Counterexample Guided Abstraction Refinement for Stability Analysis

Pavithra Prabhakar[1] and Miriam García Soto[2]

[1] Kansas State University, Manhattan, KS, USA
pprabhakar@ksu.edu
[2] IMDEA Software Institute & Universidad Politécnica de Madrid, Spain
miriam.garcia@imdea.org

Abstract. In this paper, we present a counterexample guided abstraction refinement (CEGAR) algorithm for stability analysis of polyhedral hybrid systems. Our results build upon a quantitative predicate abstraction and model-checking algorithm for stability analysis, which returns a counterexample indicating a potential reason for instability. The main contributions of this paper include the validation of the counterexample and refinement of the abstraction based on the analysis of the counterexample. The counterexample returned by the quantitative predicate abstraction analysis is a cycle such that the product of the weights on its edges is greater than 1. Validation involves checking if there exists an infinite diverging execution which follows the cycle infinitely many times. Unlike in the case of CEGAR for safety, the validation problem is not a bounded model-checking problem. Using novel insights, we present a simple characterization for the existence of an infinite diverging execution in terms of the satisfaction of a first order logic formula which can be efficiently solved. Similarly, the refinement is more involved, since, there is a priori no bound on the number of predecessor computation steps that need to be performed to invalidate the abstract counterexample. We present strategies for refinement based on the insights from the validation step. We have implemented the validation and refinement algorithms and use the stability verification tool AVERIST in the back end for performing the abstraction and model-checking. We compare the CEGAR algorithm with AVERIST and report experimental results demonstrating the benefits of counterexample guided refinement.

1 Introduction

Hybrid systems refer to systems exhibiting mixed discrete continuous behaviors. These manifest naturally in embedded control systems as a result of the interaction of embedded software, which executes in discrete steps, with physical systems, which evolve continuously in dense real-time. In particular, we consider switched hybrid systems [13] in which the continuous state does not change during a mode switching. These are apt for modeling supervisory control, wherein a supervisor continuously senses the state of a plant and takes mode change decisions based on that.

In this paper, we focus on automated stability analysis of switched hybrid systems. Stability is a fundamental property in control system design and captures robustness of the system with respect to initial states or inputs. We consider a classical notion of stability, namely, *Lyapunov stability* with respect to an equilibrium point — a state of the system which does not change with time evolution. Intuitively, an equilibrium point is Lyapunov stable if the executions starting in a small neighborhood of the equilibrium point remain close to it.

The classical methods for stability analysis are based on exhibiting a function from the state-space to the non-negative reals called a Lyapunov function (see, for instance, [11]), that ensures that the value of the function decreases along any execution of the system. Automated methods for stability analysis rely on a template based search for a Lyapunov function. For instance, a polynomial function with coefficients as parameters is chosen as a candidate Lyapunov function. The parameters are computed by solving Linear Matrix Inequalities or Sum-of-Squares [17] programming which arise while encoding the constraints of the Lyapunov function.

One of the challenges with Lyapunov based methods is the ingenuity of the user required in choosing the right templates. An exhaustive search over all templates (for instance, polynomials of increasing degrees) becomes unmanageable for relative small degrees of polynomials. In [20, 21], the authors present an alternate stability analysis method based on abstractions for a subclass of hybrid systems called polyhedral hybrid systems. Polyhedral hybrid systems are an interesting class of systems that can be used to abstract linear and non-linear hybrid systems [12, 3]. The authors propose a quantitative predicate abstraction method that constructs a finite weighted graph, and analyse the latter for the existence of cycles with product of edge weights ≥ 1. The absence of such cycles indicates that the system is Lyapunov stable. It is suggested that better abstractions can be obtained by choosing a larger set of predicates. However, no efficient strategies for the selection of the same is discussed. Here we take the quantitative predicate abstraction based analysis a step further, and discuss strategies for refinement based on counterexample validation. This is popularly referred to as CEGAR (counterexample guided abstraction refinement [6]).

The main contributions of the paper are the validation and refinement algorithms. Validation consists of checking if an abstract counterexample actually corresponds to a concrete counterexample. In the context of safety analysis [6], an abstract counterexample typically consists of a finite sequence of abstract states or nodes in a finite graph from the initial state to an unsafe state. Validation consists of checking if there exists a finite execution of the system which follows the sequence of abstract states. However, the validation problem we encounter is not a bounded model-checking problem as above. Instead, it consists of checking if there exists an infinite diverging trajectory that follows the cycle infinitely many times. This property cannot, as is, be encoded as the satisfiability of a formula in a finitary logic. We provide a novel characterization of the existence of an infinite diverging execution in terms of the existence of a finite execution that follows the cycle once from a continuous state x to a continuous state y

such that $y = \alpha x$ for some $\alpha > 1$. This provides an algorithmic procedure to perform validation, since, the latter can be encoded as the satisfiability problem of a first order logic formula, and efficiently solved.

Refinement in safety analysis consists of computing, iteratively, subsets of concrete states corresponding to abstract states that can reach the unsafe states. If the counterexample is spurious, one of the computed sets is empty, referred to as the point of refinement, and a refinement occurs by examining some local abstract states around this point of refinement. Since, the concrete counterexample required for validation has a finite length m, upper bounded by the abstract counterexample length, the point of refinement is reached within m steps. In the case of refinement for stability, we show that though a priori no such bound on the point of refinement exists, if the counterexample is spurious, it is definitely reached.

We propose two refinement strategies — one of which is applicable always, however, does not eliminate a large fraction of counterexamples; the other is applicable only in certain cases, but eliminates a large fraction of counterexamples. If the validation procedure infers that the counterexample not only does not have an infinite diverging execution corresponding to it, but also does not have any infinite executions corresponding to it; then our refinement algorithms ignores the edge weights and aims to "eliminate" the cycle. Otherwise, it considers the weights and only aims to reduce the weights on the cycle.

We have implemented the CEGAR algorithm, which uses AVERIST in the backend to perform the abstraction and model-checking steps of the CEGAR. We report experimental comparisons between the CEGAR algorithm and the AVERIST algorithm. For the latter, we consider refinement based on the naive strategy of uniformly adding new predicates. Our experimental results demonstrate the benefits of CEGAR both in terms of reduced computation time and smaller abstractions that result as a result of careful refinement in each iteration. Future work will consist of extending the CEGAR framework for stability analysis to more general classes of hybrid systems, and related notions such as asymptotic stability.

Related Work. We briefly discuss related work. There is a large body of work on Lyapunov function based stability analysis for linear and non-linear hybrid systems, see the surveys [4] and [14]. There is some work on automated verification of stability of linear systems by iteratively refining partitions [16, 15, 24], however, it is not an abstraction based approach and the refinements are not guided by counterexample analyses. CEGAR has been explored for safety verification of hybrid systems [2, 5, 18] and region stability analysis [8]. However, unlike Lyapunov and asymptotic stability, safety and region stability are bisimulation invariant properties. Recently, there is some work on learning the templates for Lyapunov functions [10].

