Invariant Checking of NRA Transition Systems via Incremental Reduction to LRA with EUF

Alessandro Cimatti[1], Alberto Griggio[1], Ahmed Irfan[1,2(✉)],
Marco Roveri[1], and Roberto Sebastiani[2]

[1] Fondazione Bruno Kessler, Trento, Italy
{cimatti,griggio,irfan,roveri}@fbk.eu
[2] University of Trento, Trento, Italy
{ahmed.irfan,roberto.sebastiani}@unitn.it

Abstract. Model checking invariant properties of designs, represented as transition systems, with non-linear real arithmetic (NRA), is an important though very hard problem. On the one hand NRA is a hard-to-solve theory; on the other hand most of the powerful model checking techniques lack support for NRA. In this paper, we present a counterexample-guided abstraction refinement (CEGAR) approach that leverages linearization techniques from differential calculus to enable the use of mature and efficient model checking algorithms for transition systems on linear real arithmetic (LRA) with uninterpreted functions (EUF). The results of an empirical evaluation confirm the validity and potential of this approach.

1 Introduction

Invariant checking for infinite-state transition systems is a fundamental research area. Based on the recent improvements of SMT technologies, effective approaches have been developed for the case of transition systems with dynamics over Linear Real Arithmetic [4,9,18,21]. However, many real-world industrial designs (e.g. aerospace, automotive) require modeling as transition systems over non-linear arithmetic (NRA). Although both problems are undecidable, proving properties of the NRA transition systems turns out to be much harder than the linear case, and has in fact received much less attention. Approaches based on BMC and k-induction [15,29] are possible, so that non-linearity is handled at the SMT-level, by means of an SMT(NRA) solver (e.g. Z3 [13], nlSAT [20], Yices [14], SMT-RAT [1]). Their power is however limited. Consider the following simple transition system: initially, $x \geq 2 \wedge y \geq 2 \wedge z = x * y$; the transition relation is defined by $x' = x + 1 \wedge y' = y + 1 \wedge z' = x' * y'$. The property "*it is always the case that* $z \geq x + y$" is not k-inductive, not even for a very large value of k. Thus, the typical proving techniques that are based on k-induction using an SMT(NRA) solver will not be able to prove it. In principle, it is also possible to

lift other approaches (e.g. interpolation, IC3 [9,26]) to handle non-linearities at the level of the solver. However, this requires the SMT(NRA) solver to carry out interpolation or quantifier elimination, and to proceed incrementally. These extra functions are usually not available, or they have a very high computational cost.

In this paper, we propose a completely different approach to tackle invariant checking for NRA transition systems. Basically, we work with an abstract version of the transition system, expressed over LRA with EUF, for which we have effective verification tools [9]. In the abstract space, nonlinear multiplication is modeled as an uninterpreted function. When spurious counter-examples are found, the abstraction is tightened by the incremental introduction of linear constraints, including tangent planes resulting from differential calculus, and monotonicity constraints.

We implemented the approach on top of the NUXMV model checker [7], leveraging the IC3 engine with Implicit Abstraction [9] for invariant checking of transition systems over LRA with EUF. We compared it, on a wide set of benchmarks, against multiple approaches working at NRA level, including BMC and k-induction using SMT(NRA), the recent interpolation-based ISAT3 engine [24], and the static abstraction approach proposed in [8]. The results demonstrate substantial superiority of our approach, that is able to solve the highest number of benchmarks.

The effectiveness of our approach is possibly explained with the following insights. On the one hand, in contrast to LRA, NRA is a hard-to-solve theory: in practice, most available complete solvers rely on CAD techniques [12], which require double exponential time in worst case. Thus, we try to avoid NRA reasoning, trading it for LRA and EUF reasoning. On the other hand, proving properties of practical NRA transition systems may not require the full power of non-linear solving. In fact, some systems are "mostly-linear" (i.e. non-linear constraints are associated to a very small part of the system), an example being the Transport Class Model (TCM) for aircraft simulation from the Simulink model library [19]. Furthermore, even NRA transition systems with significant non-linear dynamics may admit a piecewise-linear invariant of the transition system that is strong enough to prove the property.

Structure. In Sect. 2 we discuss the related work, and in Sect. 3 introduce some background. In Sect. 4 we discuss the approach in the setting of SMT(NRA). In Sect. 5 we present the verification algorithm for NRA transition systems. In Sect. 6 we describe the results of the experimental evaluation. In Sect. 7 we conclude and outline the directions for future research.

2   Related Work

There are not many tools that deal with NRA transition systems. The most relevant is the recently proposed ISAT3 [28], that uses an interpolation-based [23,24] approach to prove invariants. In addition to NRA, it also supports trascendental functions and some form of differential equations. ISAT3 is built on an SMT solver based on numeric techniques (interval arithmetic), and is able to provide results that are accurate up to the specified precision. In fact, in addition to
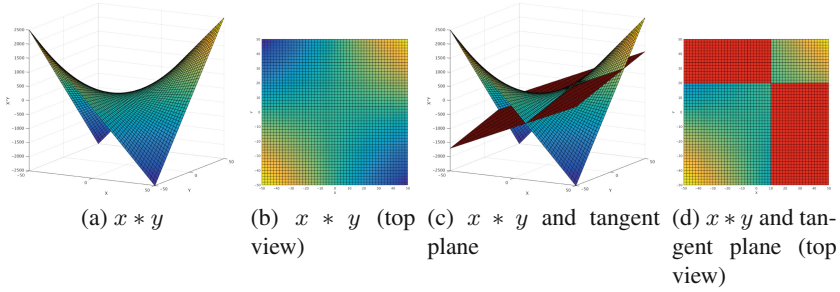
(a) $x * y$ (b) $x * y$ (top view)   (c) $x * y$ and tangent plane   (d) $x * y$ and tangent plane (top view)

Fig. 1. Multiplication function and tangent plane.

"safe" and "unsafe" answers, ISAT3 may return "maybe unsafe" when it finds an envelope of given precision that may (but is not guaranteed to) contain a counterexample. Another relevant tool is DREACH [22], a bounded model checker implemented on top of the DREAL [16] SMT solver, that adopts numerical techniques similar to ISAT3. DREACH has an expressiveness similar to ISAT3, but being a bounded model checker it is unable to prove properties.

The work in [8] follows a reduction-based approach to check invariants of NRA transition systems. It over-approximates the non-linear terms with a coarse abstraction, encoding into LRA some weak properties of multiplication like identity and sign. Another reduction-based approach is presented in [25] in the context of program analysis. The idea is to find a (tighter) convex approximation of polynomials in form of polyhedra, thus obtaining a conservative linear transition system. The key differences of our approach with respect to [8, 25] are that we iteratively refine the abstraction, and we adopt a reduction to LRA+EUF. Furthermore, to the best of our knowledge, there is no available implementation of the approach [25] in a program analysis tool – it has been only shown to work on SMT problems.

