Abstraction based verification of stability of polyhedral switched systems

Miriam García Soto[a,1,*], Pavithra Prabhakar[b,2]

[a]*IST Austria, Am Campus 1, 3400 Klosterneuburg, Austria*
[b]*Kansas State University, 2176 Engineering Hall, Manhattan, KS 66506, USA*

Abstract

This paper presents a novel abstraction technique for analyzing Lyapunov and asymptotic stability of polyhedral switched systems. A polyhedral switched system is a hybrid system in which the continuous dynamics is specified by polyhedral differential inclusions, the invariants and guards are specified by polyhedral sets and the switching between the modes do not involve reset of variables. A finite state weighted graph abstracting the polyhedral switched system is constructed from a finite partition of the state-space, such that the satisfaction of certain graph conditions, such as the absence of cycles with product of weights on the edges greater than (or equal) to 1, implies the stability of the system. However, the graph is in general conservative and hence, the violation of the graph conditions does not imply instability. If the analysis fails to establish stability due to the conservativeness in the approximation, a counterexample (cycle with product of edge weights greater than or equal to 1) indicating a potential reason for the failure is returned. Further, a more precise approximation of the switched system can be constructed by considering a finer partition of the state-space in the construction of the finite weighted graph. We present experimental results on analyzing

*Corresponding author

Email addresses: miriam.garciasoto@ist.ac.at (Miriam García Soto), pprabhakar@ksu.edu (Pavithra Prabhakar)

[1]This author was partially supported by BES-2013-065076 grant from the Spanish Ministry of Economy and Competitiveness and by S11402-N23 (RiSE/SHiNE) and Z211-N23 (Wittgenstein Award) grants from the Austrian Science Fund (FWF).

[2]This author was partially supported by NSF CAREER Award No. 1552668 and ONR YIP Award No. N00014-17-1-257.

stability of switched systems using the above method.

Keywords: polyhedral switched systems, stability verification, abstraction, model-checking techniques, counterexample generation

---

1. INTRODUCTION

Stability is a fundamental property in control design. It captures the notion that small perturbations in the initial state or input to the system result in only small variations in the behavior of the system. In this paper, we focus on the problem of automated stability verification of switched systems.

Switched systems [1] are a special class of hybrid systems [2] - systems exhibiting mixed discrete continuous behaviors - in which the continuous state of the system does not change during a mode switch. Switched systems are a natural model in supervisory control, wherein the plant consists of a finite number of operational modes, and the supervisor continuously observes the state of the system and takes decisions regarding the mode switches. Stability has been extensively investigated in the context of switched systems, and several sufficient conditions on the system and the switching behavior which ensure stability have been proposed (see [1, 3] and references therein).

One of the widely used approaches to stability verification of switched system is based on the notions of common and multiple Lyapunov functions [4, 5, 3, 6, 7, 8], wherein either a common function which acts as a Lyapunov function for each mode or a distinct function serving as a Lyapunov function for each mode along with consistency conditions on the switching, is sought. Automated verification of stability based on Lyapunov function can be characterized as deductive verification in the formal methods terminology. It encompasses a search for a Lyapunov function based on a template, such as a polynomial with coefficients as parameters, which serves as a candidate function. The requirements of Lyapunov function are encoded as a sum-of-squares programming problem over the template, which can be efficiently solved using tools such as SOSTOOLS [9, 10, 11].

One of the major limiting factors of the template based search is the ingenuity required in providing the right texmplates; and automatically learning the templates, with the exception of [12], is a challenge which has not been adequately addressed. Moreover, if a template fails to satisfy the conditions of Lyapunov function, then it does not provide insights into the potential reasons for instability or towards the choice of a better template. To overcome

these limitations, we propose an algorithmic approach — graph exploration based algorithm — that consists of constructing a weighted graph whose analysis either determines stability or returns a *counterexample* indicating a potential reason for failure. Further, the counterexample can provide insights into the choice of subsequent abstractions (refinement); we do not explore here automated refinement techniques based on counterexamples.

In this paper, we focus on the class of polyhedral switched systems (*PSS*). These are systems in which the invariants for the modes and the guards on the switching are convex polyhedral sets; further, the dynamics in each mode is specified as a polyhedral differential inclusion $\dot{x} \in P$, where P is a compact convex polyhedral set. The core contribution is the development of a quantitative predicate abstraction technique over *PSS*. Various methods include predicate and hybrid abstractions [13, 14, 15, 16] for computing abstract models oriented to safety verification. These standard methods do not suffice for stability analysis, so we introduce a novel predicate abstraction technique for stability preservation.

Our approach consists of constructing a finite weighted graph which represents a conservative approximation of the switched system, and inferring stability by analyzing certain properties of the graph. The algorithm takes as input a *PSS* $\mathcal{H}$ and a finite partition $\mathcal{P}$ of the state-space into convex polyhedral sets (so-called elements), and outputs a finite weighted graph $\mathcal{G}$. The vertices of the graph correspond to pairs consisting of a mode of the system and an element of the partition. An edge between two mode-element pairs indicates the existence of an execution starting from the first mode and a point on the first element to the second mode and a point on the second element such that it remains in a single element at all the intermediate time instances. The weight on an edge corresponds to the maximum scaling between the starting and the ending continuous states over all such executions, that is, an upper bound on the ratio of the distance to the equilibrium point at the end of the execution to that at the beginning. Hence, corresponding to every execution of the system, there exists a path in the graph which tracks the scalings associated with various time points in the execution. In particular, the existence of an edge with weight $+\infty$ implies a possibility of a diverging execution. Similarly, existence of a cycle in the graph such that the product of the weights is strictly greater than 1, implies the possibility of a diverging execution obtained by traversing the cycle infinitely many times. Absence of the above entities in the graph implies Lyapunov stability. We provide criteria based on graph analysis, which provide sufficient conditions

for Lyapunov and asymptotic stability.

One interesting feature of this analysis is a potential counterexample in the event of a failure to prove stability. For example, a cycle such that the product of the weights on the edges > 1 is a potential counterexample for Lyapunov stability. Another interesting feature is the ability to construct a less conservative abstraction, by considering a finer partition. A finer partition can be obtained, for example, by splitting each element in the current partition based on a linear constraint.

Construction of the finite weighted graph involves computing a non-trivial reachability predicate which captures all pairs of states of the system for which there is an execution from the first state to second while remaining within a single element of the partition. Existence of an edge then corresponds to satisfiability of the predicate and the weight corresponds to solving an optimization problem over the predicate. We show that we can construct a formula which is a boolean combination of linear constraints which is equivalent to the reachability predicate and hence compute the weight by solving a finite set of linear programming problems. The construction of the formula is involved owing to the fact that the number of mode switches that can occur during an execution within an element of the partition is unbounded due to the presence of cycles in the underlying switching graph. We reduce the analysis to that of an acyclic graph using the notion of strongly connected components and hence, bound the number of switches for the purpose of analysis.

The algorithm has been implemented in a tool called Averist (Algorithmic VERIfier for STability) [17]. We illustrate the merits of the algorithmic approach on an example using the tool. We employ variations of the example for evaluating the performance of the tool. The tool returns either a stability proof or a counterexample. The counterexample either leads to an instability answer or can be used to construct new constraints in order to refine the partition and obtain a more accurate one.

Currently, the choice of the appropriate predicates is carried out mainly through manual examination of the counterexample. There exists work focused on automating this process through a counterexample guided abstraction refinement approach [18].

Our choice of polyhedral switched systems is motivated by the fact that certain computations involved in our analysis are simple for this class, such as the representation of the reachability predicate. However, several classes of hybrid systems can be efficiently abstracted to this class [19, 20].

A preliminary version of this paper, appeared in [21]. In comparison to the former version, this work contains more detailed explanations, complete proofs and elaborate experiments. In particular, the replacement of executions by piecewise executions in stability definitions is justified in here by stating Proposition 2 and adding its proof. Proposition 3 and its proof appear only as an observation in the previous work. The equivalence in terms of stability between a polyhedral switched system and its normal form is proved in detail. Proofs of Theorem 7 and Theorem 8 are included. Also Algorithm 1 for abstraction construction and Algorithm 2 for scaling computation are added, accompanied by details of the functions contained in them. Table 1 with experimental results does not appear in the preliminary version.

2. Preliminaries

Let $\mathbb{R}$, $\mathbb{R}_{\geqslant 0}$, and $\mathbb{N}$ denote the set of reals, non-negative reals and natural numbers, respectively. Given a function F, we use $\mathtt{dom}(F)$ to denote the domain of F. Given a set $A \subseteq \mathtt{dom}(F)$, we denote by $F{\restriction_A}$, the restriction of F to the domain A.

Sequences: A *sequence domain* is a either a finite subset of $\mathbb{N}$ of the form $\{0, 1, \ldots, n\}$ for some $n \in \mathbb{N}$ or the infinite set $\mathbb{N}$. A *sequence* over a set A is a function $S : D \to A$, where D is a sequence domain. The *length* of a sequence S, denoted $\mathtt{len}(S)$, is the least upper bound of the elements in $\mathtt{dom}(S)$, that is, $\mathtt{last}(\mathtt{dom}(S))$. S is also represented by enumerating its elements as in $S(0)S(1)\ldots$.

Intervals, time domain and interval domain: An *interval* is a closed convex subset of $\mathbb{R}$. A *time domain* is an interval I such that $\mathtt{first}(I) = 0$. The concatenation of a time domain I_2 to a finite time domain I_1 is the time domain I, denoted $I_1 \circ I_2$, given by $I_1 \cup (I_2 + \{\mathtt{last}(I_1)\})$. An *interval domain* is a finite or infinite sequence of intervals $\iota = I_0 I_1 \cdots$ such that $\mathtt{first}(I_0) = 0$, $\mathtt{last}(I_i) = \mathtt{first}(I_{i+1})$ for $0 \leq i < \mathtt{len}(\iota)$, and if $\mathtt{dom}(\iota)$ is infinite, then for every $n \in \mathbb{N}$, there exists $m \in \mathbb{N}$ such that $n \in I_m$. We denote by $[\![\iota]\!]$ the interval $\cup_{i \in \mathtt{dom}(\iota)} \iota(i)$. The concatenation of an interval domain ι_2 to an interval domain ι_1 with $\mathtt{dom}(\iota_1) < +\infty$, denoted $\iota_1 \circ \iota_2$, is the interval domain ι whose domain is $\mathtt{dom}(\iota_1) \cup (\mathtt{dom}(\iota_2) + \mathtt{len}(\iota_1))$, and $\iota(i) = \iota_1(i)$ if $i \in \mathtt{dom}(\iota_1)$, and is $\iota_2(i - \mathtt{len}(\iota_1)) + \{\mathtt{last}(\iota_1(\mathtt{len}(\iota_1)))\}$, otherwise.

