Reachability Analysis of Nonlinear Analog Circuits through Iterative Reachable Set Reduction

Seyed Nematollah Ahmadyan, Shobha Vasudevan
Coordinated Science Lab, Electrical and Computer Engineering Department,
University of Illinois at Urbana-Champaign,
{ahmadya2, shohbav}@illinois.edu

***Abstract*—We propose a methodology for reachability analysis of nonlinear analog circuits to verify safety properties. Our iterative reachable set reduction algorithm initially considers the entire state space as reachable. Our algorithm iteratively determines which regions in the state space are unreachable and removes those unreachable regions from the over approximated reachable set. We use the State Partitioning Tree (SPT) algorithm to recursively partition the reachable set into convex polytopes. We determine the reachability of adjacent neighbor polytopes by analyzing the direction of state space trajectories at the common faces between two adjacent polytopes. We model the direction of the trajectories as a reachability decision function that we solve using a sound root counting method. We are faithful to the nonlinearities of the system. We demonstrate the memory efficiency of our algorithm through computation of the reachable set of Van der Pol oscillation circuit.**

I. Introduction

Formal verification of analog circuits is a lofty, but highly desirable goal. Some strides have been taken in analog verification research. However, since many of these techniques linearize and discretize analog circuit behavior, their practical applicability remains limited. A major challenge in formal analog verification is proving the safety properties of the system. Safety is an indication that the systems operation would always remain inside the safe regions within the state space.

Reachability analysis is a solution to the safety verification problem. Reachability analysis focuses on computing the *reachable set* of the system. The reachable set is the union of all possible trajectories generated by the system from every initial state for all input signals. To prove safety, we must show that the reachable set of the system does not intersect with any unsafe set. Generally, computing the reachable set of the nonlinear analog circuit is computationally undecidable [2]. Over the last decade, many researchers have been investigating the reachability problem [17], [1], [6], [4], [7], [13]. A common problem in previous works toward reachability analysis is memory explosion due to the inefficiency of the data structure involved in modeling the state space [6]. Importantly, these methods do not directly handle nonlinear systems, but use linearization or interval arithmetic to model nonlinearities. Both these modeling techniques result in introduction of large and often unrealistic approximation errors.

Although computation of the exact reachable set is undecidable [2], it is possible to prove the safety of a system by computing an over approximation of the reachable set [1]. Therefore, in a safe system, there is a feasible trajectory from the initial set of states to an erroneous or undesirable set of states (specified by the user). If the over approximated reachable set is safe, we can conclude that the exact reachable set is safe as well. However, if the over approximated reachable set intersects with the unsafe regions, we cannot determine the safety of the system. Over approximation introduces its own errors to the analysis. Hence, minimizing the approximation error while maintaining computational efficiency is a challenge.

In this paper, we propose a methodology for reachability analysis of nonlinear analog circuits. Our method reduces the approximation error and is computationally efficient. Our technique can be applied to general nonlinear systems while providing a precise analysis for handling polynomial nonlinear systems. Our algorithm is faithful to the nonlinear nature of the system and does not linearize the system at any point. Consequently, it provides a tightly over approximated reachable set that is close to the exact reachable set. Most of the previous techniques compute reachability starting with the initial state and iteratively growing the reachable set [6], [7]. That approach is called forward reachability analysis [6]. In contrast, we start with an over approximation that constitutes the entire reachable space of the system. We compute the boundaries of the reachable set by iteratively determining which regions in the over approximated reachable set are unreachable. Next, we remove those regions from the reachable set to reduce its size. We call our method *iterative reachable set reduction*.

Our algorithm works as follows. Initially, the entire state space is marked as the reachable set. Then we compute and refine the boundaries of the reachable set from the outside. At every iteration, our algorithm recursively partitions the reachable space into convex polytopes. For a given polytope, an adjacent polytope is one that shares a face with it. We determine if every polytope is reachable from its adjacent reachable polytopes. If we determine that a polytope is not reachable from any of its adjacent neighbor polytopes, then that polytope is marked as unreachable. We remove those unreachable polytopes from the reachable set to refine the over approximated reachable set.

A polytope is reachable if there is a feasible trajectory toward it from any of its adjacent reachable polytopes. Therefore, we examine the direction of state space trajectories over the common face of every adjacent neighbor polytope. The direction of a state space trajectory is modeled as a multi-variable *reachability decision function* whose domain is the common face between the two adjacent polytopes. We call a function *existentially positive* if there exists a point in its domain where the sign of the function is positive[1]. If the reachability decision function is existentially positive on the common face between the target and its adjacent polytope, we determine that the target polytope is reachable. If none of the corresponding reachability decision functions are existentially positive, we declare the target polytope unreachable.

To determine whether a function is existentially positive on its domain, we check whether that function has any roots in that domain. Current root finding algorithms are known to be numerically unstable [15]. However, in our context, we would like to determine the existence of a root in a domain rather than finding the location of the root. Hence it is sufficient to count the roots without actually finding them. We employ a root counting method based on *Sturms theorem* [14] for nonlinear polynomial systems. Although the root counting method provides precise analysis for polynomial nonlinear circuits, in the case of general nonlinear circuits, it is not applicable. In the general case, we use root finding methods (like the *Newton-Raphson* method or the *Quasi-Newton* method [15]) to determine whether the function is existentially positive. By using root counting for polynomial nonlinear systems, we provide an accurate solution for proving reachability without linearizing the system at any point. The polytopes that represent the partitions of the state space get progressively smaller with every iteration. The over approximation error of the reachable set is non-increasing and becomes smaller.