The idea of approximating a univariate function (in particular the natural logarithm ln) with tangent lines is used in [30]. Here we abstract a bivariate function (multiplication), and use tangent planes for the refinement. We also exploit other properties (e.g. monotonicity) to derive additional axioms. The idea of using tangent planes (spaces) has been explored in [27], limited to the case of SMT solving. Another key differences is that the tangent planes area used to under-approximate predicates, while we use them to refine the over-approximation of the multiplication function.

3   Background

Properties of the Multiplication Function. Geometrically, the surface generated by the multiplication function $f(x, y) \stackrel{\text{def}}{=} x * y$ is shown in Fig. 1a and b. This kind of surface is known in geometry as hyperbolic paraboloid. A hyperbolic paraboloid is a doubly-ruled surface, i.e. for every point on the surface, there are two distinct lines projected from the surface such that they pass through the point. In case of the multiplication surface, the projected lines basically lie on the surface.

Tangent Plane. The tangent plane to a surface at a point of interest (a, b) is a plane that "just touches" the surface at the point. The tangent planes can be used to linearly approximate the surface at the point of interest. An important property of the tangent plane to a hyperbolic paraboliod is that the two projected lines from the surface are also in the tangent plane, and they define how the plane cuts the surface (see Fig. 1c and d). The tangent plane $Tmul_{a,b}(x, y)$ to the multiplication function $f(x, y)$ at point (a, b) is calculated as follows:

$$Tmul_{a,b}(x, y) \stackrel{\text{def}}{=} f(a, b) + \frac{d}{dx} f(x, y)_{|(a,b)} * (x - a) + \frac{d}{dy} f(x, y)_{|(a,b)} * (y - b)$$

where $\frac{d}{dx} f(x, y)_{|(a,b)}$ and $\frac{d}{dy} f(x, y)_{|(a,b)}$ are the first-order partial derivatives of $f(x, y)$ w.r.t. x and y respectively, evaluated at (a, b). $Tmul_{a,b}(x, y)$ simplifies to:

$$Tmul_{a,b}(x, y) \stackrel{\text{def}}{=} b * x + a * y - a * b \tag{1}$$

Logic and Satisfiability. We assume the standard first-order quantifier-free logical setting and standard notions of theory, model, satisfiability, and logical consequence. If φ is a formula, we denote with $\text{vars}(\varphi)$ the set of its variables, and with $\text{atoms}(\varphi)$ the set of its atoms. We write $\varphi(X)$ to denote that $\text{vars}(\varphi) \subseteq X$. If x and y are two variables, we denote with $\varphi\{x \mapsto y\}$ the formula obtained by replacing all the occurrences of x in φ with y. We extend this notation to ordered sequences of variables in the natural way. If μ is a model and x is a variable, we write $\mu[x]$ to denote the value of x in μ, and we extend this notation to terms in the usual way. If X is a set of variables, we denote with X' the set obtained by replacing each element $x \in X$ with x', and with $X^{\langle i \rangle}$ the set obtained by replacing x with $x^{\langle i \rangle}$. If Γ is a set of formulas, we write $\bigwedge \Gamma$ to denote the formula obtained by taking the conjunction of all its elements. If $\bigwedge \Gamma$ is unsatisfiable (modulo some theory T), an unsatisfiable core is a set $C \subseteq \Gamma$ such that $\bigwedge C$ is still unsatisfiable.

Symbolic Transition Systems. A symbolic transition system $\mathcal{S} \stackrel{\text{def}}{=} \langle X, I, T \rangle$ is a tuple where X is a finite set of (state) variables, $I(X)$ is a formula denoting the initial states of the system, and $T(X, X')$ is a formula expressing its transition relation. A state s_i of $\mathcal{S}$ is an assignment to the variables X. A path (execution trace) $\pi = s_0, s_1, s_2, \ldots, s_{k-1}$ of length k (possibly infinite) for $\mathcal{S}$ is a sequence of states such that $s_0 \models I$ and $s_i \wedge s_{i+1}\{X \mapsto X'\} \models T$ for all $0 \leq i < k - 2$. We call an unrolling of $\mathcal{S}$ of length k the formula $I\{X \mapsto X^{\langle 0 \rangle}\} \wedge \bigwedge_{i=0}^{k-1} T\{X \mapsto X^{\langle i \rangle}\}\{X' \mapsto X^{\langle i+1 \rangle}\}$.

Let $P(X)$ be a formula whose assignments represent a property (good states) over the state variables X. The invariant verification problem, denoted with $S \models P$, is the problem of checking if for all the finite paths $s_0, s_1, \ldots, s_k$ of $\mathcal{S}$, for all i, $0 \leq i \leq k$, $s_i \models P$. Its dual formulation in terms of reachability of $\neg P$ is the problem of finding a path $s_0, s_1, \ldots, s_k$ of $\mathcal{S}$ such that $s_k \models \neg P$. P represents the "good" states, while $\neg P$ represents the "bad" states.

```
bool SMT-NRA-check-abstract (φ):
 1.  φ̂ = initial-abstraction(φ)
 2.  res, data = SMT-NRA-check-abstract-ext(φ̂)
 3.  return res

⟨bool, axiom set or model⟩ SMT-NRA-check-abstract-ext (φ̂):
 4.  Γ = ∅
 5.  while true:
 6.      if budget-exhausted(): abort
 7.      res, μ̂ = SMT-LRA+EUF-check(φ̂ ∧ ⋀ Γ)
 8.      if not res: return false, Γ
 9.      ok, μ = get-NRA-model(φ̂, μ̂)
10.      if ok: return true, μ
11.      else: Γ = Γ ∪ refine(μ̂)
```

Fig. 2. Solving SMT(NRA) via abstraction to SMT(LRA+EUF).

4   Solving SMT(NRA) via SMT(LRA+EUF)

Top-Level Algorithm. The main idea of this paper is that of solving an SMT formula containing non-linear polynomial constraints (i.e., expressed in the NRA theory) by overapproximating it with a formula over the combined theory of linear arithmetic and uninterpreted functions (LRA+EUF). Our main SMT solving procedure follows a classic abstraction refinement loop, in which at each iteration the current overapproximation of the input SMT formula is refined by adding new constraints that rule out one (or possibly more) spurious solutions, until one of the following occurs: (i) the SMT formula becomes unsatisfiable in the LRA+EUF theory; or (ii) the LRA+EUF model for the current overapproximation can be lifted to an NRA model for the original SMT formula; or (iii) the resource budget (e.g. time, memory, number of iterations) is exhausted.

The pseudocode for the top-level algorithm is shown in Fig. 2. We provide more details about its main components in the rest of this section.

Initial Abstraction. The function initial-abstraction takes as input an SMT(NRA) formula φ and returns an overapproximation $\widehat{\varphi}$ of it in the LRA+EUF theory.

