(V) − R ≤ V ≤ M(V) + R};             // for a parameter R > 0
  return B';
}
```

artificial bounds [7]. Using such a core leads to performing yet another (useless) iteration of domain enlargement. Unfortunately the choice of the unsatisfiable core depends on the way the search space is explored, which does not take into account whether bounds are original or artificial so as not to interfere with the Boolean engine heuristics. On the other hand, model-guided approaches always detect when the linearization is unsatisfiable. As for SAT instances, the number of solved problems of bcl-cores is similar to that of bcl-maxsmt, but the latter tends to be faster.

Regarding bcl-omt, it turns out that, in general, the additional iterations required in the simplex algorithm to perform the optimization are too expensive. Moreover, after inspecting the traces we have confirmed that as Example 3.5 suggested, bcl-omt enlarges the domains too slowly, which is hindering the search.

3.4 Variants

According to the experiments in Section 3.3, altogether the approach based on Max-SMT(QF-LIA) gives the best results among our methods. In this section we propose several ideas for improving it further.

3.4.1 Non-incremental Strategy. A common feature of the procedures for solving SMT(QF-NIA) described in Sections 2.2, 3.1 and 3.2 is that, when no model of the linearization is found that satisfies all artificial bounds, the domains are enlarged. Thus, iteration after iteration, the number of case splitting clauses increases. In this sense, the aforementioned methods are *incremental*. A disadvantage of this incrementality is that, after some iterations, the formula size may easily become too big to be manageable.

On the other hand, instead of enlarging a domain, one can follow a *non-incremental* strategy and *replace* the domain by another one that might not include it. For example, in the model-guided approaches, when computing the new domain for a variable one may discard the current domain and for the next iteration take an interval centered at the value in the minimal model (procedure new_domains_min_models_non_inc, described in Algorithm 10). The update of the formula has to be adapted accordingly too, so that the case splitting clauses correspond to the values in the artificial domains (procedure update_non_inc, shown in Algorithm 11): for each of the variables whose domain has changed, the case splitting clauses of the values in the old domain must be removed, and case splitting clauses for the values in the new domain must be added. In this fashion one can control the number of case splitting clauses, and therefore the size of the formula.

Since monotonicity of domains from one iteration to the next one is now not maintained, this approach requires bookkeeping so as to avoid repeating the same choice of artificial domains. One way to achieve this is to add clauses that forbid each of the combinations of artificial bounds that

[7] For the sake of efficiency, bcl-cores does not guarantee that cores are minimal with respect to subset inclusion: computing *minimal unsatisfiable sets* [6] to eliminate irrelevant clauses implies an overhead that in our experience does not pay off. But even if minimality were always achieved, there could still be unsatisfiable cores in $F \wedge B$ using artificial bounds.

ALGORITHM 11: Procedure update_non_inc

```
formula update_non_inc(formula F, set B, set B′) {
  W = {var(b) | b ∈ B′ − B};                          // variables whose domain has changed
  F′ = F;
  for (V in W)
    for (Q such that V == linearization_variable(Q)) {  // V was used to linearize monomial Q

      let L, U such that L ≤ V ≤ U ∈ B;                // remove old case splitting clauses
      for (K in [L, U])
        F′ = F′ − { V = K → V_Q = evaluate(Q, V, K)};  // V_Q is the variable standing for Q

      let L′, U′ such that L′ ≤ V ≤ U′ ∈ B′;           // add new case splitting clauses
      for (K in [L′, U′])
        F′ = F′ ∪ { V = K → V_Q = evaluate(Q, V, K)};
    }
  return F′ ∪ {⋁_{b∈B} ¬b};                           // forbid B: no solution there
}
```

have already been tried and with which no model of the original formula was found. Namely, let B be such a combination of artificial bounds. We add the (hard) clause $\vee_{b\in B}\neg b$, which forces that at least one of the bounds in B cannot hold (see procedure update_non_inc in Algorithm 11). The following lemma ensures that, in any following iteration, domains cannot all be included in those of B (and therefore, work is not repeated).

Lemma 3.6. *Let F_0 be an arbitrary formula in QF-NIA, and F be any linearization of F_0 in QF-LIA obtained using the procedure linearize with artificial bounds B.*

Let us assume that in procedure solve_SMT_QF_NIA_min_models the cost functions are admissible and that the call optimize_QF_LIA(F, B) returns a model with positive cost, so that further iterations of the loop of solve_SMT_QF_NIA_min_models are required.

Let B' be a set of artificial bounds returned in one of those further iterations by a call to procedure new_domains_min_models_non_inc. Then the domains defined by B' are not all included in those of B: there exists an assignment α of values to variables such that $\alpha \models B'$ but $\alpha \not\models B$.

Proof. Let B'' be the set of bounds and let M be the minimal model such that we have $B' =$ new_domains_min_models_non_inc(B'', M). Now notice that $M \models B'$ since the procedure new_domains_min_models_non_inc takes exactly the variables that violate their artificial bounds in B'' and replaces these bounds by new ones that are satisfied by M. Notice also that $M \not\models B''$: if it were $M \models B''$ then, by the admissibility of the cost functions, the cost of M would be 0 and the call to new_domains_min_models_non_inc would not be made.

Now let us prove that $M \not\models B$. We distinguish two cases. First, if B' are the bounds at the iteration right after that of B, then $B = B''$ and $M \not\models B$ by the previous observation.

Otherwise, procedure update_non_inc has already added clause $\vee_{b\in B}\neg b$ to the formula. As M satisfies this clause, $M \not\models B$. □

Note that the above proof uses that, in procedure new_domains_min_models_non_inc, *all* variables that violate the artificial bounds in the minimal model get their domain updated. Hence the situation is different from the incremental setting, in which procedure new_domains_min_models has the freedom to choose for which variables the domain will be changed. In practice however this restriction is not relevant, as in our implementation of the incremental approach all variables that violate the artificial bounds are updated as well.

Finally also notice that, although this alternative non-incremental strategy for producing new artificial bounds can in principle be adapted to either of the model-guided methods, it makes the most sense for the Max-SMT(QF-LIA)-based procedure. The reason is that, being model-guided, in this approach the next domains to be considered are determined by the minimal model and, as already observed in Section 3.1, this minimal model may assign large values to variables and thus lead to intractable formula growth.

Example 3.7. Let us take the formula and artificial bounds of the running example. We resume Example 3.3, where the following minimal solution of cost 1 was shown: $t = v_{t^2} = 1$, $x = v_{tx} = 4$ and $w = v_{w^2} = v_{t^2 w^2} = y = v_{y^2} = v_{x^2} = 0$, being $x \le 1$ the only violated artificial bound. Now, taking a radius $R = 2$ for the interval around $x = 4$, in the next iteration the following artificial bounds would be considered: $-1 \le t, y, w \le 1$ and $2 \le x \le 6$. Moreover, the following clause would be added to the linearization:

$$-1 > t \quad \vee \quad 1 < t \quad \vee \quad -1 > x \quad \vee \quad 1 < x \quad \vee \quad -1 > y \quad \vee \quad 1 < y \quad \vee \quad -1 > w \quad \vee \quad 1 < w$$

together with

$$\begin{array}{lllr} (x = 2 & \longrightarrow & v_{x^2} = 4) & \wedge \\ (x = 3 & \longrightarrow & v_{x^2} = 9) & \wedge \\ (x = 4 & \longrightarrow & v_{x^2} = 16) & \wedge \\ (x = 5 & \longrightarrow & v_{x^2} = 25) & \wedge \\ (x = 6 & \longrightarrow & v_{x^2} = 36), & \end{array}$$

while clauses

$$\begin{array}{lllr} (x = -1 & \longrightarrow & v_{x^2} = 1) & \wedge \\ (x = 0 & \longrightarrow & v_{x^2} = 0) & \wedge \\ (x = 1 & \longrightarrow & v_{x^2} = 1) & \end{array}$$

would be removed. ■

3.4.2 Optimality Cores. When following the approach presented in Section 3.4.1, one needs to keep track of the combinations of domains that have already been attempted, in order to avoid repeating work and possibly entering into cycles. As pointed out above, this can be achieved for instance by adding clauses that exclude these combinations of domains. From the SMT perspective, these clauses can be viewed as *conflict explanations*, if one understands a conflict as a choice of artificial domains that does not lead to a solution to the original non-linear problem. Following the SMT analogy, it is important that explanations are as short as possible. In this section we present a technique aimed at reducing the *size* of each of these explanations [8].

Following the same reasoning as in Section 3.4.1, let us focus on the Max-SMT (QF-LIA) approach. The next definition is convenient to simplify notation:

Definition 3.8. Let W be a set of weighted clauses in QF-LIA. Given an assignment α, we define its *cost* with respect to W as $cost_W(\alpha) = \sum\{\omega \mid [C, \omega] \in W, \alpha \not\models C\}$.

Now we are ready to introduce *optimality cores*:

Definition 3.9. Let (F, B) be a weighted formula in QF-LIA with hard clauses F and soft clauses B. A set of weighted clauses O is an *optimality core* of (F, B) if the following conditions hold:

[8]Note however that this is different from reducing the *number* of explanations, that is, the number of iterations of the loop in procedure solve_SMT_QF_NIA_min_models.

(1) $O \subseteq B$; and
(2) $\min\{cost_O(M) \mid M \models F\} = \min\{cost_B(M) \mid M \models F\}$.

For the sake of simplicity, in what follows in this section we will assume that weights of soft clauses are all 1, and therefore weighted clauses can be represented like ordinary clauses.