2 Polyhedral Switched System (*PSS*)

A hybrid automaton [1] is a popular formalism for modeling mixed discrete-continuous behaviors. It extends the finite state automaton model for discrete dynamics by annotating the modes with differential equations or inclusions for modeling the physical systems. In addition, invariants on the modes and guards on the edges provides constraints that need to be satisfied during evolution and mode switching, respectively. A polyhedral switched system *PSS* is a special kind of hybrid automaton in which each mode is associated with a polyhedral differential inclusion and the invariants and guards are specified by linear constraints.

Definition 1. *A n-dimensional* polyhedral switched system *(PSS) is a tuple* $\mathcal{H} = (Loc, Edges, X, Flow, Inv, Guard)$*, where:*

- Loc *is a finite set of locations;*
- $Edges \subseteq Loc \times Loc$ *is a finite set of edges;*
- $X = \mathbb{R}^n$ *is the continuous state-space;*
- $Flow : Loc \to CPolySets(n)$ *is the flow function;*
- $Inv : Loc \to PolySets(n)$ *is the invariant function; and*
- $Guard : Edges \to PolySets(n)$ *is the guard function.*

where $PolySets(X)$ *denotes the set of all convex polyhedral subsets of* X*, and* $CPolySets(X)$ *denotes the set of compact convex polyhedral sets.*

Notation. From now on, we will denote each of the elements in a *PSS* $\mathcal{H}$, with $\mathcal{H}$ as a subscript, for instance, the invariant function will be referred to as $Inv_{\mathcal{H}}$.

Example. Figure 1 shows a 3-dimensional polyhedral switched system along the $x-y$ plane when the value along the z-axis is taken to be 1. Essentially, the polyhedral sets from A to F are pyramids centered at $x = 0, y = 0$. V_A through V_F represent the polyhedra in the polyhedral differential inclusions corresponding to the regions A to F. We assume that $\dot{z} = 1$ everywhere. A sample execution of the system is shown using a sequence of directed thin lines.

A switched system starts evolving in a mode $q \in Loc$ and a continuous state x. In this mode q, the continuous state evolves inside $Inv(q)$ such that the differential of the evolution at any time lies within $Flow(q)$. If (q_1, q_2) is an edge of the system and the continuous state satisfies the guard, a switch from q_1 to q_2 can occur. The continuous state does not change during the mode switching. The semantics of a *PSS* $\mathcal{H}$ are given by the set of executions exhibited by the system.

Definition 2. *An* execution σ *of a PSS of dimension* n *is a triple* (ι, η, γ)*, where* ι *is a sequence of time intervals* $I_0, I_1, \ldots$ *which refer to the times spent by the execution in a particular location,* $\eta : \mathcal{I}(\iota) \to X$*, where* $\mathcal{I}(\iota) = \cup_i \iota(i)$*, represents the continuous state at all times, and* γ *maps* i *to the location the execution evolves in during the interval* I_i*.*

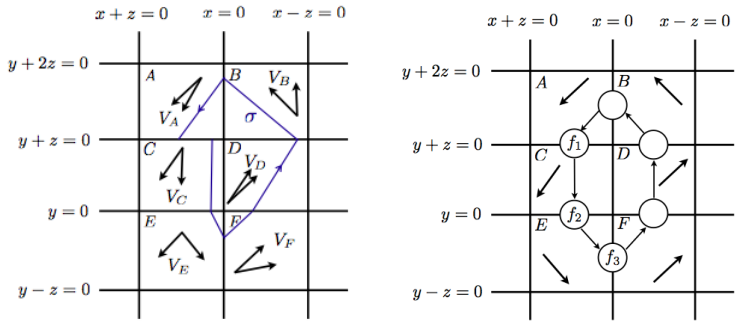

Fig. 1. (Left) Polyhedral switched system (Right) Abstract counterexample

An execution $\sigma = (\iota, \eta, \gamma)$ of $\mathcal{H}$ is said to be *complete* if $\mathcal{I}(\iota)$ is $[0, \infty)$; otherwise, it is called *finite*. The set of all executions of $\mathcal{H}$ will be denoted by $Exec(\mathcal{H})$, and the set of all complete executions by $CExec(\mathcal{H})$.

2.1 Reachability relations

We introduce certain predicates related to reachability which we will need in the sequel. Let us fix an n-dimensional *PSS* $\mathcal{H}$, two locations q_1 and q_2 in $Loc_{\mathcal{H}}$, and three polyhedral sets P_1, P_2 and P over $\mathbb{R}^n$ for the rest of the section.

$$ReachRel_{\mathcal{H}}((q_1, P_1), P, (q_2, P_2)) = \{(x, y) \in \mathbb{R}^n \times \mathbb{R}^n \mid \exists \text{ finite execution } \sigma = (\iota, \eta, \gamma)$$
$$\in Exec(\mathcal{H}), \mathcal{I}(\iota) = [0, T], x = \eta(0) \in P_1, y = \eta(T) \in P_2, \eta(t) \in P \ \forall t \in (0, T),$$
$$\gamma(0) = q_1 \text{ and } \gamma(Last(dom(\iota))) = q_2\}$$

It captures the set of points $(x, y) \in P_1 \times P_2$ such that there exists an execution which starts at (q_1, x) and ends at (q_2, y) and remains in P at all intermediate time points. It is shown in [21] that the $ReachRel_{\mathcal{H}}$ is computable and can be represented as a $2n$-dimensional polyhedral set. Next, we define the predecessor and successor operators denoted by $pre_{\mathcal{H}}$ and $post_{\mathcal{H}}$, and their weighted counterparts $wpre_{\mathcal{H}}$ and $wpost_{\mathcal{H}}$, respectively.

- $pre_{\mathcal{H}}((q_1, P_1), P, (q_2, P_2)) = \{x \in P_1 \mid \exists y \in P_2 : (x, y) \in ReachRel_{\mathcal{H}}((q_1, P_1), P, (q_2, P_2))\}$
- $post_{\mathcal{H}}((q_1, P_1), P, (q_2, P_2)) = \{y \in P_2 \mid \exists x \in P_1 : (x, y) \in ReachRel_{\mathcal{H}}((q_1, P_1), P, (q_2, P_2))\}$
- $wpre_{\mathcal{H}}((q_1, P_1), P, w, (q_2, P_2)) =$
  $= \{x \in P_1 | \exists y \in P_2, (x, y) \in ReachRel_{\mathcal{H}}((q_1, P_1), P, (q_2, P_2)), \frac{||y||}{||x||} = w\}$
- $wpost_{\mathcal{H}}((q_1, P_1), P, w, (q_2, P_2)) =$
  $= \{y \in P_2 | \exists x \in P_1, (x, y) \in ReachRel_{\mathcal{H}}((q_1, P_1), P, (q_2, P_2)), \frac{||y||}{||x||} = w\}$

Here, w is a positive real number and e $||\cdot||$ denotes the infinity norm of an element in $\mathbb{R}^n$.