First, each multiplication expression $x * y$ between two variables[1] occurring in φ is replaced by $fmul(x, y)$, where $fmul()$ is a binary uninterpreted function returning a real. We remark that this happens only for non-linear multiplications: expressions like $c * x$ or $x * c$ in which c is a constant are not rewritten.

Then, some simple axioms about multiplication are added to $\widehat{\varphi}$ via static learning. For each $fmul(x, y) \in \widehat{\varphi}$, we add the following axioms:

---

[1] To simplify the presentation, we assume (here and in the rest of the paper) that all multiplications in φ are either between two variables or between one constant and one variable.

Commutativity: $fmul(x, y) = fmul(y, x)$

Sign: $fmul(x, y) = fmul(-x, -y) \land fmul(x, y) = -fmul(-x, y) \land$
$fmul(x, y) = -fmul(x, -y)$

Zero: $((x = 0 \lor y = 0) \leftrightarrow fmul(x, y) = 0) \land$
$(((x > 0 \land y > 0) \lor (x < 0 \land y < 0)) \rightarrow fmul(x, y) > 0) \land$
$(((x < 0 \land y > 0) \lor (x < 0 \land y > 0)) \rightarrow fmul(x, y) < 0)$

Abstraction Refinement. If the SMT check on the LRA+EUF abstraction returns false (line 7 of Fig. 2), we can conclude that the input formula is unsatisfiable. In this case, Γ contains all the lemmas (discussed later in this section) that were added in the earlier refinements (line 11 of Fig. 2).

Otherwise, we have to check whether the model $\widehat{\mu}$ found for $\widehat{\varphi}$ is also a model for the original NRA formula φ. Let $Fmuls$ be the set of all $fmul(x, y)$ terms occurring in $\widehat{\varphi}$. In its simplest version, the function get-NRA-model checks whether, for all $fmul(x, y)$ in $Fmuls$, $\widehat{\mu}[fmul(x, y)] = \widehat{\mu}[x] * \widehat{\mu}[y]$. If this is the case, then $\widehat{\mu}$ is also a model for the original formula, and get-NRA-model returns true. (We present more sophisticated versions of get-NRA-model below.) Otherwise, let $CFmuls$ be the set of all $fmul(x, y)$ terms whose value in $\widehat{\mu}$ is different from $\widehat{\mu}[x] * \widehat{\mu}[y]$. The function refine generates a set of axioms Γ' such that there exists at least one element $fmul(x, y)$ of $CFmuls$ such that the formula $\widehat{\varphi} \land \bigwedge \Gamma'$ has no model $\widehat{\mu'}$ that agrees with $\widehat{\mu}$ on the values of x, y and $fmul(x, y)$ (i.e. such that $\widehat{\mu'}[fmul(x, y)] = \widehat{\mu}[fmul(x, y)]$, $\widehat{\mu'}[x] = \widehat{\mu}[x]$ and $\widehat{\mu'}[y] = \widehat{\mu}[y]$). Intuitively, the axioms Γ' block the bad model values for $fmul(x, y)$, making the abstraction more precise by restricting the set of spurious solutions.

In our current implementation, two kinds of lemmas are generated during refinement: *tangent lemmas* and *monotonicity lemmas*.

Tangent Lemmas. We use the model values $\widehat{\mu}[fmul(x, y)]$, $\widehat{\mu}[x]$ and $\widehat{\mu}[y]$ and (1) to generate tangent plane lemmas for $fmul(x, y)$:

$$fmul(a, y) = a * y \quad \land \quad fmul(x, b) = b * x \quad \land$$
$$(((x > a \land y < b) \lor (x < a \land y > b)) \rightarrow fmul(x, y) < Tmul_{a,b}(x, y)) \quad \land \quad (2)$$
$$(((x < a \land y < b) \lor (x > a \land y > b)) \rightarrow fmul(x, y) > Tmul_{a,b}(x, y))$$

where we can choose a and b as:

$$a \stackrel{\text{def}}{=} \widehat{\mu}[x] \text{ and } b \stackrel{\text{def}}{=} \widehat{\mu}[y] \tag{3}$$

$$a \stackrel{\text{def}}{=} \frac{1}{\widehat{\mu}[fmul(x, y)]} \text{ and } b \stackrel{\text{def}}{=} \widehat{\mu}[y] \tag{4}$$

$$a \stackrel{\text{def}}{=} \widehat{\mu}[x] \text{ and } b \stackrel{\text{def}}{=} \frac{1}{\widehat{\mu}[fmul(x, y)]}. \tag{5}$$

Basically the equalities in the tangent lemma are providing multiplication lines that enforce the correct value of $fmul(x, y)$ when $x = a$ or $y = b$. Moreover, the inequalities of the tangent lemma are providing bounds for $fmul(x, y)$ when x and y are not on the multiplication lines.

⟨**bool**, model⟩ get-NRA-model ($\widehat{\varphi}$, $\widehat{\mu}$):
1. $\widehat{\psi}$ = get-assignment($\widehat{\mu}$) # *truth assignment induced by $\widehat{\mu}$ on the atoms of $\widehat{\varphi}$*
2. ψ = concretize($\widehat{\psi}$) # *replace each $fmul(x, y)$ in $\widehat{\psi}$ with $x * y$*
3. **return** SMT-NRA-check (ψ) # *check with a complete NRA solver*

Fig. 3. A complete procedure using an NRA solver.

Monotonicity Lemmas. Let $fmul(x, y)$ and $fmul(w, z)$ be two terms in $\widehat{\varphi}$, such that $|\widehat{\mu}[x]| \leq |\widehat{\mu}[w]|$, $|\widehat{\mu}[y]| \leq |\widehat{\mu}[z]|$, and $|\widehat{\mu}[fmul(x, y)]| > |\widehat{\mu}[fmul(w, z)]|$. Then, we add the monotonicity lemma

$$(abs(x) \leq abs(w) \wedge abs(y) \leq abs(z)) \rightarrow abs(fmul(x, y)) \leq abs(fmul(w, z)), \quad (6)$$

where $abs(t)$ stands for $\text{ite}(t < 0, -t, t)$.

Finding Models. It is easy to see that our algorithm is expected to perform much better for unsatisfiable instances than for satisfiable ones. The algorithm can return true (meaning that the formula is satisfiable) only if the LRA+EUF solver "guesses" a model that is consistent with all the nonlinear multiplications. In an infinite and dense domain like the reals, the chances that this will happen are close to zero in general.

   Moreover, our approach is inherently limited, because it can only find models over the rationals. If the input formula is satisfiable, but all its models contain some irrational values, then our algorithm will always abort (or never terminate, if there is no resource budget set). In practice, it is very likely that the same will happen even for formulas admitting a rational solution.

