Minimal-Model-Guided Approaches to Solving Polynomial Constraints and Extensions*

Daniel Larraz, Albert Oliveras, Enric Rodríguez-Carbonell, and Albert Rubio

Universitat Politècnica de Catalunya, Barcelona, Spain

Abstract. In this paper we present new methods for deciding the satisfiability of formulas involving integer polynomial constraints. In previous work we proposed to solve SMT(NIA) problems by reducing them to SMT(LIA): non-linear monomials are linearized by abstracting them with fresh variables and by performing case splitting on integer variables with finite domain. When variables do not have finite domains, artificial ones can be introduced by imposing a lower and an upper bound, and made iteratively larger until a solution is found (or the procedure times out). For the approach to be practical, unsatisfiable cores are used to guide which domains have to be relaxed (i.e., enlarged) from one iteration to the following one. However, it is not clear then how large they have to be made, which is critical.

Here we propose to guide the domain relaxation step by analyzing minimal models produced by the SMT(LIA) solver. Namely, we consider two different cost functions: the number of violated artificial domain bounds, and the distance with respect to the artificial domains. We compare these approaches with other techniques on benchmarks coming from constraint-based program analysis and show the potential of the method. Finally, we describe how one of these minimal-model-guided techniques can be smoothly adapted to deal with the extension Max-SMT of SMT(NIA) and then applied to program termination proving.

1 Introduction

Polynomial constraints are ubiquitous. They arise naturally in many contexts, ranging from the analysis, verification and synthesis of software and cyber-physical systems [1–5] to, e.g., game theory [6]. In all these cases, it is critical to have efficient automatic solvers that, given a formula involving polynomial constraints with integer or real variables, either return a solution or report that the formula is unsatisfiable.

However, solving this kind of formulas has been a challenging problem since the early beginnings of mathematics. A landmark result is due to Tarski [7], who constructively proved that the problem is decidable for the first-order theory of real closed fields, in particular if variables are reals. Still, the algorithm in the proof has no use in practice as it has non-elementary complexity. More feasible procedures for solving polynomial constraints on the reals are based on cylindrical algebraic decomposition (CAD) [8, 9]. However, their applicability is limited, as their complexity is still doubly exponential.

With the breakthrough of SAT and SMT solving [10, 11], numerous techniques and tools have been developed which exploit the efficiency and automaticity of this

* Partially supported by Spanish MEC/MICINN under grant TIN 2010-21062-C02-01.

technology. Many of these approaches for solving polynomial constraints on the reals are numerically-driven. E.g., in [12] interval constraint propagation is integrated with SMT(LRA) solving. In [13], non-linear formulas are pre-processed and then fed to an off-the-shelf SMT(LRA) solver. Other works for instance integrate interval-based arithmetic constraint solving in the SAT engine [14], combine interval arithmetic and testing [15], or focus on particular kinds of constraints like convex constraints [16]. In order to address the ever-present concern that numerical errors can result in incorrect answers in these methods, it has been proposed to relax constraints and consider δ-complete decision procedures [17, 18]. As opposed to numerically-driven approaches, recently symbolic CAD-based techniques have been successfully integrated in a model-constructing DPLL(T)-style procedure [19, 20], and several libraries and toolboxes have been made publicly available for the development of symbolically-driven solvers [21, 22].

On the other hand, when variables must take integer values, even the problem of solving a single polynomial equation is undecidable (Hilbert's 10th problem, [23]). In spite of this theoretical limitation, and similarly to the real case, several methods that take advantage of the advancements in SAT and SMT solving have been proposed for solving integer polynomial constraints. The common idea of these methods is to reduce instances of integer non-linear arithmetic into problems of a simpler language that can be directly handled by existing SAT/SMT tools, e.g., propositional logic [24], linear bit-vector arithmetic [25], or linear integer arithmetic [26]. All these approaches are satisfiability-oriented, which makes them more convenient in contexts in which finding solutions is more relevant than proving that none exists (e.g., in invariant generation [27]).

In this paper we build upon our previous method [26] for deciding the satisfiability of formulas involving integer polynomial constraints. In that work, non-linear 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, and made iteratively larger until a solution is found (or the procedure times out). For the approach to be useful in practice, unsatisfiable cores are employed to guide which bounds have to be relaxed (i.e., enlarged) from one iteration to the following one. However, one of the shortcomings of the approach is that unsatisfiable cores provide no hint on how large the new bounds have to be made. This is critical, since the size of the new formula (and hence the time required to determine its satisfiability) can increase significantly depending on the number of new cases that must be added.

The contributions of this paper are twofold:

1. We propose heuristics for guiding the domain relaxation step by means of the analysis of minimal models generated by the SMT(LIA) solver. More specifically, we consider two different cost functions: first, the number of violated artificial domain bounds, which leads to *Maximum Satisfiability Modulo Theories* (Max-SMT, [28, 29]) problems; and second, the distance with respect to the artificial domains, which boils down to *Optimization Modulo Theories* (OMT, [30, 31]) problems. The results of comparing these approaches with other techniques show the potential of the method.
2. We extend the first of these approaches to handle problems in Max-SMT(NIA).

This paper is structured as follows. Section 2 reviews basic background on SMT, Max-SMT and OMT, and also on our previous approach in [26]. In Section 3 two different heuristics for guiding the domain relaxation step are proposed, together with an experimental evaluation. Then Section 4 presents the extension of the technique from SMT(NIA) to Max-SMT(NIA). Finally, Section 5 summarizes the conclusions of this work and sketches lines for future research.

2 Preliminaries

2.1 SMT, Max-SMT and OMT

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 (CNF) *propositional formula* is a conjunction of clauses $C_1 \wedge \cdots \wedge C_n$. The problem of *propositional satisfiability* (abbreviated SAT) consists in, given a propositional formula, to determine whether it is *satisfiable*, i.e., if it has a *model*: an assignment of Boolean values to variables that satisfies the formula.