Typical implementations of polytope partitioning suffer from memory-related efficiency issues. We circumvent these problems by using the *Space Partitioning Tree (SPT)* data structures to model the state space. SPT is the generalized Binary Space Partitioning Tree (BSPT) in higher dimensions[22]. Previously, BSPT has been used in computer graphics [22], CAD, and verification [1]. We use

[1]Existential positivity is defined over a ball in $\mathbb{R}^n$ space. A function can be existentially positive and negative over a ball at the same time.

978-3-9815370-0-0/DATE13/

SPT for recursive partitioning of the state space and as a data structure for storing and accessing geometric objects. Partitioning of the state space into polytopes results in generation of many polytopes for modeling the state space. SPT models polytopes in the state space using hyperplane division instead of modeling each polytope individually. Consequently, the complexity order of the number of generated polytopes becomes polynomial. Also, SPT is very efficient at enumerating adjacent polytopes [22] and extracting the boundaries of the reachable set [5]. Those properties make SPT a suitable underlying data structure for our reachability algorithm.

Our major contributions are as follows.

- We propose the iterative reachable set reduction algorithm, for reachability analysis of analog circuit systems. Our algorithm iteratively reduces the volume of the reachable set in an outside-in manner and converges quickly on a result.
- Our algorithm can be used to verify nonlinear analog circuits. We are faithful to the nonlinearities of the system. Our algorithm is accurate and does not introduce any linearization error into the reachable set.
- Our algorithm utilizes Space Partitioning Trees (SPT) to efficiently model the state space partitioning. Due to usage of SPT, our algorithm is more memory efficient as compared to the state-of-the-art.

Since our over approximations are conservative abstractions of the reachable set, our algorithm will never declare an unsafe state as safe, but it might declare safe state as unsafe. We prove this soundness of our algorithm. Our algorithm will always converge to the exact reachable set, or an over approximation of it. We demonstrate empirically that the algorithm converges in a few iterations to a tight approximation. We compute the reachable set of a nonlinear *Van der Pol* oscillation circuit, a standard circuit used in this analysis. To increase the confidence in our results, we run several transient simulations using a numerical simulator to approximately delineate and illustrate the reachable set. The transient simulations closely follow the output of our algorithm.

The rest of this paper is organized as follows. Section II describes the preliminaries and background on the reachability problem. We describe our algorithm in detail in Section III. Section IV contains the experimental results for our algorithm. Finally, we review the related work and conclude in Section V.

II. Preliminaries

We now define the terminology that we use for modeling nonlinear systems. We also present some background on the reachability problem and its definition.

A. Model for Nonlinear Analog Circuits

A nonlinear circuit is modeled as a set of ordinary differential equations (ODEs) through modified nodal analysis (MNA) [15] of the circuit's netlist. Let f denote the nonlinear function governing the dynamics of the circuit, and $t \in [0, \infty)$. Let $\mathbb{S} \subseteq \mathbb{R}^n$ denote the continuous state space of the circuit. Let $\mathbb{U} \subseteq \mathbb{R}^m$ denote the input space of the circuit. $\mathbf{x}$ denotes the state variables, and $\mathbf{u}$ denotes the input variables of the circuit. $\mathbf{x}(t)$ denotes the state of the circuit at time t. The initial state of the circuit is $\mathbf{x}(0)$. Therefore, a nonlinear analog circuit is described by:

$$f(\mathbf{x}(t), \dot{\mathbf{x}}(t), \mathbf{u}(t)) = 0 \tag{1}$$

B. Reachability Analysis and Safety Definition

A *trajectory* of the circuit in the time interval $[t_1\ t_2]$ is the path taken by the circuit from state $\mathbf{x}(t_1)$ to state $\mathbf{x}(t_2)$. For a given state $\mathbf{x}(t_1)$ and input $\mathbf{u}(t_1)$, the differential constraints in Equation 1 determine the trajectory of the circuit in the interval $t \in [t_1\ t_2]$. A state trajectory derived from an action trajectory for some initial state $\mathbf{x}(t_1)$ at time $t = t_1$ is defined by:

$$\mathbf{x}(t) = \mathbf{x}(t_1) + \int_{t_1}^{t} f(\mathbf{x}(t'), \mathbf{u}(t'))dt' \tag{2}$$

The *reachable set* is the set of all states that are reachable from the initial set of states for all possible trajectories (paths), for all admissible input signals U.

$$R_{x(0)}(U) = \bigcup_{x \in x(0)} \bigcup_{u \in U} \bigcup_{t \in [0, +\infty)} R(x, u, t) \tag{3}$$

where $R(x, u, t)$ is the state trajectory from state x. $R_{x(0)}$ denotes the reachable set from the initial set $x(0)$ for all $u \in \mathbb{U}$.

The over approximated reachable set $\overline{R_{x(0)}}$ is defined as a set that satisfies $R_{x(0)} \subseteq \overline{R_{x(0)}} \subset \mathbb{S}$. Given the state space of an analog circuit $\mathbb{S}$, we want to verify safety properties. We define safety properties by specified sets of unsafe regions in the state space $\mathbb{R}_{\text{unsafe}}$. A safety property is satisfied if there is no possible trajectory from any of the initial states toward $\mathbb{R}_{\text{unsafe}}$. We conclude that the safety property has been satisfied when

$$R_{x(0)} \cap R_{\text{unsafe}} = \emptyset \tag{4}$$

On the other hand, $\overline{R_{x(0)}} \cap R_{\text{unsafe}} \neq \emptyset$ is not necessarily an indication of safety violation. This is an implication that we cannot yet determine the safety of the circuit.