   One possibility for addressing this limitation would be to couple our procedure with a complete solver for NRA, to be used for detecting satisfiable cases, in order to implement a more effective version of get-NRA-model. One such possibility is shown in Fig. 3, where we extract the truth assignment $\widehat{\psi}$ induced by the LRA+EUF model $\widehat{\mu}$ on the atoms of $\widehat{\varphi}$:

$$\widehat{\psi} \stackrel{\text{def}}{=} \bigwedge_{[\widehat{a_i} \in \text{atoms}(\widehat{\varphi}) \ s.t. \ \widehat{\mu} \models \widehat{a_i}]} \widehat{a_i} \ \wedge \bigwedge_{[\widehat{a_i} \in \text{atoms}(\widehat{\varphi}) \ s.t. \ \widehat{\mu} \not\models \widehat{a_i}]} \neg \widehat{a_i}, \quad (7)$$

We concretize it by replacing each $fmul(x, y)$ in $\widehat{\psi}$ with $x * y$, and invoke the complete NRA theory solver on the resulting conjunction of NRA-literals ψ, to check whether it contains at least one solution. Although in general the problem is expected to be simpler than the original input formula because the Boolean structure of φ is disregarded, invoking a complete NRA theory solver at each loop iteration of SMT-NRA-check-abstract-ext could be very expensive. Moreover, this would still require a complete NRA theory solver, which might not always be available.

   As an alternative, we propose the procedure outlined in Fig. 4, where we extract the truth assignment $\widehat{\psi}$ induced by the LRA+EUF model $\widehat{\mu}$ on the atoms of $\widehat{\varphi}$, and we conjoin to it the *multiplication lines*:

```
⟨bool, model⟩ get-NRA-model (φ̂, μ̂):
1.  ψ̂ = get-assignment(μ̂)  # truth assignment induced by μ̂ on the atoms of φ̂
2.  ψ̂* = ψ̂ ∧ linearization-axioms(ψ̂)  # add multiplication-line axioms to ψ̂
3.  return SMT-LRA+EUF-check (ψ̂*)
```

Fig. 4. An incomplete procedure using an SMT(LRA+EUF) solver.

$$\widehat{\psi}^* = \widehat{\psi} \ \wedge \bigwedge_{fmul(x,y)\in Fmuls} \left(\begin{array}{l} (x = \widehat{\mu}[x] \wedge fmul(x,y) = \widehat{\mu}[x]*y) \ \vee \\ (y = \widehat{\mu}[y] \wedge fmul(x,y) = \widehat{\mu}[y]*x) \end{array} \right), \quad (8)$$

$Fmuls$ being the usual set of all $fmul(x,y)$ terms occurring in $\widehat{\varphi}$.

The main idea is to build an LRA+EUF underapproximation $\widehat{\psi}^*$ of the NRA formula ψ of Fig. 3, in which all multiplications are forced to be linear. Compared to the previous solution, this has the advantage of requiring a complete SMT(LRA+EUF) solver rather than a (much more expensive) complete NRA solver. Moreover, given the simplicity of the Boolean structure of the underapproximated formula, the check should in general be very cheap. The drawback is that this is (clearly) still an incomplete procedure. However, in our experiments (for which we refer to Sect. 6) we have found it to be surprisingly effective for many problems.

Unlike with the basic implementation of get-NRA-model which considers only one single candidate model at a time, the implementations in Figs. 3 and 4 consider an infinite amount of them, drastically increasing the chances of finding a model.

Correctness and Progress. We notice that the procedure in Fig. 2 is *correct*. In fact, it returns false only if φ is NRA-unsatisfiable because by construction $\widehat{\varphi}$ is an over-approximation of φ, and all axioms in Γ are valid in any theory interpreting $fmul(x,y)$ as $x*y$. Also, it returns true only if φ is NRA-satisfiable:

- if get-NRA-model is based only on evaluation, then by construction μ is an LRA+EUF-model for $\widehat{\varphi}$ s.t. each $fmul(x,y)$ equals $x*y$ in μ, so that μ is also a model for φ;
- if get-NRA-model is as in Fig. 3, then μ is an NRA-model of a conjunction of literals ψ which tautologically entails φ, so that μ is a model for φ;
- if get-NRA-model is as in Fig. 4, then μ is an LRA+EUF-model of a conjunction of literals $\widehat{\psi}^*$ which tautologically entails $\widehat{\varphi}$ and it is s.t. each $fmul(x,y)$ equals $x*y$ in μ, so that μ is a also model for φ.

We also notice that the progress of the procedure in Fig. 2 is guaranteed by the refinement step, which rules out significant parts of the search space at every loop by means of the added lemmas.

Important Heuristics for Refinement. The description of refine provided above leaves some flexibility in deciding what axioms to add (and how many of
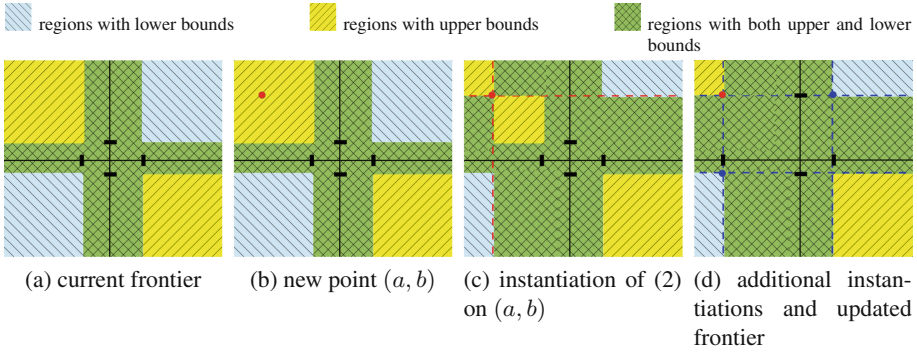
(a) current frontier   (b) new point (a, b) (c) instantiation of (2) on (a, b) (d) additional instantiations and updated frontier

Fig. 5. Illustration of the tangent lemma frontier strategy.

them) at each iteration. It is possible to conceive strategies with an increasing degree of eagerness, from very lazy (e.g. adding only a single axiom per iteration) to more aggressive ones. In our current implementation, we eagerly add all the axioms (2)–(6) that are violated by the current abstract solution $\widehat{\mu}$, leaving the investigation of alternative strategies as future work. However, we found the following two strategies to be crucial for performance.