Now let us consider a weighted formula (F, B) where F is the linearization of a formula in QF-NIA using artificial bounds B. If O is an optimality core of (F, B), then $O \subseteq B$ and the clause $\vee_{b \in O} \neg b$ has at most as many literals as the clause $\vee_{b \in B} \neg b$. The main idea in this section is that we can safely replace the latter by the former in procedure update_non_inc. Indeed, the following lemma shows that no solution of the original non-linear formula is lost:

Lemma 3.10. *Let F_0 be an arbitrary formula in QF-NIA, and F be any linearization of F_0 in QF-LIA obtained using the procedure linearize with artificial bounds B.*
Let O be an optimality core of the weighted formula (F, B). If $\min\{cost_B(M) \mid M \models F\} > 0$, then $F_0 \models \vee_{b \in O} \neg b$.

Proof. Let us reason by contradiction. Let us assume that there exists a model M_0 of F_0 such that $M_0 \not\models \vee_{b \in O} \neg b$, i.e., $M_0 \models \wedge_{b \in O} b$. Now let M be the assignment that extends M_0 by giving, to each of the variables introduced in the linearization, the value that results from evaluating the corresponding non-linear monomial. Then $M \models F$. Moreover, as $M_0 \models b$ for any $b \in O$, we have that $cost_O(M) = 0$. Therefore $\min\{cost_O(M) \mid M \models F\} = 0$. But since O is an optimality core of (F, B), this implies that $\min\{cost_B(M) \mid M \models F\} = 0$, which is a contradiction. □

As regards repeating domains and entering into cycles, the same argument of Section 3.4.1 holds: since $O \subseteq B$, we have that $\vee_{b \in O} \neg b \models \vee_{b \in B} \neg b$, and so the proof of Lemma 3.6 applies as well.

Finally, let us describe how optimality cores may be obtained. In a similar way to refutations obtained from executions of a DPLL(T) procedure on unsatisfiable instances [45], after each call to the Max-SMT(QF-LIA) solver on the linearization with soft artificial bounds one may retrieve a lower bound certificate [52]. This certificate consists essentially of a tree of *cost resolution* steps, and proves that any model of the linearization will violate at least as many artificial bounds as the reported optimal model. Now the set of artificial bounds that appear as leaves of this tree form an optimality core.

Example 3.11. Once more let us consider the running example. We proceed as in Example 3.7, but instead of adding the clause

$$-1 > t \;\vee\; 1 < t \;\vee\; -1 > x \;\vee\; 1 < x \;\vee\; -1 > y \;\vee\; 1 < y \;\vee\; -1 > w \;\vee\; 1 < w$$

we add

$$-1 > t \;\vee\; 1 < t \;\vee\; -1 > x \;\vee\; 1 < x \;\vee\; 1 < y,$$

i.e., we discard the literals $-1 > y$, $-1 > w$ and $1 < w$, since

$$\{[-1 \le t, 1], \quad [t \le 1, 1], \quad [-1 \le x, 1], \quad [x \le 1, 1], \quad [y \le 1, 1]\}$$

is an optimality core. Indeed, to satisfy $tx + y \ge 4$ at least one of these bounds must be violated. The lower bound certificate from which these bounds can be obtained semantically corresponds to the following refutation, which proves that not all bounds can be satisfied:

$$\dfrac{\dfrac{\dfrac{-1 \le t \qquad t \le 1 \qquad -1 \le x \qquad x \le 1}{tx \le 1} \qquad y \le 1}{tx + y \le 2} \qquad tx + y \ge 4}{\square}$$

Table 3. Experimental evaluation of SMT(QF-NIA) solvers on benchmark families Term, AProVE, calypto and LassoRanker.

		Term (1934)		AProVE (8829)		calypto (177)		LassoRanker (120)	
		# p.	time	# p.	time	# p.	time	# p.	time
yices-2	SAT	1,830	79,764.09	7,959	3,293.65	79	6.53	4	0.16
	UNSAT	69	940.15	**764**	**4,964.66**	97	488.38	97	875.44
z3	SAT	194	77,397.16	8,023	14,790.21	79	943.03	4	13.16
	UNSAT	**70**	**3,459.77**	286	7,989.62	96	1,932.11	100	3,527.34
bcl-maxsmt	SAT	1,857	811.54	8,027	1,763.70	80	5.74	**4**	**0.08**
	UNSAT	67	31.33	202	51.50	97	994.17	**103**	**2.96**
bcl-ninc	SAT	**1,857**	**276.20**	**8,028**	**1,777.97**	**80**	**5.50**	4	0.10
	UNSAT	67	191.66	202	51.72	**97**	**155.32**	103	13.63
bcl-ninc-cores	SAT	1,857	349.48	8,028	1,825.93	80	5.54	4	0.10
	UNSAT	67	184.41	202	57.52	97	273.15	103	9.31

■

3.5 Experimental Evaluation of Max-SMT(QF-LIA)-based Approaches

In this section we evaluate experimentally the variations of the Max-SMT(QF-LIA) approach proposed in Sections 3.4.1 and 3.4.2. In addition to the benchmarks used in Section 3.3, we have additionally considered instances produced by our constraint-based termination prover VeryMax (http://www.cs.upc.edu/~albert/VeryMax.html) on the divisions of the termination competition termCOMP 2016 (http://termination-portal.org/wiki/Termination_Competition) in which it participated, namely Integer Transition Systems and C Integer. Since internally VeryMax generates Max-SMT(QF-NIA) rather than SMT(QF-NIA) problems, soft clauses were removed. Given the huge number of obtained examples, of the order of tens of thousands, we could not afford carrying out the experiments will all tools considered in Section 3.3, and had to restrict the evaluation to the competing solvers that overall performed the best, namely z3 and yices-2. Hence, in addition to these two, the following solvers are considered here:

- bcl-maxsmt, the Max-SMT(QF-LIA)-based approach as in Section 3.3;
- bcl-ninc, the non-incremental algorithm from Section 3.4.1;
- bcl-ninc-cores, the non-incremental algorithm that uses optimality cores from Section 3.4.2.

Moreover, to further reduce the time required by the experiments, we decided to discard those benchmarks which could be solved both by yices-2 and bcl-maxsmt in negligible time (less than 0.5 seconds). After this filtering, finally 20354 and 2019 benchmarks were included in families Integer Transition Systems and C Integer, respectively.

Results are displayed in Tables 3, 4 and 5, following the same format as in Section 3.3. These results confirm that, in general, our techniques work well on SAT instances: except for families leipzig, mcm and UA, the best tool is one of the bcl-* solvers. The gap with respect to yices-2 and

Table 4. Experimental evaluation of SMT(QF-NIA) solvers on benchmark families leipzig, mcm, UltimateAutomizer (UA) and UltimateLassoRanker (ULR).

		leipzig (167)		mcm (186)		UA (7)		ULR (32)	
		# p.	time	# p.	time	# p.	time	# p.	time
yices-2	SAT	92	715.04	11	5,816.44	0	0.00	**6**	**0.05**
	UNSAT	**1**	**0.01**	0	0.00	**7**	**0.02**	**26**	**11.07**
z3	SAT	**162**	**1,472.00**	**23**	**3,906.84**	0	0.00	6	0.34
	UNSAT	0	0.00	**7**	**7,127.61**	7	0.54	26	45.20
bcl-maxsmt	SAT	153	4,978.91	17	1,004.25	0	0.00	6	0.31
	UNSAT	0	0.00	0	0.00	1	0.02	26	28.56
bcl-ninc	SAT	155	5,193.20	23	3,983.74	0	0.00	6	0.33
	UNSAT	0	0.00	0	0.00	1	0.02	26	28.47
bcl-ninc-cores	SAT	156	3,602.32	19	2,037.77	0	0.00	6	0.40
	UNSAT	0	0.00	0	0.00	1	0.02	26	31.28

Table 5. Experimental evaluation of SMT(QF-NIA) solvers on benchmark families LCTES, Integer Transition Systems (ITS) and C Integer (CI).

		LCTES (2)		ITS (20354)		CI (2019)	
		# p.	time	# p.	time	# p.	time
yices-2	SAT	0	0.00	8,408	471,160.33	714	84,986.50
	UNSAT	0	0.00	**4,085**	**142,965.19**	246	24,498.79
z3	SAT	0	0.00	5,993	784,681.66	566	16,827.79
	UNSAT	0	0.00	2,249	504,022.31	**504**	**17,919.88**
bcl-maxsmt	SAT	0	0.00	11,321	262,793.96	895	6,530.07
	UNSAT	0	0.00	2,618	35,838.06	148	14,481.72
bcl-ninc	SAT	0	0.00	**11,522**	**246,918.87**	**943**	**15,074.33**
	UNSAT	0	0.00	2,502	51,699.62	129	1,722.72
bcl-ninc-cores	SAT	0	0.00	11,504	244,201.03	941	12,174.59
	UNSAT	0	0.00	2,573	49,572.32	142	7,394.77

z3 is particularly remarkable on benchmarks coming from our termination proving applications (families Term, Integer Transition Systems and C Integer).

On the other hand, as was already justified in Section 3.3, regarding UNSAT problems, in some families the bcl-* solvers are clearly outperformed by the CAD-based techniques of yices-2 and z3. This suggests that a mixed approach that used our methods as a filter and that fell back to CAD after some time threshold could possibly take the best of both worlds.