III. Iterative Reachable Set Reduction Algorithm

The objective of reachability analysis is to determine if there exists a trajectory from the set of initial states that eventually reaches the set of unsafe states under the circuit's differential regime. Our algorithm achieves this objective by iteratively identifying unreachable states. To identify the unreachable regions, we will recursively partition the over approximated reachable set $\overline{\mathbb{R}}_x$ into convex polytopes.

Our iterative reachable set reduction algorithm consists of four major components: i) the main iterative reachable set reduction loop, ii) the state space partitioning algorithm, iii) a process for determining the reachability of adjacent neighbor regions, and iv) an SPT data structure for state space modeling. Figure 1 illustrates the important phases of our algorithm that are described in the following subsections.

The inputs to our algorithm are i) the state space of a nonlinear analog circuit, along with ii) the governing differential equations, iii) the set of initial states in the state space, and iv) the set of unsafe states.

Let $\mathbb{S} \subseteq \mathbb{R}^d$ be the continuous state space of an analog circuit where d is the number of the state variables. Let $R_{x(0)} \subseteq \mathbb{S}$ be the reachable set of the circuit from the initial set $x(0)$, and $\overline{R} \subseteq \mathbb{S}$ be an over approximation of R_x, so $R_x \subseteq \overline{R}_x$. We assume the state space is bounded. That assumption is not limiting, because the target of our algorithm is an analog circuit. For example, a user can define the voltage variable to be bounded by $[-V_{cc}, +V_{cc}]$, where V_{cc} is the value of the voltage source, and so on. We assume that any region outside those bounds is unreachable, and we consider region inside the bounds to be the state space of the circuit.

A. Iterative Reachable Set Reduction

In this section, we describe the primary loop of our algorithm, the iterative reachable set reduction. Algorithm 1 shows that this loop will recursively remove the unreachable regions from the reachable state space.

We model the partitioned regions as convex polytopes which are identified through intersections of hyperplanes. At every iteration, we analyze the polytopes that are at the boundaries of the reachable set. This implies that those polytopes share boundaries with some unreachable set. For every generated polytope P_i, if P_i is adjacent to some unreachable set, then we analyze the reachability of P_i from its neighbors. We analyze the reachability of the polytope by checking the direction of state space trajectories of adjacent partitions in the reachable set. If we prove there is no feasible trajectory from any of the adjacent reachable regions toward the polytope P_i, we determine that P_i is unreachable and we remove it from the reachable set $\overline{\mathbb{R}}_x$. Otherwise, we recursively partition the P_i to further refine the reachable set.