Tangent Lemma Frontiers. The tangent lemmas of (2) for a given point (a, b) are based on the fact that the multiplication function $x * y$ is a hyperbolic paraboloid surface, and a tangent plane to such surface cuts the surface into four regions such that in two of the regions the tangent plane is above the surface, whereas in the other two regions the tangent plane is below the surface (see Fig. 1). Each instantiation of (2) for a given point, therefore, can only provide either a lower or an upper bound for a given region. In some cases, this might lead to an infinite refinement loop in which at each iteration the "wrong" bound is refined. In order to address the problem, we use the following strategy. For each $fmul(x, y)$ in the input formula, we maintain a *frontier* $\langle l_x, u_x, l_y, u_y \rangle$ with the invariant that whenever x is in the interval $[l_x, u_x]$ or y is in the interval $[l_y, u_y]$, then $fmul(x, y)$ has both an upper and a lower bound. Initially, the frontiers are set to $\langle 0, 0, 0, 0 \rangle$. Whenever a lemma (2) for $fmul(x, y)$ is instantiated on a point (a, b), we generate further instantiations of (2) and update the frontier as follows:

case $a < l_x$ and $b < l_y$: instantiate (2) on (a, u_y) and on (u_x, b), and set the frontier to $\langle a, u_x, b, u_y \rangle$;

case $a < l_x$ and $b > u_y$: instantiate (2) on (a, l_y) and on (u_x, b), and set the frontier to $\langle a, u_x, l_y, b \rangle$;

case $a > u_x$ and $b > u_y$: instantiate (2) on (a, l_y) and on (l_x, b), and set the frontier to $\langle l_x, a, l_y, b \rangle$;

case $a > u_x$ and $b < l_y$: instantiate (2) on (a, u_y) and on (l_x, b), and set the frontier to $\langle l_x, a, b, u_y \rangle$.

Figure 5 shows a graphical illustration of the strategy.

Tangent Lemma Rounding. The instantiation of a tangent lemma at the point (a, b) has the side-effect of adding the rational constants a, b and $a * b$ to the formula that is solved by the LRA+EUF solver. If such values have large numerators and/or denominators, they might be a source of a significant slow-down for the LRA solver (which works on exact, arbitrary-precision rational arithmetic). We address this issue by observing that, in order to block a bad model $\widehat{\mu}$ such that $\widehat{\mu}[fmul(x,y)] \neq \widehat{\mu}[x] * \widehat{\mu}[y]$, it is sufficient to add one of the two equalities of (2); therefore, instead of instantiating a tangent lemma at (a, b), we can instantiate it at either $(a+\delta, b)$ or at $(a, b+\delta)$, for any value of δ. In practice, if a (resp. b) is a rational constant with a very large numerator or denominator, instead of instantiating a tangent lemma at (a, b), we instantiate two tangent lemmas at $(\lfloor a \rfloor, b)$ and $(\lceil a \rceil, b)$.

5   From Satisfiability to Verification

We now move from satisfiability checking to verification.

Overview. In principle, the solver described in the previous section could be integrated as a "black box" in any off-the-shelf SMT-based verification algorithm, such as BMC, k-induction, or one of the many extensions of IC3 to the SMT case (e.g. [4,9,18,21]). In practice, however, such black-box integration would hardly be effective, especially in the case of state-of-the-art algorithms like IC3. IC3 requires a very incremental interaction with the underlying SMT engine, which is asked to solve a large number of relatively-cheap queries. The procedure of Sect. 4, however, can be very expensive, especially for satisfiable queries, which are very common in an IC3-like algorithm.[2] Moreover, some of the IC3 extensions mentioned above require the ability of performing (approximated) quantifier eliminations, a functionality not provided by the algorithm of Fig. 2.

   We propose therefore a white-box integration, in which we lift the abstraction refinement approach of Sect. 4 at the transition system level. We generate an abstract LRA+EUF version of the input NRA transition system, which is then checked with the IC3-based procedure of [9]. In case a counterexample is produced, we use the SMT-NRA-check-abstract-ext algorithm of Fig. 2 to check whether it is spurious. If so, the axioms generated by SMT-NRA-check-abstract-ext are then used to refine the abstraction of the transition system. The pseudo-code of this algorithm is reported in Fig. 6. Similarly to the satisfiability checking case, the initial-abstraction function replaces every non-linear multiplication $x * y$ in the input transition system and property with a $fmul(x, y)$ term, and adds some simple axioms about the behaviour of multiplication to the initial-state and transition-relation formulas of the transition system (see Sect. 4). In the rest of this section, we describe the abstraction refinement algorithm in more detail.

---

[2] In fact, as already discussed in Sect. 4, the procedure is biased towards unsatisfiable instances, and might easily diverge on satisfiable ones.

```
bool IC3-NRA-prove (S : transition system ⟨X, I, T⟩, φ : invarant property):
1.    Ŝ, φ̂ = initial-abstraction(S, φ)
2.    while true:
3.        if budget-exhausted(): abort
4.        ok, π̂ = IC3-LRA+EUF-prove(Ŝ, φ̂)
5.        if ok: return true  # property proved
6.        ψ = get-cex-formula(Ŝ, φ̂, π̂)
7.        is_cex, Γ = SMT-NRA-check-abstract-ext(ψ)
8.        if is_cex: return false  # counterexample found
9.        else: Ŝ = refine-transition-system(Ŝ, Γ)
```

Fig. 6. Verification of NRA transition systems via abstraction to LRA+EUF.

Counterexample Checking and Refinement. When IC3-LRA+EUF-prove returns a counterexample trace $\hat{\pi}$ for the abstract system $\widehat{S}$, we use SMT-NRA-check-abstract-ext to check for its spuriousness. The function get-cex-formula builds a formula ψ to feed to SMT-NRA-check-abstract-ext, whose unsatisfiability implies that $\hat{\pi}$ is spurious. The formula ψ is built by unrolling the transition relation of $\widehat{S}$, and optionally adding constraints that restrict the allowed transitions to be compatible with the states in $\hat{\pi}$. Various heuristics are possible, trading generality for complexity: ψ could be fully constrained by the states in $\hat{\pi}$ (thus checking only one abstract counterexample path per iteration); it could be only partially constrained (e.g. by considering only the Boolean variables and/or the state variables occurring only in linear constraints); or it could be left unconstrained, considering only the length of the abstract counterexample. In our current implementation (see Sect. 6), we use the last option, i.e. we only consider the length of $\hat{\pi}$ to build a BMC formula that checks for any counterexample of the given length, leaving the investigation of alternative strategies to future work.

If SMT-NRA-check-abstract-ext returns true, the property is violated. In this case, we can use the model found by SMT-NRA-check-abstract-ext to build a counterexample trace for the input system and property.

If SMT-NRA-check-abstract-ext returns false, we use the axioms Γ produced during search to refine the transition system $\widehat{S}$, using the procedure shown in Fig. 7. Essentially, refine-transition-system translates back the axioms from their unrolled version (i.e. on variables $X^{\langle 0 \rangle}, X^{\langle 1 \rangle}, \ldots$) to their "single step" version (on variables X and X'), adding each of them either to the initial-states formula or to the transition relation formula. In case an axiom γ spans more than a single transition step (lines 9–10 of Fig. 7), we arbitrarily choose to map the variables with the lowest index as current state variables X, and all the others as next-state variables X'. Notice that this might cause some refinement failure, as discussed in the next paragraph.