Comparing our techniques among themselves, there is not an overall clear winner. For SAT examples, it can be seen that the non-incremental approach is indeed a useful heuristic: bcl-ninc tends to perform better, especially in the Integer Transition Systems and C Integer families. As regards optimality cores, as could be expected on SAT instances they do not prove profitable and result into a slight overhead of bcl-ninc-cores with respect to bcl-ninc. On the other hand, on UNSAT examples quite often (namely, families Term, LassoRanker, Integer Transition Systems and C Integer) the shorter conflict clauses discarding previous combinations of artificial domains help in detecting unsatisfiability more efficiently. Still, for this kind of instances bcl-maxsmt is usually the

ALGORITHM 12: Algorithm for solving Max-SMT(QF-NIA)

```
⟨status, map⟩ solve_Max_SMT_QF_NIA(formula F0) {        // returns if H0 is satisfiable and best model wrt. S0
                                                          // H0 are the hard clauses of F0 and S0 the soft ones
  B = artificial_bounds(F0);
  ⟨H, S⟩ = linearize(F0, B);
  best_so_far = ⊥;                                        // best model found so far
  max_soft_cost = ∞;                                      // maximum soft cost we can afford
  while (not timed_out()) {
    ⟨ST, M⟩ = optimize_QF_LIA_Max_SMT_threshold(H, S, B, max_soft_cost);
    if (ST == UNSAT)
      if (best_so_far == ⊥) return ⟨UNSAT, ⊥⟩;
      else                  return ⟨SAT, best_so_far⟩;
    else if (cost_B(M) == 0) {
      best_so_far = M;
      max_soft_cost = cost_S(M) − 1;                      // let us assume costs are natural numbers
    }
    else {
      B′ = new_domains_min_models(B, M);
      H = update(H, B, B′);                               // add case splitting clauses to the hard part
      B = B′;
    }
  }
  return ⟨UNKNOWN, ⊥⟩;
}
```

best of the three, since fewer iterations of the loop in procedure solve_SMT_QF_NIA_min_models are required to prove that the formula is unsatisfiable.

4 SOLVING MAX-SMT(QF-NIA)

This section is devoted to the extension of our techniques for SMT(QF-NIA) to Max-SMT(QF-NIA), which has a wide range of applications, e.g. in termination and non-termination proving [48, 49] as well as safety analysis [16]. Taking into account the results of the experiments in Sections 3.3 and 3.5, we will choose the Max-SMT(QF-LIA) approaches as SMT(QF-NIA) solving engines for the rest of this article. In particular, in the description of the following algorithms we will take as a reference the first version explained in Section 3.1, since adapting the algorithms to the variations from Sections 3.4.1 and 3.4.2 is easy.

4.1 Algorithm

We will represent the input F_0 of a Max-SMT(QF-NIA) instance as a conjunction of a set of hard clauses $H_0 = \{C_1, \cdots, C_n\}$ and a set of soft clauses $S_0 = \{[D_1, \Omega_1], \cdots, [D_m, \Omega_m]\}$. The aim is to decide whether there exist assignments α such that $\alpha \models H_0$, and if so, to find one such that $\sum\{\Omega \mid [D, \Omega] \in S_0, \alpha \not\models D\}$ is minimized.

Procedure solve_Max_SMT_QF_NIA for solving Max-SMT(QF-NIA) is shown in Algorithm 12. In its first step, as usual the initial artificial bounds B[9] are chosen (procedure artificial_bounds), with which the input formula $F_0 \equiv H_0 \wedge S_0$ is linearized (procedure linearize). As a result, a weighted linear formula is obtained with hard clauses H and soft clauses S, where:

[9]We will abuse notation and represent with B both the set of artificial bounds and also the corresponding set of weighted clauses. The exact meaning will be clear from the context.

ALGORITHM 13: Procedure optimize_QF_LIA_Max_SMT_threshold

```
⟨status, map⟩ optimize_QF_LIA_Max_SMT_threshold(formula H, formula S, set B, number msc) {
  F′ = H ∪ S;                              // H are hard clauses, S are soft
  for ([b, ω] in B)                        // typically ω = 1 is chosen
    F′ = F′ ∪ [b, ω];
  return solve_Max_SMT_QF_LIA(F′, msc);    // call to Max-SMT solver
}
```

- H results from replacing the non-linear monomials in H_0 by their corresponding fresh variables, and adding the case splitting clauses;
- S results from replacing the non-linear monomials in S_0 by their corresponding fresh variables.

Now notice that there are two kinds of weights: those from the original soft clauses, and those introduced in the linearization. As they have different meanings, it is convenient to consider them separately. Thus, given an assignment α, we define its *(total) cost* as $cost(\alpha) = (cost_B(\alpha), cost_S(\alpha))$, where $cost_B(\alpha) = \sum\{\omega \mid [b, \omega] \in B, \alpha \not\models b\}$ is the *bound cost*, i.e., the contribution to the total cost due to artificial bounds, and $cost_S(\alpha) = \sum\{\Omega \mid [D, \Omega] \in S, \alpha \not\models D\}$ is the *soft cost*, corresponding to the original soft clauses. Equivalently, if weights are written as pairs, so that artificial bound clauses become of the form $[C, (\omega, 0)]$ and soft clauses become of the form $[C, (0, \Omega)]$, we can write $cost(\alpha) = \sum\{(\omega, \Omega) \mid [C, (\omega, \Omega)] \in S \cup B, \alpha \not\models C\}$, where the sum of the pairs is component-wise. In what follows, pairs $(cost_B(\alpha), cost_S(\alpha))$ will be lexicographically compared, so that the bound cost (which measures the consistency with respect to the theory of QF-NIA) is more relevant than the soft cost. Hence, by taking this cost function and this ordering, we have a Max-SMT(QF-LIA) instance in which weights are not natural or non-negative real numbers, but pairs of them.

In the next step of solve_Max_SMT_QF_NIA procedure optimize_QF_LIA_Max_SMT_threshold (described in Algorithm 13) dispatches this Max-SMT instance. This procedure is like that presented in Algorithm 8, with the only difference that now a parameter *max_soft_cost* is passed to the Max-SMT(QF-LIA) solver. This parameter restrains the models of the hard clauses the solver will consider: only assignments α such that $cost_S(\alpha) \leq$ *max_soft_cost* will be taken into account. That is, this adapted Max-SMT solver computes, among the models α of the hard clauses such that $cost_S(\alpha) \leq$ *max_soft_cost* (if any), one that minimizes $cost(\alpha)$. Thus, the search can be pruned when it is detected that it is not possible to improve the best soft cost found so far. This adjustment is not difficult to implement if the Max-SMT solver follows a branch-and-bound scheme (see Section 3.1), as it is our case.

Then the algorithm examines the result of the call to the Max-SMT solver. If it is UNSAT, then there are no models of the hard clauses with soft cost at most *max_soft_cost*. Therefore, the algorithm can stop and report the best solution found so far, if any.

Otherwise, M satisfies the hard clauses and has soft cost at most *max_soft_cost*. If it has null bound cost, and hence is a true model of the hard clauses of the original formula, then the best solution found so far and *max_soft_cost* are updated, in order to search for a solution with better soft cost. Finally, if the bound cost is not null, then domains are enlarged as described in Section 3.1, in order to widen the search space. In any case, the algorithm jumps back and a new iteration is performed.

For the sake of simplicity, Algorithm 12 returns ⟨UNKNOWN, ⊥⟩ when time is exhausted. However, the best model found so far *best_so_far* can also be reported, as it can still be useful in practice.

The following theorem states the correctness of procedure solve_Max_SMT_QF_NIA:

THEOREM 4.1. *Procedure solve_Max_SMT_QF_NIA is correct. That is, given a weighted formula F_0 in QF-NIA with hard clauses H_0 and soft clauses S_0:*

(1) *if solve_Max_SMT_QF_NIA(F_0) returns $\langle \mathsf{SAT}, M\rangle$ then H_0 is satisfiable, and M is a model of H_0 that minimizes the sum of the weights of the falsified clauses in S_0; and*
(2) *if solve_Max_SMT_QF_NIA(F_0) returns $\langle \mathsf{UNSAT}, \perp\rangle$ then H_0 is unsatisfiable.*

PROOF. Let us assume that solve_Max_SMT_QF_NIA(F_0) returns $\langle \mathsf{SAT}, M\rangle$. The assignment M is different from $\perp$, and therefore has been previously computed in a call to the procedure optimize_QF_LIA_Max_SMT_threshold(H, S, B, *max_soft_cost*) such that $cost_B(M) = 0$. So M respects all artificial bounds in B. Thanks to the case splitting clauses in H, this ensures that auxiliary variables representing non-linear monomials have the right values. Therefore M satisfies H_0, which is what we wanted to prove. Now we just need to check that indeed M minimizes the sum of the weights of the falsified clauses in S_0. Notice that, from the specification of optimize_QF_LIA_Max_SMT_threshold, we know that there is no model of H such that its soft cost is strictly less than $cost_S(M)$. Now let M' be a model of H_0. By extending M' so that auxiliary variables representing non-linear monomials are assigned to their corresponding values, we have $M' \models H$. By the previous observation, $cost_{S_0}(M') = cost_S(M') \geq cost_S(M) = cost_{S_0}(M)$.