Splitting of sequences, intervals and interval domains: Sequences S_1 and S_2 form a splitting of a sequence S, denoted $S = S_1 \circ S_2$, if $S_1(\mathtt{len}(S_1)) = S_2(0)$, $\mathtt{len}(S) = \mathtt{len}(S_1) + \mathtt{len}(S_2)$, and $S(i) = S_1(i)$ if $i \in \mathtt{dom}(S_1)$, and

$S_2(i - \texttt{len}(S))$, otherwise. Note that for a given index of S, there is a unique way to split it into two sequences.

Intervals I_1 and I_2 form a splitting of an interval I, denoted $I = I_1 \circ I_2$, if $\texttt{last}(I_1) = \texttt{first}(I_2)$, and $I = I_1 \cup I_2$. Interval domains ι_1 and ι_2 form a splitting of an interval domain ι, denoted $\iota = \iota_1 \circ \iota_2$, if $\texttt{len}(\iota) = \texttt{len}(\iota_1) + \texttt{len}(\iota_2)$, and $\iota(i) = \iota_1(i)$ if $i < \texttt{len}(\iota_1)$ and $\iota(i) = \iota_2(i - \texttt{len}(\iota_1)) + \{\texttt{last}(\iota_1(\texttt{len}(\iota_1)))\})$ otherwise.

Euclidean space: The n-dimensional Euclidean space is given by $\mathbb{R}^n$. We use $|x|$ to denote the infinity norm of $x \in \mathbb{R}^n$ and $x \cdot y$ to denote the dot product of $x, y \in \mathbb{R}^n$. Given $\epsilon \in \mathbb{R}_{\geqslant 0}$, we use $B_\epsilon(x)$ to denote an open ball around x of radius ϵ, that is, $B_\epsilon(x) = \{y \,|\, |x-y| < \epsilon\}$. A set $S \subseteq \mathbb{R}^n$ is *open* if for every $x \in S$, there exists a $\delta > 0$ such that $B_\delta(x) \subseteq S$; and a set $S \subseteq \mathbb{R}^n$ is closed if $\mathbb{R}^n \backslash S$ is open. Note that open sets are closed under arbitrary union and closed sets are closed under arbitrary intersection. Hence, given any set $S \subseteq \mathbb{R}^n$, the *interior* of S, denoted $\mathring{S}$, is the largest open subset of S; and the *closure* of S, denoted $\overline{S}$ is the smallest closed set containing S. Given sets $X, Y \subseteq \mathbb{R}^n$, $X + Y$ denotes the Minkowski sum $\{x + y \,|\, x \in X, y \in Y\}$. Given a set $X \subseteq \mathbb{R}^n$, we use $\texttt{chull}(X)$ to denote the smallest convex set containing X.

Convex polyhedral sets: A *linear constraint* is an expression of the form $a \cdot x + b \sim 0$, where $a \in \mathbb{R}^n$, $x = (x_1, \cdots, x_n)$ is a tuple of variables, $b \in \mathbb{R}$ and relation $\sim \in \{<, \leq, =\}$. It is called *homogeneous* if $b = 0$. The set defined by a linear constraint $c \equiv a \cdot x + b \sim 0$, denoted $[\![c]\!]$, is the set of valuations $v = (v_1, \ldots, v_n) \in \mathbb{R}^n$, such that $a \cdot v + b \sim 0$ holds. A *half-space* is a set defined by a linear constraint with inequality relation and a *hyperplane* is a set defined by a linear constraint with equality, so it has dimension $n-1$. Note that a half-space is either of the two parts into which a hyperplane divides the space. We denote the set of all hyperplanes of space $\mathbb{R}^n$ by $\texttt{hyper}(n)$. Given a set $p \in \mathbb{R}^n$ of dimension lower than n, $\texttt{plane}(p)$ is the intersection of all hyperplanes which contain p, $\texttt{plane}(p) = \bigcap_{h \in \texttt{hyper}(n),\, p \subseteq h} h$, while in case of dimension equal to n, $\texttt{plane}(p) = \mathbb{R}^n$. A *convex polyhedral set* is an intersection of finitely many half-spaces and hyperplanes; it is said to be *pointed* if the half-spaces and hyperplanes are defined by homogenous linear constraints and *elementary* if the linear constraints defining the half-spaces only use the relation $\{<\}$. We will use $\texttt{polysets}(X)$ to denote the set of all convex polyhedral subsets of X, and $\texttt{cpolysets}(X)$ to denote the set of compact convex polyhedral sets. In case of $X = \mathbb{R}^n$ we denote $\texttt{polysets}(n)$

and $\texttt{cpolysets}(n)$.

A *partition* $\mathcal{P}$ of $S \subseteq \mathbb{R}^n$ into convex polyhedral sets is a finite set of convex polyhedral sets $\{P_1, \cdots, P_k\}$ such that $\cup_{i=1}^{k} P_i = S$ and for each $i \neq j$, $\mathring{P}_i \cap \mathring{P}_j = \emptyset$. An *elementary partition* is a partition in which all the convex polyhedral sets are elementary; the sets are referred to as elements.

Linear, piecewise-linear and differentiable functions: A *trajectory* η is a function from a time domain D to $\mathbb{R}^n$ for some n. A trajectory η is said to be *linear* if there exist $a, b \in \mathbb{R}^n$, such that for every $t \in \texttt{dom}(\eta)$, $\eta(t) = at + b$; and *piecewise-linear* if there exists an interval domain ι such that $[\![\iota]\!] = \texttt{dom}(\eta)$ and $\eta\lfloor_{\iota(i)}$ is linear for every $i \in \texttt{dom}(\iota)$. A trajectory η is said to be *differentiable* if its derivative exists at all point of the domain, and the derivative of η is denoted by $\dot{\eta}$.

Graphs and weighted graphs: A *graph* $\mathcal{G}$ is a pair (V, E), where V is a finite set of vertices and $E \subseteq V \times V$ is a finite set of edges. Cardinality of the graph is denoted as $|\mathcal{G}|$ and corresponds to the number of nodes. A *path* of a graph is a finite or infinite sequence of vertices $\pi = v_0 v_1 \ldots$ such that (v_i, v_{i+1}) is an edge for each $i < \texttt{len}(\pi)$. Let $\texttt{paths}(\mathcal{G})$ denote the set of paths of $\mathcal{G}$. A *cycle* is a finite path where the first and the last vertices are the same; and it is *simple* if all the vertices except the last are distinct.

A *weighted graph* $\mathcal{G} = (V, E, \texttt{w})$ where (V, E) is a graph and $\texttt{w} : E \to \mathbb{R}_{\geqslant 0} \cup \{+\infty\}$ is a weighting function on the edges. The weight of a finite path π is $\texttt{w}(\pi) = \Pi_{i \in \texttt{dom}(\pi)} \texttt{w}(\pi(i))$. Given a set of vertices $V' \subseteq V$, we use $\mathcal{G}[V']$ to represent $(V', E \cap V' \times V', \texttt{w}\lfloor_{V' \times V'})$.

A *strongly connected component* $\texttt{scc}$ of a graph $\mathcal{G}$ is a set of vertices V' of G such that for every $v_1, v_2 \in V'$, there is a path in $\mathcal{G}[V']$ from v_1 to v_2. A strongly connected component is *maximal* if there is no bigger strongly connected component containing it. Let $\texttt{scc}(\mathcal{G})$ denote the set of maximal strongly connected components of $\mathcal{G}$. Note that $\texttt{scc}(\mathcal{G})$ is a partition of the vertices of $\mathcal{G}$. The *quotient graph* $\mathcal{G}/\texttt{scc}$ is a graph where the set of vertices is $\texttt{scc}(\mathcal{G})$ and the edges correspond to pairs $(C_1, C_2) \in \texttt{scc}(\mathcal{G})$ such that there exist $c_1 \in C_1$ and $c_2 \in C_2$ with (c_1, c_2) an edge of $\mathcal{G}$. Note that the quotient graph is an acyclic graph.

Satisfiability Modulo Theory (SMT): An SMT formula $\varphi(x)$ over linear real arithmetic is a boolean combination of linear constraints over the variable in x. We will refer to it simply as an SMT formula from now on. A valuation $v \in \mathbb{R}^n$ is said to *satisfy* an SMT formula $\varphi(x)$, denoted $v \models \varphi(x)$, over variables $x = (x_1, \ldots, x_n)$, if the constraint obtained by substituting v_i to x_i is true. An SMT formula $\varphi(x)$ is *satisfiable* if there exists a valuation which

satisfies it. Checking satisfiability of an SMT formula over linear arithmetic is decidable and there exist efficient tools [22] to compute it.

The following observation will be used later.

Proposition 1. *Given a polyhedral set* $P \in \texttt{polysets}(n)$ *defined by the linear constraints* $a_i \cdot x \sim b_i$ *for* $1 \leq i \leq k$*, an* n*-tuple of variables* $x = (x_1, \ldots, x_n)$ *and a variable* t*, the set of all valuations for* x *and* t *satisfying the constraints* $x/t \in P$ *can be represented by the following SMT formula:* $\bigwedge_{i=1}^{k} a_i \cdot x \sim b_i t$.

3. Polyhedral Switched Systems

A switched system [1] models supervisory control in which the supervisor observes the state of the system and switches between a finite number of operational modes of the system. In each mode, the continuous state evolves according to a pre-assigned continuous dynamics and satisfies certain invariant conditions. Mode switch occurs when certain guards are satisfied, and in particular, the continuous state remains the same during the switch. We focus on the class of switched systems in which the continuous dynamics is specified using polyhedral differential inclusions, and the invariants and guards are specified using convex polyhedral sets.

Definition 1. An n-dimensional *polyhedral switched system* (*PSS*) is a tuple $\mathcal{H} = (Q, E, X, F, I, G)$, where:

- Q is a finite set of control modes or locations;
- $E \subseteq Q \times Q$ is a finite set of edges;
- $X = \mathbb{R}^n$, for some n, is the continuous state space;
- $F : Q \to \texttt{cpolysets}(n)$ is the flow function;
- $I : Q \to \texttt{polysets}(n)$ is the invariant function; and
- $G : E \to \texttt{polysets}(n)$ is the guard function.

Notation 1. 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 $I_{\mathcal{H}}$.