Reducing the Number of Axioms to Add. In general, not all the axioms generated during a call to SMT-NRA-check-abstract-ext are needed to successfully block a counterexample, especially if eager strategies like those described in Sect. 4 are

```
transition system refine-transition-system (Ŝ : transition system, Γ : set of axioms):
1.  let ⟨X, Î, T̂⟩ = Ŝ
2.  Γ_I, Γ_T = ∅, ∅
3.  for each γ in Γ:
4.      if vars(γ) ⊆ X^⟨0⟩:
5.          Γ_I = Γ_I ∪ {γ{X^⟨0⟩ ↦ X}}
6.      else if there exists i > 0 s.t. vars(γ) ⊆ X^⟨i⟩:
7.          Γ_T = Γ_T ∪ {γ{X^⟨i⟩ ↦ X}, γ{X^⟨i⟩ ↦ X'}}
8.      else
9.          let i be the smallest index s.t. vars(γ) ∩ X^⟨i⟩ ≠ ∅
10.         Γ_T = Γ_T ∪ {γ{X^⟨i⟩ ↦ X}{X^⟨i+1⟩ ∪ ... ∪ X^⟨i+k⟩ ↦ X'}}
11. return ⟨X, Î ∧ ⋀ Γ_I, T̂ ∧ ⋀ Γ_T⟩
```

Fig. 7. Refinement of the LRA+EUF transition system.

```
transition system reduce-axioms (⟨X, Î, T̂⟩, φ̂, π̂, ⟨Γ_I, Γ_T⟩):
1.  ψ = get-cex-formula(⟨X, Î ∧ ⋀ Γ_I, T̂ ∧ ⋀ Γ_T⟩, π̂)
2.  if not SMT-LRA+EUF-check(ψ):
3.      let C be an unsatisfiable core of ψ
4.      Γ_I = {γ ∈ Γ_I | γ{X ↦ X^⟨0⟩} ∈ C}
5.      Γ_T = {γ ∈ Γ_T | ∃j > 0 s.t. γ{X ↦ X^⟨j⟩}{X' ↦ X^⟨j+1⟩} ∈ C}
6.      return ⟨X, Î ∧ ⋀ Γ_I, T̂ ∧ ⋀ Γ_T⟩
7.  else
8.      abort  # refinement failure
```

Fig. 8. Reducing the axioms needed for refinement.

used. In the long run, having a large number of redundant axioms can be quite harmful for performance. In order to mitigate this problem, we apply a filtering strategy (based on unsatisfiable cores) to the set of axioms, before adding them to the transition system. Instead of adding Γ_I and Γ_T directly to $\widehat{\mathcal{S}}$, we invoke the function shown in Fig. 8. Note that due to the flattening of multi-step axioms described above (lines 9–10 of Fig. 7), the refinement might fail. In this case, our current implementation simply aborts the execution.[3]

6   Experimental Analysis

Implementation and Comparisons. We have implemented a prototype of the IC3-NRA-prove procedure using the IC3 engine of NUXMV [7] for IC3-LRA+EUF-prove. The code is written in Python, using the PYSMT library [17]. Our implementation, benchmarks, and experimental data are available at https://es-static.fbk.eu/people/griggio/papers/tacas17-ic3-nra.tar.gz. We have used the following tools for our evaluation.

NUXMV-*LRA-static:* we apply the upfront abstraction of NRA to LRA proposed in [8], running the IC3 engine of NUXMV on the resulting transition system.

---

[3] We remark however that so far we have never observed this behaviour during our experiments.

NRA-BMC-{Z3, DREAL} *and NRA-K-induction-*{Z3, DREAL}*:* we have implemented the BMC [3] and k-induction [29] algorithms in Python (using PYSMT), using either Z3 (NRA) or DREAL (NRA) as back-end SMT solver.

iSAT3[1e-1] and iSAT3[1e-9]: we have used the latest version of the ISAT3 solver [24], which combines an SMT solver integrating CDCL and interval constraint propagation techniques with an interpolation-based abstraction/refinement algorithm for verification. ISAT3 supports both transition systems and software programs encoded as control flow graphs. Similarly to DREAL, ISAT3 may return a "maybe unsafe" answer and provide a candidate solution identifying the upper and lower bounds on the variables. In the experiments, iSAT3[1e-1] is the configuration suggested by the ISAT3 authors[4] and iSAT3[1e-9] is the same except that the *minimum splitting width* (`msw`) parameter is set to 10^{-9}. We have used a smaller value for the `msw` to get more precise answers, i.e. "safe" or "unsafe", as suggested in the ISAT3 user manual.

Benchmarks. We have collected a total of 114 NRA benchmarks from various sources.

Handcrafted. This set contains 14 hand-written instances, 13 safe and 1 unsafe.

HyComp. The second set contains 7 benchmarks (3 safe, 4 unsafe) which are taken from [11] and converted to NRA transition systems using HYCOMP [10].

HYST. This is the biggest set, consisting of 65 benchmarks. These are generated from the Hybrid examples that come with the HYST [2] distribution, by approximating the continuous time by sampling at a fixed time interval. This process is done automatically using an extended version of HYST. Since the generated benchmarks are approximations, we do not know their safety status. The benchmarks contain mostly non-linear behaviour.

iSAT3 and iSAT3-CFG. The 11 benchmarks in this set (7 safe, 4 unsafe) are taken from [24] and the ISAT3 examples available online.

nuXmv. In this set, we have 2 safe benchmarks which we collected from the nuXmv users' mailing list. These benchmarks have complex boolean structure.

SAS13. These 13 benchmarks are generated from the C programs used in [5], but interpreted over NRA instead of the theory of IEEE floating-point numbers. This makes some of the instances unsafe.

TCM. We have generated 2 safe benchmarks from the Simulink models (taken from the case study [6]) by first generating the C code using the Embedded Coder[5] and then encoding the program into a symbolic transition system.

Results. We ran our experiments on a cluster of machines with 2.67 GHz Xeon X5650 CPUs and 96 GB of RAM, running Scientific Linux 6.7. We used 6 GB memory limit and 3600 s CPU timeout.

---

[4] `-I --use-craig-interpolation --use-cegar --cegar-abstraction-inductive --interpolant-rules-mcmillan --interpolant-a-biased --interpolation-offset --interpolant-offset 2`.
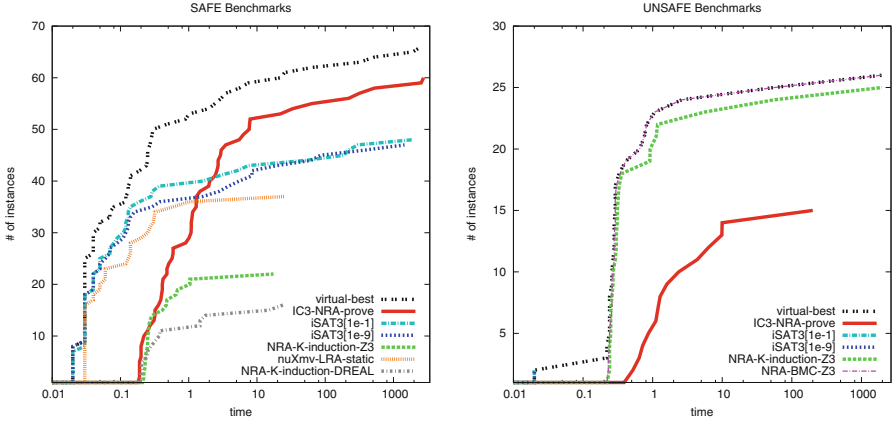
[5] https://www.mathworks.com/products/embedded-coder/.