A generalization of SAT is the *satisfiability modulo theories (SMT)* problem: to decide the satisfiability of a given 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. Here we will focus on integer variables and the theories of *linear integer arithmetic (LIA)*, where literals are linear inequalities, and the more general theory of *non-linear integer arithmetic (NIA)*, where literals are polynomial inequalities.[1]

Another generalization of SAT is *Max-SAT*, 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 $\mathcal{F}$, to find an assignment such that the number of satisfied clauses in $\mathcal{F}$ 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 C_i of $\mathcal{F}$ has a *weight* ω_i (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 $\mathcal{F}$ 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 a model that minimizes the sum of the weights of the falsified clauses in the background theory.

[1] In some classes of formulas of practical interest, real variables can also be handled by our methods. See Section 2.2 for details.

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 $\mathcal{F}$ involving a particular variable called *cost*, to find the model of $\mathcal{F}$ 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 functions, etc.).

2.2 Solving SMT(NIA) with Unsatisfiable Cores

In [26], we proposed a method for solving SMT(NIA) problems based on encoding them into SMT(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 SMT(NIA) formula and x must satisfy $0 \leq x \leq 2$, 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*: $x = 0 \rightarrow v_{x^2yz} = 0$, $x = 1 \rightarrow v_{x^2yz} = yz$ and $x = 2 \rightarrow v_{x^2yz} = 4yz$. 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 SMT(LIA) is obtained, for which efficient decision procedures exist [32].

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* bounds can be introduced for the variables that require them. In principle, this implies that the procedure is no longer complete, since a linearized formula with artificial bounds may be unsatisfiable while the original SMT(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 are to be relaxed 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 (for an account of techniques for computing unsatisfiable cores, see [33]). Note that the method tells *which* bounds should be enlarged, but does not provide any guidance in regard to *how large* the new bounds should be. This is critical, as the size of the formula in the next iteration (and so the time needed to determine its satisfiability) can grow significantly depending on the number of new case splitting clauses that have to be added.

Altogether, the overall algorithm in [26] for solving a given formula in SMT(NIA) is as follows (see Figure 1). First, the needed artificial bounds are added (procedure *initial_bounds*) and the linearized formula (procedure *linearize*) is passed to an SMT(LIA) solver (procedure *solve_LIA*). If the solver returns SAT, we are done. If the solver returns UNSAT, then an unsatisfiable core is computed. If this core does not contain any of the artificial bounds, then the original non-linear formula must be unsatisfiable, and again we are done. Otherwise, at least one of the artificial bounds appearing in the core must be chosen for relaxation (procedure *relax_domains*). Once the domains are enlarged and the formula is updated (procedure *update*), the new linearized formula is tested for satisfiability, and the process is repeated (typically, while a prefixed time limit is not exceeded).

```
status solve_NIA(Formula F0) {
  b = initial_bounds (F0); // enough artificial bounds to linearize F0
  F = linearize(F0, b);
  while (not timed_out ()) {
    ⟨st, core⟩ = solve_LIA(F); // core computed here to ease presentation
    if (st == SAT) return SAT;
    else if (b ∩ core == ∅) return UNSAT;
    else {
      b = relax_domains(b, core); // at least one in the intersection is relaxed
      F = update(F, b); // add new bounds and case splitting clauses
  } }
  return UNKNOWN;
}
```

Fig. 1. Algorithm in [26] based on unsatisfiable cores

Finally, notice that the assumption that all variables should have integer type can be weakened, since it suffices that there are *enough* finite domain variables to perform the linearization. For example, this can be exploited in our SMT problems coming from constraint-based program analysis [27, 34]. Those formulas are produced by applying Farkas' Lemma [35], and therefore only quadratic monomials of the form $\lambda \cdot u$ appear. Although in principle both λ and u are real unknowns, in the context of invariant and ranking function generation it is reasonable to assume that u should be integer. Hence, by case splitting on u one can linearize the monomial and does not need to force λ to take integer values. Moreover, when analyzing programs with integer variables, one often needs to be able to reason taking into account the integrality of the variables. In this situation integer versions of Farkas' Lemma [36] can be used, which when applied in the context of, e.g., invariant generation, require again the unknowns u to be in $\mathbb{Z}$.

3 Domain Relaxation with Minimal Models

Taking into account the limitations of the method based on cores when domains have to be enlarged, in this section we propose a model-guided approach to perform this step. The idea is to replace the satisfiability check in linear arithmetic with an optimization call, so that the best model found by the linear solver can be used as a reference for relaxing bounds (e.g., by extending the domains up to the value in that best model for those bounds that have to be relaxed).

Thus, the high-level algorithm we propose for solving a given formula in SMT(NIA) is shown in Figure 2 (cf. Figure 1). Here the SMT(LIA) black box does not just decide satisfiability, but finds the minimum model of the formula according to a prefixed non-negative cost function (procedure *optimize_LIA*). This function must have the property that the models of the linearized formula with cost 0 are true models of the original non-linear formula, and that if the linearization is unsatisfiable then so is the original formula. In addition to procedure *optimize_LIA*, the concrete implementations of procedures *linearize*, *relax_domains* and *update* also depend on the cost function.

```
status solve_NIA(Formula F_0) {
  b = initial_bounds (F_0); // enough artificial bounds to linearize F_0
  F = linearize(F_0, b);
  while (not timed_out ()) {
    ⟨st, model⟩ = optimize_LIA(F);
    if (st == UNSAT) return UNSAT;
    else if (cost(model) == 0) return SAT;
    else {
      b = relax_domains(b, model);
      F = update(F, b); // add new bounds and case splitting clauses
  } }
  return UNKNOWN;
}
```

Fig. 2. Algorithm for solving SMT(NIA) based on minimal models

Below we suggest two such cost functions: the number of violated artificial bounds (Section 3.1), and the distance with respect to the artificial domains (Section 3.2).