Now let us assume that solve_Max_SMT_QF_NIA(F_0) returns $\langle \mathsf{UNSAT}, \perp\rangle$. Let us also assume that there exists M' a model of H_0, and we will get a contradiction. Indeed, again extending M' as necessary, we have that $M' \models H$. If solve_Max_SMT_QF_NIA(F_0) returns $\langle \mathsf{UNSAT}, \perp\rangle$, then the previous call to optimize_QF_LIA_Max_SMT_threshold(H, S, B, *max_soft_cost*) has returned $\langle \mathsf{UNSAT}, \perp\rangle$, and moreover no previous call to optimize_QF_LIA_Max_SMT_threshold has produced a model with null bound cost. This means that *max_soft_cost* has not changed its initial value, namely ∞. Therefore H must be unsatisfiable, a contradiction. □

Example 4.2. Let F_0 be the weighted formula with hard clauses

$$H_0 \equiv tx + y \;\geq\; 4 \;\;\wedge\;\; t^2w^2 + t^2 + x^2 + y^2 + w^2 \;\leq\; 13$$

(the same of previous examples) and a single soft clause

$$S_0 \equiv [t^2 + x^2 + y^2 \leq 1,\; 1].$$

Let us take $-1 \leq t, x, y, w \leq 1$ as artificial bounds. After linearization, we get a weighted linear formula with hard clauses:

$$H \equiv \left(\begin{array}{l} v_{tx} + y \geq 4 \;\;\wedge\;\; v_{t^2w^2} + v_{t^2} + v_{x^2} + v_{y^2} + v_{w^2} \leq 13 \;\;\wedge \\ \begin{array}{lllllllll} (t = -1 & \rightarrow & v_{tx} = -x) & \wedge & (t = -1 & \rightarrow & v_{t^2w^2} = v_{w^2}) & \wedge \\ (t = 0 & \rightarrow & v_{tx} = 0) & \wedge & (t = 0 & \rightarrow & v_{t^2w^2} = 0) & \wedge \\ (t = 1 & \rightarrow & v_{tx} = x) & \wedge & (t = 1 & \rightarrow & v_{t^2w^2} = v_{w^2}) & \wedge \\ \\ (t = -1 & \rightarrow & v_{t^2} = 1) & \wedge & (x = -1 & \rightarrow & v_{x^2} = 1) & \wedge \\ (t = 0 & \rightarrow & v_{t^2} = 0) & \wedge & (x = 0 & \rightarrow & v_{x^2} = 0) & \wedge \\ (t = 1 & \rightarrow & v_{t^2} = 1) & \wedge & (x = 1 & \rightarrow & v_{x^2} = 1) & \wedge \\ \\ (y = -1 & \rightarrow & v_{y^2} = 1) & \wedge & (w = -1 & \rightarrow & v_{w^2} = 1) & \\ (y = 0 & \rightarrow & v_{y^2} = 0) & \wedge & (w = 0 & \rightarrow & v_{w^2} = 0) & \\ (y = 1 & \rightarrow & v_{y^2} = 1) & \wedge & (w = 1 & \rightarrow & v_{w^2} = 1) & \end{array} \end{array}\right)$$

and soft clauses

$$S \equiv [v_{t^2} + v_{x^2} + v_{y^2} \leq 1,\ (0,1)]$$

$$B \equiv \left(\begin{array}{cccc} [-1 \leq t,\ (1,0)] & \wedge & [t \leq 1,\ (1,0)] & \wedge \\ [-1 \leq x,\ (1,0)] & \wedge & [x \leq 1,\ (1,0)] & \wedge \\ [-1 \leq y,\ (1,0)] & \wedge & [y \leq 1,\ (1,0)] & \wedge \\ [-1 \leq w,\ (1,0)] & \wedge & [w \leq 1,\ (1,0)] & \end{array} \right),$$

where weights are already represented as pairs (bound cost, soft cost) as explained above.

In the first call to optimize_QF_LIA_Max_SMT_threshold(H, S, B, ∞), the optimal cost is $(1,0)$. An assignment with this cost that may be returned is, for example, $t = v_{t^2} = 1$, $x = v_{tx} = 4$ and $w = v_{w^2} = v_{t^2w^2} = y = v_{y^2} = v_{x^2} = 0$, the same in as Example 3.3. In this assignment, the only soft clause that is violated is $[x \leq 1, (1,0)]$.

Since the bound cost is not null, new artificial bounds should be introduced. Following Example 3.3, the new upper bound for x becomes $x \leq 4$. Hence, the soft clause $[x \leq 1, (1,0)]$ is replaced by $[x \leq 4, (1,0)]$), and the following hard clauses are added:

$$\begin{array}{lcl} x = 2 & \longrightarrow & v_{x^2} = 4 \\ x = 3 & \longrightarrow & v_{x^2} = 9 \\ x = 4 & \longrightarrow & v_{x^2} = 16 \end{array}$$

The following call to optimize_QF_LIA_Max_SMT_threshold returns an assignment with cost $(0,1)$, e.g., $t = v_{t^2} = w = v_{w^2} = v_{t^2w^2} = y = v_{y^2} = 1$, $x = v_{tx} = 3$ and $v_{x^2} = 9$. Since the bound cost is null, this assignment is recorded as the best model found so far and *max_soft_cost* is set to 0. This forces that, from now on, only solutions with null soft cost are considered, i.e., the soft clause $v_{t^2} + v_{x^2} + v_{y^2} \leq 1$ must hold. Since $t^2 + x^2 + y^2 \leq 1$ implies $|t|, |x|, |y| \leq 1$, which contradicts $tx + y \geq 4$, there is actually no solution of cost $(0,0)$. Hence next calls to optimize_QF_LIA_Max_SMT_threshold will unsuccessfully look for non-linear models with null soft cost, and eventually the search will time out. Note that, with the current set of clauses, the linear solver cannot prove unsatisfiability.

The previous example illustrates that showing optimality of the best model found so far requires proving unsatisfiability, more precisely that there cannot be a model with a better cost. Since our techniques are incomplete, this is a weakness of our approach. For this reason, to alleviate this problem additional redundant clauses can be introduced to describe the values of variables outside the finite domains. This enables the solver to prove unsatisfiability if linear reasoning with these clauses is sufficient.

Example 4.3. If the following clauses that describe the values outside the finite domains are introduced:

$$\begin{array}{lcl} x \leq -2 & \longrightarrow & v_{x^2} \geq 4 \\ x \geq 5 & \longrightarrow & v_{x^2} \geq 25\,, \end{array}$$

the unsatisfiability in the last step of the example will be detected by the procedure optimize_QF_LIA_Max_SMT_threshold. Then, instead of timing out, solve_Max_SMT_QF_NIA will terminate reporting that the minimum cost (with respect to the original soft clauses S_0) is 1, and that a model with that cost is given by $t = y = w = 1$ and $x = 3$. ∎

Table 6. Experimental evaluation of Max-SMT(QF-NIA) solvers on benchmark family Integer Transition Systems (20354 benchmarks).

	bcl-maxsmt		bcl-ninc		bcl-ninc-cores		z3	
	# p.	time	# p.	time	# p.	time	# p.	time
UNSAT	**2,618**	**32,947.80**	2,490	49,351.13	2,573	46,750.26	2,571	624,204.13
OPT	**7,644**	**449,806.47**	6,720	174,062.35	6,908	228,974.31	0	0.00
OPT + SAT	**8,311**	**490,204.00**	7,390	218,202.00	7,583	276,237.00	2,165	652,295.00

Table 7. Experimental evaluation of Max-SMT(QF-NIA) solvers on benchmark family C Integer (2019 benchmarks).

	bcl-maxsmt		bcl-ninc		bcl-ninc-cores		z3	
	# p.	time	# p.	time	# p.	time	# p.	time
UNSAT	144	9,027.27	121	3,993.28	136	8,930.16	**257**	**7,855.24**
OPT	453	9,090.26	466	9,177.07	**469**	**10,768.57**	0	0.00
OPT + SAT	522	9,108.00	535	9,194.00	**539**	**10,797.00**	207	23,579.00

4.2 Experimental Evaluation

In this section we evaluate experimentally the approach proposed in Section 4.1 for solving Max-SMT(QF-NIA). We adapt the method to each of the three Max-SMT(QF-LIA)-based variants for solving SMT(QF-NIA). Thus, following the same names as in Section 3.5, here we consider the solvers bcl-maxsmt, bcl-ninc and bcl-ninc-cores. We also include in the experiments z3, which is the only competing tool that, up to our knowledge, can handle Max-SMT(QF-NIA) too. As regards benchmarks, we use the original Max-SMT(QF-NIA) versions (that is, keeping soft clauses) of the examples Integer Transition Systems and C Integer employed in Section 3.5.

Tables 6 and 7 show the results of the experiments on the families Integer Transition Systems and C Integer, respectively. In each table, row UNSAT indicates the number of instances that were proved to be unsatisfiable, and row OPT counts the instances for which optimality of the reported model could be established. A third row OPT + SAT adds to row OPT the number of problems in which a model was found, but could not be proved to be optimal. For the sake of succinctness, as in previous tables other outcomes (timeouts, UNKNOWN answer, etc.) are not made explicit. Columns represent systems and show either the number of problems that were solved with outcome UNSAT/OPT/OPT or SAT respectively (subcolumn "# p."), or the total time in seconds to process all problems of the family with that outcome (subcolumn "time"). The best solver in each case is highlighted in bold face.

From the tables it can be observed that bcl-ninc-cores is more effective than bcl-ninc for Max-SMT. This is natural: proving the optimality of the best model found so far implicitly involves proving unsatisfiability, more precisely that there cannot be a model with a better cost. And as was already remarked in Section 3.5, optimality cores help the non-incremental approach to detect unsatisfiability more quickly. Regarding the incremental approach, the results are inconclusive: depending on the benchmarks, bcl-maxsmt may perform better than bcl-ninc-cores, or the other way around. Finally, z3 is competitive or even superior when dealing with unsatisfiable problems, while it significantly lags behind for the rest of the instances.

5 SOLVING SMT AND MAX-SMT($\exists\mathbb{Z}\,\forall\mathbb{R}$-NIRA)

In this section we will extend our techniques for SMT and Max-SMT(QF-NIA) to the theory of $\exists\mathbb{Z}\,\forall\mathbb{R}$-NIRA. In this fragment of the first-order theory of non-linear real and integer arithmetic, formulas are of the form $\exists x\,\forall y\,F(x,y)$, where F is a quantifier-free formula whose literals are polynomial inequalities. Moreover, the existentially quantified variables have integer type, whereas the universally quantified ones are real. In particular we will focus on a subset of this logic, namely, those formulas in which monomials never contain the product of two universally quantified variables.