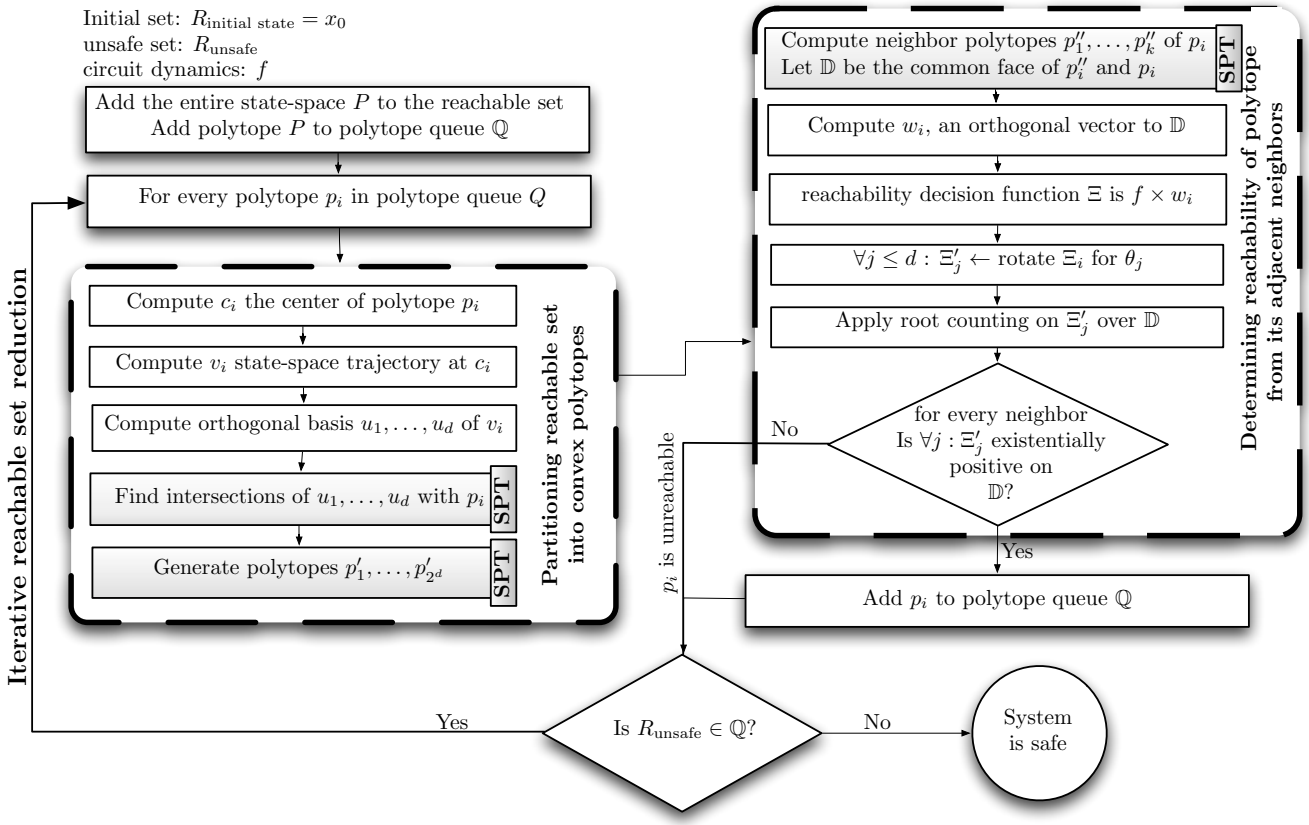

Fig. 1: Overview of the iterative reachable set reduction algorithm. The exterior loop is the iterative reachable set reduction algorithm. For each polytope, our algorithm partitions it. Then, for each new partitions, our algorithm decides on the reachability of those partitions from the reachable set. The parts of our algorithm that use SPT for computation are marked with SPT labels.

Algorithm 1 Iterative reachable set reduction algorithm

```
Data: Circuit 𝕊, Initial States 𝕀, Iteration bound n
Queue ℚ ;
ReachableSet R_x ;
ℚ.push(R_x) ;
while ¬ ℚ.empty() do
    Polytope ℙ ← ℚ.pop() ;
    if ℙ is adjacent to an unreachable region or iteration < n then
        iteration ← iteration+1 ;
        partition(ℙ) ;
        foreach P_j in ℙ.getChildren() do
            Determine the reachability of P_j from its adjacent neighbors;
            if P_j is reachable then
                ℚ.push( P_j ) ;
            end
        end
    end
end
```

In Algorithm 1, we create a queue of reachable polytopes. For each of those polytopes P_i, if P_i is at the boundary of reachable set, we further divide P_i to get a finer partition. Next, for each of the sub partitions of P_i (called P_i's children) we determine if they are reachable from the reachable set. We enqueue any of those sub partitions that are reachable and discard other sub partitions to get a more accurate over approximation. This process continues until a predefined number of polytope division n is reached; at that point, the algorithm terminates. In our algorithm, the volume of polytopes rapidly get smaller and our algorithm converges to an approximated reachable set very fast.

To determine the reachability of each sub partitions, we analyze the direction of the state space trajectories toward those sub partitions. The polytope is reachable under either of two conditions: i) the polytope is part of the initial state, or ii) there exists a trajectory from at least one of the polytope's reachable adjacent neighbors toward it. In those cases, we conclude that the polytope is reachable.

B. Partitioning the Reachable Set into Convex Polytopes

We partition the continuous space of analog circuits to obtain a discrete model for the state space. Our algorithm partitions the reachable state space into convex polytopes. The partitioning is based on the direction of state trajectories at the center of each polytope. Let d denote the dimension of the system.

Algorithm 2 Partitioning the polytope

```
Data: Circuit 𝕊, Convex Polytope P
c = center point of P ;
v = 𝕊.state space trajectory at (c) ;
u_1, ..., u_d = GramSchmidt(𝕊, v) ;
i_1, ..., i_d = Compute intersections of u_1, ..., u_d hyperplanes with P ;
P_1, ..., P_2d = Polytopes defined by i_1, ..., i_d points and P ;
return P_1, ..., P_2d ;
```

Algorithm 2 shows an overview of our partitioning algorithm for a given polytope P. Initially, the entire state space constitutes the first polytope. At every subsequent iteration, we recursively divide the polytope by partitioning it using hyperplanes. We compute those hyperplanes using a vector basis that is orthogonal to the state space trajectory at the center of the polytope. Let c be the center of convex polytope P. Let v be the trajectory vector of the system at c (Figure 2.a). Using Gram-Schmidt orthonormalization process [8], [20], we construct an orthogonal basis vector set u such that $\{u_1, \ldots, u_d : u_i.u_j = 0, \forall 1 \leq i, j \leq d, i \neq j, v \in u\}$ (Figure 2.b). Vectors $u_1, \ldots, u_d$ form a set of d hyperplanes that divides the polytope P into 2^d convex polytopes $P_1, \ldots, P_{2^d}$ (Figure 2.c). Accordingly, our algorithm computes the intersection of each hyperplane with the faces of the polytope P. Then we compute the polytopes

generated by the intersection of those hyperplanes and the polytope P. For example, in Figure 2.c, the new polytope P_1 is defined by the sequence of points $< c, i_4, q_5, q_1, i_1 >$ and so on. For recursively partitioning the state space, we utilize space partitioning tree (SPT) algorithm. SPT divides the state space into convex polytopes defined by intersection of hyperplanes. SPT algorithm allows an efficient storing and accessing of the polytopes in polynomial time [22].

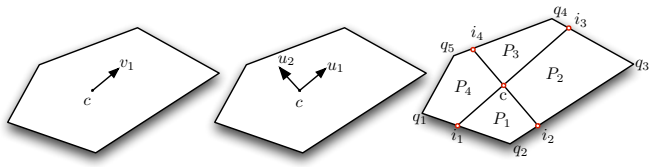

Fig. 2: Partitioning a polytope based on state space trajectories.