3.1 A Max-SMT(LIA) Approach

A possibility is to define the cost of an assignment as the number of violated artificial domain 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(LIA) tool. In this setting, *linearize* works as in the core-based algorithm, with the following difference: the clauses of the original formula (after being linearized by replacing non-linear monomials with fresh variables) together with the case splitting clauses are considered to be hard, while the artificial bounds are soft (with weight 1). Following the same construction, procedure *update* updates the soft clauses with the relaxed bounds, and adds the new case splitting clauses as hard clauses.

As regards the optimization step, procedure *optimize_LIA* boils down to making a call to a Max-SMT(LIA) solver on the linearized formula. In this case, the status *st* in Figure 2 corresponds to the satisfiability of the hard clauses. It is clear that if this status is UNSAT, then the original non-linear clause set is also unsatisfiable, given that the models of the original formula are a subset of the models of the hard clauses of the linearized formula. Another important property is that, if a model of the linearization has cost 0, then it is a true model of the non-linear formula.

Finally, procedure *relax_domains* determines the bounds to be relaxed by inspecting the soft clauses that are falsified. Moreover, as outlined above, the bounds are enlarged as follows. Let us assume that $x \leq u$ is an artificial bound that is falsified in the minimal model. If x is assigned value U in that model (and, hence, $u < U$), then $x \leq U$ becomes the new upper bound of x. A similar construction applies for lower bounds.

Regarding the weights of the soft clauses, in general 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_NIA* to the next one.

Example 1. Let us consider the formula $tx + wy \geq 4 \ \wedge \ t^2 + x^2 + w^2 + y^2 \leq 12$, where variables t, x, w, y are integer. Let us also assume that we add the following artificial bounds in order to linearize: $-1 \leq t, x, w, y \leq 1$. Then we obtain the following linearized weighted formula:

$$
\left.
\begin{array}{l}
v_{tx} + v_{wy} \geq 4 \ \wedge \ v_{t^2} + v_{x^2} + v_{w^2} + v_{y^2} \leq 12 \ \wedge \\
t = -1 \ \rightarrow v_{tx} = -x \quad \wedge \quad w = -1 \rightarrow v_{wy} = -y \quad \wedge \\
t = 0 \ \ \rightarrow v_{tx} = 0 \quad \wedge \quad w = 0 \ \ \rightarrow v_{wy} = 0 \quad \wedge \\
t = 1 \ \ \rightarrow v_{tx} = x \quad \wedge \quad w = 1 \ \ \rightarrow v_{wy} = y \quad \wedge \\
\\
t = -1 \ \rightarrow v_{t^2} = 1 \quad \wedge \quad w = -1 \rightarrow v_{w^2} = 1 \quad \wedge \\
t = 0 \ \ \rightarrow v_{t^2} = 0 \quad \wedge \quad w = 0 \ \ \rightarrow v_{w^2} = 0 \quad \wedge \\
t = 1 \ \ \rightarrow v_{t^2} = 1 \quad \wedge \quad w = 1 \ \ \rightarrow v_{w^2} = 1 \quad \wedge \\
\\
x = -1 \rightarrow v_{x^2} = 1 \quad \wedge \quad y = -1 \rightarrow v_{y^2} = 1 \quad \wedge \\
x = 0 \ \ \rightarrow v_{x^2} = 0 \quad \wedge \quad y = 0 \ \ \rightarrow v_{y^2} = 0 \quad \wedge \\
x = 1 \ \ \rightarrow v_{x^2} = 1 \quad \wedge \quad y = 1 \ \ \rightarrow v_{y^2} = 1 \quad \wedge
\end{array}
\right\} (\star)
$$

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

where $v_{tx}, v_{wy}, v_{t^2}, v_{x^2}, v_{w^2}, v_{y^2}$ are integer fresh variables standing for non-linear monomials. Soft clauses are written $[C, \omega]$, while clauses without weight are hard clauses.

In this case minimal solutions have cost 1, since at least one of the artificial bounds has to be violated so as to satisfy $v_{tx} + v_{wy} \geq 4$. For instance, the Max-SMT(LIA) solver could return the assignment: $t = 1$, $x = 4$, $v_{tx} = 4$, $w = y = v_{wy} = v_{w^2} = v_{y^2} = 0$, $v_{t^2} = 1$ and $v_{x^2} = 0$, 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 be $x \leq 4$ (so the soft clause $[x \leq 1, 1]$ would be replaced by $[x \leq 4, 1]$), and the following hard clauses would be added: $x = 2 \rightarrow v_{x^2} = 4$, $x = 3 \rightarrow v_{x^2} = 9$ and $x = 4 \rightarrow v_{x^2} = 16$. In the next iteration there are solutions with cost 0, e.g., $t = 1$, $x = 3$, $v_{tx} = 3$, $w = y = v_{wy} = v_{w^2} = v_{y^2} = 1$, $v_{t^2} = 1$ and $v_{x^2} = 9$. ■

One of the disadvantages of this approach is that potentially the Max-SAT(LIA) solver could return models with numerical values much larger than necessary. Since the model is used for extending the domains, it could be the case that a prohibitive number of case splitting clauses are added, and at the next iteration the Max-SAT(LIA) solver is not able to handle the formula with a reasonable amount of resources. For instance, in Example 1, it could have been the case that the Max-SAT(LIA) solver returned $u = y = 0$, $t = 1$, $x = 10^5$, $v_{x^2} = 0$, etc. However, 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(LIA) Approach

Another possibility is to define the cost of an assignment as the distance with respect to the artificial domains. This can be cast as a problem in OMT(LIA) as follows.

