A CEGAR-based Approach for Proving Invariant Properties of Transition Systems on Non-Linear Real Arithmetic

(Extended Abstract)

Alessandro Cimatti*, Ahmed Irfan*†, Alberto Griggio*, Marco Roveri* and Roberto Sebastiani†
*Fondazione Bruno Kessler, Italy
Email: [lastname]@fbk.eu
†University of Trento, Italy
Email: [firstname].[lastname]@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 work-in-progress 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).**

I. Introduction

Typical real-world industrial designs, for instance from the aerospace and automotive domains, often involve non-linear dynamics. Due to the safety-critical nature, their formal verification is of great importance. A prominent method is to model them as transition systems on non-linear arithmetic (NRA) and prove properties by applying model checking techniques. Unlike transition systems on linear real arithmetic (LRA), proving properties of the transition systems on NRA has received much less attention. In this paper, we focus on proving invariant properties of such systems that involve non-linear polynomial constraints.

Usual approaches for proving invariant properties of transition systems on NRA handle non-linearity at the SMT-level – by using a non-linear SMT solver like Z3 [1], nlSAT [2], CVC4 [3], Yices [4], SMT-RAT [5]. We argue such techniques are not mature, most of them extend the ideas of BMC and k-induction, whereas exploring more powerful and efficient proving techniques is an active research area.

Consider the following simple transition system: in the initial state we have $x \geq 2 \wedge y \geq 2 \wedge z = x * y$ and in the transition relation we have $x' = x+1 \wedge y' = y+1 \wedge z' = x' * y'$. Suppose that we are interested in checking the property: "*it is always the case that* $z \geq x + y$". Notice that the property is not k-inductive, not even for a very large value of k. Thus the typical proving techniques that are based on k-induction algorithm [6], [7] using non-linear SMT solver, will not be able to prove this property. The more powerful techniques [8], [9] require some extra functionalities like interpolants, quantifier elimination from the non-linear SMT solver. These extra functionalities are usually not available or they have a very high computational cost. Note that the more efficient non-linear SMT solvers like iSAT [10], dreal [11] that based on the delta-complete [12] idea, will also fail. The reason for their failure is that these solvers require that upper and lower bounds for each real variable are given, which is not the case in this example.

In contrast to the usual approaches, we propose to handle non-linearity at the model checking level, by using linearization techniques from differential calculus. Our ideas are motivated by the fact that NRA is a hard-to-solve theory, and for proving properties of a transition system on NRA may not require full power of non-linear solving. One such case is when there exists a piecewise linear invariant of the transition system that is strong enough to prove the property. Another case is when a system is "mostly-linear" and non-linear constraints are very small part of the system, for instance the simulink model of the Transport Class Model (TCM) for aircraft simulation [13] has this characteristic.

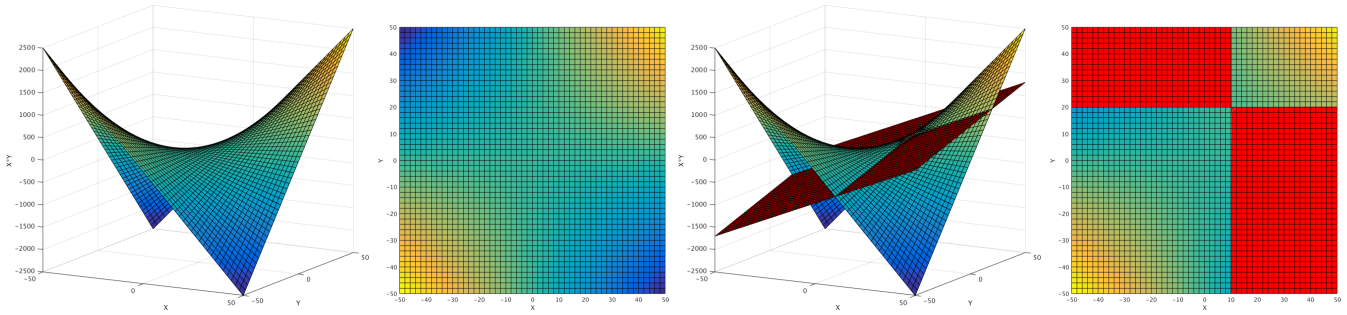
We are working on a CEGAR-based idea: by using uninterpreted functions to abstract the multiplication and the tangent planes to refine the abstraction. We intend to exploit the powerful techniques for transition systems on linear real arithmetic (LRA) and equality with uninterpreted functions (EUF) to model check the abstract system.

The paper is structured as follows: Sec. II gives a brief background about the ideas of differential calculus, transition systems, and CEGAR framework. In Sec. III we present the key ideas of our work. Sec. IV and Sec. V briefly discuss some interesting strategies and examples. We mention some related works in Sec. VI and we conclude in Sec. VII.