C. Determining the Reachability of Adjacent Polytopes

After generating a new polytope, we determine whether that polytope is reachable from the adjacent reachable polytopes. To determine the reachability between adjacent polytopes, we evaluate the direction of trajectories at the common faces of the polytope and adjacent polytopes from the reachable set. If we can prove that for all common faces with the reachable set, there is no trajectory from the reachable set toward that polytope, we can deduce that polytope is unreachable and remove it from the reachable set. As shown in Figure 3, in 2-dimensions, the shared faces between two adjacent polytope R_1 and R_2 is the line from p_1 to p_2. Therefore for checking reachability of R_1 from its adjacent reachable neighbor R_2, we should check if there is any trajectory from R_2 to R_1 at the common face between two polytopes. We need to find at least a single trajectory from R_2 that goes toward R_1. We reformulate this as an analytical *reachability decision function*.

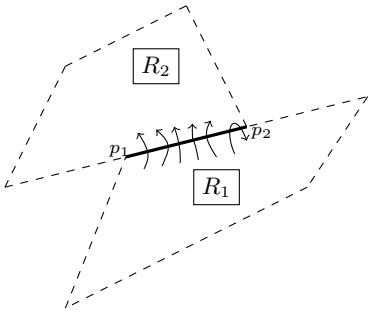

Fig. 3: Determining reachability of two adjacent polytopes.

The direction of the trajectories over the bounded face of the polytope is presented as a multi variable reachability decision function. Let w be an orthogonal vector from the center of the p_1, p_2 face toward R_1. We use a cross product of the RHS of the circuit's ODE f (Section II-A) with the w vector to determine the direction of trajectories. For example, for a 2-dimensional system, the direction of vector trajectories is defined by the following *reachability decision function* Ξ:

$$\Xi(x,y) = f(x,y) \times w = det \begin{pmatrix} p_2.x - p_1.x & f_1 - p_1.x \\ p_2.y - p_1.y & f_2 - p_1.y \end{pmatrix} \quad (5)$$

where $p_1 =< x_1, y_1 >$ and $p_2 =< x_2, y_2 >$.

We define *existential positivity* of a function to determine if there is any interval in which the function ξ is positive on its domain. The existential positivity property for function ξ is defined as if there exists any *ball* $B_\epsilon(t) \in D_\xi$ such that for some $\mathbf{x} \in B_\epsilon(t)$ we have $\xi(\mathbf{x}) > 0$, where D_ξ denotes to domain of ξ.

The existence of a trajectory from R_2 toward R_1 is equivalent to the existential positivity of the reachability decision function Ξ on the function's domain (line p_1 to p_2, i.e., $\mathbb{D} = \{< x, y >: \lambda p_1 + (1-\lambda)p_2 =< x, y >, \lambda \in [0,1]\}$). Therefore, when $\Xi(x,y) > 0$, the direction of the trajectories is toward R_1 and $\Xi(x,y) < 0$ is an indication of the direction of the trajectories toward R_2.

To determine existential positivity of higher dimensional functions, we transform them to lower dimensions. Therefore, we reduce that function to a weaker form by transforming it from a single d-dimensional function into d single-dimensional functions using rotation transformation. At lower dimensions (like $d = 1$), we use a root-counting method using the *Sturm's theorem* [14] and root-finding method using the *Newton-Raphson* algorithm [15] to determine the existential positivity of the function. We call these single-dimensional functions the *basis functions* of the reachability decision function.

The domain of the basis function becomes paraxial by applying rotation transformation to the common faces between two adjacent polytopes. Therefore except for only one axis, all other variables are constant. By applying d rotation transformations to the decision function for each axis, the reachability decision function is reduced to d single-dimension basis functions. If all of those d single-dimension functions are existentially positive on their basis domains (which are the intervals obtained by rotating the domains of the decision functions), we conclude that the reachability decision function is existentially positive on its domain. For example, in two dimensions the rotation transformation is as follows.

$$\begin{bmatrix} \Phi(x) \\ \Phi(y) \end{bmatrix} = \begin{bmatrix} \Xi_1(x,y) \\ \Xi_2(x,y) \end{bmatrix} \times \begin{bmatrix} \cos(-\theta) & -\sin(-\theta) \\ \sin(-\theta) & \cos(-\theta) \end{bmatrix}$$

where Ξ_1 and Ξ_2 are the RHS of the system's ODE, and θ is the angle between the face and the axis. $\Phi(x)$ and $\Phi(y)$ are the basis functions.

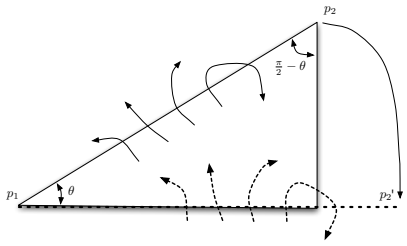

Fig. 4: Determining existential positivity of the reachability decision function. Our algorithm rotates the function θ degrees to align it to the axis. Therefore, other variables become constant, and the reachability decision function becomes a single-dimensional basis function.

For a single dimensional basis function, our algorithm uses two different methods for determining its existential positivity. Existential positivity for a function depends on whether the function has any roots in its domain. If the root does not exist, it means that the function is not changing signs on its interval. Therefore we evaluate the basis function at the midpoint of the function's domain. If the function evaluates to positive, then we can conclude existential positivity of the basis function. Otherwise, the basis function is not existentially positive. We use two methods for determining the existence of a root in the basis function.