First of all, given a non-linear formula $\mathcal{F}_0$, the linearization $\mathcal{F}$ (procedure *linearize*) is computed like in the algorithm based on cores, except for the fact that artificial bounds are not included in the linearization: $\mathcal{F}$ consists only of the clauses of $\mathcal{F}_0$ (after being linearized), and of the case splitting clauses (together with other constraints to express the cost function, to be described below).

Now, let S be the set of variables x for which an artificial domain $[\lambda_x, \nu_x]$ is added in the linearization. Formally, the cost function is $\sum_{x \in S} \delta(x, [\lambda_x, \nu_x])$, where $\delta(z, [\lambda, \nu])$ is the *distance* of z with respect to $[\lambda, \nu]$:

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

Note that, in the definition of the cost function, one could also include true original bounds: the contribution to the cost of these is null, since they are part of the formula and therefore must be respected.

In procedure *optimize_LIA*, the OMT(LIA) solver minimizes this function, expressed in the following way. Let *cost* be the variable that the solver minimizes. For each variable $x \in S$ with domain $[\lambda_x, \nu_x]$, let us introduce once and for all two extra integer variables l_x and u_x (meaning the distance with respect to the lower and to the upper bound of the domain of x, respectively) and the *auxiliary constraints* $l_x \geq 0$, $l_x \geq \lambda_x - x$, $u_x \geq 0$, $u_x \geq x - \nu_x$. Then the cost function is determined by the equation $cost = \sum_{x \in S} (l_x + u_x)$, which is added to the linearization together with the auxiliary constraints listed above.

Note that a model of the linearization that has cost 0 must assign values within the bounds for all variables. Therefore the variables standing for non-linear monomials must be assigned consistent values with their semantics, by virtue of the case splitting clauses. Thus, models of the linearization with null cost are models of the original non-linear formula. Moreover, if the linearized formula is unsatisfiable, then the original formula must be unsatisfiable too, since the models of the original formula are included in the models of the linearized formula.

As regards domain relaxation, procedure *relax_domains* determines the bounds to be enlarged by identifying the variables l_x, u_x that are assigned a non-null value. Further, again the bounds are enlarged by taking the optimal model as a reference: similarly as in Section 3.1, if $x \leq u$ is an artificial bound to be relaxed and x is assigned value U in the best model, then $x \leq U$ becomes the new upper bound. Then procedure *update* updates the auxiliary constraints (e.g., $u_x \geq x - u$ is replaced by $u_x \geq x - U$), and adds the new case splitting clauses (for the $U - u$ cases $x = u + 1$, ..., $x = U$, etc.). Note that precisely the value of u_x in the optimal model is $U - u > 0$. Hence, intuitively the cost function corresponds to the *number of new cases* that will have to be taken into account for the next iteration of the loop of *solve_NIA*.

It is also possible to consider a slightly different cost function, which corresponds to the *number of new clauses* that will have to be added for the next iteration. For that purpose, it is only necessary to multiply variables l_x, u_x in the equation that defines

cost by the number of monomials whose value is determined by case splitting on x. In general, similarly to Section 3.1, one may have a generic cost function of the form $cost = \sum_{x \in S} (\alpha_x l_x + \beta_x u_x)$, where $\alpha_x, \beta_x > 0$ for all $x \in S$. Further, again these coefficients may be changed from one iteration to the next one.

Example 2. Let us consider again the same non-linear formula from Example 1: $tx + wy \geq 4 \wedge t^2 + x^2 + w^2 + y^2 \leq 12$, where variables t, x, w, y are integer. Let us also assume that we add the following artificial bounds in order to linearize: $-1 \leq t, x, w, y \leq 1$. Then we obtain the following OMT(LIA) problem:

$$\min \ cost \quad \text{subject to}$$

$$\begin{array}{c} \text{constraints } (\star) \text{ from Example 1} \ \wedge \\ cost = l_t + u_t + l_x + u_x + l_w + u_w + l_y + u_y \ \wedge \\ \begin{array}{cccccccc} l_t \geq 0 & \wedge & l_t \geq -1 - t & \wedge & u_t \geq 0 & \wedge & u_t \geq t - 1 & \wedge \\ l_x \geq 0 & \wedge & l_x \geq -1 - x & \wedge & u_x \geq 0 & \wedge & u_x \geq x - 1 & \wedge \\ l_w \geq 0 & \wedge & l_w \geq -1 - w & \wedge & u_w \geq 0 & \wedge & u_w \geq w - 1 & \wedge \\ l_y \geq 0 & \wedge & l_y \geq -1 - y & \wedge & u_y \geq 0 & \wedge & u_y \geq y - 1 & \end{array} \end{array}$$

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

At the next iteration there is still no solution with cost 0, and 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 Experiments

In this section we evaluate experimentally the performance of the two minimal-model-guided approaches proposed above, and compare them with other competing non-linear solvers. Namely, we consider the following tools[2] :

- bcl-maxsmt, our Max-SMT-based algorithm from Section 3.1;
- bcl-omt, our OMT-based algorithm from Section 3.2;
- bcl-cores, our core-based algorithm [26];
- Z3 version 4.3.1 [37].

[2] We also experimented with other tools, namely dReal [18], SMT-RAT [21] and MiniSMT [25]. It turned out that the kind of instances we are considering here are not well-suited for these solvers, and many timeouts were obtained.

The experiments were carried out on an Intel Core i7 with 3.40GHz clock speed and 16 GB of RAM. We set a timeout of 60 seconds.