II. Background

We assume the standard first-order quantifier-free logical setting and standard notions of theory, satisfiability, and logical consequence. In the rest of the section we introduce some notions that will facilitate in understanding this work.

This work was performed as part of the H2020-FETOPEN-2016-2017-CSA project SC^2 (712689)



(a) $x * y$ (b) $x * y$ (top view) (c) $x * y$ and tangent plane (d) $x * y$ and tangent plane (top view)

Fig. 1. Plots – Multiplication and Tangent Plane.

A. Calculus Notions

We review some of the basic calculus notions. In the following we adopt the following notation. Given a function $f(x_1, ..., x_n) : \mathcal{R} \times ... \times \mathcal{R} \mapsto \mathcal{R}$, we denote with $f_{x_i} \doteq \frac{d}{dx_i} f$ the first-order partial derivative of f w.r.t. variable x_i.

A *Tangent Line* to a uni-variate function $f(x)$ at a point of interest $x = a$ is a straight line that "just touches" the function at the point, and represents the instantaneous rate of change of the function f at that one point. One particular use of the tangent lines is to get linear approximation of a given function.

The tangent line $T_{f(x)}(a)$ to $f(x)$ at point $x = a$ is the straight line in (1).

$$T_{f(x)}(a) \doteq f(a) + f_x(a) * (x - a) \tag{1}$$

The *Secant Line* to a uni-variate function $f(x)$ is a straight line that connects two points on the function plot. The equation for the secant line $S_{f(x)}(a1, a2)$ to a function $f(x)$ between points $x = a_1$ and $x = a_2$ is represented by (2)

$$S_{f(x)}(a_1, a_2) \doteq m_{f(x)}(a_1, a_2)(x - a_1) + f(a_1) \tag{2}$$

where $m_{f(x)}(a_1, a_2) \doteq \frac{f(a_1) - f(a_2)}{a_1 - a_2}$ is the slope of f between the two point a_1 and a_2.

The *Tangent Plane* to a two-variable function $f(x, y)$, essentially a surface, at a point of interest (a, b) is a plane that "just touches" the surface at the point. Similar to the tangent lines, the tangent planes can be used to linearly approximate the surface at the point of interest. We use first-order partial derivatives to calculate the tangent plane $T_{f(x,y)}(a, b)$ to function $f(x, y)$ at point (a, b).

$$\begin{aligned} T_{f(x,y)}(a, b) \doteq f(a, b) + f_x(a, b) * (x - a) \\ + f_y(a, b) * (y - b) \end{aligned} \tag{3}$$

In this work, we are interested in abstracting the multiplication function between two real variables (this will become clear in Sec. III). Geometrically the surface generated by the multiplication function $f(x, y) \doteq x * y$ is shown in Fig. 1a and Fig. 1b. This kind of surface is known in geometry as Hyperbolic Paraboloid. An *Hyperbolic Paraboloid* surface is a doubly-ruled surface, i.e. for every point on the surface, there are two distinct straight lines that lie on the surface such that they pass through the point. Moreover these two lines are also in the tangent plane of the surface at the point, and they define how the plane cuts the surface. Fig. 1c and Fig. 1d show the multiplication surface with tangent plane in a 3-dimension plot and 2-dimension top view, respectively. Using (3), the tangent plane of multiplication $x * y$ at point (a, b) is:

$$Tmul_{x,y}(a, b) \doteq b * x + a * y - a * b \tag{4}$$

B. Symbolic Transition System

A symbolic transition system $\mathcal{S}$ is a tuple $\mathcal{S} = \langle X, I, T \rangle$ where X is a finite set of state variables, a state s_i of $\mathcal{S}$ is an assignment to the variables X, $I(X)$ is a formula whose assignments represent the set of initial states, and $T(X, X')$ is a formula whose assignments represent the transitions (we use X' to denote the set of next state variables). A path (execution trace) $\pi = s_0, s_1, s_2, \ldots$ for $\mathcal{S}$ is such that $s_0 \models I(X_0)$ and $s_i \wedge s_{i+1} \models T(X_i, X_{i+1})$ for all $i \geq 0$, where X_0, X_i and X_{i+1} are renamings of the variables representing the state variables at time 0, i and $i + 1$, respectively.

Let $P(X)$ be a formula whose assignments represent a property (good states) over the state variables X. The model checking problem $\mathcal{S} \models \mathbf{G}P$ is the problem of deciding that for all the paths $\pi = s_0, s_1, \ldots$ of $\mathcal{S}$, the following holds: $\forall i, 0 \leq i, s_i \models P$. The dual is the reachability problem, which is to check whether bad states ($\neg P(X)$) are reachable in $\mathcal{S}$, i.e., there exists a path $\pi = s_0, s_1, \ldots, s_k$ of $\mathcal{S}$ such that $s_k \models \neg P(X)$ and $s_i \models P(X)$ for $0 \leq i < k$.