Figure 1 shows an automaton which represents a 2-dimensional polyhedral switched system with 5 modes, q_1, q_2, q_3, q_4 and q_5. The continuous state space corresponds to $X = \mathbb{R}^2$ and (x, y) represents the continuous state of the system. Every mode q_i is related to a convex polyhedral set I_i which is the invariant, that is $I(q_i) = I_i$. The continuous state belongs to I_i in mode q_i. Also every mode q_i is related to a compact convex polyhedral set F_i which specifies the flow function, $F(q_i) = F_i$. The derivative with respect to time of the continuous state, $(\dot{x}, \dot{y})$ belongs to F_i when in mode q_i. The edges of the system correspond to the arrows between the modes. These edges are tagged with predicates of the form $(x, y) \in G_i$ where G_i are convex polyhedral sets determined by the guard function. The predicates determine the condition when a switch between modes can be committed. Figure 2a and Figure 2b are two graphical instances of the *PSS* in Figure 1. The invariants are determined as follows, $I_1 = \{(x, y) \in X : x > 0, y > 0\}$, $I_2 = \{(x, y) \in X : x < 0, y > 0\}$, $I_3 = \{(x, y) \in X : x < 0, y < 0\}$, $I_4 = \{(x, y) \in X : x > 0, y < 0\}$ and $I_5 = \{(x, y) \in X : x - 2y > 0, y > 0\}$. The polyhedra F_i are determined by the convex combination of the vectors depicted in Figure 2. Observe that in Figure 2 the difference between the stable and unstable instances arises from the description of F_5. This corresponds to the flow of the system in the wedge-shaped region I_5. The 3 different guards tagging the edges are determined by the following predicates in both instances, $G_1 = \{(x, y) : x = 0\}$, $G_2 = \{(x, y) : y = 0\}$, and $G_3 = \{(x, y) : x - 2y \geqslant 0\}$.

The switched system starts in a location q and a continuous state x. In a mode q, the continuous state evolves inside $I(q)$ such that the differential of the evolution at anytime lies within $F(q)$. The mode can switch from q_1 to q_2 if (q_1, q_2) is an edge of the system and the continuous state at the switching satisfies the guard associated with the edge. Switched systems differ from a hybrid system in that the continuous state does not change in a switched system during a mode switch.

3.1. Semantics

Next, we present the semantics of a polyhedral switched system as a set of executions of the system.

Definition 2. An *execution* σ of a *PSS* $\mathcal{H} = (Q, E, X, F, I, G)$ of dimension n is a triple (ι, η, γ), such that:

- ι is an interval domain;

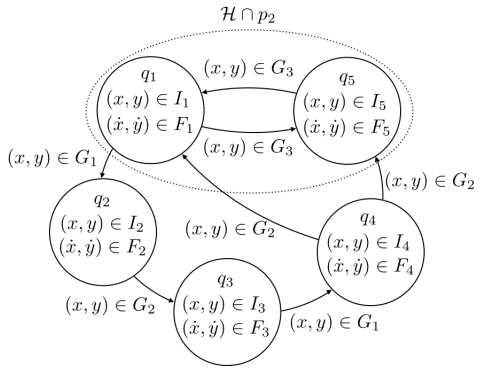
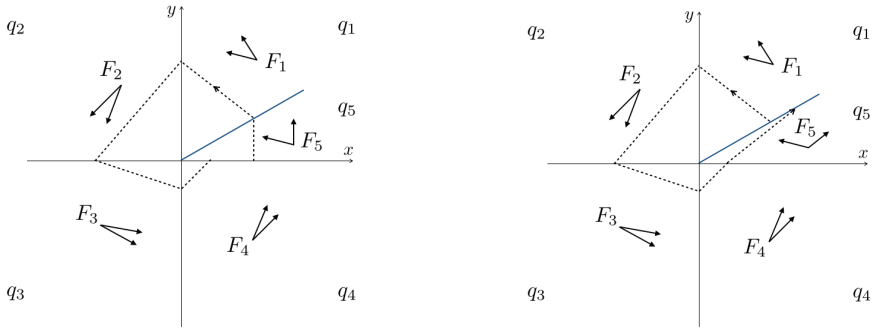


Figure 1: Polyhedral switched system automaton



(a) Stable (b) Unstable

Figure 2: Instances of polyhedral switched system in Figure 1

- $\eta : [\![\iota]\!] \to X$ such that for each $i \in \texttt{dom}(\iota)$, $\eta \restriction_{\iota(i)}$ is a differentiable function;
- $\gamma : \texttt{dom}(\iota) \to Q$ such that:
  - for all $i \in \texttt{dom}(\iota)$, for all $t \in \iota(i)$, $\eta(t) \in I(\gamma(i))$ and $\dot{\eta}(t) \in F(\gamma(i))$;
  - for all $0 \leq i < \texttt{len}(\iota)$, $(\gamma(i), \gamma(i+1)) \in E$ and $\eta(\texttt{last}(\iota(i))) \in G((\gamma(i), \gamma(i+1)))$.

We denote the set of all executions of $\mathcal{H}$ by $\texttt{exec}(\mathcal{H})$. The intersection between an execution $\sigma = (\iota, \eta, \gamma)$ and a set $A \subseteq X$ is defined as $\sigma \cap A = \{\eta(t) \mid t \in [\![\iota]\!]\} \cap A$, and extended to $\texttt{exec}(\mathcal{H}) \cap A = \cup_{\sigma \in \texttt{exec}(\mathcal{H})}(\sigma \cap A)$.

Definition 3. An execution $\sigma = (\iota, \eta, \gamma)$ of $\mathcal{H}$ is said to be *complete* if $\texttt{dom}(\eta) = [0, \infty)$; otherwise, it is called *finite*.

Every execution of the *PSS* in Figure 1 evolves inside I_i when located in control mode q_i by following some of the vectors in F_i. Notice that the invariants of locations q_1 and q_5 are overlapped and in the overlapped region, which coincides with the wedge-shaped region, the polyhedral set determining the dynamics consists of the convex hull of the polyhedra F_1 and F_5. In Figure 2, two sample finite executions are shown. They correspond to the dotted lines.

Notation 2. We will use $\texttt{fs}(\sigma)$ to refer to the first state $(\gamma(0), \eta(0))$. If σ is finite, we will use $\texttt{ls}(\sigma)$ to refer to the last state $(\gamma(\texttt{len}(\gamma)), \eta(\texttt{last}([\![\iota]\!])))$. Given an execution σ_i, its components will be referred to as $(\iota_i, \eta_i, \gamma_i)$.

3.2. Piecewise-linear executions and Splitting

Definition 4. An execution $\sigma = (\iota, \eta, \gamma)$ is said to be *piecewise-linear* if η is piecewise linear.

We denote the set of all piecewise-linear executions of $\mathcal{H}$ by $\texttt{pwexec}(\mathcal{H})$. The next proposition states that if there exists an execution of $\mathcal{H}$ between two states, then there also exists a piecewise linear execution between the states.

Proposition 2. *[Lemma 1 in [23]] Let $\mathcal{H}$ be a polyhedral switched system. Given any finite execution $\sigma \in \texttt{exec}(\mathcal{H})$, there exists a piecewise-linear execution $\sigma' \in \texttt{pwexec}(\mathcal{H})$ such that $\texttt{fs}(\sigma) = \texttt{fs}(\sigma')$ and $\texttt{ls}(\sigma) = \texttt{ls}(\sigma')$.*

Definition 5. (Splitting of an execution)
Executions $\sigma_1 = (\iota_1, \eta_1, \gamma_1)$ and $\sigma_2 = (\iota_2, \eta_2, \gamma_2)$ form a splitting of an execution $\sigma = (\iota, \eta, \gamma)$, denoted $\sigma = \sigma_1 \circ \sigma_2$, if the following hold:

- $\iota = \iota_1 \circ \iota_2$;
- $\eta_1(t) = \eta(t)$ for all $t \in \texttt{dom}(\eta_1)$, and $\eta_2(t) = \eta(t + \texttt{last}(\texttt{dom}(\eta_1)))$ for all $t \in \texttt{dom}(\eta_2)$.
- $\gamma_1(i) = \gamma(i)$ for all $i \in \texttt{dom}(\gamma_1)$, and $\gamma_2(i) = \gamma(i + \texttt{len}(\gamma_1))$ for all $i \in \texttt{dom}(\gamma_2)$.

Note that the splitting predicate is associative, that is, if $\sigma = (\sigma_1 \circ \sigma_2) \circ \sigma_3$ holds, then so does $\sigma = \sigma_1 \circ (\sigma_2 \circ \sigma_3)$. Hence, we will denote a splitting of a an execution σ into a finite sequence of executions $\sigma_1\sigma_2 \ldots \sigma_k$ as $\sigma = \sigma_1 \circ \sigma_2 \circ \ldots \circ \sigma_k$, without the parentheses.

An infinite sequence $\sigma_1\sigma_2 \cdots$ is a splitting of a complete execution σ, denoted $\sigma = \sigma_1 \circ \sigma_2 \circ \ldots$, if there exists an infinite sequence of complete executions $\sigma'_2, \sigma'_3, \ldots$ such that $\sigma = \sigma_1 \circ \sigma'_2$, $\sigma'_i = \sigma_i \circ \sigma'_{i+1}$ for $i \geq 2$.

3.3. Functions on switched systems

Let us fix a *PSS* $\mathcal{H} = (Q, E, X, F, I, G)$ of dimension n. The underlying graph of a hybrid system $\mathcal{H}$, denoted $\mathcal{UG}(\mathcal{H})$, is the pair of the set of locations and the set of edges (Q, E).

Given a set $p \subseteq \mathbb{R}^n$, define $\mathcal{H} \cap p$ to be the *PSS* $(Q_p, E_p, X_p, F_p, I_p, G_p)$, where:

- $Q_p = \{q \in Q \,|\, p \subseteq I(q)\}$,
- $E_p = \{(q_1, q_2) \in (Q_p \times Q_p) \cap E \,|\, p \subseteq G((q_1, q_2))\}$,
- $F_p(q) = F(q) \cap \texttt{plane}(p)$ for each $q \in Q_p$,
- $I_p(q) = p$ for each $q \in Q_p$, and
- $G_p(e) = p$, for each $e \in E_p$.

Let $\texttt{reach}(\mathcal{H}, p)$ denote the set of pairs $(q_1, q_2) \in Q_p \times Q_p$ such that there exists a path in $\mathcal{UG}(\mathcal{H} \cap p)$ from q_1 to q_2.

By considering the polyhedral switched system in Figure 1, $\mathcal{H}$, and the polyhedral set $p_2 = \{(x, y) : y > 0, x - 2y > 0\}$, observe the *PSS* $\mathcal{H} \cap p_2$ enclosed by a dotted line.

4. Stability: Lyapunov and Asymptotic

In this section, we define two classical notions of stability in control theory, and state preliminary results about the stability of polyhedral switched systems.

Definition 6. A point $x \in \mathbb{R}^n$ is an *equilibrium point* of an n-dimensional *PSS*, if every execution $\sigma \in \texttt{exec}(\mathcal{H})$ with $\eta(0) = x$ satisfies $\eta(t) = x$ for all $t \in \texttt{dom}(\eta)$.