3 Stability

In this section, we define a classical notion of stability in control theory, namely, Lyapunov stability. We consider stability of the system with respect to the origin $\bar{0}$, which we assume is an equilibrium point. Intuitively, Lyapunov stability captures the notion that an execution starting close to the equilibrium point remains close to it. Le $B_\epsilon(\bar{0})$ be an open ball of radius ϵ around $\bar{0}$, which denotes $\{x \,|\, ||x|| < \epsilon\}$.

Definition 3. *A PSS $\mathcal{H}$ is said to be* Lyapunov stable*, if for every $\epsilon > 0$, there exists a $\delta > 0$ such that for every execution $\sigma = (\iota, \eta, \gamma) \in Exec(\mathcal{H})$ with $\eta(0) \in B_\delta(\bar{0})$, $\eta(t) \in B_\epsilon(\bar{0})$ for every $t \in \mathcal{I}(\iota)$.*

Observe that Lyapunov is a local property whose satisfaction depends on the behaviors of the system in a small neighborhood around the origin. Hence, the only polyhedral sets of the *PSS* which play a role in stability analysis are those which contain the $\bar{0}$. Therefore, we will assume without loss of generality that the *PSS* is in a normal form [20, 21].

Definition 4. *A polyhedral set P is* closed under positive scaling *if for every $x \in P$ and $\alpha > 0$, $\alpha x \in P$.*

Definition 5. *A PSS $\mathcal{H}$ is in* normal form *if for every $q \in Loc$ and for every $e \in Edges$, $Inv_{\mathcal{H}}(q)$ and $Guard_{\mathcal{H}}(e)$ are positive scaling closed.*

4 Counterexample guided abstraction refinement

In this section, we present the CEGAR framework for stability analysis. The algorithm is summarized in Algorithm 1. First, we briefly review the abstraction and model-checking algorithms for stability analysis of polyhedral switched systems from [21]. Then, we present the new validation and refinement algorithms.

4.1 Abstraction

The abstraction procedure is a modification of the standard predicate abstraction [9] which constructs a finite state system using a finite set of predicates, which simulates the concrete system. It was shown in [19] that stability is not preserved by simulation and instead stronger notions which strengthen the simulation relation with continuity conditions are required. Hence, the abstraction procedure in [20] constructs a finite weighted graph as illustrated in Figure 1. More precisely, the vertices of the graph correspond to pairs of location and facet of the partition (instead of the regions). An edge exists between two vertices if there exists an execution from one pair of location and facet to the other by remaining in the common region of the facets. Further, the weights on the edges store quantitative information, which track by what factor the execution moves closer to the origin when it reaches the target facet as compared to where it started on the source facet. Next, we present the formal construction of the abstract system, for what we introduce some auxiliary definitions.

Algorithm 1 CEGAR for stability analysis

Require: $\mathcal{H}, \mathcal{P}, \mathcal{F}$
Ensure: Stable/Unstable
if *Check-explosion*$(\mathcal{H}, \mathcal{P})$ **then**
  return Unstable
$\mathcal{A} = Abs(\mathcal{H}, \mathcal{P}, \mathcal{F})$
while true do
  $\pi :=$ *Model-checking*$(\mathcal{A})$
  if π not counterexample **then**
    return Stable
  $\psi :=$ *Encode-*$\psi_\pi(m > 1)$
  if *Check-satisfiability*(ψ) **then**
    return Unstable
  else
    $\psi :=$ *Encode-*$\psi_\pi(m \leqslant 1)$
    if *Check-satisfiability*(ψ) **then**
      $\mathcal{A} :=$*Weighted-refinement*$(\mathcal{A}, \pi)$
    else
      $\mathcal{A} :=$*Refinement*$(\mathcal{A}, \pi)$

Definition 6. *A* polyhedral partition $\mathcal{P}$ *of* $X \subseteq \mathbb{R}^n$ *is a finite set of closed convex polyhedral sets,* $\{P_1, \ldots, P_k\}$*, such that* $X = \cup_{i=1}^{k} P_i$ *and* $interior(P_i) \cap interior(P_j) = \emptyset$*, for* $1 \leqslant i, j \leqslant k$*.*

The elements of a polyhedral partition are referred to as regions. A polyhedral partition is said to *respect* a *PSS* $\mathcal{H}$ if for every $P \in \mathcal{P}$, $q \in Loc_{\mathcal{H}}$ and $e \in Edges_{\mathcal{H}}$, either $P \subseteq Inv_{\mathcal{H}}(q)$ or $P \cap Inv_{\mathcal{H}}(q) = \emptyset$ and either $P \subseteq Guard_{\mathcal{H}}(e)$ or $P \cap Guard_{\mathcal{H}}(e) = \emptyset$.

Definition 7. *A* facet partition $\mathcal{F}$ *of a polyhedral partition* $\mathcal{P}$ *is a polyhedral partition of* $\cup_{P \in \mathcal{P}} \partial(P)$*, where* $\partial(P)$ *is the boundary of* P*.*

Definition 8. *Let us fix a concrete PSS* $\mathcal{H}$*. Let* $\mathcal{P}$ *be a polyhedral partition of* X *and* $\mathcal{F}$ *be a facet partition of* $\mathcal{P}$*. The abstract system is the finite weighted graph* $Abs(\mathcal{H}, \mathcal{P}, \mathcal{F}) = (V, E, W)$ *defined as follows.*

- $V = Loc \times \mathcal{F}$.
- $E \subseteq V \times \mathcal{P} \times V$ *is* $\{((q_1, f_1), P, (q_2, f_2)) \mid ReachRel((q_1, f_1), P, (q_2, f_2)) \neq \emptyset\}$.
- $W : E \to \mathbb{R}_{\geq 0} \cup \{\infty\}$*, such that for* $e = ((q_1, f_1), P, (q_2, f_2)) \in E$*,*

$$W(e) = \sup\{||y||/||x|| \mid (x, y) \in ReachRel((q_1, f_1), P, (q_2, f_2))\}$$

The weight computation on the edges of the abstract graph can be constructed by solving an optimization problem on the reachability relation polyhedral set [20, 21].

4.2 Model-checking and counterexample generation

For every execution of the concrete system, there is a path in the weighted graph such that the product of the weights of its edges is an upper bound on the scaling of the execution - the ratio of the distance of its end point from the origin to the distance of its starting point from the origin. Therefore, the following theorem provides sufficient conditions on the finite weighted graph which imply stability of the concrete system. We say that a region is exploding in $\mathcal{H}$ if there exists an execution which always remains in the region and diverges (goes arbitrarily far from the origin). Consider a partition $\mathcal{P}$ which respects $\mathcal{H}$, then for every region $P \in \mathcal{P}$ there exists $q \in Loc_{\mathcal{H}}$ such that $P \subseteq Inv_{\mathcal{H}}(q)$. The region P is exploding in $\mathcal{H}$ in the case of $P \cap Flow_{\mathcal{H}}(q) \neq \emptyset$. Given a path π, let $W(\pi)$ denote the product of the weights on the edges of π.