There are several algorithms to solve the model checking problem. We have bounded model checking technique (BMC) [14] (useful for disproving properties), and to prove properties we can leverage on k-induction [6], on interpolation-based [8], or on IC3 with implicit abstraction [9] techniques. These last techniques can either prove the property holds, or generate a counterexample. We briefly summarize here the incremental formulation of the BMC technique, since we will be using it later. The incremental BMC technique checks if the transition system violates a property P or not in exactly k steps, by checking the satisfiability of the following formula:

```
CEGAR(I,T,P):
[Î,T̂,P̂] := InitialAbstraction(I,T,P);
while True:
  [result,π̂_k] := ModelCheck(Î,T̂,P̂);
  if result = SAFE:
    return SAFE;
  if result = UNSAFE:
    [res,π_k] := CheckSpuriousness(Î,T̂,P̂,π̂_k);
    if res = SPURIOUS:
      [Î,T̂,P̂] := Refine(Î,T̂,P̂,π̂_k);
    else
      return [UNSAFE, π_k];
```

Fig. 2. CEGAR Procedure

$$BMC_k(I,T,P) \doteq \tag{5}$$
$$I(X_0) \wedge \bigwedge_{i=0}^{k-1}(T(X_i,X_{i+1}) \wedge P(X_i)) \wedge \neg P(X_k)$$

where X_i for $0 \leq i \leq k$ are renamings of the variables representing the state variables at time i. If the property is violated, then the above formula will be satisfiable, and we will have a counterexample path π_k such that:

$$\pi_k \models BMC_k(I,T,P) \tag{6}$$

If the formula is unsatisfiable, then there is no counterexample path of length k, but there might be one for a length $k+1$ and we check that by incrementing the value of k. We denote with $\pi_k[t_i]$ the interpretation of a term t_i in $BMC_k(I,T,P)$ on the counterexample (path) π_k.

C. The CEGAR Framework

Counterexample guided abstraction refinement (CEGAR) [15], in the context of model checking, is an iterative framework consisting of three main phases: initial abstraction, model checking, and refinement. Fig. 2 shows the generic procedure of the CEGAR framework.

Typically the CEGAR loop starts with a very coarse initial abstraction (InitialAbstraction) of the system and applies some model checking procedure (ModelCheck) on the abstract system. If the result of model checking procedure says that the property holds in the abstract system, then it also holds in the concrete system and we return SAFE. If the property fails in the abstract system, then one has to check whether the abstract counterexample $\hat{\pi}_k$ can be concretized into a concrete counterexample π_k or not (CheckSpuriousness). If there exists a concrete counterexample π_k refinement of $\hat{\pi}_k$, then the property P is violated in the original transition system and we can return UNSAFE and the computed concrete counterexample. Otherwise, the abstract counterexample $\hat{\pi}_k$ is spurious. The spurious counterexample is used to refine the abstract transition system (Refine) in such a way that the counterexample $\hat{\pi}_k$ does not hold anymore in the abstract transition system.

III. CEGAR for NRA

To simplify the presentation we assume the transition system contains only non-linear multiplication constraints (with no division), and we assume the multiplication terms occurs only between two variables. This is not a limitation, indeed, it is possible to normalize any transition system $[\hat{I},\hat{T},\hat{P}]$ to fulfill these constraint by using the following normalization rules, until no more rewriting is possible.

1) *Non-linear division as multiplication in a formula.* This can be done by replacing the term of form $\frac{x}{y}$ by a fresh variable z and conjoining the constraint $y \neq 0 \rightarrow x = y * z$ to the formula.
2) *Non-linear multiplication occurs between two variables in a formula.* This can be done by introducing fresh variables to convert a multiplication expression with generic term into a multiplication expression with only two variables, e.g. $fmul(t_i,t_j)$ can be rewritten as $fmul(u,v)$ where u,v are fresh variables and $u = t_i \wedge v = t_j$ is conjoined to the formula.

In what follows we describe the different phases of the proposed CEGAR loop for the verification of transition systems with NRA constraints.

A. Initial Abstraction

The initial abstraction $[\hat{I},\hat{T},\hat{P}]$ is obtained by replacing each multiplication expressions $x * y$ with a fresh uninterpreted function expression $fmul(x,y)$ in $[I,T,P]$. For each term $fmul(x,y)$ occurring in $[\hat{I},\hat{T},\hat{P}]$, we also conjoin the equality constraint $fmul(x,y) = fmul(y,x)$ to $\hat{I}$, or $\hat{T}$ or $\hat{P}$, depending on where $fmul(x,y)$ appears.