We will assume without loss of generality that the origin $\bar{0}$, is the equilibrium point. In the case of the equilibrium point is not $\bar{0}$, we can apply a translation to the equilibrium point and every guard and invariant in the *PSS* by the vector from the equilibrium point to the origin. This translation does not affect to the dynamical behaviour with respect to the equilibrium.

Remark 1. The stability verification algorithm described in the paper does not require the input system to have $\bar{0}$ as an equilibrium point. However, if the algorithm deduces that the system is stable, then it also implies that $\bar{0}$ is an equilibrium point.

Intuitively, Lyapunov stability captures the notion that the executions of the system starting close to the equilibrium point remain close to it. Asymptotic stability, in addition, requires that executions starting in a small enough neighborhood around the equilibrium point converge to it.

Given $\epsilon, \delta \in \mathbb{R}_{\geqslant 0}$ and a set of executions Σ, let the predicate $\texttt{lyap}(\Sigma, \epsilon, \delta)$ denote the fact that for every execution $\sigma \in \Sigma$ with $\eta(0) \in B_\delta(\bar{0})$, $\eta(t) \in B_\epsilon(\bar{0})$ for every $t \in \texttt{dom}(\eta)$.

Definition 7. A set of executions Σ of a *PSS* $\mathcal{H}$ is said to be *Lyapunov stable* with respect to $\bar{0}$, if for every real $\epsilon > 0$, there exists a real $\delta > 0$ such that $\texttt{lyap}(\Sigma, \epsilon, \delta)$.

Given an execution σ, it is said to converge to $\bar{0}$, denoted $\texttt{conv}(\sigma, \bar{0})$, if for every real $\epsilon > 0$, there exists a $T \in \texttt{dom}(\eta)$, such that $\eta(t) \in B_\epsilon(\bar{0})$ for every $t \geq T$. Further, we denote by $\texttt{asymp}(\Sigma, \delta)$ the fact that every complete execution $\sigma \in \Sigma$ with $\eta(0) \in B_\delta(\bar{0})$ satisfies $\texttt{conv}(\sigma, \bar{0})$.

Definition 8. A set of executions Σ of a *PSS* $\mathcal{H}$ is said to be *asymptotically stable* with respect to $\bar{0}$, if it is Lyapunov stable with respect to $\bar{0}$ and there exists a $\delta > 0$ such that $\texttt{asymp}(\Sigma, \delta)$.

A polyhedral switched system $\mathcal{H}$ is said to be Lyapunov (asymptotically) stable if $\texttt{exec}(\mathcal{H})$ is Lyapunov (asymptotically) stable with respect to $\bar{0}$. We observe in Figure 2a a stable *PSS* while in Figure 2b, with different dynamics in q_5, an unstable *PSS*.

4.1. Some properties of stability

Next, we prove some properties about stability. The first property states that the Lyapunov stability of a system depends on the executions in a small neighborhood around $\bar{0}$ of the system.

Proposition 3. *A set of executions Σ is Lyapunov stable if and only if there exists an $\epsilon' > 0$ such that for every $0 < \epsilon < \epsilon'$, there exists a $\delta > 0$ for which* $\texttt{lyap}(\Sigma, \epsilon, \delta)$ *holds.*

In a small enough neighborhood around $\bar{0}$, the executions of a *PSS* are determined by the homogenous linear constraints of the guards and the invariants. Hence, we can assume that the system is in a normal form as stated in the next proposition.

Definition 9. A polyhedral switched system is in *normal form* if the invariants and the guards of the system are pointed.

Consider a polyhedral switched system $\mathcal{H} = (Q_{\mathcal{H}}, E_{\mathcal{H}}, X, F_{\mathcal{H}}, I_{\mathcal{H}}, G_{\mathcal{H}})$ and an equilibrium point $x \in X$, we define a *PSS* in *normal form* $\mathcal{H}'$ based on it, as the tuple $(Q_{\mathcal{H}'}, E_{\mathcal{H}'}, X, F_{\mathcal{H}'}, I_{\mathcal{H}'}, G_{\mathcal{H}'})$, where $Q_{\mathcal{H}'} = \{q \in Q_{\mathcal{H}} : x \in \overline{I(q)}\}$, $E_{\mathcal{H}'} = \{e \in E_{\mathcal{H}} : x \in \overline{G(e)}\}$, $F_{\mathcal{H}'}(q) = F_{\mathcal{H}}(q)$, $I_{\mathcal{H}'}(q) = \{\alpha y : y \in I_{\mathcal{H}}(q), \alpha > 0\}$ and $G_{\mathcal{H}'}(e) = \{\alpha y : y \in G_{\mathcal{H}}(e), \alpha > 0\}$. Observe that the local behaviour close to $\bar{0}$ of both systems is analogous, it is the same as the behaviour of $\mathcal{H}$ restricted to $B_\epsilon(\bar{0})$ for a small enough ϵ value.

Proposition 4. *Let $\mathcal{H}$ be a polyhedral switched system and $\mathcal{H}'$ be the polyhedral switched system in normal form based on $\mathcal{H}$. Then, there exists a value $\epsilon > 0$ such that*

$$\texttt{exec}(\mathcal{H}) \cap B_\epsilon(\bar{0}) = \texttt{exec}(\mathcal{H}') \cap B_\epsilon(\bar{0}).$$

Proof. It is enough to fix an $\epsilon-$ball around the origin such that the invariants and guards of the *PSS* $\mathcal{H}$ not containing $\bar{0}$ are out of the ball. Fix $\epsilon > 0$ such that $B_\epsilon(\bar{0}) \cap I_{\mathcal{H}}(q) = \emptyset$ for every $q \in Q_{\mathcal{H}}$ with $I_{\mathcal{H}}(q)$ not pointed and $B_\epsilon(\bar{0}) \cap G_{\mathcal{H}}(q_1, q_2) = \emptyset$ for every pair $(q_1, q_2) \in E_{\mathcal{H}}$ such that $G_{\mathcal{H}}(q_1, q_2)$ not pointed. Consider $\sigma \in \texttt{exec}(\mathcal{H}) \cap B_\epsilon(\bar{0})$. Then, $\sigma = (\iota, \eta, \gamma)$ where ι is an interval domain; $\eta : [\![\iota]\!] \to \mathbb{R}^n$ such that for each $i \in \texttt{dom}(\iota)$, $\eta \restriction_{\iota(i)}$ is a differentiable function. $\gamma : \texttt{dom}(\iota) \to Q_{\mathcal{H}}$ is such that for all $i \in \texttt{dom}(\iota)$, $I_{\mathcal{H}}(\gamma(i))$ is pointed. Then, $\gamma : \texttt{dom}(\iota) \to Q_{\mathcal{H}'}$. For all $i \in \texttt{dom}(\iota)$,

for all $t \in \iota(i)$, $\eta(t) \in I_{\mathcal{H}}(\gamma(i)) \cap B_\epsilon(\bar{0})$ and $\dot{\eta}(t) \in F_{\mathcal{H}}(\gamma(i))$. By construction of the normal *PSS* $\mathcal{H}'$ based on $\mathcal{H}$, $I_{\mathcal{H}}(\gamma(i)) \subseteq I_{\mathcal{H}'}(\gamma(i))$ and hence $I_{\mathcal{H}}(\gamma(i)) \cap B_\epsilon(\bar{0}) \subseteq I_{\mathcal{H}'}(\gamma(i)) \cap B_\epsilon(\bar{0})$. Let us denote $A = \cup_{j \in \mathtt{dom}(\iota)} \{I_{\mathcal{H}}(\gamma(j)) : \bar{0} \notin I_{\mathcal{H}}(\gamma(j))\}$. We know that $I_{\mathcal{H}'}(\gamma(i)) = \{\alpha x : x \in I_{\mathcal{H}}(\gamma(i)), \alpha > 0\} \subseteq I_{\mathcal{H}}(\gamma(i)) \cup A$, which implies that $I_{\mathcal{H}'}(\gamma(i)) \cap B_\epsilon(\bar{0}) \subseteq (I_{\mathcal{H}}(\gamma(i)) \cup A) \cap B_\epsilon(\bar{0})$. And this last expression, by definition of ϵ, is equal to $I_{\mathcal{H}}(\gamma(i)) \cap B_\epsilon(\bar{0})$. Therefore, $I_{\mathcal{H}}(\gamma(i)) \cap B_\epsilon(\bar{0}) = I_{\mathcal{H}'}(\gamma(i)) \cap B_\epsilon(\bar{0})$. By construction of $\mathcal{H}'$, $F_{\mathcal{H}}(\gamma(i)) = F_{\mathcal{H}'}(\gamma(i))$. For all $0 \leq i < \mathtt{len}(\iota)$, $(\gamma(i), \gamma(i+1)) \in E_{\mathcal{H}}$ and $\eta(\mathtt{last}(\iota(i))) \in G_{\mathcal{H}}((\gamma(i), \gamma(i+1))) \cap B_\epsilon(\bar{0})$, where $G_{\mathcal{H}}((\gamma(i), \gamma(i+1)))$ is pointed. Then, $G_{\mathcal{H}}((\gamma(i), \gamma(i+1))) \cap B_\epsilon(\bar{0}) = G_{\mathcal{H}'}((\gamma(i), \gamma(i+1))) \cap B_\epsilon(\bar{0})$, by construction of the normal *PSS* $\mathcal{H}'$ based on $\mathcal{H}$. Hence, $\sigma \in \mathtt{exec}(\mathcal{H}') \cap B_\epsilon(\bar{0})$. On the other hand, if we consider $\sigma \in \mathtt{exec}(\mathcal{H}') \cap B_\epsilon(\bar{0})$, by an analogous reasoning, $\sigma \in \mathtt{exec}(\mathcal{H}) \cap B_\epsilon(\bar{0})$ is obtained.