All bcl-* solvers[3] share essentially the same underlying SAT engine and LIA theory solver. Moreover, some strategies are also common:

- procedure *initial_bounds* uses a greedy algorithm to approximate the minimal set of variables that have to be introduced in the linearization [26]. For each of them, 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 in some cases 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 has been chosen to be enlarged is handled specially. Let us assume it is the first time that a lower bound (respectively, an upper bound) of x has to be enlarged. By virtue of the remark above, the bound must be of the form $x \geq -1$ (respectively, $x \leq 1$). Now, if x has a true bound of the form $x \geq l$ (respectively, $x \leq u$), then the new bound is the true bound. Otherwise, if x 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 a bound has to be enlarged onwards, domain relaxation of bcl-maxsmt and bcl-omt follows basically what is described in Section 3, except for a correction factor aimed at instances where solutions have some large values. Namely, if $x \leq u$ has to be enlarged and in the minimal model x is assigned value U, then the new upper bound is $U + \alpha \cdot \min(\beta, \frac{n}{m})$, where α and β are parameters, n is the number of times the upper bound of x has been relaxed, and m is the number of occurrences of x in the original formula. 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.

In this evaluation we considered two different sets of benchmarks. The first benchmark suite consists of 1934 instances generated by our constraint-based termination prover [34]. As pointed out in Section 2.2, in these problems non-linear monomials are quadratic. Moreover, since it makes sense in our application, for each benchmark we have run Z3 (which cannot solve any of our non-linear integer instances) on versions of the instances where all variables are reals. This has been done in order to perform a fairer comparison, since unlike our approaches, Z3 is targeted to the real case. Results can be seen in Table 1, where columns represent systems and rows possible outcomes (SAT, UNSAT, UNKNOWN and TIMEOUT). Each cell contains the number of problems with that outcome obtained with the corresponding system, or the total time to process them.

The second benchmark suite consists of 36 examples of SMT(NIA) generated by the QArmc-Hsf(c) tool [38], a predicate-abstraction-based model checker with a special focus on liveness properties. In these problems all variables are integer, and monomi-

[3] Available at `www.lsi.upc.edu/~albert/sat14.tgz`.

Table 1. Experiments with benchmarks from Termination prover

	z3		bcl-cores		bcl-maxsmt		bcl-omt	
	#prob	secs	#prob	secs	#prob	secs	#prob	secs
SAT	1136	2578	1838	5464	1852	3198	1798	7896
UNSAT	0	0	0	0	4	0	62	112
UNKNOWN	11	2	0	0	0	0	0	0
TIMEOUT	787	47220	96	5760	78	4680	74	4440

Table 2. Experiments with benchmarks from model checking

	z3		bcl-cores		bcl-maxsmt		bcl-omt	
	#prob	secs	#prob	secs	#prob	secs	#prob	secs
SAT	30	2	35	55	35	72	34	263
UNSAT	1	0	1	0	1	0	1	0
UNKNOWN	0	0	0	0	0	0	0	0
TIMEOUT	5	300	0	0	0	0	1	60

als beyond quadratic appear. Results are in Table 2 and follow the same format as in Table 1.

As we can see in the tables, bcl-cores and bcl-maxsmt are the most efficient systems on satisfiable instances. While bcl-omt is doing slightly worse, Z3 is clearly outperformed, even when variables have real type. After inspecting the traces, we have seen that bcl-omt enlarges the domains too slowly, which is hindering the search.

Regarding unsatisfiable instances, it can also be observed that bcl-cores performs worse than the model-guided approaches, and that in particular bcl-omt is surprisingly effective. The reason is that, while the latter will always identify when the linear abstraction of the formula is unsatisfiable, this may not be the case with the former, which depending on the computed core may detect or not the unsatisfiability. In particular, bcl-cores does not guarantee that cores are minimal with respect to subset inclusion, and attempts to eliminate irrelevant clauses would imply an overhead that in most cases would not pay off.

Finally, as a side note, it is worth mentioning that we also experimented with a mixed version of the Max-SMT and OMT approaches. This version works as follows. Once the Max-SMT(LIA) finds a propositional model of the (propositional skeleton of the) linearization that minimizes the number of violations of the artificial bounds (this is the Max-SMT part), instead of taking any of the solutions that satisfy this propositional model, one finds a solution among those that minimizes the distance with respect to the artificial domains (this is the OMT part). This hybridization did not perform significantly better than the Max-SMT approach, because most often the solution computed by default by the Max-SMT(LIA) solver turns out to be already optimal with respect to the distance cost function, and in general the gain obtained with this final optimization does not compensate the overhead it incurs in the total execution time.

4 Extension to Max-SMT(NIA)

As we showed in previous work [34], the framework of Max-SMT(NIA) is particularly appropriate for constraint-based termination proving. Other applications of Max-SMT(NIA) in program analysis can be envisioned given the enormous expressive power of its language. For the feasibility of this kind of applications, it is of paramount importance that efficient solvers are available. For this reason, this section will be devoted to the extension of our techniques for SMT(NIA) to Max-SMT(NIA).

More specifically, the experiments in Section 3.3 indicate that, when applied to satisfiable instances of SMT(NIA), the Max-SMT(LIA) approach behaves better than the OMT(LIA) one, and similarly to the core-based one, although on the instances coming from our program analysis applications it tends to perform better. Because of this, in Section 4.1 the Max-SMT(LIA) approach will be taken as a basis upon which a new algorithm for Max-SMT(NIA) will be proposed, which is more simple and natural than what a Max-SMT(NIA) system built on top of a core-based SMT(NIA) solver would be. Finally, in Section 4.2 we will report on the application of an implementation of this algorithm to program termination.