We define the set of the $fmul()$ arguments as:

$$FmulArgs(\hat{I},\hat{T},\hat{P}) \doteq \tag{7}$$
$$\{(x,y) \mid fmul(x,y) \text{ appear in } \hat{I} \text{ or in } \hat{T} \text{ or in } \hat{P}\}$$

B. Model Checking

For the model checking we rely on any of the SMT-based model checking procedures for the theories of linear real arithmetic (LRA) and uninterpreted functions (UF), e.g. k-induction [6], interpolation-based [8], IC3 with implicit abstraction [9]. These techniques can either prove that the property holds, or generate a counterexample.

C. Counterexample Checking and Refinement

The counterexample checking and refinement phase performs two kinds of actions: the monotonicity and the multiplication check&refinement. Both produce a tuple of refinement formulas $[\hat{I}_{ref},\hat{T}_{ref}]$ – initial and transition respectively. These refinement formulas are conjoined to the abstraction.

Suppose that we have an abstract counterexample $\hat{\pi}_k$ of length k. We know from (6) that $\hat{\pi}_k \models BMC_k(\hat{I},\hat{T},\hat{P})$. Then the counterexample is spurious if the monotonicity spuriousness or the multiplication spuriousness checks return true.

The *monotonicity spuriousness check* returns true if there exists at least two terms $fmul(x_i, y_j)$ and $fmul(u_l, v_m)$ in $BMC_k(\hat{I}, \hat{T}, \hat{P})$ such that

$$\begin{vmatrix}\hat{\pi}_k[x_i]\end{vmatrix} \leq \begin{vmatrix}\hat{\pi}_k[u_l]\end{vmatrix} \ \wedge \ \begin{vmatrix}\hat{\pi}_k[y_j]\end{vmatrix} \leq \begin{vmatrix}\hat{\pi}_k[v_m]\end{vmatrix} \ \wedge \\ \begin{vmatrix}\hat{\pi}_k[fmul(x_i, y_j)]\end{vmatrix} > \begin{vmatrix}\hat{\pi}_k[fmul(u_l, v_m)]\end{vmatrix}.$$

If this is the case, we call the pair $(fmul(x_i, y_j), fmul(u_l, v_m))$ a *monotonicity conflict*.

The *multiplication spuriousness check* returns true if there exists at least one term $fmul(x_i, y_j)$ in $BMC_k(\hat{I}, \hat{T}, \hat{P})$ such that

$$\hat{\pi}_k[fmul(x_i, y_j)] \neq \hat{\pi}_k[x_i] * \hat{\pi}_k[x_j].$$

If this is the case, then we call $fmul(x_i, y_j)$ a *multiplication conflict*.

Let $FmonoConfSet(\hat{I}, \hat{T}, \hat{P}, \hat{\pi}_k)$ be the set of monotonicity conflicts. The *monotonicity refinement* will return the tuple $[\hat{I}_{ref}, \hat{T}_{ref}]$ that basically contains the multiplication monotonicity lemmas produced for a subset of $FmonoConfSet(\hat{I}, \hat{T}, \hat{P}, \hat{\pi}_k)$. Given a monotonicity conflict $(fmul(x_i, y_j), fmul(u_l, v_m))$, and the spurious counterexample $\hat{\pi}_k$, where (x_i, y_j) and (u_l, v_m) are essentially BMC renaming of (x, y) and (u, v) from the set $FmulArgs(\hat{I}, \hat{T}, \hat{P})$, then the monotonicity lemma is given by:

$$\begin{aligned}&(abs(x) \leq abs(u) \wedge abs(y) \leq abs(v)) \\ &\quad \rightarrow abs(fmul(x, y)) \leq abs(fmul(u, v))\end{aligned} \tag{8}$$

where $abs()$ function is a shorthand for encoding the absolute value of a term, i.e. $abs(t) \doteq IfThenElse(t \geq 0, t, -t)$. The monotonicity lemma is basically stating a weaker form of the property that the absolute multiplication is monotonic. The monotonicity lemma is part of $\hat{I}_{ref}$, if $i = j = l = m = 0$, otherwise it is part of $\hat{T}_{ref}$.