Proposition 5. *Given a polyhedral switched system $\mathcal{H}$, the polyhedral switched system $\mathcal{H}'$ in normal form based on $\mathcal{H}$, is such that:*

- *$\mathcal{H}$ is Lyapunov stable if and only if $\mathcal{H}'$ is Lyapunov stable.*
- *$\mathcal{H}$ is asymptotically stable if and only if $\mathcal{H}'$ is asymptotically stable.*

Proof. For a system in normal form it is enough to prove stability in a small neighborhood to the origin [24]. Suppose $\mathcal{H}$ is Lyapunov stable. By Proposition 4, we fix $\epsilon > 0$ such that $\mathtt{exec}(\mathcal{H}) \cap B_\epsilon(\bar{0}) \subseteq \mathtt{exec}(\mathcal{H}')$. We know that there exists $\delta > 0$ such that $\mathtt{lyap}(\mathtt{exec}(\mathcal{H}), \epsilon, \delta)$. Then, by construction of $\mathcal{H}'$, it is clear that $\mathtt{lyap}(\mathtt{exec}(\mathcal{H}'), \epsilon, \delta)$ and that for every $\epsilon' = w\epsilon > 0$ we can choose $\delta' = w\delta$, which satisfies $\mathtt{lyap}(\mathtt{exec}(\mathcal{H}'), \epsilon', \delta')$. Therefore, we infer that $\mathcal{H}'$ is Lyapunov stable. Now, suppose $\mathcal{H}'$ is Lyapunov stable. Choose $\epsilon' > 0$ such that $\mathtt{exec}(\mathcal{H}) \cap B_{\epsilon'}(\bar{0}) \subseteq \mathtt{exec}(\mathcal{H}')$. For every $\epsilon < \epsilon'$ we know that there exists $\delta > 0$ such that $\mathtt{lyap}(\mathtt{exec}(\mathcal{H}'), \epsilon, \delta)$. Since $\mathtt{exec}(\mathcal{H}) \cap B_{\epsilon'}(\bar{0}) \subseteq \mathtt{exec}(\mathcal{H}')$, for every $\epsilon < \epsilon'$ we have $\mathtt{lyap}(\mathtt{exec}(\mathcal{H}), \epsilon, \delta)$. Then, by Proposition 3, $\mathtt{exec}(\mathcal{H})$ is Lyapunov stable. Next, we need to prove the second point, about asymptotic stability. Suppose first $\mathcal{H}'$ is asymptotically stable. Then, by definition, there exists $\delta > 0$ such that $\mathtt{asymp}(\mathtt{exec}(\mathcal{H}'), \delta)$ is satisfied. Also, by Proposition 4, there exists $\epsilon > 0$ such that $\mathtt{exec}(\mathcal{H}) \cap B_\epsilon(\bar{0}) \subseteq \mathtt{exec}(\mathcal{H}')$. In addition, since Lyapunov stability of $\mathcal{H}'$ implies Lyapunov stability of $\mathcal{H}$, it is known that there exists $\delta_\epsilon > 0$ such that $\mathtt{lyap}(\mathtt{exec}(\mathcal{H}), \epsilon, \delta_\epsilon)$. Therefore, every execution $\sigma \in \mathtt{exec}(\mathcal{H})$ starting from $B_{\delta_\epsilon}(\bar{0})$ remains in $B_\epsilon(\bar{0})$, which implies that $\sigma \in \mathtt{exec}(\mathcal{H}')$. Therefore,

in the case of $\delta_\epsilon \leqslant \delta$, σ satisfies $\texttt{conv}(\sigma, \bar{0})$ since $B_\epsilon(\bar{0}) \subseteq B_\epsilon(\bar{0})$, which implies that $\texttt{asymp}(\texttt{exec}(\mathcal{H}), \delta_\epsilon)$. While in the case of $\delta_\epsilon > \delta$, σ does not satisfy $\texttt{conv}(\sigma, \bar{0})$. Then, consider σ that, in addition, starts from $B_\delta(\bar{0})$. These kind of executions satisfy $\texttt{conv}(\sigma, \bar{0})$ and consequently $\texttt{asymp}(\texttt{exec}(\mathcal{H}), \delta)$. Now, for the case of $\mathcal{H}$ is asymptotically stable, the proof is analogous.

From now on, we will assume that the *PSS* is in normal form. The *PSS* in Figure 2 are in normal form.

The next lemma states that the stability of a polyhedral switched system is determined completely by the stability of the set of piecewise-linear executions of the system.

Lemma 6. *Let $\mathcal{H}$ be a polyhedral switched system. Then:*

- $\texttt{exec}(\mathcal{H})$ *is Lyapunov stable if and only if* $\texttt{pwexec}(\mathcal{H})$ *is Lyapunov stable.*
- $\texttt{exec}(\mathcal{H})$ *is asymptotically stable if and only if* $\texttt{pwexec}(\mathcal{H})$ *is asymptotically stable.*

Proof. Suppose $\texttt{exec}(\mathcal{H})$ is Lyapunov (asymptotically) stable, it is clear that $\texttt{pwexec}(\mathcal{H})$ is Lyapunov (asymptotically) stable since $\texttt{pwexec}(\mathcal{H}) \subseteq \texttt{exec}(\mathcal{H})$.

Next, suppose that $\texttt{pwexec}(\mathcal{H})$ is Lyapunov stable. We prove that this implies $\texttt{exec}(\mathcal{H})$ is Lyapunov stable. Suppose that $\texttt{exec}(\mathcal{H})$ is not Lyapunov stable, then there exists an $\epsilon > 0$ such that for every value $\delta > 0$ there exists an execution $\sigma \in \texttt{exec}(\mathcal{H})$ starting from $B_\delta(\bar{0})$ which at some time $T > 0$ goes out from $B_\epsilon(\bar{0})$, that is, it is such $\eta(0) \in B_\delta(\bar{0})$ and $\eta(T) \notin B_\epsilon(\bar{0})$. The restriction of σ to the time interval $[0, T]$ is a finite execution in $\mathcal{H}$. By applying Proposition 2, construct a piecewise-linear execution $\sigma' \in \texttt{pwexec}(\mathcal{H})$ such that $\texttt{fs}(\sigma) = \texttt{fs}(\sigma')$ and $\texttt{ls}(\sigma) = \texttt{ls}(\sigma')$. This means that for every $\delta > 0$, there exists a piecewise-linear execution $\sigma' \in \texttt{pwexec}(\mathcal{H})$ starting from $B_\delta(\bar{0})$ and at time $T > 0$ goes out of $B_\epsilon(\bar{0})$, which contradicts the fact that $\texttt{pwexec}(\mathcal{H})$ is Lyapunov stable.

Finally, we prove that $\texttt{pwexec}(\mathcal{H})$ is asymptotically stable implies $\texttt{exec}(\mathcal{H})$ is asymptotically stable. Suppose that $\texttt{asymp}(\texttt{pwexec}(\mathcal{H}), \delta)$ holds. We will show that $\texttt{asymp}(\texttt{exec}(\mathcal{H}), \delta)$ holds. Let σ be an execution in $\mathcal{H}$ with $\eta(0) \in B_\delta(\bar{0})$. Suppose that $\texttt{conv}(\sigma, \bar{0})$ does not hold. Then there exists $\delta' > 0$, such that the trajectory η does not eventually remain inside $B_{\delta'}(\bar{0})$.

Choose an infinite sequence of diverging times $t_0 = 0 < t_1 < t_2 < \cdots$ such that $\eta(t_i) \notin B_{\delta'}(\bar{0})$. Construct a piecewise-linear execution σ' of $\mathcal{H}$ using Proposition 2 by replacing σ in each of the intervals $[t_i, t_{i+1}]$ by a piecewise-linear execution. Then $\texttt{conv}(\sigma', \bar{0})$ does not hold, which contradicts $\texttt{asymp}(\texttt{pwexec}(\mathcal{H}), \delta)$.

5. Stability verification procedure

In this section, we present an algorithmic approach for verifying stability of *PSS*s. This is an extension of the algorithms in [25] for piecewise constant derivative systems and in [24] for two dimensional rectangular switched systems. The verification procedure consists of two parts:

1. Extracting a finite weighted graph from the *PSS* using an elementary partition of the state-space.
2. Analyzing the graph for deducing stability.

We discuss the two parts in detail in the following. However, we will defer the computational aspects to the next section.

5.1. Formal definition of the graph

The graph construction takes as input an elementary partition of the state-space and a polyhedral switched system, and outputs a finite weighted graph. The graph captures the sequence of elements the executions of the system traverse using the notion of an almost-inside element execution.

Definition 10. Let $\mathcal{H}$ be a *PSS* and $\mathcal{P}$ an elementary partition of its state space. Given an element p of $\mathcal{P}$, a *p-execution* is an execution σ such that $\eta(t) \in p$ for every $0 < t < \texttt{last}(\texttt{dom}(\eta))$. An *element execution* is an execution which is a p-execution for some element p of $\mathcal{P}$.

The weights in the graph correspond to scalings of executions which measure the relative distance of the end-points of the executions to the origin. Given a finite execution σ, its scaling, denoted $\texttt{scaling}(\sigma)$ is given by:

$$\texttt{scaling}(\sigma) = \frac{|\eta(\texttt{last}(\texttt{dom}(\eta)))|}{|\eta(0)|}.$$

The vertices of the graph correspond to location-element pairs, edges between two location-element pairs correspond to the presence of an element

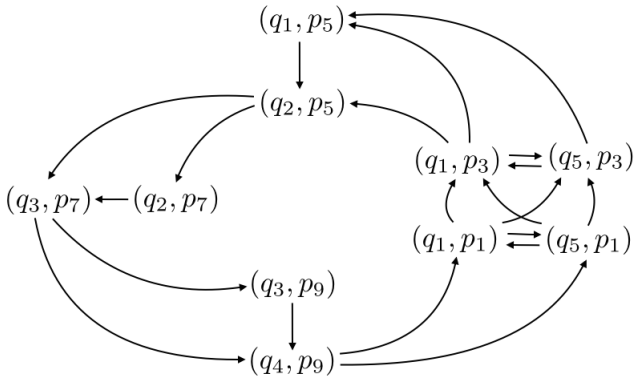


Figure 3: Weighted graph

execution from the first location-element pair to the second, and weights on edges correspond to an upper-bound on the scaling of the executions corresponding to the edge.

Let us fix a *PSS* $\mathcal{H} = (Q, E, X, F, I, G)$ in normal form, and an element partition $\mathcal{P}$ of the state space. We will assume that every element p of the partition $\mathcal{P}$ is such that for each invariant or guard r, p is either contained in r or is disjoint from r. Given $q_1, q_2 \in Q$ and $p_1, p_2 \in \mathcal{P}$, let $\mathtt{ai}(\sigma, (q_1, p_1), (q_2, p_2))$ denote the fact that σ is an element execution from a state in (q_1, p_1) to a state in (q_2, p_2).

The weighted graph $\mathcal{G} = (V, E, \mathtt{w})$ is determined by $\mathcal{H}$ and $\mathcal{P}$, denoted as $\mathcal{G}(\mathcal{H}, \mathcal{P})$ and it is defined as follows:

1. The set of vertices V is given by $Q \times \mathcal{P}$;
2. The set of edges $E \subseteq V \times V$ is given by $\{((q_1, p_1), (q_2, p_2)) \mid \exists \sigma, \mathtt{ai}(\sigma, (q_1, p_1), (q_2, p_2))\}$;
3. The weighting function $\mathtt{w}$ is given by the supremum of a set of scaling values as follows, $\mathtt{w}(((q_1, p_1), (q_2, p_2))) = \sup\{\mathtt{scaling}(\sigma) \mid \exists \sigma, \mathtt{ai}(\sigma, (q_1, p_1), (q_2, p_2))\}$.