- Root counting method. For polynomial systems, we count the number of roots in any given domain using Sturm's theorem [14]. Sturm's theorem defines the number of real roots of a polynomial system in any interval using the changes in the signs of the values of the Sturm's sequence.
  Therefore if the total number of roots of the basis function in its domain is more than one, we deduce that the basis function is existentially positive on its domain. The benefits of root counting method is that it always returns an exact result. However, Sturm's theorem can only be applied to nonlinear polynomial systems.
- Root finding method. Instead of counting the number of roots, our algorithm computes the roots of the function. If our algo-

rithm is able to find at least one root in the domain of the basis function, we conclude that the basis function is existentially positive on its domain. Our algorithm uses the Newton-Raphson [15] algorithm to find roots of nonlinear functions.

D. Modeling the State Space Using a Space Partitioning Tree (SPT)

We use a Space Partitioning Tree (SPT) algorithm to model and store the polytopes generated in the state space. The space partitioning tree algorithm is the general n-dimensional case of the binary space partitioning algorithm used in [1], [22].

The polytopes are modeled in the SPT tree as shown in Figure 5. First, the entire state space is modeled as the root of tree. Then we partition the root into 2^2 polytopes using two hyperplanes. Those hyperplanes are added to the SPT to model the generated polytopes. The tree is built and maintained on-the-fly during execution of the reachability algorithm.

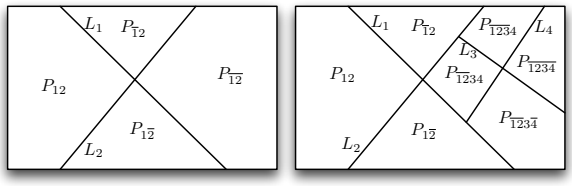

Fig. 5: State space partitioning using hyperplanes. The polytopes are defined by the intersections of the hyperplanes in the state-space.

Let d denote the number of dimensions. We are constructing the SPT in R^d space. At each iteration, we add d hyperplanes to the tree to model the 2^d convex polytopes. Each hyperplane can be defined in $\mathcal{O}(d)$ memory. Therefore the memory complexity of our algorithm is $\mathcal{O}(nd^2)$ where n is the number of divisions that our algorithm performs. SPT allows access of logarithmic complexity to each polytope inside the state space while it utilize a moderately small memory foot-print. SPT also facilitates an efficient enumeration of adjacent polytopes to a region. Finally, SPT allows for fast computation of boundary of reachable set without computing the union of all reachable sets [5].

E. Sketch of Soundness Proof

To prove soundness of iterative reachable set reduction algorithm, we show that if the region is unsafe, our algorithm will never declare that the region is safe. On the other hand, if the region is safe, our algorithm does not make any guarantees on it. We do not generate any false positives. Let $\mathbb{S} \subseteq \mathbb{R}^n$ denote the continuous state space of the circuit. Let $R_{x(0)}$ and R_{unsafe} denote the initial set and unsafe set of states respectively. A region is a set of states in $\mathbb{S}$. An unsafe region $\mathbb{X}$ is a region such that $\mathbb{X} \in R_{\text{unsafe}}$. The safe regions are in $\mathbb{S}-\mathbb{X}$. Let the sequence $< p_i, p_{i+1}, \ldots, p_j >$ denote the set of convex polytopes in the state space such that p_t and p_{t+1} are adjacent.

We use proof by contradiction. Given an unsafe region $\mathbb{U} \in R_{\text{unsafe}}$, let us assume that our algorithm declares this unsafe region as safe. Initially, the algorithm considers the entire set of states as reachable. This includes the safe as well as unsafe states in the system. The algorithm does not declare any unsafe region as safe. Hence, it is initially sound. Let us say that in iteration i, the algorithm declares (erroneously) that a region $\mathbf{U}$ is safe. In every subsequent iteration, it will declare $\mathbf{U}$ as safe. Initially we assumed that the system was unsafe. This implies that there is at least one trajectory from initial state $x_0 \in R_{x(0)}$ to state $x_f \in \mathbf{U}$. Let us call this trajectory T. By construction, T will cross some subsequence of adjacent polytopes $< p_0, p_1, \ldots, p_k >$ such that $x_0 \in p_0$ and $x_f \in p_k$. The polytope p_0 is reachable because $p_0 \cap R_{x(0)} \neq \emptyset$. Since the algorithm declared the system safe, the polytope p_k should be unreachable. Therefore, for some $1 < j \leq k$ our algorithm has determined that p_j is reachable but p_{j+1} is unreachable. This would mean that there is no trajectory from p_j toward $p_j + 1$. But T is a trajectory from p_0 to p_1 to $\ldots p_j$ to p_{j+1} to $\ldots$ to p_k. Therefore there is a trajectory from p_j to p_{j+1}. This is a contradiction. Hence the soundness of the algorithm is proved.