Let $FmulConfSet(\hat{I}, \hat{T}, \hat{P}, \hat{\pi}_k)$ be the set of multiplication conflicts. Then the *multiplication refinement* will use a subset of $FmulConfSet(\hat{I}, \hat{T}, \hat{P}, \hat{\pi}_k)$ to compute tangents lemmas for producing $[\hat{I}_{ref}, \hat{T}_{ref}]$, such that the spurious counterexample will not hold in the refined abstract system. Given a multiplication conflict $fmul(x_i, y_j)$, essentially (x_i, y_j) is a BMC renaming of $(x, y) \in FmulArgs(\hat{I}, \hat{T}, \hat{P})$, and the spurious counterexample $\hat{\pi}_k$, then the tangent lemma is:

$$\begin{aligned}&((x \geq a \wedge y \leq b) \rightarrow fmul(x, y) \leq Tmul_{x,y}(a, b)) \wedge \\ &((x \leq a \wedge y \geq b) \rightarrow fmul(x, y) \leq Tmul_{x,y}(a, b)) \wedge \\ &((x \leq a \wedge y \leq b) \rightarrow fmul(x, y) \geq Tmul_{x,y}(a, b)) \wedge \\ &((x \geq a \wedge y \geq b) \rightarrow fmul(x, y) \geq Tmul_{x,y}(a, b))\end{aligned} \tag{9}$$

where we can choose a and b as:

$$a := \hat{\pi}_k[x_i] \text{ and } b := \hat{\pi}_k[y_j] \tag{10}$$

$$a := \frac{1}{\hat{\pi}_k[fmul(x_i, y_j)]} \text{ and } b := \hat{\pi}_k[y_j] \tag{11}$$

$$a := \hat{\pi}_k[x_i] \text{ and } b := \frac{1}{\hat{\pi}_k[fmul(x_i, y_j)]} \tag{12}$$

The tangent lemma is part of $\hat{I}_{ref}$, if $i = j = 0$, otherwise it is part of $\hat{T}_{ref}$. The predicates in the tangent lemmas are extracted using the fact that the $x * y$ multiplication is a hyperbolic paraboloid surface and a tangent plane to such surface cuts the surface into four regions – 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, as shown in Fig. 1c and Fig. 1d. Note that the tangent lemma is taking into account two among the possible three values (shown in (10), (11), (12)). To get more or different tangent lemmas, we can use all or different values pairs for calculating the tangent lemma. Note that we can use (11) and (12) only when $\hat{\pi}_k[fmul(x_i, y_j)] \neq 0$.

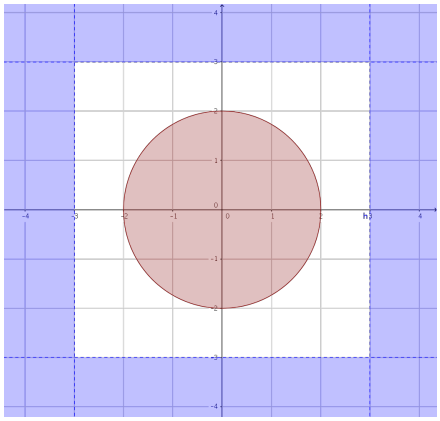


Fig. 3. Example where tangent lemmas are enough

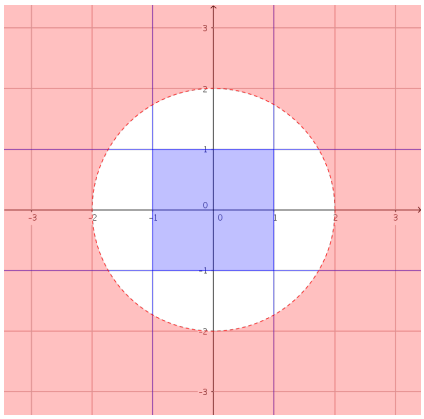


Fig. 4. Example where tangent lemmas are not enough

We have a special case for the tangent lemma when the multiplication function has the same arguments (basically a square function), e.g. $fmul(x, x)$. In that case, the tangent lemma for the multiplication conflict $fmul(x_i, x_i)$, after simplification becomes:

$$f(a, a) \leq a^2 \wedge f(x, x) \geq Tmul_{x,x}(a, a) \tag{13}$$

where we get $Tmul_{x,x}(a, a) \doteq 2ax + a^2$ after simplifying (4). We can get the same result by using tangent line for the univariate square function – see (1).

Note that the tangent lemma for $x * y$ provides lower bounds in two regions and upper bounds in other two regions. Similarly the tangent lemma for x^2 provides a generic lower

bound for the function, however the upper bound is point-based. In certain cases, these partial bounds are not enough. For instance, consider the following examples:

A) $I := x^2 + y^2 \leq 2$ and $T := x' = x \wedge y' = y$ and $P := -3 \leq x \leq 3 \wedge -3 \leq y \leq 3$;
B) $I := -1 \leq x \leq 1 \wedge -1 \leq y \leq 1$ and $T := x' = x \wedge y' = y$ and $P := x^2 + y^2 \leq 4$.

In example A) we are able to prove the property using lower bounds provided by the tangent planes. Fig. 3 shows the reachable states inside the circle and the bad states as the outer part of the square. Basically we are able to put a separation between the circle and bad states using the lower bounds provided by the tangent lemmas.