A weighted graph is represented in Figure 3. It is constructed by considering the *PSS* $\mathcal{H}$ in Figure 2a and an element partition $\mathcal{P}$. The element partition, depicted in Figure 5a, consists of the following polyhedral sets, $p_1 = \{(x, y) : x > 0, y = 0\}, p_2 = \{(x, y) : x - 2y > 0, y > 0\}, p_3 = \{(x, y) : x - 2y = 0, y > 0\}, p_4 = \{(x, y) : x - 2y < 0, x > 0\}, p_5 = \{(x, y) : x =$

$0, y > 0\}, p_6 = \{(x, y) : x < 0, y > 0\}, p_7 = \{(x, y) : x < 0, y = 0\}, p_8 = \{(x, y) : x < 0, y < 0\}, p_9 = \{(x, y) : x = 0, y < 0\}$ and $p_{10} = \{(x, y) : x > 0, y < 0\}$. For clarity, the weighted graph is depicted only for the elements p_1, p_3, p_5, p_7 and p_9, since they are enough to abstract the *PSS*. The set of vertices for $\mathcal{G}(\mathcal{H}, \mathcal{P})$ corresponds to the pairs (q_1, p_1), (q_5, p_1), (q_1, p_3), (q_5, p_3), (q_1, p_5), (q_2, p_5), (q_2, p_7), (q_3, p_7), (q_3, p_9), (q_4, p_9), and the edges are $((q_1, p_1), (q_5, p_1))$, $((q_5, p_1), (q_1, p_1))$, $((q_5, p_1), (q_5, p_3))$, $((q_1, p_1), (q_5, p_3))$, $((q_5, p_1), (q_1, p_3))$, $((q_1, p_1), (q_1, p_3))$, $((q_5, p_3), (q_1, p_3))$, $((q_1, p_3), (q_5, p_3))$, $((q_5, p_3), (q_1, p_5))$, $((q_1, p_3), (q_1, p_5))$, $((q_1, p_3), (q_2, p_5))$, $((q_1, p_5), (q_2, p_5))$, $((q_2, p_5), (q_3, p_7))$, $((q_2, p_5), (q_2, p_7))$, $((q_2, p_7), (q_3, p_7))$, $((q_3, p_7), (q_3, p_9))$, $((q_3, p_7), (q_4, p_9))$, $((q_3, p_9), (q_4, p_9))$, $((q_4, p_9), (q_1, p_1))$ and $((q_4, p_9), (q_5, p_1))$. The computation of the weight values associated with the edges will be presented later.

5.2. Stability Criteria

The weighted graph $\mathcal{G} = (V, E, \mathtt{w})$ is analysed for the satisfaction of the following properties:

P1 For every edge $e \in E$, $\mathtt{w}(e) \in \mathbb{R}_{\geqslant 0}$.

P2 Every simple cycle π of $\mathcal{G}$ is such that $\mathtt{w}(\pi) \leqslant 1$.

P3 Every simple cycle π of $\mathcal{G}$ is such that $\mathtt{w}(\pi) < 1$.

In addition, we need to verify the following property:

P4 For every element $p \in \mathcal{P}$, there exists no complete execution σ such that $\eta(t) \in p$, for all $t \in [0, \infty)$ and $\mathtt{conv}(\sigma, \overline{0})$ does not hold.

The following theorems characterize the criteria for stability in terms of the above properties.

Theorem 7. *(Lyapunov stability) If Conditions [P1] and [P2] hold for $\mathcal{G}$, then $\mathcal{H}$ is Lyapunov stable.*

Theorem 8. *(Asymptotic stability) If Conditions [P1], [P3] and [P4] hold for $\mathcal{G}$, then $\mathcal{H}$ is asymptotically stable.*

5.3. Correctness of the stability criteria

Below, we provide insights into the correctness of the theorems. It relies crucially on the following decomposition of the executions into element executions.

Lemma 9. *Let $\mathcal{H}$ be a polyhedral switched system and $\mathcal{P}$ an elementary partition. For every execution $\sigma \in \texttt{exec}(\mathcal{H})$, there exists a finite or infinite sequence $\sigma_0\sigma_1\ldots$ of executions in $\texttt{exec}(\mathcal{H})$ such that for each i, σ_i is a p-execution for some $p \in \mathcal{P}$, and $\sigma = \sigma_0 \circ \sigma_1 \circ \ldots$.*

Proof. Consider $\sigma \in \texttt{exec}(\mathcal{H})$. We know $\sigma = (\iota, \eta, \gamma)$ and $\forall i \in \texttt{dom}(\iota)$ and $\forall t \in \iota(i), \eta(t) \in I(\gamma(i))$. The polyhedral set $I(\gamma(i))$ is divided into k_i finite parts by partition $\mathcal{P}$, $\{I(\gamma(i)) \cap p_1, \ldots, I(\gamma(i)) \cap p_{k_i}\}$. Then, the interval $\iota(i)$ can be separated into k_i time intervals, $\{\iota(i)_1, \ldots, \iota(i)_{k_i}\}$such that $\eta(t) \in p_j$ for every $t \in \iota(i)_j$ and $1 \leq j \leq k_i$. Let us denote $\texttt{last}(\texttt{dom}(\iota))$ as l and assign to the interval set $\{\iota(1)_1, \ldots, \iota(1)_{k_1}, \iota(2)_1, \ldots, \iota(2)_{k_2}, \ldots, \iota(l)_1, \ldots, \iota(l)_{k_l}\}$ the indexes from 1 to $k = \sum_{1 \leq j \leq l} k_j$. We define the set of executions $\sigma_j = (\iota_j, \eta_j, \gamma_j)$ for $1 \leq j \leq k$ such that:

- $\iota_j = \iota(1)_j$ if $1 \leq j \leq k_1$, $\iota_j = \iota(2)_j$ if $k_1 < j \leq k_1 + k_2$, ... and $\iota_j = \iota(l)_j$ if $k_{l-1} < j \leq k_l$,
- $\eta_j(t) = \eta(t)$ for all $t \in \texttt{dom}(\eta_j)$, and $\eta_{j+1}(t) = \eta(t + \texttt{last}(\texttt{dom}(\eta_j)))$ for all $t \in \texttt{dom}(\eta_{j+1})$.
- $\gamma_j(1)$ is equal to $\gamma(1)$ if $1 \leq j \leq k_1$, $\gamma(2)$ if $k_1 < j \leq k_1 + k_2$, and so on, till being equal to $\gamma(\texttt{dom}(\iota))$ if $k_{l-1} < j \leq k_l$.

Hence, the execution σ is split into a set of p-executions in $\texttt{exec}(\mathcal{H})$, $\sigma = \sigma_1 \circ \ldots \circ \sigma_k$.

Proof of Theorem 7. A sufficient condition for proving Lyapunov stability is to establish a global bound on the scaling of all executions of the system. This is precisely what conditions [P1] and [P2] capture. Consider $\sigma \in \texttt{exec}(\mathcal{H})$ being a p-execution, hence there exist $q_1, q_2 \in Q$ such that $\texttt{ai}(\sigma, (q_1, p), (q_2, p))$ with $\texttt{scaling}(\sigma) \leqslant \texttt{w}((q_1, p), (q_2, p))$. $\texttt{w}((q_1, p), (q_2, p)) \in \mathbb{R}_{\geqslant 0}$ because of condition [P1], and corresponds to an edge in $\mathcal{G}$ which is finite; therefore we can define

$$b1 = \max_{e \in E} \texttt{w}(e)$$

as the bound of p-executions. Consider $\sigma \in \texttt{exec}(\mathcal{H})$ such that $\sigma = \sigma_0 \circ \sigma_1 \circ \ldots$ where for every i, σ_i is p_i-execution for some $p_i \in \mathcal{P}$. Then there exists $q_i \in Q$ for all i such that $\texttt{ai}(\sigma_i, (q_i, p_i), (q_{i+1}, p_{i+1}))$ with $\texttt{scaling}(\sigma_i) \leqslant \texttt{w}((q_i, p_i), (q_{i+1}, p_{i+1}))$. Now we distinguish between σ finite or complete. Let us consider σ finite of length k, which implies $\texttt{w}(\sigma) = \prod_{i=0}^{k} \texttt{w}((q_i, p_i), (q_{i+1}, p_{i+1}))$. Since condition [P1] establishes that $\texttt{w}((q_i, p_i), (q_{i+1}, p_{i+1})) \in \mathbb{R}_{\geqslant 0}$ and the number of simple paths in $\mathcal{G}$ is bounded we define

$$b_2 = \max_{\substack{\sigma \in \texttt{exec}(\mathcal{H}) \\ \sigma \text{ finite}}} \texttt{w}(\sigma).$$

In case of σ complete, $\texttt{w}(\sigma) = \prod_{i=0}^{\infty} \texttt{w}((q_i, p_i), (q_{i+1}, p_{i+1}))$ and due to the fact of $\mathcal{G}$ is finite, σ is abstracted by a finite number of simple paths and a finite set of cycles, getting $\texttt{w}(\sigma) = \prod_{\pi \text{ cycle}} \texttt{w}(\pi) \cdot \prod_{\pi \text{ simple path}} \texttt{w}(\pi)$ where the first term in the product is less than or equal to 1 because of condition [P2] and the second one belongs to $\mathbb{R}_{\geqslant 0}$ because of condition [P1]. Then, we define

$$b_3 = \max_{\substack{\sigma \in \texttt{exec}(\mathcal{H}) \\ \sigma \text{ infinite}}} \texttt{w}(\sigma) \in \mathbb{R}_{\geqslant 0}.$$

Finally, by choosing $B = \max\{b_1, b_2, b_3\}$, we claim that for every $\sigma \in \texttt{exec}(\mathcal{H})$, $\texttt{w}(\sigma) \leqslant B$, it means that $|\eta(t)| \leqslant B \cdot |\eta(0)|$ for all $t \in \texttt{dom}(\eta)$, hence $\forall \epsilon \geq 0$, $\texttt{lyap}(\texttt{exec}(\mathcal{H}), \epsilon, \frac{\epsilon}{B})$ holds.