IV. Experimental Results

We implemented the iterative reachable set reduction algorithm in a prototype tool in the C++ language to evaluate its accuracy and efficiency. We chose an Apple Macbook Pro laptop equipped with a Core i7 processor and 8 GB memory as our computing platform. We ran our algorithm over a Van der Pol oscillation circuit to compute its reachable set. A Van der Pol oscillator is a nonconservative oscillator with nonlinear damping. A Van der pol oscillator is a fundamental example in nonlinear oscillation theory [12]. It has a periodic solution that attracts every solution in the state space (except the zero solution). It is governed by two dimensional equations.

$$\dot{x} = y \tag{6}$$

$$\dot{y} = \epsilon(1 - x^2) \times y - x \tag{7}$$

In our experiment, we let $\epsilon = 1$, which was a medium value for ϵ and resulted in a medium distortion in the oscillation. The state space was defined as a bounded box $\mathbb{S} = [-10, 10] \times [-10, 10] \subset \mathbb{R}^2$. The initial set was a box $[-3.0, -2.8] \times [3, 3.2] \subset \mathbb{S}$. Figure 6 shows the reachable set of the Van der Pol oscillator circuit. The reachable set is the grey region. The unreachable states are in white. Figure 6 also shows the hyperplanes and the polytopes that our algorithm generated to compute the reachable set. As shown in the figure, our algorithm rapidly removed huge portions of the state space that were unreachable from the reachable set during the first few iterations. Then our algorithm eventually converged on the more refined and accurate reachable set. Our algorithm took **0.05 seconds** to compute the reachable set on our computing platform.

The iterative reachable set reduction algorithm can be effectively used for safety verification through computation of the reachable set. In many real test cases, only a few iterations are required to generate a coarse approximation of the reachable set and hence prove safety. That makes our algorithm a suitable candidate for safety verification. At any point during the execution, if we can prove safety, our algorithm terminates. If after reaching a predefined number of polytope partitioning, we have been unable to prove safety, our algorithm terminates without deciding on the safety of the system.

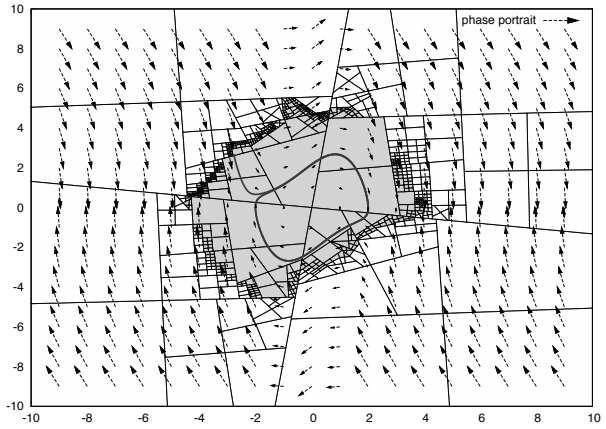

Fig. 6: Reachable set for the Van der Pol oscillator using our iterative reachable set reduction algorithm. The reachable set is in grey and the unreachable states are in white. The polytopes at the boundaries of the reachable set shrink rapidly in volume.

To make Figure 6 more informative, we added a quiver plot of the vector field (marked with arrows). We have also shown a transient simulation from a sampled point in the initial set. The transient trace was simulated using a numerical ODE solver (explicit embedded Runge-Kutta Prince-Dormand (8,9) method available in GNU-gsl-odeiv2 package) to delineate the reachable set. The circuit was simulated for $t = 20$ with $\delta t = 0.02$ time steps.

Table I shows some statistics of the space partitioning tree. We terminated the execution after 250 iterations. The SPT was built in two dimensions on-the-fly during the execution of our algorithm. In the end, the algorithm explored and divided 751 polytope out of the possible 4^9 polytopes. The maximum depth of the tree was a logarithmic order of the generated polytope. In the end, the boundary of the reachable set consisted of 214 polytopes.

Statistical information obtained from the SPT	Value
Number of generated polytopes	1000
Number of leaves in the tree	751
Number of reachable leaves in the tree	491
Number of generated hyperplanes	500
Maximum depth of the SPT	9
Average depth of the SPT	7.34
Volume of smallest polytope in the leaf	4.63e-4 †
Volume of biggest polytope in the leaf	26.40 †
Volume of biggest reachable polytope in the leaf	6.25 †
Average volume of a polytope in the leaf	0.53 †
Average volume of a reachable polytope in the leaf	0.14 †

TABLE I: Space partitioning tree statistics. During the execution of the iterative reachable set reduction algorithms, most of the generated polytopes are at the boundaries of the reachable set and they rapidly get smaller in volume. † indicates that the number is a two-dimensional volume.

Among the different components of our iterative reachable set reduction algorithm, the component that allows for optimizations is the partitioning technique. Hence, a change in the partitioning algorithm significantly impacts the quality of results. Initially, we used hyper-rectangle partitioning, where each polytope's face was aligned to the axes. Similar to [20], we observed that the hyperbox data structure was causing a "massive over approximation" of the reachable state space. Also, we tried other methods for polytope partitioning ,such as partitioning of each polytope into two polytopes using the binary space partitioning tree (instead of d polytopes). However, the results were unsatisfactory. With those observations, we decided that polytope partitioning with d hyperplanes, resulting in 2^d polytopes at each iteration, is the optimum implementation of our algorithm.

To evaluate the reachability determination between adjacent polytope algorithm, we used a sampling-based method. To determine the reachability of a polytope from its adjacent polytopes, our algorithm incorporated a sampling scheme as follows. Our algorithm created many sample points at the borders of the polytope and simulated them for a δt time. Then our algorithm determined if the final state of the simulation had ended up inside the polytope or in the adjacent polytope. As a result, for 100 samples for each common face of each polytope, there was no significant difference between the sampling based reachability decision and our algorithm. However the sampling-based method took significantly more time (**4.48 seconds**) to compute the reachable set.