This fragment of quantified non-linear arithmetic is relevant to many applications. For example, it appears in verification and synthesis problems when the so-called *template-based method* [21] is employed. In this framework, one attempts to discover an object of interest (e.g., an invariant, or a ranking function) by introducing a *template*, usually a linear inequality or expression, and solving a formula that represents the conditions the object should meet. For instance, let us find an invariant for the next loop:

$$\textbf{real}\ \ y\ =\ 0;\ \ \textbf{while}\ (y \le 2)\ \ y\ =\ y{+}1;$$

A loop invariant $I(y)$ must satisfy the following *initiation* and *inductiveness* conditions:

- **Initiation:** $\forall y_0\ \ (y_0 = 0\ \rightarrow\ I(y_0))$
- **Inductiveness:** $\forall y_1, y_2\ \ (I(y_1)\ \wedge\ y_1 \le 2\ \wedge\ y_2 = y_1 + 1\ \ \rightarrow\ \ I(y_2))$

Now a linear template $x_0\,y \le x_1$ is introduced as a candidate for $I(y)$, where x_0, x_1 are unknowns and y is the program variable. Then the conditions needed for $I(y)$ to be an invariant can be expressed in terms of template unknowns and program variables as an $\exists\forall$ formula:

$$\begin{array}{ll}\exists x_0, x_1\ \ \forall y_0, y_1, y_2 & \big((y_0 = 0\ \rightarrow\ x_0\,y_0 \le x_1) \wedge \\ & (x_0\,y_1 \le x_1\ \wedge\ y_1 \le 2\ \wedge\ y_2 = y_1 + 1\ \ \rightarrow\ \ x_0\,y_2 \le x_1)\big)\end{array}$$

This falls into the logical fragment considered here. Indeed note that, since the template is linear, the non-linear monomials in the formula always consist of the product of a template unknown and a program variable. Moreover, we can regard that we are interested in integer coefficients, so the existential variables are integers, while the universal variables are reals, since this is the type of program variable y. On the other hand, if one is interested in finding models with other type patterns, the following can be taken into account: in general, if a formula

$$\exists x \in \mathbb{Z}\ \ \forall y \in \mathbb{R}\ \ F(x,y)$$

is satisfiable, then so are

- $\exists x \in \mathbb{R}\ \ \forall y \in \mathbb{R}\ \ F(x,y)$,
- $\exists x \in \mathbb{Z}\ \ \forall y \in \mathbb{Z}\ \ F(x,y)$,
- $\exists x \in \mathbb{R}\ \ \forall y \in \mathbb{Z}\ \ F(x,y)$,

since the same witness x can be taken.

5.1 Algorithm

Let us first describe how to deal with the satisfiability problem given a formula $\exists x\,\forall y\,F(x,y)$, and then the technique will extend to the more general Max-SMT($\exists\mathbb{Z}\,\forall\mathbb{R}$-NIRA) problem naturally. Note that the requirement that monomials cannot contain the product of two universal variables allows writing the literals in F as linear polynomials in variables y, i.e., in the form $a_1(x)\,y_1 + \cdots + a_n(x)\,y_n \le b(x)$. Hence, if for instance F is a clause, we can write it as

$$\neg\Big(\bigwedge_{i=1}^{m} a_{i1}(x)\,y_1 + \cdots + a_{in}(x)\,y_n \le b_i(x) \;\wedge\; \bigwedge_{j=1}^{l} c_{j1}(x)\,y_1 + \cdots + c_{jn}(x)\,y_n < d_j(x)\Big),$$

or more compactly using matrix notation as $\neg\Big(A(x)\,y \le b(x) \;\wedge\; C(x)\,y < d(x)\Big)$.

The key idea (borrowed from [21][10]) is to apply the following result from polyhedral geometry to eliminate the quantifier alternation and transform the problem into a purely existential one:

Theorem 5.1 (Motzkin's Transposition Theorem [67]). *Let $A \in \mathbb{R}^{m\times n}$, $y \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, $C \in \mathbb{R}^{l\times n}$ and $d \in \mathbb{R}^l$. The system $Ay \le b \wedge Cy < d$ is unsatisfiable if and only if there are $\lambda \in \mathbb{R}^m$ and $\mu \in \mathbb{R}^l$ such that $\lambda \ge 0$, $\mu \ge 0$, $\lambda^T A + \mu^T C = 0$, $\lambda^T b + \mu^T d \le 0$, and $\lambda^T b < 0$ or $\mu \ne 0$.*

Thanks to Motzkin's Transposition Theorem, we have that formulas

$$\exists x\; \forall y\; \neg\big(A(x)\,y \le b(x) \;\wedge\; C(x)\,y < d(x)\big)$$

and

$$\exists x\, \exists \lambda\, \exists \mu\, \Big(\lambda, \mu \ge 0 \;\wedge\; \lambda^T A(x) + \mu^T C(x) = 0 \;\wedge\; \lambda^T b(x) + \mu^T d(x) \le 0 \;\wedge\; (\lambda^T b(x) < 0 \;\vee\; \mu \ne 0)\Big)$$

are equisatisfiable. In general, if the formula F in $\exists x\, \forall y\, F(x, y)$ is a CNF, this transformation is applied locally to each of the clauses with fresh multipliers.

Note that the formula resulting from applying Motzkin's Transposition Theorem is non-linear, but the existentially quantified variables λ and μ have real type. Fortunately, our techniques from Section 3 do not actually require that all variables are integer: it suffices that there are *enough* finite domain variables to perform the linearization. And this is indeed the case, since every non-linear monomial of the transformed formula has at most one occurrence of a λ or a μ variable, and all other variables are integer. All in all, we have reduced the problem of satisfiability of the fragment of $\exists\mathbb{Z}\,\forall\mathbb{R}$-NIRA under consideration to satisfiability of non-linear formulas that our approach can deal with.

Finally, as for Max-SMT, the technique extends clause-wise in a natural way. Given a weighted CNF, hard clauses are transformed using Motzkin's Transposition Theorem as in the SMT case. As for soft clauses, let $[S, \Omega]$ be such a clause, where S is of the form $\neg(A(x)\,y \le b(x) \wedge C(x)\,y < d(x))$. Then a fresh propositional symbol p_S is introduced, and $[S, \Omega]$ is replaced by a soft clause $[p_S, \Omega]$ and hard clauses corresponding to the double implication

$$\Big(\lambda, \mu \ge 0 \;\wedge\; \lambda^T A(x) + \mu^T C(x) = 0 \;\wedge\; \lambda^T b(x) + \mu^T d(x) \le 0 \;\wedge\; (\lambda^T b(x) < 0 \;\vee\; \mu \ne 0)\Big) \leftrightarrow p_S.$$

Therefore, similarly to satisfiability, we can solve the Max-SMT problem for the fragment of $\exists\mathbb{Z}\,\forall\mathbb{R}$-NIRA of interest by reducing it to instances that can be handled with the techniques presented in Section 4.

Example 5.2. Let us consider again the problem of finding an invariant for the loop:

real $y = 0$; **while** $(y \le 2)$ $y = y{+}1$;

However, now we will make the initiation condition soft, say with weight 1, while the inductiveness condition will remain hard (as done in [16]). The rationale is that, if the initiation condition can be satisfied, then we have a true invariant; and if it is not, then at least we have a *conditional invariant*: a property that, if at some iteration holds, then from that iteration on it always holds.

[10] In [21], Farkas' Lemma is used instead of the generalization presented here.

Using the same template as above, the formula to be solved is (quantifiers are omitted for the sake of presentation):

$$\begin{array}{l}[y_0 = 0 \;\rightarrow\; x_0\, y_0 \le x_1,\; 1] \wedge \\ (x_0\, y_1 \le x_1 \;\wedge\; y_1 \le 2 \;\wedge\; y_2 = y_1 + 1 \;\;\rightarrow\;\; x_0\, y_2 \le x_1)\end{array}$$

After moving the right-hand side of the implication to the left, and applying some simplifications, it results into:

$$\begin{array}{l}[0 \le x_1,\; 1] \wedge \\ \neg(x_0\, y_1 \le x_1 \;\wedge\; y_1 \le 2 \;\wedge\; x_0\,(y_1 + 1) > x_1)\end{array}$$

Now the transformation is performed clause by clause. Since the first clause $[0 \le x_1,\; 1]$ does no longer contain universally quantified variables, it can be left as it is. As regards the second one, we introduce three fresh multipliers λ_1, λ_2, and μ and replace

$$\neg(x_0\, y_1 \le x_1 \;\wedge\; y_1 \le 2 \;\wedge\; x_0\,(y_1 + 1) > x_1)$$

by

$$\Big(\lambda_1 \ge 0 \wedge \lambda_2 \ge 0 \wedge \mu \ge 0 \wedge \lambda_1 x_0 + \lambda_2 - \mu x_0 = 0 \;\wedge$$
$$\lambda_1 x_1 + 2\lambda_2 + \mu(x_0 - x_1) \le 0 \wedge (\lambda_1 x_1 + 2\lambda_2 < 0 \vee \mu \ne 0)\Big)$$

All in all, the following Max-SMT instance must be solved:

$$[0 \le x_1,\; 1] \;\wedge$$
$$\Big(\lambda_1 \ge 0 \wedge \lambda_2 \ge 0 \wedge \mu \ge 0 \wedge \lambda_1 x_0 + \lambda_2 - \mu x_0 = 0 \;\wedge$$
$$\lambda_1 x_1 + 2\lambda_2 + \mu(x_0 - x_1) \le 0 \wedge (\lambda_1 x_1 + 2\lambda_2 < 0 \vee \mu \ne 0)\Big)$$

There exist many solutions with cost 0, each of them corresponding to a loop invariant; for instance, $x_0 = 1, x_1 = 3, \lambda_1 = 0, \lambda_2 = 1, \mu = 1$ (which represents the invariant $y \le 3$). ■

5.2 Experimental Evaluation

In this section we evaluate experimentally the approach proposed in Section 5.1 for solving Max-SMT($\exists\mathbb{Z}\,\forall\mathbb{R}$-NIRA). Similarly to Section 4.2, again we instantiate the method for the three Max-SMT(QF-LIA)-based variants for solving SMT(QF-NIA). So, using the same names as in Sections 3.5 and 4.2, in this evaluation we consider the solvers bcl-maxsmt, bcl-ninc and bcl-ninc-cores. Unfortunately, as far as we know no competing tool can handle the problems of Max-SMT($\exists\mathbb{Z}\,\forall\mathbb{R}$-NIRA) effectively. Hence, we have to limit our experiments to our own tools.