In example B) instead lower bounds provided by the tangent lemmas are not enough to prove the property. Fig. 4 shows the reachable states inside the square and the bad states as the outer part of the circle. Actually we require upper bounds, however the upper bounds by tangent lemmas are point-based and we will require infinitely many point-based upper bounds to prove the property.

Here we give an idea how to solve this problem of upper bounds in the case when we have square function: we can lift the same idea (with some modifications) to the case of $x * y$ multiplication function. Basically we use secant lines to get better interval-based upper bounds. Using (2), we have the following lemmas depending on the choice of c:

- when $a \leq c$

$$(x \geq a \wedge x \leq c) \rightarrow fmul(x, x) \leq S_{x*x}(a, c) \quad (14)$$

- when $a > c$

$$(x \geq c \wedge x \leq a) \rightarrow fmul(x, x) \leq S_{x*x}(a, c) \quad (15)$$

where

$$a := \hat{\pi}_k[x_i] \quad (16)$$
$$c := \pm\sqrt{\hat{\pi}_k[fmul(x_i, x_i)]} \quad (17)$$

Fig. 5 shows a plot of the square function with tangent and secant lines. Using these extra lemmas we are able to prove the example B) as shown in Fig. 4. The idea of secant lines can be lifted to some sort of secant planes to the surface of the multiplication function $x * y$. We are aware that it will not be straight forward, since the cut generated by the (secant) plane is likely to be non-linear, i.e. a curve which we can linearize by using tangent line.

IV. Strategies

We have seen in Sec. III, that the refinement phase performs two different kind of refinements. Moreover, for each kind there are different choices in terms of producing refinements. These choices may contribute to the search strategy of the CEGAR loop. In this paper, we are not going into the details about how to combine the two refinements and what different choices to make; rather, we provide some brief ideas.

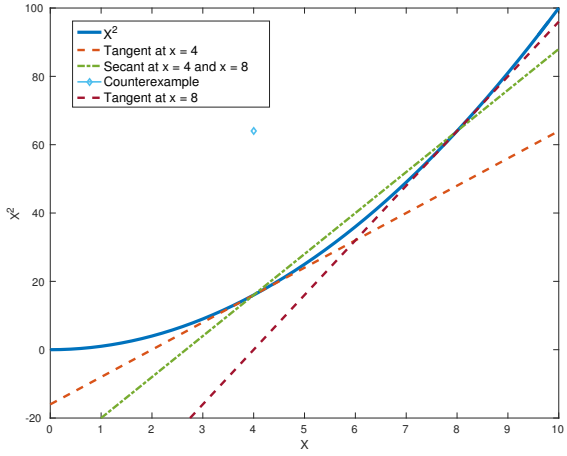


Fig. 5. Square function plot with tangent and secant lines

A. CEGAR Structure

We can think of having one CEGAR loop with different priorities to the monotonicity refinement and the multiplication refinement. Another possibility is to have a two-level nested CEGAR loop where one level corresponds to the monotonicity refinement and the other corresponds to the multiplication refinement.

B. Eager vs Lazy Strategy

The eager strategy focuses on learning lemmas for all the conflicts that are encountered during the refinement phase. This basically means that we try to add monotonicity lemmas and tangent lemmas for every conflict in the $FmonoConfSet$ and $FmulConfSet$ sets. This also suggests to add all the three tangent lemmas for the different choices of the parameters a and b in (9). The lazy strategy, instead, tries to learn enough lemmas during the refinement phase, such that the spurious counterexample will not hold in the abstract system. This means that we add monotonicity lemmas and tangent lemmas for a subset of the $FmonoConfSet$ and $FmulConfSet$ sets. This approach has also to deal with the choice of the parameters a and b in (9).

V. Proof of Concept

As proof of concept, we manually simulated the CEGAR-based approach discussed in this paper on the example presented in Sec. I and on some other paradigmatic examples. We leverage on the nuXmv model checker [16] for performing the ModelCheck step on the abstract model.

A. Example 1

This is the example presented in Sec. I.
Original Problem:
$I := x \geq 2 \wedge y \geq 2 \wedge z = x * y$
$T := x' = x + 1 \wedge y' = y + 1 \wedge z' = x' * y'$
$P := z \geq x + y$
Initial Abstraction:
$\hat{I} := x \geq 2 \wedge y \geq 2 \wedge z = fmul(x, y)$

$\hat{T} := x' = x + 1 \wedge y' = y + 1 \wedge z' = fmul(x', y')$
$\hat{P} := z \geq x + y$
We model check and get the following abstraction counterexample of length 1:

$$0: \; fmul(4, 2) = 5; fmul(0, 0) = 0; x = 4; y = 2; z = 5.$$

We notice that the abstract counterexample is spurious, since $4*2 \neq 5$. Therefore we eagerly (as discussed in Sec. IV) refine with the three tangent lemmas using (9), (10), (11), (12). We model check again and this time we get the following abstract counterexample of length 1:

$$0: \; fmul(2, 4) = 5; fmul(0, 0) = 0; x = 2; y = 4; z = 5.$$

This counterexample is also spurious as $2 * 4 \neq 5$. We again eagerly refine using the three tangent lemmas and model check again. This time we are able to prove the property.