Fig. 9. Plots of SAFE and UNSAFE results.

Table 1. Summary of experimental results.

	Total	Handcrafted	HyComp	HYST	ISAT3	ISAT3-CFG	nuXmv	SAS13	TCM
	(114)	**(14)**	**(7)**	**(65)**	**(1)**	**(10)**	**(2)**	**(13)**	**(2)**
IC3-NRA-prove	**60**/15	**9**/1	**3**/0	33/7	0/0	**6**/2	**2**/0	**5**/5	**2**/0
iSAT3[1e-1]	48/2(47)	2/0(8)	0/0(3)	**34**/2(23)	0/0	**6**/0(4)	0/0	4/0(9)	**2**/0
iSAT3[1e-9]	47/2(19)	2/0(3)	0/0(2)	32/2(3)	0/0	**6**/0(3)	0/0	**5**/0(8)	**2**/0
NRA-K-induction-Z3	22/25	2/1	0/2	12/**15**	0/0	**6**/2	0/0	0/5	**2**/0
NUXMV-LRA-static	37/0	4/0	1/0	19/0	0/0	4/0	**2**/0	**5**/0	**2**/0
NRA-BMC-Z3	0/**26**	0/1	0/2	0/**15**	0/0	0/3	0/0	0/5	0/0
NRA-K-induction-DREAL	16/0(32)	2/0(4)	0/0(2)	9/0(19)	0/0	5/0(2)	0/0	0/0(5)	0/0
NRA-BMC-DREAL	0/0(39)	0/0(8)	0/0(2)	0/0(19)	0/0	0/0(3)	0/0	0/0(7)	0/0
virtual-best	66/26	9/1	3/2	38/15	0/0	7/3	2/0	5/5	2/0

Each column shows a benchmark family, and each entry gives the number of safe/unsafe instances found. For tools working over interval arithmetic, the number of "maybe unsafe" is reported in parentheses.

The results are summarized in Tables 1 and 2 and in Fig. 9. The plots show the time to solve an instance on the x-axis and the total number of solved instances on the y-axis. Table 1 reports a summary of the solved instances by family, whereas Table 2 shows a comparitive analysis by reporting for each tool the number of uniquely solved instances and the difference of solved instances w.r.t IC3-NRA-prove. We can make the following observations from the experimental results:

Table 2. Comparitive summary of total solved benchmarks.

	# Solved	# Uniquely Solved	Difference wrt. IC3-NRA-prove	Gained	Lost
IC3-NRA-prove	**60**/15	**9**/0	-	-	-
nuXmv-LRA-static	37/0	0/0	-38	1/0	24/15
iSAT3[1e-1]	48/2(47)	4/0	-25	4/0	16/13
iSAT3[1e-9]	47/2(19)		-26	3/0	16/13
NRA-K-induction-Z3	22/25	0/11	-28	2/11	40/1
NRA-BMC-Z3	0/**26**		-49	0/11	60/0
NRA-K-induction-DREAL	16/2(32)	0/0	-59	2/0	46/15
NRA-BMC-DREAL	0/0(39)		-75	0/0	60/15
virtual-best	66/26	-	17	6/11	0

– IC3-NRA-prove is the best performer overall, and it significantly outperforms all the other approaches on safe instances (where it can solve 9 problems that are out of reach for all the other tools). Interestingly, despite its simplicity, our model finding approach (as outlined in Sect. 4) is surprisingly effective, allowing IC3-NRA-prove to find 15 counterexample traces.

– The simple abstraction proposed in [8] is quite effective for many families, allowing nuXmv-LRA-static to verify more properties than the approaches based on K-induction with an NRA solver. However, IC3-NRA-prove results in a clear and very significant improvement, solving more than twice as many instances than nuXmv-LRA-static (and losing only 1).

– None of the other tools (with the exception of nuXmv-LRA-static) is able to solve any safe benchmark in the HyComp and nuXmv families. These benchmarks have a non-trivial Boolean structure and a significant linear component. Both IC3-NRA-prove and nuXmv-LRA-static are able to fully exploit the effectiveness of the underlying IC3 engine of nuXmv, outperforming the competitors. However, IC3-NRA-prove is very competitive also on the HYST family, whose instances are mostly non-linear and have very little Boolean structure.

– Increasing the default precision of iSAT3 significantly reduces the number of "maybe unsafe" answers, but it doesn't seem to help in solving more benchmarks. In fact, we remark that even with the increased precision iSAT3[1e-9] classifies 2 safe instances as "maybe unsafe" (whereas in the default configuration, 6 safe instances are classified as "maybe unsafe").

7   Conclusions and Future Work

We presented a novel abstraction-refinement approach to the verification of transition systems with nonlinear dynamics expressed in the NRA theory. We abstract non-linear multiplication as an uninterpreted function, leveraging efficient invariant checkers for transition systems over LRA and EUF to solve the problem in the abstract space. In case of spurious counterexample, the abstraction of multiplication is incrementally refined by introducing suitable axioms,

based on the idea of tangent planes. An extensive experimental evaluation demonstrates that the proposed approach is significantly more effective than approaches directly based on SMT(NRA) solving.

This work opens up several important directions. First, we are going to improve the implementation, by integrating all the steps within the NUXMV [7] model checker, and to perform a thorough analysis of the various heuristic choices. Second, we will investigate the potential of the approach for SMT, both for other theories (e.g. NIA) and for extended functionalities (e.g. interpolation). We will also extend the scope of the approach to deal with transcendental functions, look-up tables, and partially axiomatized functions (e.g. gain functions known to be monotonic and of restricted co-domain).

Finally, we are going to investigate the generalization of the approach from transition systems to continuous-time hybrid systems with nonlinear characteristic functions.