Theorem 1. *[21] Let $\mathcal{H}$ be a PSS, $\mathcal{P}$ be a polyhedral partition respecting $\mathcal{H}$ and $\mathcal{F}$ be a facet partition of $\mathcal{P}$. Then, the PSS $\mathcal{H}$ is Lyapunov stable if for every simple cycle π, $W(\pi) \leq 1$ and there is no region in $\mathcal{P}$ which explodes in $\mathcal{H}$.*

The conditions on the abstract system can be efficiently checked [20, 21]. The model-checking procedure will either return that $\mathcal{H}$ is stable or in the case that the abstract system does not satisfy the conditions of Theorem 1 return an abstract counterexample in the form of a simple cycle with weight > 1 or say that the system has an exploding region. In the first case, we know that the system is stable, and in the third case, it is unstable. For the second case, the CEGAR algorithm proceeds to the validation phase.

Example. Consider the 3-dimensional *PSS* shown in Figure 1 (Left). Now, the picture on the right shows part of the abstract system. The nodes are superimposed over the facets they represent and the edges show the existence of an execution between such facets evolving through the common polyhedral set. For instance, we observe that there exists an execution from facet f_1 to f_2 evolving through the polyhedron C. The cycle shown is an abstract counterexample since the weight associated with it is greater than 1. Validation will check if there exists an actual execution along the cycle which can witness instability.

Remark 1. The conditions in Theorem 1 are, in fact, both necessary and sufficient in the case of 2-dimensional *PSS*s [23], however, it is only sufficient in 3 or more dimensions. There are two reasons for the conservativeness. First, the edges are not transitively closed, because they are existential with respect to the executions in the concrete system. More precisely, existence of an execution from a facet f_1 to f_2 and an execution from f_2 to f_3 does not imply that there is a single execution which goes from f_1 to f_2 to f_3. Secondly, a similar transitivity may not hold on the weights. Suppose that the weight on the edge from f_1 to f_2 is w_1 and from f_2 to f_3 is w_2. There exists an execution from some point in f_1 to some point in f_2 with scaling w_1 and an execution from some point in f_2 to a point in f_3 with weight w_2. However, there may not be a single execution from f_1 to f_3 through f_2 such that the scaling corresponding to the prefix from f_1 to f_2 is w_1, while that from f_2 to f_3 is w_2.

4.3 Validation

We present some preliminaries and define the validation problem. Next, the validation procedure and its theoretical basis are presented.

Definition 9. *A simple cycle π in $Abs(\mathcal{H}, \mathcal{P}, \mathcal{F})$ is an* abstract counterexample *if $W(\pi) > 1$.*

A. Validation Problem. Validation consists of checking if the abstract counterexample corresponds to a violation of stability in the concrete system. Let us fix a counterexample $\pi = (q_0, f_0), P_0, (q_1, f_1), P_1, \ldots, (q_{k-1}, f_{k-1}), P_{k-1}, (q_0, f_0)$ of $\mathcal{A} = Abs(\mathcal{H}, \mathcal{P}, \mathcal{F})$.The following definition states a connection between the abstract counterexample and the executions in the concrete system.

Definition 10. *An execution $\sigma = (\iota, \eta, \gamma)$ of $\mathcal{H}$ is said to* follow *the abstract counterexample π of $\mathcal{A} = Abs(\mathcal{H}, \mathcal{P}, \mathcal{F})$, denoted $\sigma \rightsquigarrow \pi$, if there exists a non-decreasing sequence of times $0 = t_0, t_1, t_2, \ldots$ such that $\eta(t_i) \in f_{i \bmod k}$, $\eta(t) \in P_i$ for $t \in [t_{(i-1) \bmod k}, t_{i \bmod k}]$ and $(\eta(t_i), \eta(t_{i+1})) \in ReachRel((q_i, f_i), P_i, (q_{i+1}, f_{i+1}))$. Further, σ is said to* follow π respecting the weights*, denoted $\sigma \overset{w}{\rightsquigarrow} \pi$, if in addition*

$$\frac{||\eta(t_{i+1})||}{||\eta(t_i)||} = W((q_{i \bmod k}, f_{i \bmod k}), P_{i \bmod k}, (q_{(i+1) \bmod k}, f_{(i+1) \bmod k})).$$

The following notion captures the violation of Lyapunov stability along π. The abstract counterexample π is a witness to the violation of Lyapunov stability by the concrete system $\mathcal{H}$ if there exist executions with arbitrary scaling which follow the cycle respecting the weights.

$$(C1)\, \exists \epsilon > 0, \forall \delta > 0, \exists \sigma \in Exec(\mathcal{H}) \text{ such that}$$

$$\sigma \overset{w}{\rightsquigarrow} \pi,\ \eta(0) \in B_\delta(\bar{0}), \exists t \in \mathcal{I}(\iota), \eta(t) \notin B_\epsilon(\bar{0})$$

The next proposition states that the above condition in fact implies that there is a complete execution along π.

Proposition 1. *Condition (C1) is equivalent to the existence of a complete execution σ of $\mathcal{H}$ such that $\sigma \overset{w}{\rightsquigarrow} \pi$.*

While (C1) can be validated exactly, a refinement corresponding to (C1) tries to eliminate just the executions which follow the weights on the edges of π exactly. In order to accelerate the progress in the CEGAR iterations, we consider a stronger validation problem, where we do not require the execution to follow the weights, but still be diverging.

Definition 11. *An abstract counterexample π is said to be* spurious *if there does not exist a divergent complete execution σ such that $\sigma \rightsquigarrow \pi$.*

Validation problem: Given an abstract counterexample π, is π spurious?

B. Validation procedure. The crux of the validation procedure is to reduce the problem of checking the existence of infinite executions to that of finite executions. Hence, for $m \in \mathbb{R}_{\geq 0}$ we define a predicate $\psi_\pi(m)$, which captures the set of points $x_0, \ldots, x_k$ such that x_k can be reached from x_0 by following the cycle once, and $x_k = mx_0$.

$$\psi_\pi(m) := \exists x_0, x_1, \ldots, x_k \in \mathbb{R}^n : x_k = mx_0, \forall 0 \leq i < k,$$

$$x_i \in f_i, (x_i, x_{i+1}) \in \mathit{ReachRel}((q_i, f_i), P_i, (q_{i+1}, f_{i+1})).$$

Next, we state the main theorem for validation.

Theorem 2. *The following holds for the abstract counterexample π:*

V1 $\exists m > 1 : \psi_\pi(m) \Rightarrow \exists \sigma \in \mathit{CExec}(\mathcal{H}) : \sigma \rightsquigarrow \pi \wedge \sigma$ diverges.
V2 $\nexists m : \psi_\pi(m) \Rightarrow \nexists \sigma \in \mathit{CExec}(\mathcal{H}) : \sigma \rightsquigarrow \pi$.
V3 $\exists m : \psi_\pi(m) \wedge \nexists m > 1 : \psi_\pi(m) \Rightarrow$
$\exists \sigma \in \mathit{CExec}(\mathcal{H}) : \sigma \rightsquigarrow \pi \wedge \nexists \sigma \in \mathit{CExec}(\mathcal{H}) : \sigma \overset{w}{\rightsquigarrow} \pi$.