4.1 Algorithm

We will represent the input $\mathcal{F}_0$ of a Max-SMT(NIA) instance as a conjunction of a set of hard clauses $\mathcal{H}_0 = \{C_1, \cdots, C_n\}$ and a set of soft clauses $\mathcal{S}_0 = \{[D_1, \omega_1], \cdots, [D_m, \omega_m]\}$. The aim is to decide whether there exist assignments α such that $\alpha \models \mathcal{H}_0$, and if so, to find one such that $\sum_{[D,\omega] \in \mathcal{S}_0 \mid \alpha \not\models D} \omega$ is minimized.

```
<Status, Model> solve_Max_SMT_NIA(Formula F_0) {
  b = initial_bounds (F_0);
  F = linearize(F_0, b);
  best_so_far = ⊥;
  max_soft_cost = ∞
  while (not timed_out ()) {
    ⟨st, model⟩ = solve_Max_SMT_LIA(F, max_soft_cost);
    if (st == UNSAT)
      if (best_so_far == ⊥) return < UNSAT, ⊥ >;
      else                  return < SAT, best_so_far >;
    else if (cost_B(model) == 0) {
      best_so_far = model ;
      max_soft_cost = cost_S(model) − 1;
    }
    else {
      b = relax_domains(b, model);
      F = update(F, b);
  } }
  return < UNKNOWN, ⊥ >;
}
```

Fig. 3. Algorithm for solving Max-SMT(NIA) based on Max-SMT(LIA)

The algorithm for solving Max-SMT(NIA) is shown in Figure 3. In its first step, as usual the initial artificial bounds are chosen (procedure *initial_bounds*) and the input formula $\mathcal{F}_0 \equiv \mathcal{H}_0 \wedge \mathcal{S}_0$ is linearized (procedure *linearize*). As a result, a weighted linear formula $\mathcal{F}$ is obtained with hard clauses $\mathcal{H} \wedge \mathcal{C}$ and soft clauses $\mathcal{S} \wedge \mathcal{B}$, where:

- $\mathcal{H}$ and $\mathcal{S}$ are the result of replacing the non-linear monomials in $\mathcal{H}_0$ and $\mathcal{S}_0$ by their corresponding fresh variables, respectively;
- $\mathcal{C}$ are the case splitting clauses;
- $\mathcal{B}$ is the set of artificial bounds of the form $[x \geq l, \Omega]$, $[x \leq u, \Omega']$, where the weights Ω, Ω' are positive numbers that are introduced in the linearization.

Now notice that there are two kinds of weights: those from the original soft clauses, and those produced by 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 $cost(\alpha) = (cost_{\mathcal{B}}(\alpha), cost_{\mathcal{S}}(\alpha))$, where $cost_{\mathcal{B}}(\alpha) = \sum_{[B,\Omega]\in\mathcal{B} \mid \alpha\not\models B} \Omega$ is the *bound cost*, i.e., the contribution to the total cost due to artificial bounds, and $cost_{\mathcal{S}}(\alpha) = \sum_{[D,\omega]\in\mathcal{S} \mid \alpha\not\models D} \omega$ 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 $cost(\alpha) = \sum_{[C,(\Omega,\omega)]\in\mathcal{B}\cup\mathcal{S} \mid \alpha\not\models C} (\Omega, \omega)$, where the sum of the pairs is component-wise.

In what follows, pairs $(cost_{\mathcal{B}}(\alpha), cost_{\mathcal{S}}(\alpha))$ will be lexicographically compared, so that the bound cost (i.e., to be consistent in NIA) is more relevant than the soft cost. Hence, by taking this cost function and this ordering we have a Max-SMT(LIA) instance in which weights are not natural or non-negative real numbers, but pairs of them.

In the next step of the algorithm, procedure *solve_Max_SMT_LIA* calls a Max-SMT(LIA) tool to solve this instance. A difference with the usual setting is that the Max-SMT(LIA) solver admits a parameter *max_soft_cost* that restrains the models of the hard clauses we are considering: only assignments α such that $cost_{\mathcal{S}}(\alpha) \leq$ *max_soft_cost* are taken into account. Thus, this adapted Max-SMT(LIA) solver computes, among the models α of the hard clauses such that $cost_{\mathcal{S}}(\alpha) \leq$ *max_soft_cost* (if any), one that minimizes $cost(\alpha)$. This allows one to prune the search lying under the Max-SMT(LIA) solver when it is detected that the best soft cost found so far cannot be improved. This is not difficult to implement if the Max-SAT solver follows a branch-and-bound scheme, as it is our case.

Now the algorithm examines the result of the call to the Max-SMT(LIA) 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, *model* satisfies the hard clauses and has soft cost at most *max_soft_cost*. If it has null bound cost, i.e., it 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 relaxed as described in Section 3.1, in order to widen the search space. In any case, the algorithm jumps back to a new call to *solve_Max_SMT_LIA*.

4.2 Application

As far as we know, none of the competing non-linear solvers is providing native support for Max-SMT, and hence no fair comparison is possible. For this reason, in order to give empirical evidence of the usefulness of the algorithm described in Section 4.1, here we opt for giving a brief summary of the application of Max-SMT to program termination [34] and, most importantly, highlighting the impact of our Max-SMT solver on the efficacy of the termination prover built on top of it.

Termination proving requires the generation of ranking functions as well as supporting invariants. Previous work [39] formulated invariant and ranking function synthesis as constraint problems, thus yielding SMT instances. In [34], Max-SMT is proposed as a more convenient framework. The crucial observation is that, albeit the goal is to show that program transitions cannot be executed infinitely by finding a ranking function or an invariant that disables them, if we only discover an invariant, or an invariant and a *quasi-ranking function* that almost fulfills all needed properties for well-foundedness, we have made some progress: either we can remove part of a transition and/or we have improved our knowledge on the behavior of the program. A natural way to implement this idea is by considering that some of the constraints are hard (the ones guaranteeing invariance) and others are soft (those guaranteeing well-foundedness).