B. Example 2

Original Problem:
$I := x \geq 0 \wedge y \geq 0 \wedge u > x \wedge v > y$
$T := x' = x + 1 \wedge y' = y + 1 \wedge u' = u + 1 \wedge v' = v + 1$
$P := x * y \leq u * v$
Initial Abstraction:
$\hat{I} := x \geq 0 \wedge y \geq 0 \wedge u > x \wedge v > y$
$\hat{T} := x' = x + 1 \wedge y' = y + 1 \wedge u' = u + 1 \wedge v' = v + 1$
$\hat{P} := fmul(x, y) \leq fmul(u, v)$
We model check with the IC3 with implicit abstraction technique and we get a following abstraction counterexample of length 1:

$$0: \; fmul(\tfrac{5}{2}, \tfrac{7}{2}) = -\tfrac{3}{2}; fmul(2, 3) = -1; fmul(0, 0) = 0; \\ x = 2; y = 3; u = \tfrac{5}{2}; v = \tfrac{7}{2};$$

We notice that the abstract counterexample is violating the monontocity property and is spurious. Therefore we refine the abstraction with the monotonicity lemma using (8). Model checking the refined abstract system shows that the property holds.

C. Example 3

This example is similar to the example presented in Sec. III and shown in Fig. 3.
Original Problem:
$I := x^2 + y^2 \leq 1$
$T := x' = x \wedge y' = y$
$P := -10 \leq x \leq 10 \wedge -10 \leq y \leq 10;$
Initial Abstraction:
$\hat{I} := fmul(x, x) + fmul(y, y) \leq 1$
$\hat{T} := x' = x \wedge y' = y$
$\hat{P} := -10 \leq x \leq 10 \wedge -10 \leq y \leq 10;$
First model checking step gives the following abstract counterexample of length 1:

$$0: \; fmul(-11, -11) = 13; fmul(-12, -12) = -13; \\ x = -12; y = -11$$

This counterexample is voilating monotonicity property. Thus we do the monotonicity refinement. Model checking again gives the following abstract counterexample of length 1:

$$0: \; fmul(-12, -12) = -2; fmul(-11, -11) = -1; \\ x = -12; y = -11$$

The counterexample is spurious since $-12 * -12 \neq -2$ and $-11 * -11 \neq -1$. We perform the multiplication refinement and check again. This time we get the following abstract counterexample of length 1:

$$0: \; fmul(-\tfrac{312}{25}, -\tfrac{312}{25}) = \tfrac{3888}{25}; \\ fmul(12, 12) = -\tfrac{3863}{25}; fmul(0, 0) = 0; x = 12; \\ y = -\tfrac{312}{25}$$

Again the counterexample is spurious since $-\frac{312}{25} * -\frac{312}{25} \neq \frac{3888}{25}$ and $12*12 \neq -\frac{3863}{25}$. We do the multiplication refinement and model check again. Now we get the following abstract counterexample of length 1:

$$0: \; fmul(11, 11) = \tfrac{169}{3}; fmul(\tfrac{133}{36}, \tfrac{133}{36}) = -\tfrac{166}{3}; \\ fmul(0, 0) = 0; x = \tfrac{133}{36}; y = 11$$

Once more the counterexample is spurious and we use the multiplication tangent lemmas to do the refinement. Model checking again shows that the property holds in the refined system.

VI. Related Works

The closest to our approach is the polyhedral approximation of polynomials [17] in the context of program analysis. However our approach is iterative, whereas the approach in [17] is not – basically it tries to find a tight convex polyhedra approximation in the beginning and then tries to prove the property.

In [18], the authors use interpolation-based [8] approach to prove invariant properties of transition systems on NRA, by using interpolants produced by iSAT SMT solver which uses the semantics of delta-complete satisfiability. Note that iSAT is sound modulo some precision error and thus also are the techniques that use iSAT.

Similar to our approach, [19] abstracts the multiplication and the division functions using uninterpreted functions. However in their approach abstraction is not iterative and is very coarse since it encodes some weak properties of the multiplication like identity and sign. In contrast to [19] our approach is based on CEGAR and it is refined on demand.

The idea of using tangent planes (spaces) for approximation has also been used in [20], however in a different setting as they use them to under-approximate predicates for SMT solving. In contrast we use them to over-approximate the multiplication function in a CEGAR-based approach to solve model checking problem.