Remark 2. Condition V1 implies that when there exists $m > 1$ such that $\psi_\pi(m)$ holds, the system is unstable. Condition V2 states that when there exist no m at all such that $\psi_\pi(m)$ is true, then the counterexample has no complete executions following it, and hence, is spurious. Condition V3 implies that there is no complete execution following π which respects the weight, however, there is some complete execution (diverging or not).

Before proving the result above, we introduce some definitions and a fixpoint theorem that we will use to establish an intermediate result. Let $\pi = (q_0, f_0), P_0, (q_1, f_1), P_1, \ldots, (q_{k-1}, f_{k-1}), P_{k-1}, (q_0, f_0)$ be an abstract counterexample of $\mathcal{A}$. Let $\mathit{PreReach}^i(S_0)$, for some $S_0 \subseteq f_0$, denote the set of points from which there is a sequence of length i following π which starts at S_0. Similarly, let $\mathit{WPreReach}^i(S_0)$ denote the points from which the executions also respect the weights. We introduce the formal definitions below. *PreReach* is defined as follows:

- $\mathit{PreReach}^0(S_0) = S_0$.
- For $i > 0$, $\mathit{PreReach}^i(S_0) = \mathit{pre}_{\mathcal{H}}((q_j, f_j), P_j, (q_{j+1}, \mathit{PreReach}^{i-1}(S_0)))$, where $j = k - (i \bmod k)$.

In addition to *WPreReach* we also define *WPostReach*.

- $\mathit{WPreReach}^0(S_0) = S_0$.
- $\mathit{WPreReach}^i(S_0) = \mathit{wpre}_{\mathcal{H}}((q_j, f_j), P_j, w_j, (q_{j+1}, \mathit{WPreReach}^{i-1}(S_0)))$, where $w_j = W((q_{j-1}, f_{j-1})(q_j, f_j))$, $i > 0$ and $j = k - (i \bmod k)$.
- $\mathit{WPostReach}^0(S_0) = S_0$.
- $\mathit{WPostReach}^i(S_0) = \mathit{wpost}_{\mathcal{H}}((q_j, f_j), P_j, w_j, (q_{j+1}, \mathit{WPostReach}^{i-1}(S_0)))$, where $w_j = W((q_{j-1}, f_{j-1})(q_j, f_j))$, $i > 0$ and $j = i \bmod k$

Theorem 3 (Kakutani's fixed point theorem). *Let $S \subseteq \mathbb{R}^n$ be a non empty, compact and convex set. Let $H : S \to 2^S$ be a set-valued function whose graph $\{(s, s') : s' \in H(s)\}$ is a closed set, and for all $s \in S$, $H(s) \neq \emptyset$ and convex. Then H has a fixed point, which means $\exists s^* \in S : s^* \in H(s^*)$.*

The existence of such kind of fixed point provides us a strategy for proving the next result.

Proposition 2. *If there exists $\sigma \in \mathit{CExec}(\mathcal{H})$ such that $\sigma \overset{w}{\rightsquigarrow} \pi$, then there exists a value m greater than 1 such that $\psi_\pi(m)$ holds.*

Proof. Suppose $\sigma \in \mathit{CExec}(\mathcal{H})$ such that $\sigma \overset{w}{\rightsquigarrow} \pi$. Let us first define a set of starting points for divergent executions following π respecting the weights. $\mathit{Kernel}(\pi) = \{x \in f_0 \mid \exists \sigma = (\iota, \eta, \gamma) \in \mathit{CExec}(\mathcal{H}) : \eta(0) = x, \sigma \overset{w}{\rightsquigarrow} \pi\}$.

$\mathit{Kernel}(\pi)$ is a closed convex set which is positive scaling closed. This follows from the following facts. Firstly, the facet f_0 is closed, convex and positive scaling closed, since it is a facet from a polyhedral partition respecting a *PSS* $\mathcal{H}$ in normal form. Next, the set $\mathit{Kernel}(\pi)$ is the intersection of $\mathit{WPreReach}^i(f_0)$ for $i \geqslant 0$ which is a multiple of k, the length of the counterexample π. (This depends on the fact that the set $Z = \bigcap_{i \bmod k = 0} \mathit{WPreReach}^i(f_0)$ has the property that $Z \subseteq \mathit{PreReach}^k(Z)$). Finally, the $\mathit{WPreReach}$ and intersection operations preserve the closedness, the convexity and the positive scaling property.

Consider a set-valued function G from f_0 to f_0 which maps $x_0 \in f_0$ to the set $\mathit{WPostReach}^k(x_0)$. Define $S = \{x \mid ||x|| \leq 1, x \in \mathit{Kernel}(\pi)\}$. Since $\mathit{Kernel}(\pi)$ is non-empty, convex, closed and closed under positive scaling, we obtain that S is non-empty, compact and convex. Compactness follows from the assumption that $\mathit{Kernel}(\pi)$ is closed and the set $||x|| \leq 1$ is compact, and hence, their intersection S is compact. The convexity of S follows from the fact that it is the intersection of the set $\mathit{Kernel}(\pi)$ and the set $||x|| \leq 1$, both of which are convex.

Define K as an upper bound for the scaling of the executions following π for one iteration and respecting the weights, so the ones from f_0 to $\mathit{WPostReach}^k(f_0)$, being k the length of π. Define the set valued function H from S to 2^S, which maps $x \in S$ to the set $\{\frac{y}{K} \mid y \in G(x)\}$.

Note that the graph $\{(x, y) \mid y \in G(x)\}$ is a closed set. Consider a sequence of points $(x_0, y_0), (x_i, y_i), \ldots$ which belong to the graph and converge to (x, y). Then x will be in the domain of the graph because of closedness of f_0. And $y \in G(x)$ because of compactness and linearity of every polyhedral set $\mathit{Flow}(q)$ for $q \in \mathit{Loc}$ which represents the dynamics.

Next, we show that H has a fixed point. For this, we apply the Kakutani's fixed point theorem. Since H defined above satisfies the hypothesis of Kakutani's theorem, there exists $s^* \in S$ such that $s^* \in H(s^*)$. Then, Note that $s^* \in \frac{G(s^*)}{K}$, it is $Ks^* \in G(s^*)$. Then the sequence of points $s^*, Ks^*, K^2s^*, \ldots$ holds $\psi_\pi(K)$, and $K > 1$ because it is an upper bound on the $W(\pi)$ and π is a counterexample, and $K^{j+1}s^* \in \mathit{WPostReach}^k(K^j s^*)$ for every $j \geqslant 0$. □

Next, we prove Theorem 2. Suppose π is an abstract counterexample.