Regarding benchmarks, again we use the weighted formulas of the families Integer Transition Systems and C Integer, employed in Section 4.2. However, here problems are expressed in Max-SMT($\exists\mathbb{Z}\,\forall\mathbb{R}$-NIRA) rather than in Max-SMT(NIA); that is, Motzkin's Transposition Theorem is applied silently inside the solver, and not in the process of generating the instances. Moreover, as illustrated in Example 5.2, Max-SMT($\exists\mathbb{Z}\,\forall\mathbb{R}$-NIRA) problems coming from the application of the template-based method can usually be simplified, e.g., by using equations to eliminate variables. In order to introduce some variation with respect to the evaluation in Section 4.2, we decided to experiment with the Max-SMT($\exists\mathbb{Z}\,\forall\mathbb{R}$-NIRA) problems in raw form, without simplifications. Another difference is that, while in Section 4.2 multipliers were considered integer variables (so that purely integer problems were obtained), in this evaluation they have real type.

Table 8. Experimental evaluation of Max-SMT($\exists\mathbb{Z}\,\forall\mathbb{R}$-NIRA) solvers on benchmark family Integer Transition Systems (20354 benchmarks).

	bcl-maxsmt		bcl-ninc		bcl-ninc-cores	
	# p.	time	# p.	time	# p.	time
UNSAT	**2,196**	**89,259.58**	2,119	121,556.46	2,031	140,585.27
OPT	**6,707**	**1,002,816.92**	5,902	405,813.78	5,856	401,333.33
OPT + SAT	**7,337**	**1,071,480.43**	6,536	475,622.68	6,485	467,898.84

Table 9. Experimental evaluation of Max-SMT($\exists\mathbb{Z}\,\forall\mathbb{R}$-NIRA) solvers on benchmark family C Integer (2019 benchmarks).

	bcl-maxsmt		bcl-ninc		bcl-ninc-cores	
	# p.	time	# p.	time	# p.	time
UNSAT	**88**	**10,095.79**	64	1,992.78	76	2,545.89
OPT	360	11,928.57	374	13,223.96	**379**	**15,173.59**
OPT + SAT	429	13,985.45	442	13,811.42	**447**	**15,818.40**

Results are shown in Tables 8 and 9, following the same format as in Section 4.2. It is worth noticing that the number of solved instances is significantly smaller than in Tables 6 and 7, respectively. This shows the usefulness of the simplifications performed when generating the Max-SMT(NIA) instances. Regarding which tool for Max-SMT($\exists\mathbb{Z}\,\forall\mathbb{R}$-NIRA) among the three is the most powerful, on SAT instances there is not a global winner, while on unsatisfiable ones bcl-maxsmt has the best results for both families.

6 CONCLUSIONS AND FUTURE WORK

In this article we have proposed two strategies to guide domain enlargement in the instantiation-based approach for solving SMT(QF-NIA) [15]. Both are based on computing minimal models with respect to a cost function, namely: the number of violated artificial domain bounds, and the distance with respect to the artificial domains. We have experimentally argued that the former gives better results than the latter and previous techniques, and have devised further improvements, based on weakening the invariant that artificial domains should grow monotonically, and exploiting optimality cores. Finally, we have developed and implemented algorithms for Max-SMT(QF-NIA) and for Max-SMT($\exists\mathbb{Z}\,\forall\mathbb{R}$-NIRA), logical fragments with important applications to program analysis and termination but which are missing effective tools.

As for future work, several directions for further research can be considered. Regarding the algorithmics, it would be interesting to look into different cost functions following the model-guided framework proposed here, as well as alternative ways for computing those minimal models (e.g., by means of *minimal correction subsets* [11, 57]). Besides, one of the shortcomings of our instantiation-based approach is that it cannot deal with unsatisfiable instances that require complex non-linear reasoning. This is particularly inconvenient in Max-SMT(QF-NIA) and Max-SMT($\exists\mathbb{Z}\,\forall\mathbb{R}$-NIRA), since solving any instance eventually requires proving optimality, that is, showing that the problem of finding a better solution than the best one found so far is unsatisfiable. In this context, the integration of real-goaled CAD techniques adapted to SMT [43] as a fallback or run in parallel appears to be a promising line of work. This would also alleviate another of the limitations of our approach, namely the handling of formulas with little Boolean structure, for which CAD-based techniques are more appropriate.

Another direction for future research concerns applications. So far we have applied our methods for Max-SMT/SMT(QF-NIA/$\exists\mathbb{Z}\,\forall\mathbb{R}$-NIRA) to array invariant generation [51], safety [16], termination [49] and non-termination [48] proving. Other problems in program analysis where we envision these techniques could help in improving the state-of-the-art are, e.g., the analysis of worst-case execution time, resource analysis, program synthesis and automatic bug fixing. Also, so far we have only considered sequential programs. The extension of Max-SMT-based techniques to concurrent programs is a promising line of work with a potentially high impact in the industry.

ACKNOWLEDGMENTS

All authors partially supported by the Spanish Ministerio de Economía y Competitividad under grant TIN2015-69175-C4-3-R (Spanish MINECO/FEDER UE) and by the Spanish Ministerio de Ciencia, Innovación y Universidades under grant RTI2018-094403-B-C33 (Spanish MICINN/FEDER UE). Albert Oliveras is partially funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement ERC-2014-CoG 648276 AUTAR).

The authors would also like to thank the anonymous referees for their helpful comments.

REFERENCES

[1] Carlos Ansótegui, Maria Luisa Bonet, and Jordi Levy. 2013. SAT-based MaxSAT algorithms. *Artif. Intell.* 196 (2013), 77–105. https://doi.org/10.1016/j.artint.2013.01.002

[2] Roberto Asín Achá, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. 2010. Practical algorithms for unsatisfiability proof and core generation in SAT solvers. *AI Communications* 23, 2-3 (2010), 145–157.

[3] Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanovic, Tim King, Andrew Reynolds, and Cesare Tinelli. 2011. CVC4. In *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings (Lecture Notes in Computer Science)*, Ganesh Gopalakrishnan and Shaz Qadeer (Eds.), Vol. 6806. Springer, 171–177. https://doi.org/10.1007/978-3-642-22110-1_14

[4] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. 2016. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`.

[5] S. Basu, R. Pollack, and M.-F. Roy. 2003. *Algorithms in Real Algebraic Geometry.* Springer-Verlag, Berlin.

[6] Anton Belov, Inês Lynce, and João Marques-Silva. 2012. Towards efficient MUS extraction. *AI Commun.* 25, 2 (2012), 97–116.

[7] Rachel Ben-Eliyahu and Rina Dechter. 1996. On Computing Minimal Models. *Ann. Math. Artif. Intell.* 18, 1 (1996), 3–27. https://doi.org/10.1007/BF02136172

[8] Rachel Ben-Eliyahu-Zohary. 2005. An incremental algorithm for generating all minimal models. *Artif. Intell.* 169, 1 (2005), 1–22. https://doi.org/10.1016/j.artint.2005.06.003

[9] Tewodros Beyene, Swarat Chaudhuri, Corneliu Popeea, and Andrey Rybalchenko. 2014. A Constraint-based Approach to Solving Games on Infinite Graphs. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '14).* ACM, New York, NY, USA, 221–233. https://doi.org/10.1145/2535838.2535860

[10] Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh (Eds.). 2009. *Handbook of Satisfiability.* Frontiers in Artificial Intelligence and Applications, Vol. 185. IOS Press. 980 pages.

[11] Nikolaj Bjørner and Nina Narodytska. 2015. Maximum Satisfiability Using Cores and Correction Sets. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, Qiang Yang and Michael Wooldridge (Eds.). AAAI Press, 246–252. http://ijcai.org/Abstract/15/041

[12] Nikolaj Bjørner, Anh-Dung Phan, and Lars Fleckenstein. 2015. νZ - An Optimizing SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings (Lecture Notes in Computer Science)*, Christel Baier and Cesare Tinelli (Eds.), Vol. 9035. Springer, 194–199. https://doi.org/10.1007/978-3-662-46681-0_14

[13] Roderick Bloem and Natasha Sharygina (Eds.). 2010. *Proceedings of 10th International Conference on Formal Methods in Computer-Aided Design, FMCAD 2010, Lugano, Switzerland, October 20-23.* IEEE.