Proof of Theorem 8. First, note that Condition [P3] implies Condition [P2], hence, [P1] and [P3] together imply Lyapunov stability. It remains to show that every complete execution converges to $\bar{0}$. If the execution eventually enters an element p and remains within it, then Condition [P4] guarantees convergence. Hence, let us consider the case where the execution, say σ, can be split into an infinite sequence of element executions, that is, corresponds to an infinite path π in the graph. An infinite path π corresponds to a splitting of a finite set of finite paths, $\pi_1, \ldots, \pi_k$, and an infinite set of simple cycles, $\pi_{k+1}, \pi_{k+2}, \ldots$. Note that the indices do not represent the order of the splitting and k varies from 0 to the total number of simple paths in the graph $\mathcal{G}$. Let b be an upper bound on the weight of all simple paths, it is $b = \max_{\substack{\pi \in \texttt{paths}(\mathcal{G}) \\ \texttt{len}(\pi) < \infty}} \texttt{w}(\sigma)$. From Condition [P3], we know that the weight of

each simple cycle is < 1. Since the number of simple cycles in the graph $\mathcal{G}$ is finite, there exists $\epsilon > 0$ such that the weight of each simple cycle is $< 1 - \epsilon$. Every prefix of the infinite path π is conformed by the finite paths, $\pi_1, \ldots, \pi_k, \pi_{k+1}, \ldots, \pi_{k+m}$ for some $m \in \mathbb{N}$, which has weight less than or equal to $b^k(1-\epsilon)^m$. The weight goes to 0 as m increases. Hence, the scaling of σ converges to 0. Therefore, σ converges to $\bar{0}$.

5.4. Verifying Conditions [P1-P4]

Given the graph $\mathcal{G}$, conditions [P1-P3] can be efficiently verified in $O(|V||E|)$ complexity where $|V|$ and $|E|$ are the number of vertices and edges in $\mathcal{G}$, respectively. Verifying Condition [P1] consists of iterating over the edges and checking if the values of the weights is $< +\infty$, which is of complexity $O(|E|)$. Conditions [P2] and [P3] can be transformed into checking for cycles whose sum of weights is negative by replacing the weight in an edge e by $-\log(\mathtt{w}(e))$. Negative cycles can be detected, for example, by Bellman-Ford algorithm [26], whose worst-case performance is $O(|V||E|)$ for V and E being the vertices and edges in $\mathcal{G}$.

Note that any complete execution which remains within a partition element p will eventually enter a strongly connected component of $\mathcal{H} \cap p$ and stay there. It will not converge if $\bar{0}$ is in the convex hull of the flows associated with the locations in that component or there exists a vector in the convex hull which points into p from the origin (and hence diverges). Hence, Condition [P4] is equivalent to checking:

P4' For each $C \in \mathtt{scc}(\mathcal{UG}(\mathcal{H} \cap p))$, $\bar{0} \notin \mathtt{chull}(\bigcup_{q \in C} F(q)) \wedge \mathtt{chull}(\bigcup_{q \in C} F(q)) \cap p = \emptyset$,

where $\mathcal{H} \cap p$ is computed in $O(|V| + |E|)$ and the search for all strongly connected components in $\mathcal{H} \cap p$ is performed by the Tarjan's algorithm with Nuutila's modifications [27] in $O(|V_p| + |E_p|)$ complexity time, where V_p and E_p are the vertices and edges in $\mathcal{UG}(\mathcal{H} \cap p)$, respectively.

In the case of Condition [P2] is not satisfied, the finite abstract graph $\mathcal{G}$ could be too coarse for proving stability and a tighter abstraction needs to be constructed. A finer abstraction comes out by adding new predicates to the state space partition. This addition can be either an automatic random process or a counterexample based approach. Both approaches are refinement strategies. The automatic random process creates predicates which partition

Algorithm 1 Weighted graph construction

Require: A *PSS* $\mathcal{H}$ and an elementary partition $\mathcal{P}$
Ensure: The weighted graph $\mathcal{G}$

for each location q in Q and element p in $\mathcal{P}$ **do**
  if $p \subseteq I(q)$ **then**
    AddNode$((q, p), \mathcal{G})$
  end if
end for
for element p in $\mathcal{P}$ **do**
  $\mathcal{G}' :=$ QuotientGraph$(\mathcal{H}, p)$
  for (q_1, p_1) and (q_2, p_2) in $\mathcal{G}$ **do**
    if Adjacent(p_1, p) **and** Adjacent(p_2, p) **then**
      $w := -\infty$
      for each pair of nodes C_1, C_2 in $\mathcal{G}'$ **do**
        $\Pi :=$ SimplePaths$(C_1, C_2, \mathcal{G}')$
        for π in Π **do**
          $w_\pi :=$ Scaling$(p_1, p_2, p, \pi, \mathcal{H})$
          $w :=$ Max(w, w_π)
        end for
      end for
      if q_1 in Enter(p_1, C_1) **and** q_2 in Exit(p_2, C_2) **then**
        AddEdge$(((q_1, p_1), (q_2, p_2), p, w), \mathcal{G})$
      end if
    end if
  end for
end for

uniformly the state-space. Another refinement strategy is an algorithmic technique which constructs predicates based on the counterexample. The theoretical foundations and results related to the refinement techniques are presented in [18].

6. Weighted graph construction

Here, we present in detail the weighted graph construction. Algorithm 1 describes the full procedure and Algorithm 2 is the most intricate function in the graph construction. Algorithm 2 determines the existence of edges

between vertices and computes the weights associated with them.

6.1. Algorithm 1

Given a *PSS* $\mathcal{H}$ and an elementary partition $\mathcal{P}$, Algorithm 1 outputs a weighted graph $\mathcal{G}$. This graph $\mathcal{G}$ corresponds to the graph $\mathcal{G}(\mathcal{H}, \mathcal{P})$ defined in Subsection 5.1. We enumerate the functions involved in Algorithm 1.

1. AddNode$((q,p),\mathcal{G})$ adds the node (q,p) to the graph $\mathcal{G}$.
2. QuotientGraph$(\mathcal{H}, p)$ constructs the quotient graph $\mathcal{UG}(\mathcal{H}\cap p)/\texttt{scc}$, where $\mathcal{UG}(\mathcal{H}\cap p)$ corresponds to the underlying graph of the *PSS* $\mathcal{H}\cap p$ as defined in Subsection 3.3.
3. Adjacent(p', p) takes as inputs two elements $p', p \in \mathcal{P}$ and returns True if they are adjacent and False otherwise. An element p' is adjacent to p if p' belongs to the boundary of p, that is if $p' \subset \overline{p}$ and $p' \cap \mathring{p} = \emptyset$.
4. SimplePaths$(C_1, C_2, \mathcal{G}')$ provides all the simple paths in the graph $\mathcal{G}'$ starting from the node C_1 and ending at the node C_2.
5. Scaling$(p_1, p_2, p, \pi, \mathcal{H})$ computes the relation between points in p_1 and p_2 such that there exists a p-execution evolving by following the dynamics involved in the path π and it will be explained in detail later.
6. Max(w, w_π) computes the maximum of the two rational values w and w_π.
7. Enter(p_1, C_1) takes as input an element p_1 and a strongly connected component C_1 of $\mathcal{UG}(\mathcal{H}\cap p)/\texttt{scc}$. It consists of three subfunctions. The first one proceeds to search the locations q in the strongly connected component C_1 such that p_1 is contained in the closure of the invariant set $I(q)$. These locations are stored into a list l_1. The second function lists into l_2 the set of locations $q \in l_1$ such that the vector field $F(q)$ can define a execution which enters in p. Finally, the third function outputs the locations in l_2 such that there exists an edge in the subgraph $\mathcal{H}\cap p_1$ to them.
8. Exit(p_2, C_2) takes as input an element p and a strongly connected component C_2 of $\mathcal{UG}(\mathcal{H} \cap p)/\texttt{scc}$. It requires three steps. The first one identifies the locations q in the strongly connected component C_2 such that $p_2 \subseteq \overline{I(q)}$. They are stored into a list l_1. The second step lists into l_2 the locations $q \in l_1$ such that the dynamics determined by the vector field $F(q)$ can exit from the element p. Then, the third step selects from l_2 the locations which form an edge in the subgraph $\mathcal{H}\cap p_2$ from them.

Algorithm 2 SCALING function

Require: Three elements p_1, p_2, p, a sequence of sets of locations $\pi = (C_0, \ldots, C_k)$ and a *PSS* $\mathcal{H}$

Ensure: Scaling w_π

1: $d :=$ COLLECTDYNAMICS$(C_0, \mathcal{H})$
2: $D :=$ CONVEXHULL(d)
3: $R :=$ REACHRELATION(p_1, p, D)
4: **for** $i = 1, \ldots, k-1$ **do**
5: $\quad d :=$ COLLECTDYNAMICS$(C_i, \mathcal{H})$
6: $\quad D :=$ CONVEXHULL(d)
7: $\quad r :=$ REACHRELATION(p, p, D)
8: $\quad R :=$ COMPOSE(R, r)
9: **end for**
10: $d :=$ COLLECTDYNAMICS$(C_k, \mathcal{H})$
11: $D :=$ CONVEXHULL(d)
12: $r :=$ REACHRELATION(p, p_2, D)
13: $R :=$ COMPOSE(R, r)
14: $w_\pi :=$ OPTIMIZATION(R)

The input to Algorithm 1 is a *PSS* $\mathcal{H}$ and an elementary partition $\mathcal{P}$, and the output is a weighted graph. The nodes of the weighted graph are pairs of a location and an element. Hence, the algorithm iterates over locations and elements. If a pair of a location q and an element p is such that the element is contained in the invariant $I(q)$ of the location, this pair is added as a node to the graph. Next, the edges and the weights associated with them will be computed by iterating over the elements of the elementary partition. For the element p, the quotient graph over the strongly connected components of $\mathcal{H} \cap p$ is constructed. Then, for every pair of nodes C_1 and C_2 in the quotient graph, all the simple paths connecting them are added into Π (line 12 in Algorithm 1). For each simple path in Π, the scaling is computed. Such a computation is performed by the function SCALING, whose inputs are a pair of elements p_1 and p_2 (both adjacent to p), the element p, the selected simple path and the initial *PSS* $\mathcal{H}$. Observe that the edges are to be between nodes determined by a pair of location in $\mathcal{H}$ and element in $\mathcal{P}$. Therefore, one step is to set the enter and exit locations in order to pair the locations with the elements p_1 and p_2 respectively. The enter locations correspond to the locations in the strongly connected component of $\mathcal{H} \cap p$ associated with C_1

such that an execution can travel from p_1 to the invariants in C_1, while exit locations are those in the strongly connected component of $\mathcal{H} \cap p$ associated with C_2 such that there exists an execution evolving from each of the location invariants to the adjacent element p_2. Therefore, for each enter location q_1 and exit location q_2 there exists an edge of the form $((q_1, p_1), (q_2, p_2))$ in the weighted graph $\mathcal{G}$. It remains to compute the weight. Each edge of the form $((q_1, p_1), (q_2, p_2))$ have the same weight for the same triple of elements p_1, p_2 and p, irrespective of the locations q_1 and q_2. The maximum of the scaling over all the simple paths will be the weight linked to and edge of the form $((q_1, p_1), (q_2, p_2))$ in case of q_1 is a location which allows to enter the component C_1 and q_2 is a location which allows to exit from the component C_2.