V1 Suppose there exists $m > 1$ and $x_0, \ldots, x_k \in \mathbb{R}^n$ such that for all $0 \leq i < k$, $x_i \in f_i$ and $(x_i, x_{i+1}) \in \mathit{ReachRel}((q_i, f_i), P_i, (q_{i+1}, f_{i+1}))$, and $x_k = mx_0$. Then consider the infinite execution $\nu = x_0, \ldots, x_{k-1}, mx_0, \ldots, mx_{k-1}, m^2x_0, \ldots, m^2x_{k-1}, \ldots$ such that $(m^j x_i, m^j x_{i+1}) \in \mathit{ReachRel}((q_i, f_i), P_i, (q_{i+1}, f_{i+1}))$ for every $j \geqslant 0$ because of linearity of the flows. Construct with such points and π an execution σ such that $\sigma \rightsquigarrow \pi$. Note that σ diverges, since $m > 1$.

V2 It can show by using a similar argument that in the proof of Proposition 2 but defining $\mathit{Kernel}(\pi)$ as $\{x \in f_0 \mid \exists \sigma = (\iota, \eta, \gamma) \in \mathit{CExec}(\mathcal{H}) : \eta(0) = x, \sigma \rightsquigarrow \pi\}$.

V3 Suppose there exists $0 < m \leqslant 1$ and $x_0, \ldots, x_k \in \mathbb{R}^n$ such that for all $0 \leq i < k$, $x_i \in f_i$ and $(x_i, x_{i+1}) \in \mathit{ReachRel}((q_i, f_i), P_i, (q_{i+1}, f_{i+1}))$, and $x_k = mx_0$. Then consider the infinite execution $\nu = x_0, \ldots, x_{k-1}, mx_0, \ldots, mx_{k-1}, m^2x_0, \ldots, m^2x_{k-1}, \ldots$. Construct with such points and π an execution σ such that $\sigma \rightsquigarrow \pi$. Note that there does not exist σ respecting the weights in π because in case of existence we would get a contradiction due to Proposition 2. □

4.4 Refinement

First, we formalize the refinement problem. Then, we present different strategies for refinement by considering the reason for the spuriousness of the abstract counterexample.

A. Refinement problem. We first introduce the notion of refinement.

Definition 12. *Given two abstract systems for $\mathcal{H}$, $\mathcal{A} = \mathit{Abs}(\mathcal{H}, \mathcal{P}, \mathcal{F}) = (V, E, W)$ and $\mathcal{A}' = \mathit{Abs}(\mathcal{H}, \mathcal{P}, \mathcal{F}') = (V', E', W')$, $\mathcal{A}'$ is said to be a* refinement *of $\mathcal{A}$, if there exists a mapping $\alpha : V' \to V$ such that if $(v_1, P, v_2) \in E'$, then $(\alpha(v_1), P, \alpha(v_2)) \in E$, and $W'(v_1, P, v_2) \leq W(\alpha(v_1), P, \alpha(v_2))$.*

Next we associate a set of triples with an abstract system which captures the potential executions and scalings along the edges.

Definition 13. *Given an abstract system $\mathcal{A} = (V, E, W)$ of $\mathcal{H}$, $\mathit{Pot}(\mathcal{A}) = \{((q_1, x), w, (q_2, y)) \mid \exists((q_1, f_1), P, (q_2, f_2)) \in E, x \in f_1, y \in f_2, ||y||/||x|| = w \leq W((q_1, f_1), P, (q_2, f_2))\}$.*

Definition 14. *An abstract system $\mathcal{A}'$ of $\mathcal{H}$ is a* strict refinement *of an abstract system $\mathcal{A}$ of $\mathcal{H}$, if $\mathcal{A}'$ is a refinement of $\mathcal{A}$ and $\mathit{Pot}(\mathcal{A}')$ is a strict subset of $\mathit{Pot}(\mathcal{A})$.*

Refinement problem: Given the concrete system $\mathcal{H}$, an abstract system $\mathcal{A}$ of $\mathcal{H}$ and a spurious abstract counterexample of $\mathcal{A}$,namely π, find a strict refinement $\mathcal{A}'$ of $\mathcal{A}$.

Remark 3. Observe that if $\mathcal{F}'$ is a facet partition which is strictly finer than $\mathcal{F}$, then $\mathit{Abs}(\mathcal{H}, \mathcal{P}, \mathcal{F}')$ is a refinement of $\mathit{Abs}(\mathcal{H}, \mathcal{P}, \mathcal{F})$, however, it may not be a strict refinement of $\mathit{Abs}(\mathcal{H}, \mathcal{P}, \mathcal{F})$. Hence, it is crucial to exploit the spuriousness of the abstract counterexample π to construct a finer facet partition $\mathcal{F}'$ such that $\mathcal{A}' = \mathit{Abs}(\mathcal{H}, \mathcal{P}, \mathcal{F}')$ is a strict refinement of $\mathcal{A}$.

B. Refinement procedure. We present two different strategies for refinement based on the reason for the spuriousness. First, we show that non-existence of a complete execution along π (respecting the weights) implies that the *PreReach* (*WPreReach*) computation terminates.

Theorem 4. *Consider a PSS $\mathcal{H}$, an abstract system $\mathcal{A}$ of $\mathcal{H}$ and a counterexample π. Then*

R1 If $\nexists \sigma \in CExec(\mathcal{H})$ such that $\sigma \rightsquigarrow \pi \Rightarrow PreReach^i(f_0) = \emptyset$ for some i.
R2 If $\nexists \sigma \in CExec(\mathcal{H})$ such that $\sigma \overset{w}{\rightsquigarrow} \pi \Rightarrow WPreReach^i(f_0) = \emptyset$ for some i.

From Theorem 2, there are two reasons for spuriousness corresponding to Conditions V2 and V3. Statements R1 and R2 suggest the refinement strategies corresponding to V2 and V3.

Refinement strategy when the premise of V2 holds. Let ι be the smallest index such that $PreReach^{\iota}(f_0)$ empty. Note that $S_1 = PreReach^{\iota-1}(f_0)$ is not empty. Let the value $\hat{k} = k-(\iota \bmod k)$. Also, $(\hat{k}+1) \bmod k$ is the index of the facet which contains S_1. It also implies that the set $S_2 = post_{\mathcal{H}}((q_{\hat{k}}, f_{\hat{k}}), P_{\hat{k}}, (q_{(\hat{k}+1) \bmod k}, f_{(\hat{k}+1) \bmod k}))$ which is also a subset of $f_{(\hat{k}+1) \bmod k}$ has an empty intersection with S_1. Refinement corresponds to refining the facet $f_{(\hat{k}+1) \bmod k}$ into $\{f^1, f^2\}$ such that it separates S_1 and S_2, that is, $S_1 \subseteq f^1$ and $S_2 \subseteq f^2$, and $S_1 \cap f^2 = \bar{0}$ and $S_2 \cap f^1 = \bar{0}$. Such a splitting is always possible since S_1 and S_2 are two closed convex polyhedral sets whose intersection contains only $\bar{0}$ and hence, there exists a hyperplane which separates them.