V. Related Work and Conclusion

Asarin et al. provide an introduction and formal definition for reachability analysis [3]. Most of reachability analysis techniques construct the reachable set from the initial set using forward reachability analysis [6]. Several techniques have investigated the usage of polytopes [6], [4], [9], zonotopes [7], or support functions [13]. Some reachability techniques are based on state space discretization methods [11], [20], [21], [10]. Another technique in control theory for verifying safety without actually computing the reachable set is using barrier certificates [16], [19].

Another related technique to ours is the backward reasoning technique [17]. Alur et. al. in their pivotal paper [1] proposed a technique for reachability analysis of linear hybrid systems using predicate abstraction. They propose to improve reachability analysis through vector field analysis and binary space partitioning to optimize predicate abstraction. Ratschan and She [18] propose recursive backward reasoning for hyper-boxes. In comparison to their work, we provide a more efficient partitioning algorithm using polytopes and a sound method for computing reachability decisions between adjacent polytopes for nonlinear analog systems.

In conclusion, we propose an iterative reachable set reduction algorithm to compute and refine the reachable set of nonlinear analog circuits. Our algorithm recursively partitions the reachable set into convex polytopes. We determine the reachability of adjacent neighbor polytopes through analyzing the direction of the state space trajectories at the common faces between two adjacent polytopes.

References

[1] R. Alur, T. Dang, and F. Ivančić. Predicate abstraction for reachability analysis of hybrid systems. *ACM Transactions on Embedded Computing Systems*, 5(1):152–199, 2006.

[2] E. Asarin, V. P. Mysore, A. Pnueli, and G. Schneider. Low dimensional hybrid systems – decidable, undecidable, don't know. *Information and Computation*, 211:138–159, Feb. 2012.

[3] E. Asarin, G. Schneider, and S. Yovine. Algorithmic analysis of polygonal hybrid systems, part i: Reachability. *Theor. Comput. Sci.*, 379(1-2):231–265, July 2007.

[4] A. Chutinan and B. H. Krogh. Computational techniques for hybrid system verification. *Automatic Control, IEEE Transactions on*, (1):1–12, 2003.

[5] J. Comba and B. Naylor. Conversion of binary space partitioning trees to boundary representation. In *Proceedings of Theory and Practice of Geometric Modeling*, Jan. 1996.

[6] G. Frehse. PHAVer: Algorithmic verification of hybrid systems past HyTech. *International Journal on Software Tools for Technology Transfer (STTT)*, 10(3):263–279, 2008.

[7] A. Girard. Reachability of uncertain linear systems using zonotopes. *Hybrid Systems: Computation and Control*, pages 1–15, Jan. 2005.

[8] G. H. Golub and C. F. van der Loan. *Matrix Computations*. The Johns Hopkins University Press, 3rd edition, 1996.

[9] M. Greenstreet and I. Mitchell. Reachability analysis using polygonal projections. *Hybrid Systems: Computation and Control*, Jan. 1999.

[10] H. A. Hansen, G. Schneider, and M. Steffen. Reachability analysis of non-linear planar autonomous systems. In *FSEN'11: Proceedings of the 4th IPM International Conference on Fundamentals of Software Engineering*. Springer-Verlag, Apr. 2011.

[11] W. Hartong, R. Klausen, and L. Hedrich. Formal verification for nonlinear analog systems: Approaches to model and equivalence checking. *Advanced Formal Verification*, pages 205–245, 2004.

[12] H. K. Khalil. *Nonlinear Systems*. Prentice Hall, 3 edition, Dec. 2001.

[13] C. Le Guernic and A. Girard. Reachability analysis of linear systems using support functions. *Nonlinear Analysis: Hybrid Systems*, 4(2):250–262, May 2010.

[14] P. Pedersen. Multivariate Sturm theory. *Applied algebra, algebraic algorithms and error-correcting codes, Lecture Notes in Computer Science*, 539:318–332, 1991.

[15] L. T. Pillage, R. A. Rohrer, and C. Visweswariah. *Electronic circuit and system simulation methods*. McGraw-Hill Professional Publishing, 1995.

[16] S. Prajna and A. Jadbabaie. Safety verification of hybrid systems using barrier certificates. *Hybrid Systems: Computation and Control, volume 2993 of Lecture Notes in Computer Science*, pages 271–274, Feb. 2004.

[17] J. Preußig, O. Stursberg, and S. Kowalewski. Reachability Analysis of a Class of Switched Continuous Systems by Integrating Rectangular Approximation and Rectangular Analysis. In *HSCC '99: Proceedings of the Second International Workshop on Hybrid Systems: Computation and Control*. Springer-Verlag, Mar. 1999.

[18] S. Ratschan and Z. She. Recursive and backward reasoning in the verification of hybrid systems. In *Proceedings of the 5th Int Conf on Informatics*, Jan. 2008.

[19] C. Sloth, G. J. Pappas, and R. Wisniewski. Compositional safety analysis using barrier certificates. *Proceedings of the 15th ACM international conference on Hybrid Systems: Computation and Control - HSCC '12*, pages 15–29, Jan. 2012.

[20] S. Steinhorst. Formal verification methodologies for nonlinear analog circuits. *PhD thesis*, Universität Frankfurt a. M., 2010.

[21] S. Steinhorst and L. Hedrich. Model checking of analog systems using an analog specification language. *2008 Design, Automation and Test in Europe*, pages 324–329, Mar. 2008.

[22] W. C. Thibault and B. F. Naylor. Set operations on polyhedra using binary space partitioning trees. *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques - SIGGRAPH '87*, pages 153–162, Jan. 1987.