References

1. Ábrahám, E., Corzilius, F., Loup, U., Sturm, T.: A lazy SMT-solver for a nonlinear subset of real algebra. In: Dagstuhl Seminar Proceedings. Schloss Dagstuhl-Leibniz-Zentrum f A1/4r Informatik (2010)
2. Bak, S., Bogomolov, S., Johnson, T.T.: HYST: a source transformation and translation tool for hybrid automaton models. In: Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control, pp. 128–133. ACM (2015)
3. Biere, A., Cimatti, A., Clarke, E., Zhu, Y.: Symbolic model checking without BDDs. In: Cleaveland, W.R. (ed.) TACAS 1999. LNCS, vol. 1579, pp. 193–207. Springer, Heidelberg (1999). doi:10.1007/3-540-49059-0_14
4. Birgmeier, J., Bradley, A.R., Weissenbacher, G.: Counterexample to induction-guided abstraction-refinement (CTIGAR). In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 831–848. Springer, Cham (2014). doi:10.1007/978-3-319-08867-9_55
5. Brain, M., D'Silva, V., Griggio, A., Haller, L., Kroening, D.: Interpolation-based verification of floating-point programs with abstract CDCL. In: Logozzo, F., Fähndrich, M. (eds.) SAS 2013. LNCS, vol. 7935, pp. 412–432. Springer, Heidelberg (2013). doi:10.1007/978-3-642-38856-9_22
6. Brat, G., Bushnell, D., Davies, M., Giannakopoulou, D., Howar, F., Kahsai, T.: Verifying the safety of a flight-critical system. In: Bjørner, N., de Boer, F. (eds.) FM 2015. LNCS, vol. 9109, pp. 308–324. Springer, Heidelberg (2015). doi:10.1007/978-3-319-19249-9_20
7. Cavada, R., et al.: The NUXMV symbolic model checker. In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 334–342. Springer, Cham (2014). doi:10.1007/978-3-319-08867-9_22

8. Champion, A., Gurfinkel, A., Kahsai, T., Tinelli, C.: CoCoSpec: a mode-aware contract language for reactive systems. In: De Nicola, R., Kühn, E. (eds.) SEFM 2016. LNCS, vol. 9763, pp. 347–366. Springer, Cham (2016). doi:10.1007/978-3-319-41591-8_24

9. Cimatti, A., Griggio, A., Mover, S., Tonetta, S.: IC3 modulo theories via implicit predicate abstraction. In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014. LNCS, vol. 8413, pp. 46–61. Springer, Heidelberg (2014). doi:10.1007/978-3-642-54862-8_4

10. Cimatti, A., Griggio, A., Mover, S., Tonetta, S.: HyComp: an SMT-based model checker for hybrid systems. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 52–67. Springer, Heidelberg (2015). doi:10.1007/978-3-662-46681-0_4

11. Cimatti, A., Mover, S., Tonetta, S.: A quantifier-free SMT encoding of non-linear hybrid automata. In: Formal Methods in Computer-Aided Design (FMCAD), pp. 187–195. IEEE (2012)

12. Collins, G.E.: Quantifier elimination for real closed fields by cylindrical algebraic decomposition-preliminary report. SIGSAM Bull. **8**(3), 80–90 (1974). http://doi.acm.org/10.1145/1086837.1086852

13. Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). doi:10.1007/978-3-540-78800-3_24

14. Dutertre, B.: Yices 2.2. In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 737–744. Springer, Heidelberg (2014). doi:10.1007/978-3-319-08867-9_49

15. Eén, N., Sörensson, N.: Temporal induction by incremental SAT solving. Electron. Notes Theor. Comput. Sci. **89**(4), 543–560 (2003)

16. Gao, S., Kong, S., Clarke, E.M.: dReal: an SMT solver for nonlinear theories over the reals. In: Bonacina, M.P. (ed.) CADE 2013. LNCS (LNAI), vol. 7898, pp. 208–214. Springer, Heidelberg (2013). doi:10.1007/978-3-642-38574-2_14

17. Gario, M., Micheli, A.: PySMT: a solver-agnostic library for fast prototyping of SMT-based algorithms. In: Proceedings of the 13th International Workshop on Satisfiability Modulo Theories (SMT), pp. 373–384 (2015)

18. Hoder, K., Bjørner, N.: Generalized property directed reachability. In: Cimatti, A., Sebastiani, R. (eds.) SAT 2012. LNCS, vol. 7317, pp. 157–171. Springer, Heidelberg (2012). doi:10.1007/978-3-642-31612-8_13

19. Hueschen, R.M.: Development of the Transport Class Model (TCM) aircraft simulation from a sub-scale Generic Transport Model (GTM) simulation. Technical report, NASA Langley Research Center (2011)

20. Jovanović, D., Moura, L.: Solving non-linear arithmetic. In: Gramlich, B., Miller, D., Sattler, U. (eds.) IJCAR 2012. LNCS (LNAI), vol. 7364, pp. 339–354. Springer, Heidelberg (2012). doi:10.1007/978-3-642-31365-3_27

21. Komuravelli, A., Gurfinkel, A., Chaki, S.: SMT-based model checking for recursive programs. Form. Methods Syst. Des. **48**(3), 175–205 (2016)

22. Kong, S., Gao, S., Chen, W., Clarke, E.: dReach: δ-reachability analysis for hybrid systems. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 200–205. Springer, Heidelberg (2015). doi:10.1007/978-3-662-46681-0_15

23. Kupferschmid, S., Becker, B.: Craig interpolation in the presence of non-linear constraints. In: Fahrenberg, U., Tripakis, S. (eds.) FORMATS 2011. LNCS, vol. 6919, pp. 240–255. Springer, Heidelberg (2011). doi:10.1007/978-3-642-24310-3_17

24. Mahdi, A., Scheibler, K., Neubauer, F., Fränzle, M., Becker, B.: Advancing software model checking beyond linear arithmetic theories. In: Bloem, R., Arbel, E. (eds.) HVC 2016. LNCS, vol. 10028, pp. 186–201. Springer, Heidelberg (2016). doi:10.1007/978-3-319-49052-6_12

25. Maréchal, A., Fouilhé, A., King, T., Monniaux, D., Périn, M.: Polyhedral approximation of multivariate polynomials using Handelman's theorem. In: Jobstmann, B., Leino, K.R.M. (eds.) VMCAI 2016. LNCS, vol. 9583, pp. 166–184. Springer, Heidelberg (2016). doi:10.1007/978-3-662-49122-5_8

26. McMillan, K.L.: Interpolation and SAT-based model checking. In: Hunt, W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 1–13. Springer, Heidelberg (2003). doi:10.1007/978-3-540-45069-6_1

27. Nuzzo, P., Puggelli, A., Seshia, S.A., Sangiovanni-Vincentelli, A.: CalCS: SMT solving for non-linear convex constraints. In: Proceedings of the 2010 Conference on Formal Methods in Computer-Aided Design, pp. 71–80. FMCAD Inc. (2010)

28. Scheibler, K., Kupferschmid, S., Becker, B.: Recent improvements in the SMT solver iSAT. MBMV **13**, 231–241 (2013)

29. Sheeran, M., Singh, S., Stålmarck, G.: Checking safety properties using induction and a SAT-solver. In: Hunt, W.A., Johnson, S.D. (eds.) FMCAD 2000. LNCS, vol. 1954, pp. 127–144. Springer, Heidelberg (2000). doi:10.1007/3-540-40922-X_8

30. Tiwari, A.: Time-aware abstractions in HybridSal. In: Kroening, D., Păsăreanu, C.S. (eds.) CAV 2015. LNCS, vol. 9206, pp. 504–510. Springer, Heidelberg (2015). doi:10.1007/978-3-319-21690-4_34