An illustration of the refinement is shown in Figure 2. The system is partitioned by the two polyhedral sets C and E, in which the flow direction is determined by the dashed lines, pointing from facet f_1 to facet f_2 and from f_2 to facet f_3. Observe that after performing predecessor operation on f_3 once we reach S_2 in f_2, and predecessor reach set of S_2 in f_1 becomes empty. From f_1 the successor reach set is computed and intersected with f_2, obtaining S_1. The two sets S_1 and S_2 are almost disjoint but for $\bar{0}$ so they can be separated by a hyperplane. A choice of a separating hyperplane is $3x + 2z = 0$.

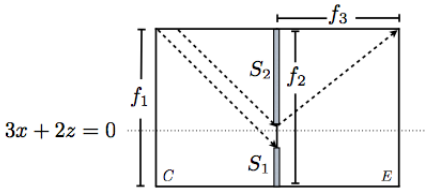

Fig. 2. Refinement

Proposition 3. *The abstract system $Abs(\mathcal{H}, \mathcal{P}, \mathcal{F}')$ is a strict refinement of the abstract system $Abs(\mathcal{H}, \mathcal{P}, \mathcal{F})$, where $\mathcal{F}' = (\mathcal{F} \backslash \{f_{(\hat{k}+1) \bmod k}\}) \cup \{f^1, f^2\}$.*

Proof. It follows from the fact that there is an edge from $(q_{\hat{k}}, f_{\hat{k}})$ to $(q_{(\hat{k}+1) \bmod k}, f^2)$, but no edge from $(q_{\hat{k}}, f_{\hat{k}})$ to $(q_{(\hat{k}+1) \bmod k}, f^1)$.

Refinement strategy when the premise of V3 holds The refinement is similar to the previous case, except that all the operators are replaced by their weighted counterparts, that is, *PreReach* is replaced by *WPreReach* and *post* by *wpost*. The

following proposition implying progress is similar to Proposition 3, however, the proof relies on the reduction of the weight rather than the removal of an edge.

Proposition 4. *The abstract system* $Abs(\mathcal{H}, \mathcal{P}, \mathcal{F}')$ *is a strict refinement of the abstract system* $Abs(\mathcal{H}, \mathcal{P}, \mathcal{F})$*, where* $\mathcal{F}' = (\mathcal{F} \backslash \{f_{(\hat{k}+1) \bmod k}\}) \cup \{f^1, f^2\}$.

Proof. Note that the weight of the edge $((q_{\hat{k}}, f_{\hat{k}}), P_{\hat{k}}, (q_{(\hat{k}+1) \bmod k}, f^1))$, if it exists, is less than the weight $w_{\hat{k}}$, the weight of the edge $((q_{\hat{k}}, f_{\hat{k}}), P_{\hat{k}}, (q_{(\hat{k}+1) \bmod k}, f_{(\hat{k}+1) \bmod k}))$.

Algorithm 1 summarizes the validation and refinement procedures. Line 8 checks if there exists an infinite diverging trajectory by constructing the formula $\psi_\pi(m > 1)$. If it is satisfiable, then a counterexample is found. If not, a refinement is required. However, to determine the type of refinement, the satisfiability of the formula $\psi_\pi(m \leq 1)$ is checked. If it is not satisfiable, then no infinite execution corresponding to the abstract counterexample exists, and we proceed with a non-weighted refinement. However, if $\psi_\pi(m \leq 1)$ is satisfiable, we know that an infinite execution exists, but we cannot conclude that it is diverging, hence, we proceed with a weighted refinement in Line 14.

5 Implementation

The validation procedure and the refinement strategies have been implemented in Python 2.7.3. We use Z3 SMT solver [7] for the validation, that is, checking the satisfiability of the formulas in Theorem 2; and use Parma Polyhedra Library (PPL) for performing polyhedral operations such as reachability computations in the refinement process. We also use Averist [22] for the abstraction and model-checking algorithms from [20].

We illustrate our Cegar algorithm on a particular class of polyhedral switched systems. The experiments are inspired by the example described in Figure 1. The 3-dimensional experiments consist of the same locations as the one in the example where the configurations of the flow function *Flow* are modified. The 4 and 5-dimensional experiments are obtained by extending every element of the example to higher dimensions.

Some of our results are summarized in Table 1. Here, *Exp* refers to the experiment number, *Dim* to the dimension of the concrete system (number of continuous variables) and *Stab* states whether the concrete system is Lyapunov stable (Y) or not (N). *Ans* is the output of the Cegar algorithm, which can be stable (S) (when the model-checking succeed), unstable (NS) (when the validation succeeds) or no answer (NA) (if the system does not terminate in a pre-set time). *Regions* is the number of regions in the polyhedral partition. *IT* refers to the number of iterations of the Cegar loop before termination, *Ref* indicates if weighted refinement strategy has been applied for some iteration, *Pre* states only predecessor reach computation and *WPre* indicates some weighted predecessor reach computation has been performed. *Size* refers to the number of nodes in the final weighted graph. The time for abstraction, *A time*, model-checking, *MC*

Exp	*Dim*	*Stab*	*Ans*	*Regions*	*IT*	*Ref*	*Size*	*A time*	*MC time*	*Val time*	*Ref time*	*Time*
1	3	Y	S	163	3	Pre	75	18.75	0.01	0.03	0.03	20.02
2	3	Y	S	287	11	Pre	153	196	0.23	0.84	0.30	204.50
3	3	N	NS	135	1	–	–	0	0	0	0	0.15
4	3	Y	NA	59	11	$WPre$	123	119.32	0.25	0.94	2.44	130.91
5	3	N	NS	9	1	–	16	0.30	ε	0.13	0	0.55
6	3	Y	S	151	2	$WPre$	74	12.55	ε	0.17	0.07	5.72
7	3	Y	S	179	4	Pre	87	31.63	0.02	0.03	0.04	33.62
8	3	Y	S	291	11	Pre	157	249.40	0.38	0.57	0.39	269.11
9	4	Y	S	537	3	Pre	341	312	0.32	0.32	0.10	319.42
10	4	Y	S	865	7	Pre	601	1543	2.13	1.18	0.28	1582.33
11	5	Y	S	1706	3	Pre	1365	4208	4.32	0.51	0.12	4252

Table 1. Experimental results for CEGAR algorithm

			AVERIST				CEGAR technique			
Exp	*Dim*	*Stab*	*Answer*	*Regions*	*Runs*	*Time*	*Answer*	*Regions*	*IT*	*Time*
2	3	Y	NA	85250	6	10658.26	S	287	11	204.50
2	3	Y	NA	4034	4	857.23	NA	59	11	130.91
3	3	Y	NA	4035	4	181.32	S	151	2	5.72
4	3	Y	NA	4035	4	187.24	S	179	4	33.62
5	4	Y	NA	27201	3	4728.61	S	537	3	319.42

Table 2. Comparison of AVERIST and CEGAR technique

time, validation, *Val time* and refinement, *Ref time*, are shown along with the total time *Time*. All the times are in seconds, and ε indicates a value smaller than 0.001.