The idea of approximating an univariate function, in particular natural log ln, with the tangent lines has been also used in [21]. Similar to that work we use the tangent lines to approximate univariate function, however the difference is we

use them to approximate the square function. In addition to the tangent lines we also use the secant lines for the square function approximation. Moreover, in our work we use tangent planes to approximate the multiplication function.

VII. Conclusion

We presented a promising CEGAR-based approach for proving invariant properties of transition systems on NRA. Our next step is to implement it in nuXmv [16] and perform a thorough experimental evaluation on real-world benchmarks. Our approach is driven towards proving properties and it is incomplete. In the future we would like to work on the completeness of the approach.

References

[1] L. De Moura and N. Bjørner, "Z3: An efficient smt solver," in *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.

[2] D. Jovanović and L. De Moura, "Solving non-linear arithmetic," in *International Joint Conference on Automated Reasoning*. Springer, 2012, pp. 339–354.

[3] C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanovic, T. King, A. Reynolds, and C. Tinelli, "CVC4," in *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, 2011, pp. 171–177.

[4] B. Dutertre, "Yices 2.2," in *Computer-Aided Verification (CAV'2014)*, ser. Lecture Notes in Computer Science, A. Biere and R. Bloem, Eds., vol. 8559. Springer, July 2014, pp. 737–744.

[5] E. Ábrahám, F. Corzilius, U. Loup, and T. Sturm, "A lazy smt-solver for a non-linear subset of real algebra," in *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum f A1/4r Informatik, 2010.

[6] M. Sheeran, S. Singh, and G. Stålmarck, "Checking safety properties using induction and a sat-solver," in *Formal Methods in Computer-Aided Design*. Springer, 2000, pp. 127–144.

[7] N. Eén and N. Sörensson, "Temporal induction by incremental sat solving," *Electronic Notes in Theoretical Computer Science*, vol. 89, no. 4, pp. 543–560, 2003.

[8] K. L. McMillan, "Interpolation and sat-based model checking," in *Computer Aided Verification*. Springer, 2003, pp. 1–13.

[9] A. Cimatti, A. Griggio, S. Mover, and S. Tonetta, "IC3 modulo theories via implicit predicate abstraction," in *TACAS*, 2014.

[10] M. Franzle, C. Herde, T. Teige, S. Ratschan, and T. Schubert, "Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 1, pp. 209–236, 2007.

[11] S. Gao, S. Kong, and E. M. Clarke, "dreal: An smt solver for nonlinear theories over the reals," in *International Conference on Automated Deduction*. Springer, 2013, pp. 208–214.

[12] S. Gao, J. Avigad, and E. M. Clarke, "δ-complete decision procedures for satisfiability over the reals," in *International Joint Conference on Automated Reasoning*. Springer, 2012, pp. 286–300.

[13] R. M. Hueschen, "Development of the transport class model (tcm) aircraft simulation from a sub-scale generic transport model (gtm) simulation," 2011.

[14] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu, "Symbolic model checking without bdds," in *TACAS*. London, UK: Springer-Verlag, 1999, pp. 193–207.

[15] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Counterexample-guided abstraction refinement," in *CAV*. London, UK: Springer-Verlag, 2000, pp. 154–169.

[16] R. Cavada, A. Cimatti, M. Dorigatti, A. Griggio, A. Mariotti, A. Micheli, S. Mover, M. Roveri, and S. Tonetta, "The nuXmv symbolic model checker," in *CAV*. Springer, 2014.

[17] A. Maréchal, A. Fouilhé, T. King, D. Monniaux, and M. Périn, "Polyhedral approximation of multivariate polynomials using handelmans theorem," in *International Conference on Verification, Model Checking, and Abstract Interpretation*. Springer, 2016, pp. 166–184.

[18] S. Kupferschmid and B. Becker, "Craig interpolation in the presence of non-linear constraints," in *International Conference on Formal Modeling and Analysis of Timed Systems*. Springer, 2011, pp. 240–255.

[19] A. Champion, A. Gurfinkel, T. Kahsai, and C. Tinelli, "Cocospec: A mode-aware contract language for reactive systems," in *Software Engineering and Formal Methods - 14th International Conference, SEFM 2016, Held as Part of STAF 2016, Vienna, Austria, July 4-8, 2016, Proceedings*, 2016, pp. 347–366.

[20] P. Nuzzo, A. Puggelli, S. A. Seshia, and A. Sangiovanni-Vincentelli, "Calcs: Smt solving for non-linear convex constraints," in *Proceedings of the 2010 Conference on Formal Methods in Computer-Aided Design*. FMCAD Inc, 2010, pp. 71–80.

[21] A. Tiwari, "Time-aware abstractions in hybridsal," in *International Conference on Computer Aided Verification*. Springer, 2015, pp. 504–510.