[14] Maria Paola Bonacina (Ed.). 2013. *Automated Deduction - CADE-24 - 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013. Proceedings.* Lecture Notes in Computer Science, Vol. 7898. Springer. https://doi.org/10.1007/978-3-642-38574-2

[15] Cristina Borralleras, Salvador Lucas, Albert Oliveras, Enric Rodríguez-Carbonell, and Albert Rubio. 2012. SAT Modulo Linear Arithmetic for Solving Polynomial Constraints. *J. Autom. Reasoning* 48, 1 (2012), 107–131.

[16] Marc Brockschmidt, Daniel Larraz, Albert Oliveras, Enric Rodríguez-Carbonell, and Albert Rubio. 2015. Compositional Safety Verification with Max-SMT. In *Formal Methods in Computer-Aided Design, FMCAD 2015, Austin, Texas, USA, September 27-30, 2015.*, Roope Kaivola and Thomas Wahl (Eds.). IEEE, 33–40.

[17] Chih-Hong Cheng, Natarajan Shankar, Harald Ruess, and Saddek Bensalem. 2013. EFSMT: A Logical Framework for Cyber-Physical Systems. *CoRR* abs/1306.3456 (2013). http://arxiv.org/abs/1306.3456

[18] Alessandro Cimatti, Anders Franzén, Alberto Griggio, Roberto Sebastiani, and Cristian Stenico. 2010. Satisfiability Modulo the Theory of Costs: Foundations and Applications. In *TACAS (Lecture Notes in Computer Science)*, Javier Esparza and Rupak Majumdar (Eds.), Vol. 6015. Springer, 99–113.

[19] George E. Collins. 1975. Hauptvortrag: Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Automata Theory and Formal Languages (Lecture Notes in Computer Science)*, H. Barkhage (Ed.), Vol. 33. Springer, 134–183.

[20] Michael Colón, Sriram Sankaranarayanan, and Henny Sipma. 2003. Linear Invariant Generation Using Non-linear Constraint Solving. In *CAV (Lecture Notes in Computer Science)*, Warren A. Hunt Jr. and Fabio Somenzi (Eds.), Vol. 2725. Springer, 420–432.

[21] Michael Colón and Henny Sipma. 2002. Practical Methods for Proving Program Termination. In *CAV (Lecture Notes in Computer Science)*, Ed Brinksma and Kim Guldstrand Larsen (Eds.), Vol. 2404. Springer, 442–454.

[22] S. Barry Cooper. 2004. *Computability Theory.* Chapman Hall/CRC Mathematics Series.

[23] Florian Corzilius and Erika Ábrahám. 2011. Virtual Substitution for SMT-Solving. In *Fundamentals of Computation Theory - 18th International Symposium, FCT 2011, Oslo, Norway, August 22-25, 2011. Proceedings (Lecture Notes in Computer Science)*, Olaf Owe, Martin Steffen, and Jan Arne Telle (Eds.), Vol. 6914. Springer, 360–371. https://doi.org/10.1007/978-3-642-22953-4_31

[24] Florian Corzilius, Gereon Kremer, Sebastian Junges, Stefan Schupp, and Erika Ábrahám. 2015. SMT-RAT: An Open Source C++ Toolbox for Strategic and Parallel SMT Solving. In *Theory and Applications of Satisfiability Testing - SAT 2015 - 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings (Lecture Notes in Computer Science)*, Marijn Heule and Sean Weaver (Eds.), Vol. 9340. Springer, 360–368. https://doi.org/10.1007/978-3-319-24318-4_26

[25] Florian Corzilius, Ulrich Loup, Sebastian Junges, and Erika Ábrahám. 2012. SMT-RAT: An SMT-Compliant Nonlinear Real Arithmetic Toolbox - (Tool Presentation). In *SAT (Lecture Notes in Computer Science)*, Alessandro Cimatti and Roberto Sebastiani (Eds.), Vol. 7317. Springer, 442–448.

[26] Leonardo Mendonça de Moura and Nikolaj Bjørner. 2008. Model-based Theory Combination. *Electr. Notes Theor. Comput. Sci.* 198, 2 (2008), 37–49. https://doi.org/10.1016/j.entcs.2008.04.079

[27] Leonardo Mendonça de Moura and Nikolaj Bjørner. 2008. Z3: An Efficient SMT Solver. In *TACAS (Lecture Notes in Computer Science)*, C. R. Ramakrishnan and Jakob Rehof (Eds.), Vol. 4963. Springer, 337–340.

[28] Leonardo Mendonça de Moura and Dejan Jovanovic. 2013. A Model-Constructing Satisfiability Calculus. In *VMCAI (Lecture Notes in Computer Science)*, Roberto Giacobazzi, Josh Berdine, and Isabella Mastroeni (Eds.), Vol. 7737. Springer, 1–12.

[29] Leonardo Mendonça de Moura and Grant Olney Passmore. 2013. Computation in Real Closed Infinitesimal and Transcendental Extensions of the Rationals, See [14], 178–192. https://doi.org/10.1007/978-3-642-38574-2

[30] Isil Dillig, Thomas Dillig, and Alex Aiken. 2011. Cuts from proofs: a complete and practical technique for solving linear inequalities over integers. *Formal Methods in System Design* 39, 3 (2011), 246–260. https://doi.org/10.1007/s10703-011-0127-z

[31] Bruno Dutertre. 2014. Yices 2.2. In *Computer-Aided Verification (CAV'2014) (Lecture Notes in Computer Science)*, Armin Biere and Roderick Bloem (Eds.), Vol. 8559. Springer, 737–744.

[32] Bruno Dutertre. 2015. Solving Exists/Forall Problems With Yices. In *13th International Workshop on Satisfiability Modulo Theories (SMT 2015)*.

[33] Bruno Dutertre and Leonardo Mendonça de Moura. 2006. A Fast Linear-Arithmetic Solver for DPLL(T). In *CAV (Lecture Notes in Computer Science)*, Thomas Ball and Robert B. Jones (Eds.), Vol. 4144. Springer, 81–94.

[34] Martin Fränzle, Christian Herde, Tino Teige, Stefan Ratschan, and Tobias Schubert. 2007. Efficient Solving of Large Non-linear Arithmetic Constraint Systems with Complex Boolean Structure. *JSAT* 1, 3-4 (2007), 209–236.

[35] Zhaohui Fu and Sharad Malik. 2006. On Solving the Partial MAX-SAT Problem. In *SAT*. 252–265.

[36] Carsten Fuhs, Jürgen Giesl, Aart Middeldorp, Peter Schneider-Kamp, René Thiemann, and Harald Zankl. 2007. SAT Solving for Termination Analysis with Polynomial Interpretations. In *SAT (Lecture Notes in Computer Science)*, João Marques-Silva and Karem A. Sakallah (Eds.), Vol. 4501. Springer, 340–354.

[37] Malay K. Ganai and Franjo Ivancic. 2009. Efficient decision procedure for non-linear arithmetic constraints using CORDIC. In *FMCAD*. IEEE, 61–68.

[38] Sicun Gao, Jeremy Avigad, and Edmund M. Clarke. 2012. δ-Complete Decision Procedures for Satisfiability over the Reals. In *IJCAR (Lecture Notes in Computer Science)*, Bernhard Gramlich, Dale Miller, and Uli Sattler (Eds.), Vol. 7364. Springer, 286–300.

[39] Sicun Gao, Malay K. Ganai, Franjo Ivancic, Aarti Gupta, Sriram Sankaranarayanan, and Edmund M. Clarke. 2010. Integrating ICP and LRA solvers for deciding nonlinear real arithmetic problems, See [13], 81–89.

[40] Sicun Gao, Soonho Kong, and Edmund M. Clarke. 2013. dReal: An SMT Solver for Nonlinear Theories over the Reals, See [14], 208–214. https://doi.org/10.1007/978-3-642-38574-2_14

[41] Adria Gascón, Pramod Subramanyan, Bruno Dutertre, Ashish Tiwari, Dejan Jovanovic, and Sharad Malik. 2014. Template-based circuit understanding. In *Formal Methods in Computer-Aided Design, FMCAD 2014, Lausanne, Switzerland, October 21-24, 2014*. IEEE, 83–90. https://doi.org/10.1109/FMCAD.2014.6987599

[42] Alberto Griggio. 2012. A Practical Approach to Satisability Modulo Linear Integer Arithmetic. *JSAT* 8, 1/2 (2012), 1–27. http://jsat.ewi.tudelft.nl/content/volume8/JSAT8_1_Griggio.pdf

[43] Dejan Jovanovic and Leonardo Mendonça de Moura. 2012. Solving Non-linear Arithmetic. In *IJCAR (Lecture Notes in Computer Science)*, Bernhard Gramlich, Dale Miller, and Uli Sattler (Eds.), Vol. 7364. Springer, 339–354.

[44] Sebastian Junges, Ulrich Loup, Florian Corzilius, and Erika Ábrahám. 2013. On Gröbner Bases in the Context of Satisfiability-Modulo-Theories Solving over the Real Numbers. In *Algebraic Informatics - 5th International Conference, CAI 2013, Porquerolles, France, September 3-6, 2013. Proceedings (Lecture Notes in Computer Science)*, Traian Muntean, Dimitrios Poulakis, and Robert Rolland (Eds.), Vol. 8080. Springer, 186–198. https://doi.org/10.1007/978-3-642-40663-8_18

[45] Guy Katz, Clark W. Barrett, Cesare Tinelli, Andrew Reynolds, and Liana Hadarean. 2016. Lazy proofs for DPLL(T)-based SMT solvers. In *2016 Formal Methods in Computer-Aided Design, FMCAD 2016, Mountain View, CA, USA, October 3-6, 2016*, Ruzica Piskac and Muralidhar Talupur (Eds.). IEEE, 93–100. https://doi.org/10.1109/FMCAD.2016.7886666

[46] To Van Khanh and Mizuhito Ogawa. 2012. SMT for Polynomial Constraints on Real Numbers. *Electr. Notes Theor. Comput. Sci.* 289 (2012), 27–40.