A limit on the number of CEGAR iterations has been set to 11, but it can be set to any arbitrary value. The CEGAR procedure terminates on most of the examples that we are reporting. In the case of experiment 4 we do not obtain any answer, while in the case of experiments 3 and 5 we obtain instability. In the experiment 3, we observe instability due to an exploding region, therefore all the times are zero. In the experiment 5, the refinement is not performed because the validation algorithm returns the existence of a concrete counterexample. Our experiments illustrate that the CEGAR framework is practically feasible, since the times added by the validation and refinement procedures can be neglected if considering the total times.

Next, we compare the CEGAR algorithm with AVERIST. AVERIST allows specification of predicates as well as built-in automated methods for generating uniform predicates based on an input granularity value. In our comparison, we run our examples on AVERIST by iteratively increasing the number of predicates using this feature. Our CEGAR algorithm on the other hand applies the new refinement strategies based on the returned counterexamples for adding the predicates. We choose the termination criterion for CEGAR to be a bound of

11 on the number of iterations, and for AVERIST, we stop when the running time is more than 5 times the total time taken by the CEGAR algorithm. We compared all the examples in Table 1 with AVERIST, however, we present only a representative subset of them in Table 2. In Table 2, *Exp* refers to the experiment number, *Dim* to the dimension of the concrete system and *Stab* states whether the concrete system is Lyapunov stable (Y) or not (N). *Answer* is the algorithmic output, which can be stable (S), unstable (NS) or no answer (NA). *Regions* is the number of regions in the last polyhedral partition. *Runs* refers to the number of times AVERIST is run with an incremented number of uniform predicates, *IT* refers to the number of iterations of the CEGAR loop before termination and *Time* is the total time. As we observe from the experiments, AVERIST does not terminate on any of the examples within time 5 times that of the CEGAR algorithm. It shows that uniformly partitioning may be slower since the new predicates added are not necessarily useful towards constructing the right abstractions that are successful in stability analysis.

6 Conclusions

In this paper, we developed a counterexample guided abstraction refinement framework for the stability analysis of polyhedral switched systems. This approach explores the search space systematically by using counterexamples. To the best of our knowledge, this is the first CEGAR framework for stability analysis. Instantiating the CEGAR algorithm for stability analysis is non-trivial, since the notion of a counterexample is more involved, and the refinement is more expensive. Future work will focus on extending the ideas in the paper to more general classes of switched systems.

Acknowledgements. This work is partially supported by the Marie Curie Career Integration Grant no. 631622 and the NSF CAREER award no. 1552668 to Pavithra Prabhakar and by the research grant no. BES-2013-065076 from the Spanish Ministry of Economy and Competitiveness to Miriam García Soto.

References

1. R. Alur, C. Courcoubetis, T.A. Henzinger, and P. Ho. Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems. In *Hybrid Systems*, pages 209–229, 1992.
2. R. Alur, T. Dang, and F. Ivancic. Counter-Example Guided Predicate Abstraction of Hybrid Systems. In *Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 208–223, 2003.
3. S. Bogomolov, G. Frehse, M. Greitschus, R. Grosu, C. Pasareanu, A. Podelski, and T. Strump. Assume-Guarantee Abstraction Refinement Meets Hybrid Systems. In *International Haifa Verification Conference*, pages 116–131, 2014.
4. M.S. Branicky. Stability of hybrid systems: state of the art. In *Conference on Decision and Control*, pages 120–125, 1997.

5. E.M. Clarke, A. Fehnker, Z. Han, B. Krogh, J. Ouaknine, O. Stursberg, and M. Theobald. Abstraction and Counterexample-Guided Refinement in Model Checking of Hybrid Systems. *International Journal on Foundations of Computer Science*, 14(4):583–604, 2003.
6. E.M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-Guided Abstraction Refinement. In *Proceedings of the International Conference on Computer Aided Verification*, pages 154–169, 2000.
7. L. de Moura and N. Bjørner. Z3: An Efficient SMT Solver. In *Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340, 2008.
8. P.S. Duggirala and S. Mitra. Abstraction Refinement for Stability. In *International Conference on Cyber-Physical Systems*, pages 22–31, 2011.
9. S. Graf and H. Saidi. Construction of abstact state graphs with PVS. In *Proceedings of the International Conference on Computer Aided Verification*, pages 72–83, 1997.
10. J. Kapinski, J.V. Deshmukh, S. Sankaranarayanan, and N. Arechiga. Simulation-guided Lyapunov Analysis for Hybrid Dynamical Systems. In *Proceedings of the International Conference on Hybrid Systems: Computation and Control*, pages 133–142, 2014.
11. H.K. Khalil. *Nonlinear Systems*. Prentice-Hall, Upper Saddle River, NJ, 1996.
12. M. Kourjanski and P. Varaiya. Stability of Hybrid Systems. In *Hybrid Systems*, pages 413–423, 1995.
13. D. Liberzon. *Switching in Systems and Control*. Boston : Birkhäuser, 2003.
14. H. Lin and P.J. Antsaklis. Stability and Stabilizability of Switched Linear Systems: A Survey of Recent Results. *IEEE Transactions on Automatic Control*, 54(2):308–322, 2009.
15. E. Möhlmann and O.E. Theel. Stabhyli: a tool for automatic stability verification of non-linear hybrid systems. In *Proceedings of the International Conference on Hybrid Systems: Computation and Control*, pages 107–112, 2013.
16. J. Oehlerking, H. Burchardt, and O.E. Theel. Fully Automated Stability Verification for Piecewise Affine Systems. In *Proceedings of the International Conference on Hybrid Systems: Computation and Control*, pages 741–745, 2007.
17. P.A. Parrilo. *Structure Semidefinite Programs and Semialgebraic Geometry Methods in Robustness and Optimization*. PhD thesis, California Institute of Technology, Pasadena, CA, May, 2000.
18. P. Prabhakar, S. Duggirala, S. Mitra, and M. Viswanathan. Hybrid Automata-based CEGAR for Rectangular Hybrid Automata. In *Proceedings of the International Conference on Verification, Model Checking, and Abstract Interpretation*, 2013.
19. P. Prabhakar, G.E. Dullerud, and M. Viswanathan. Pre-orders for Reasoning about Stability. In *Proceedings of the International Conference on Hybrid Systems: Computation and Control*, pages 197–206, 2012.
20. P. Prabhakar and M.G. Soto. Abstraction Based Model-Checking of Stability of Hybrid Systems. In *Proceedings of the International Conference on Computer Aided Verification*, pages 280–295, 2013.
21. P. Prabhakar and M.G. Soto. An algorithmic approach to stability verification of polyhedral switched system. In *American Control Conference*, 2014.
22. P. Prabhakar and M.G. Soto. AVERIST: An Algorithmic Verifier for Stability. *Electronic Notes in Theoretical Computer Science*, 317:133–139, 2015.
23. P. Prabhakar and M. Viswanathan. On the Decidability of Stability of Hybrid Systems. In *Proceedings of the International Conference on Hybrid Systems: Computation and Control*, 2013.

24. C.A. Yfoulis and R. Shorten. A Numerical Technique for Stability Analysis of Linear Switched Systems. In *Proceedings of the International Conference on Hybrid Systems: Computation and Control*, pages 631–645, 2004.