Thus, efficient Max-SMT solvers open the door to more refined analyses of termination, which in turn allows one to prove more programs terminating. In order to support this claim, we carried out the experiment reported in Table 3, where we considered two termination provers:

- The tool (SMT) implements the generation of invariants and ranking functions using a translation to SMT(NIA), where all constraints are hard.
- The tool (Max-SMT) is based on the same infrastructure, but expresses the synthesis of invariants and ranking functions as Max-SMT(NIA) problems. As outline above, this allows performing more refined analyses.

Table 3 presents the number of instances (#ins.) in each benchmark suite we considered (from [40]) and the number of those that respectively each system proved terminating (with a timeout of 300 seconds). As can be seen in the results, there is a non-

Table 3. Comparison of SMT-based and Max-SMT-based termination provers

	#ins.	SMT	Max-SMT
Set1	449	212	228
Set2	472	245	262

negligible improvement in the number of programs proved terminating thanks to the adoption of the Max-SMT approach and the efficiency of our Max-SMT(NIA) solver.

5 Conclusions and Future Work

In this paper we have proposed two strategies to guide domain relaxation in the instantiation-based approach for solving SMT(NIA) [26]. Both are based on computing minimal models with respect to a cost function, respectively, the number of violated artificial domain bounds, and the distance with respect to the artificial domains. The results of comparing them with other techniques show their potential. Moreover, we have developed an algorithm for Max-SMT(NIA) building upon the first of these approaches, and have shown its impact on the application of Max-SMT(NIA) to program termination.

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 minimal-model-guided framework proposed here. On the other hand, one of the shortcomings of our instantiation-based approach for solving Max-SMT/SMT(NIA) is that unsatisfiable instances that require non-trivial non-linear reasoning cannot be captured. In this context, the integration of real-goaled CAD techniques adapted to SMT [19] appears to be a promising line of work.

Another direction for future research concerns applications. So far we have applied Max-SMT(NIA) to array invariant generation [27] and termination proving [34]. Other problems in program analysis where we envision the Max-SMT(NIA) framework could help in improving the state-of-the-art are, e.g., the analysis of worst-case execution time and the analysis of non-termination. 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. We thank C. Popeea and A. Rybalchenko for their benchmarks.

References

1. Colón, M., Sankaranarayanan, S., Sipma, H.: Linear Invariant Generation Using Non-linear Constraint Solving. In Jr., W.A.H., Somenzi, F., eds.: CAV. Volume 2725 of Lecture Notes in Computer Science., Springer (2003) 420–432
2. Sankaranarayanan, S., Sipma, H., Manna, Z.: Non-linear loop invariant generation using Gröbner bases. In Jones, N.D., Leroy, X., eds.: POPL, ACM (2004) 318–329
3. Sankaranarayanan, S., Sipma, H.B., Manna, Z.: Constructing invariants for hybrid systems. Formal Methods in System Design **32**(1) (2008) 25–55
4. Platzer, A., Quesel, J.D., Rümmer, P.: Real world verification. In Schmidt, R.A., ed.: CADE. Volume 5663 of Lecture Notes in Computer Science., Springer (2009) 485–501
5. Cheng, C.H., Shankar, N., Ruess, H., Bensalem, S.: EFSMT: A Logical Framework for Cyber-Physical Systems (2013) CoRR abs/1306.3456.
6. Beyene, T., Chaudhuri, S., Popeea, C., Rybalchenko, A.: 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, New York, NY, USA, ACM (2014) 221–233
7. Tarski, A.: A decision method for elementary algebra and geometry. Bulletin of the American Mathematical Society **59** (1951)