[47] Sebastian Krings, Jens Bendisposto, and Michael Leuschel. 2015. From Failure to Proof: The ProB Disprover for B and Event-B. In *Software Engineering and Formal Methods - 13th International Conference, SEFM 2015, York, UK, September 7-11, 2015. Proceedings (Lecture Notes in Computer Science)*, Radu Calinescu and Bernhard Rumpe (Eds.), Vol. 9276. Springer, 199–214. https://doi.org/10.1007/978-3-319-22969-0_15

[48] Daniel Larraz, Kaustubh Nimkar, Albert Oliveras, Enric Rodríguez-Carbonell, and Albert Rubio. 2014. Proving Non-termination Using Max-SMT. In *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings (Lecture Notes in Computer Science)*, Armin Biere and Roderick Bloem (Eds.), Vol. 8559. Springer, 779–796. https://doi.org/10.1007/978-3-319-08867-9_52

[49] Daniel Larraz, Albert Oliveras, Enric Rodríguez-Carbonell, and Albert Rubio. 2013. Proving termination of imperative programs using Max-SMT. In *FMCAD*. IEEE, 218–225.

[50] Daniel Larraz, Albert Oliveras, Enric Rodríguez-Carbonell, and Albert Rubio. 2014. Minimal-Model-Guided Approaches to Solving Polynomial Constraints and Extensions. In *Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings (Lecture Notes in Computer Science)*, Carsten Sinz and Uwe Egly (Eds.), Vol. 8561. Springer, 333–350. https://doi.org/10.1007/978-3-319-09284-3_25

[51] Daniel Larraz, Enric Rodríguez-Carbonell, and Albert Rubio. 2013. SMT-Based Array Invariant Generation. In *VMCAI (Lecture Notes in Computer Science)*, Roberto Giacobazzi, Josh Berdine, and Isabella Mastroeni (Eds.), Vol. 7737. Springer, 169–188.

[52] Javier Larrosa, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. 2011. A Framework for Certified Boolean Branch-and-Bound Optimization. *J. Autom. Reasoning* 46, 1 (2011), 81–102. https://doi.org/10.1007/s10817-010-9176-z

[53] Chu Min Li and Felip Manyà. 2009. MaxSAT, Hard and Soft Constraints. In *Handbook of Satisfiability*, Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh (Eds.). Frontiers in Artificial Intelligence and Applications, Vol. 185. IOS Press, 613–631. https://doi.org/10.3233/978-1-58603-929-5-613

[54] Yi Li, Aws Albarghouthi, Zachary Kincaid, Arie Gurfinkel, and Marsha Chechik. 2014. Symbolic optimization with SMT solvers. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, Suresh Jagannathan and Peter Sewell (Eds.). ACM, 607–618. https://doi.org/10.1145/2535838.2535857

[55] Rüdiger Loos and Volker Weispfenning. 1993. Applying Linear Quantifier Elimination. *Comput. J.* 36, 5 (1993), 450–462. https://doi.org/10.1093/comjnl/36.5.450

[56] Alexandre Maréchal, Alexis Fouilhé, Tim King, David Monniaux, and Michaël Périn. 2016. Polyhedral Approximation of Multivariate Polynomials Using Handelman's Theorem. In *Verification, Model Checking, and Abstract Interpretation - 17th International Conference, VMCAI 2016, St. Petersburg, FL, USA, January 17-19, 2016. Proceedings*

(Lecture Notes in Computer Science), Barbara Jobstmann and K. Rustan M. Leino (Eds.), Vol. 9583. Springer, 166–184. https://doi.org/10.1007/978-3-662-49122-5_8

[57] João Marques-Silva, Federico Heras, Mikolás Janota, Alessandro Previti, and Anton Belov. 2013. On Computing Minimal Correction Subsets. In *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, Francesca Rossi (Ed.). IJCAI/AAAI. http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6922

[58] António Morgado, Federico Heras, Mark H. Liffiton, Jordi Planes, and João Marques-Silva. 2013. Iterative and core-guided MaxSAT solving: A survey and assessment. *Constraints* 18, 4 (2013), 478–534. https://doi.org/10.1007/s10601-013-9146-2

[59] Robert Nieuwenhuis and Albert Oliveras. 2006. On SAT Modulo Theories and Optimization Problems. In *SAT (Lecture Notes in Computer Science)*, Armin Biere and Carla P. Gomes (Eds.), Vol. 4121. Springer, 156–169.

[60] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. 2006. Solving SAT and SAT Modulo Theories: from an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T). *J. ACM* 53, 6 (Nov. 2006), 937–977.

[61] Pierluigi Nuzzo, Alberto Puggelli, Sanjit A. Seshia, and Alberto L. Sangiovanni-Vincentelli. 2010. CalCS: SMT solving for non-linear convex constraints, See [13], 71–79.

[62] Roc Oliver. 2012. *Optimization Modulo Theories.* Master's thesis. Universitat Politècnica de Catalunya, Spain. (January 2012).

[63] Grant Olney Passmore, Leonardo Mendonça de Moura, and Paul B. Jackson. 2010. Gröbner Basis Construction Algorithms Based on Theorem Proving Saturation Loops. In *Decision Procedures in Software, Hardware and Bioware, 18.04. - 23.04.2010 (Dagstuhl Seminar Proceedings)*, Nikolaj Bjørner, Robert Nieuwenhuis, Helmut Veith, and Andrei Voronkov (Eds.), Vol. 10161. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany. http://drops.dagstuhl.de/opus/volltexte/2010/2734/

[64] André Platzer, Jan-David Quesel, and Philipp Rümmer. 2009. Real World Verification. In *CADE (Lecture Notes in Computer Science)*, Renate A. Schmidt (Ed.), Vol. 5663. Springer, 485–501.

[65] Sriram Sankaranarayanan, Henny Sipma, and Zohar Manna. 2004. Non-linear loop invariant generation using Gröbner bases. In *POPL*, Neil D. Jones and Xavier Leroy (Eds.). ACM, 318–329.

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

[67] Alexander Schrijver. 1998. *Theory of Linear and Integer Programming.* Wiley.

[68] Roberto Sebastiani and Silvia Tomasi. 2015. Optimization Modulo Theories with Linear Rational Costs. *ACM Trans. Comput. Log.* 16, 2 (2015), 12:1–12:43. https://doi.org/10.1145/2699915

[69] Roberto Sebastiani and Patrick Trentin. 2015. OptiMathSAT: A Tool for Optimization Modulo Theories. In *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I (Lecture Notes in Computer Science)*, Daniel Kroening and Corina S. Pasareanu (Eds.), Vol. 9206. Springer, 447–454. https://doi.org/10.1007/978-3-319-21690-4_27

[70] Takehide Soh and Katsumi Inoue. 2010. Identifying Necessary Reactions in Metabolic Pathways by Minimal Model Generation. In *ECAI 2010 - 19th European Conference on Artificial Intelligence, Lisbon, Portugal, August 16-20, 2010, Proceedings (Frontiers in Artificial Intelligence and Applications)*, Helder Coelho, Rudi Studer, and Michael Wooldridge (Eds.), Vol. 215. IOS Press, 277–282. http://www.booksonline.iospress.nl/Content/View.aspx?piid=17757

[71] Aaron Stump, Geoff Sutcliffe, and Cesare Tinelli. 2014. StarExec: a cross-community infrastructure for logic solving. In *IJCAR.*

[72] Alfred Tarski. 1951. A decision method for elementary algebra and geometry. *Bull. Amer. Math. Soc.* 59 (1951).

[73] Ashish Tiwari, Adria Gascón, and Bruno Dutertre. 2015. Program Synthesis Using Dual Interpretation. In *Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings (Lecture Notes in Computer Science)*, Amy P. Felty and Aart Middeldorp (Eds.), Vol. 9195. Springer, 482–497. https://doi.org/10.1007/978-3-319-21401-6_33

[74] Vu Xuan Tung, To Van Khanh, and Mizuhito Ogawa. 2016. raSAT: An SMT Solver for Polynomial Constraints. In *Automated Reasoning - 8th International Joint Conference, IJCAR 2016, Coimbra, Portugal, June 27 - July 2, 2016, Proceedings (Lecture Notes in Computer Science)*, Nicola Olivetti and Ashish Tiwari (Eds.), Vol. 9706. Springer, 228–237. https://doi.org/10.1007/978-3-319-40229-1_16

[75] Volker Weispfenning. 1988. The Complexity of Linear Problems in Fields. *J. Symb. Comput.* 5, 1/2 (1988), 3–27. https://doi.org/10.1016/S0747-7171(88)80003-8

[76] Volker Weispfenning. 1997. Quantifier Elimination for Real Algebra - the Quadratic Case and Beyond. *Appl. Algebra Eng. Commun. Comput.* 8, 2 (1997), 85–101. https://doi.org/10.1007/s002000050055

[77] Harald Zankl and Aart Middeldorp. 2010. Satisfiability of Non-linear (Ir)rational Arithmetic. In *LPAR (Dakar) (Lecture Notes in Computer Science)*, Edmund M. Clarke and Andrei Voronkov (Eds.), Vol. 6355. Springer, 481–500.

[78] Lintao Zhang and Sharad Malik. 2003. Validating SAT Solvers Using an Independent Resolution-Based Checker: Practical Implementations and Other Applications. *2008 Design, Automation and Test in Europe* 1 (2003), 10880.