8. Collins, G.E.: Hauptvortrag: Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In Barkhage, H., ed.: Automata Theory and Formal Languages. Volume 33 of Lecture Notes in Computer Science., Springer (1975) 134–183
9. Basu, S., Pollack, R., Roy., M.F.: Algorithms in Real Algebraic Geometry. Springer-Verlag, Berlin (2003)
10. Biere, A., Heule, M.J.H., van Maaren, H., Walsh, T., eds.: Handbook of Satisfiability. Volume 185 of Frontiers in Artificial Intelligence and Applications. IOS Press (February 2009)
11. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT Modulo Theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). J. ACM **53**(6) (2006) 937–977
12. Gao, S., Ganai, M.K., Ivancic, F., Gupta, A., Sankaranarayanan, S., Clarke, E.M.: Integrating ICP and LRA solvers for deciding nonlinear real arithmetic problems. [41] 81–89
13. Ganai, M.K., Ivancic, F.: Efficient decision procedure for non-linear arithmetic constraints using CORDIC. In: FMCAD, IEEE (2009) 61–68
14. Fränzle, M., Herde, C., Teige, T., Ratschan, S., Schubert, T.: Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. JSAT **1**(3-4) (2007) 209–236
15. Khanh, T.V., Ogawa, M.: SMT for Polynomial Constraints on Real Numbers. Electr. Notes Theor. Comput. Sci. **289** (2012) 27–40
16. Nuzzo, P., Puggelli, A., Seshia, S.A., Sangiovanni-Vincentelli, A.L.: CalCS: SMT solving for non-linear convex constraints. [41] 71–79
17. Gao, S., Avigad, J., Clarke, E.M.: δ-complete decision procedures for satisfiability over the reals. In Gramlich, B., Miller, D., Sattler, U., eds.: IJCAR. Volume 7364 of Lecture Notes in Computer Science., Springer (2012) 286–300
18. Gao, S., Kong, S., Clarke, E.M.: dReal: An SMT Solver for Nonlinear Theories over the Reals. [42] 208–214
19. Jovanovic, D., de Moura, L.M.: Solving non-linear arithmetic. In Gramlich, B., Miller, D., Sattler, U., eds.: IJCAR. Volume 7364 of Lecture Notes in Computer Science., Springer (2012) 339–354
20. de Moura, L.M., Jovanovic, D.: A model-constructing satisfiability calculus. In Giacobazzi, R., Berdine, J., Mastroeni, I., eds.: VMCAI. Volume 7737 of Lecture Notes in Computer Science., Springer (2013) 1–12
21. Corzilius, F., Loup, U., Junges, S., Ábrahám, E.: SMT-RAT: An SMT-Compliant Nonlinear Real Arithmetic Toolbox - (Tool Presentation). In Cimatti, A., Sebastiani, R., eds.: SAT. Volume 7317 of Lecture Notes in Computer Science., Springer (2012) 442–448
22. de Moura, L.M., Passmore, G.O.: Computation in real closed infinitesimal and transcendental extensions of the rationals. [42] 178–192
23. Cooper, S.B.: Computability Theory. Chapman Hall/CRC Mathematics Series (2004)
24. Fuhs, C., Giesl, J., Middeldorp, A., Schneider-Kamp, P., Thiemann, R., Zankl, H.: SAT Solving for Termination Analysis with Polynomial Interpretations. In Marques-Silva, J., Sakallah, K.A., eds.: SAT. Volume 4501 of Lecture Notes in Computer Science., Springer (2007) 340–354
25. Zankl, H., Middeldorp, A.: Satisfiability of non-linear (ir)rational arithmetic. In Clarke, E.M., Voronkov, A., eds.: LPAR (Dakar). Volume 6355 of Lecture Notes in Computer Science., Springer (2010) 481–500
26. Borralleras, C., Lucas, S., Oliveras, A., Rodríguez-Carbonell, E., Rubio, A.: SAT Modulo Linear Arithmetic for Solving Polynomial Constraints. J. Autom. Reasoning **48**(1) (2012) 107–131
27. Larraz, D., Rodríguez-Carbonell, E., Rubio, A.: SMT-Based Array Invariant Generation. In Giacobazzi, R., Berdine, J., Mastroeni, I., eds.: VMCAI. Volume 7737 of Lecture Notes in Computer Science., Springer (2013) 169–188

28. Nieuwenhuis, R., Oliveras, A.: On SAT Modulo Theories and Optimization Problems. In Biere, A., Gomes, C.P., eds.: SAT. Volume 4121 of Lecture Notes in Computer Science., Springer (2006) 156–169
29. Cimatti, A., Franzén, A., Griggio, A., Sebastiani, R., Stenico, C.: Satisfiability Modulo the Theory of Costs: Foundations and Applications. In Esparza, J., Majumdar, R., eds.: TACAS. Volume 6015 of Lecture Notes in Computer Science., Springer (2010) 99–113
30. Sebastiani, R., Tomasi, S.: Optimization in SMT with $\mathcal{LA}(\mathbb{Q})$ Cost Functions. In Gramlich, B., Miller, D., Sattler, U., eds.: IJCAR. Volume 7364 of Lecture Notes in Computer Science., Springer (2012) 484–498
31. Oliver, R.: Optimization Modulo Theories. Master's thesis, Universitat Politècnica de Catalunya, Spain (January 2012).
32. Dutertre, B., de Moura, L.M.: A Fast Linear-Arithmetic Solver for DPLL(T). In Ball, T., Jones, R.B., eds.: CAV. Volume 4144 of Lecture Notes in Computer Science., Springer (2006) 81–94
33. Asín Achá, R., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E.: Practical algorithms for unsatisfiability proof and core generation in SAT solvers. AI Communications **23**(2-3) (2010) 145–157
34. Larraz, D., Oliveras, A., Rodríguez-Carbonell, E., Rubio, A.: Proving termination of imperative programs using Max-SMT. In: FMCAD, IEEE (2013) 218–225
35. Schrijver, A.: Theory of Linear and Integer Programming. Wiley (June 1998)
36. Robinson, J.A., Voronkov, A., eds.: Handbook of Automated Reasoning (in 2 volumes). Elsevier and MIT Press (2001)
37. de Moura, L.M., Bjørner, N.: Z3: An efficient smt solver. In Ramakrishnan, C.R., Rehof, J., eds.: TACAS. Volume 4963 of Lecture Notes in Computer Science., Springer (2008) 337–340
38. Grebenshchikov, S., Gupta, A., Lopes, N.P., Popeea, C., Rybalchenko, A.: HSF(C): A Software Verifier Based on Horn Clauses - (Competition Contribution). In Flanagan, C., König, B., eds.: TACAS. Volume 7214 of Lecture Notes in Computer Science., Springer (2012) 549–551
39. Bradley, A.R., Manna, Z., Sipma, H.B.: Linear ranking with reachability. In Etessami, K., Rajamani, S.K., eds.: CAV. Volume 3576 of Lecture Notes in Computer Science., Springer (2005) 491–504
40. Brockschmidt, M., Cook, B., Fuhs, C.: Better termination proving through cooperation. In Sharygina, N., Veith, H., eds.: CAV. Volume 8044 of Lecture Notes in Computer Science., Springer (2013) 413–429
41. Bloem, R., Sharygina, N., eds.: Proceedings of 10th International Conference on Formal Methods in Computer-Aided Design, FMCAD 2010, Lugano, Switzerland, October 20-23. In Bloem, R., Sharygina, N., eds.: FMCAD, IEEE (2010)
42. Bonacina, M.P., ed.: Automated Deduction - CADE-24 - 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013. Proceedings. In Bonacina, M.P., ed.: CADE. Volume 7898 of Lecture Notes in Computer Science., Springer (2013)