6.2. Algorithm 2

Next, we explain in detail the function $\textsc{Scaling}(p_1, p_2, p, \pi, \mathcal{H})$, described in Algorithm 2. Algorithm 2 takes as input three elements $p_1, p_2, p \in \mathcal{P}$ and a *PSS* $\mathcal{H}$, then it computes the relation between points in p_1 and p_2 such that there exists a p-execution evolving by following the dynamics involved in the path π. It consists of five functions:

1. $\textsc{CollectDynamics}(C, \mathcal{H})$ collects all the flows $F(q)$ in the *PSS* $\mathcal{H}$, where $q \in C$, into a list.
2. $\textsc{ConvexHull}(d)$ takes as input a list of flows, say d, and computes the convex hull polyhedron of these flows.
3. $\textsc{ReachRelation}(p_1, p, D)$ returns a polyhedron which represents the relation between points from element p_1 travelling to points in element p by following any vector dynamics contained in the polyhedron D. A detailed description on the relation polyhedron construction is exposed later.
4. $\textsc{Compose}(R, r)$ composes two given reach relations, R and r by returning a new one, such that if $(x, y) \in R$ and $(y, z) \in r$, then (x, z) will be contained in the composed reach relation.
5. $\textsc{Optimization}(R)$ returns the pair element in R which maximizes the scaling, that is the pair $(x, y) \in R$ with maximum value $|y|/|x|$.

The input to the Algorithm 2, that is the scaling function, is three elements, p_1, p_2 and p, a *PSS* H and a sequence of sets of locations or components in $\mathcal{H}$, $\pi = (C_0, \ldots, C_k)$. The output is a rational value. It constructs a composed reach relation polyhedral set by iterating over the sets of locations.

For each set of locations C_i in π, the flows determined by them are collected into a list d. After, the convex hull of the collected flows is computed and called D. Then the reach relation polyhedral set is constructed for the pair of corresponding elements, p_1, p for C_0, p, p_2 for C_k and p, p for the rest of components. All the computed reach relation polyhedra are composed into the polyhedron R. Finally, an optimization is performed over the composed relation polyhedron R in order to obtain the maximum scaling between an initial point in p_1 and a final point in p_2 for a p-execution. This is the scaling value returned by $\textsc{Scaling}(p_1, p_2, p, \pi, \mathcal{H})$.

Illustration. We illustrate the main steps in Algorithm 1. Consider the polyhedral switched system $\mathcal{H}$ depicted in Figure 1, and the partition $\mathcal{P}$ defined at the end of Section 5.1. The weighted graph $\mathcal{G}(\mathcal{H}, \mathcal{P})$ is represented in Figure 3. We pick one of the elements in the partition, for instance $p_2 = \{(x, y) \in \mathbb{R}^2 : x - 2y > 0, y > 0\}$. The element p_2 is contained in the invariant of two different modes of the *PSS*. The function $\textsc{Quotient-Graph}(\mathcal{H}, p_2)$ constructs first $\mathcal{UG}(\mathcal{H} \cap p_2)$ as defined in Subsection 3.3. This subgraph is circled by a dotted line in Figure 1. Then, the quotient graph obtained from $\mathcal{UG}(\mathcal{H} \cap p_2)$ returns just one strongly connected component, which consists of the locations q_1 and q_5. The adjacent elements to p_2 contained in the partition $\mathcal{P}$, defined in Subsection 5.1, are p_1 and p_3. Let us consider p_1 as the initial adjacent element and p_3 as the final adjacent element. We explain in the following subsection how to define the polyhedral set which represents the relation between points in p_1 and p_3 such that there exists an execution starting at a point in p_1, ending in p_3 and evolving through the element p_2.

6.3. Reach Relation polyhedron

Here, we present a detailed description of the polyhedral construction implemented in $\textsc{ReachRelation}(p_1, p_2, D)$. This function constructs a polyhedron which represents the relation between points from the element p_1 travelling to points in the element p_2 by following some of the dynamics described in the polyhedron D. Consider d_1 and d_2 the affine dimensions of elements p_1 and p_2 respectively. To perform such construction, three different sets are defined:

- A polyhedral set which duplicates the initial polyhedron p_1. It consists of a polyhedron of affine dimension $2d_1$ where the new variables are

constrained in the same fashion as the old ones and equalized to them. The new polyhedral set for an element defined as $\{x \in \mathbb{R}^n : a_1 \cdot x \sim 0, \ldots, a_m \cdot x \sim 0\}$ is $\{(x, y) \in \mathbb{R}^{2n} : a_1 \cdot x \sim 0, \ldots, a_m \cdot x \sim 0, a_1 \cdot y \sim 0, \ldots, a_m \cdot y \sim 0, x = y\}$.

- A new dynamical polyhedron duplicates the dimension of the initial dynamic polyhedron, D, preserving the coefficients of the constraints in the second half part of the new coefficients and adding constraints to equalize the half first variables to zero. Each dynamical polyhedron of the form $\{x \in \mathbb{R} : a_1 \cdot x \sim 0, \ldots, a_m \cdot x \sim 0\}$ becomes $\{(x, y) \in \mathbb{R}^{2n} : a_1 \cdot y \sim 0, \ldots, a_m \cdot y \sim 0, x = \bar{0}\}$.

- A polyhedral set based on the final polyhedron p_2. It will be one with affine dimension $2d_2$, where the first half part of the coefficients in the new constraints are whatever value and the second half part of the coefficients correspond to the ones in the old constraints. Consider a final polyhedron p_2 of the form $\{x \in \mathbb{R}^n : a_1 \cdot x \sim 0, \ldots, a_m \cdot x \sim 0\}$, then the new polyhedral set is defined as follows, $\{(x, y) \in \mathbb{R}^{2n} : a_1 \cdot y \sim 0, \ldots, a_m \cdot y \sim 0\}$.

Now, by considering Figure 5a, we construct the relation polyhedron for the elements p_1 and p_3 such that there exists a p_2-execution evolving through the list of dynamics in the strongly connected component formed by locations q_1 and q_5 in the *PSS* of Figure 1. The involved dynamics are determined by the sets F_1 and F_5. We define these sets for the stable system in Figure 2a, $F_1 = \{(x, y) : x + y < 0, x + 2y > 0\}$ and $F_5 = \{(x, y) : x < 0, x + 2y > 0\}$, so $d = \{F_1, F_5\}$. Then, we have all the ingredients for constructing the relation polyhedron $\textsc{ReachRelation}(p_1, p_3, D)$. Let us first construct the duplicated polyhedron for p_1. It will be defined as

$$P_1 = \{(x, y, z, t) : x > 0, y = 0, z > 0, t = 0, z = x, t = y\}.$$

Then, the duplicated polyhedron for the final element p_3 is

$$P_2 = \{(x, y, z, t) : z - 2t = 0, t > 0\}.$$

For the polyhedra F_1 and F_5 representing the dynamics, the following duplicated sets are computed, $\{(x, y, z, t) : x = 0, y = 0, z + t < 0, z + 2t > 0\}$ and $\{(x, y, z, t) : x = 0, y = 0, z < 0, z + 2t > 0\}$ respectively. The convex hull generated by them is determined as

$$D = \{(x, y, z, t) : x = 0, y = 0, z + 2t > 0, z < 0\}.$$

Finally, the relation polyhedron returned by the function ReachRelation (p_1, p_3, D) will be the one constructed by considering the initial polyhedron P_1, the final polyhedron P_2 and the dynamics polyhedron $\{(x, y, z, t) : z - 2t = 0, y = 0, x - z >= 0, z > 0, -x + 2z >= 0\}$.

7. Implementation

The stability verification procedure has been implemented in Python and integrated in a tool called Averist (Algorithmic VERIfier for STability) [21, 17], which can be found in `software.imdea.org/projects/averist/index.html`. It uses the Parma Polyhedral Library (PPL) [28] to deal with polyhedral operations, the NetworkX Python package [29] to manage and analyse graphs and the GLPK library [30] for solving optimization problems.

The experiments have been performed on Mac OS X 10.10 with processor 2.8 GHz Intel Core i5 and 8GB 1600 MHz DDR3 memory. They are performed, first, to demonstrate the feasibility of the approach and, second, to evaluate the running times. Comparison with stability verification tools is not possible because, as far as we know, the unique stability-verification available tool works only for switched linear and non-linear systems [31].

We illustrate the algorithm on an example which has been verified using Averist. The switched system is shown in Figure 4 and a pictorial illustration of the system when the value of the variable $z = 1$ is shown in Figure 5b. The system executions do not diverge while remaining in any single cell, and as we will see from the proof of stability given by the tool, the system does not have any infinite executions. Hence, the scaling associated with any execution is bounded and the system is both Lyapunov and asymptotically stable.

Part of the graph constructed by the algorithm is shown in Figure 5b. It has a cycle with weight > 1. The same is returned by the tool. However, observe that the path e_1 to e_2 to e_3 is infeasible even if both the edges e_1 to e_2 and e_2 to e_3 are feasible. We add the constraint $x + y = 0$ shown by the dotted line in Figure 5b to the partition to create a finer partition. This breaks the cycle and Averist returns that the system is stable.

We have experimented with variations of the example described in Figure 4, considering different flow functions, and we extend it to some 4 and 5 dimensional examples. Our results are summarized in Table 1. Stability refers to Lyapunov stability. Here, *Exp* refers to the experiment number,

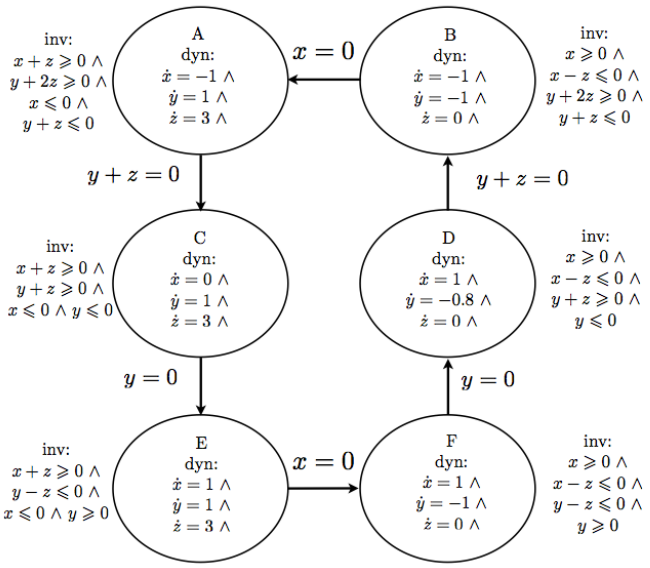
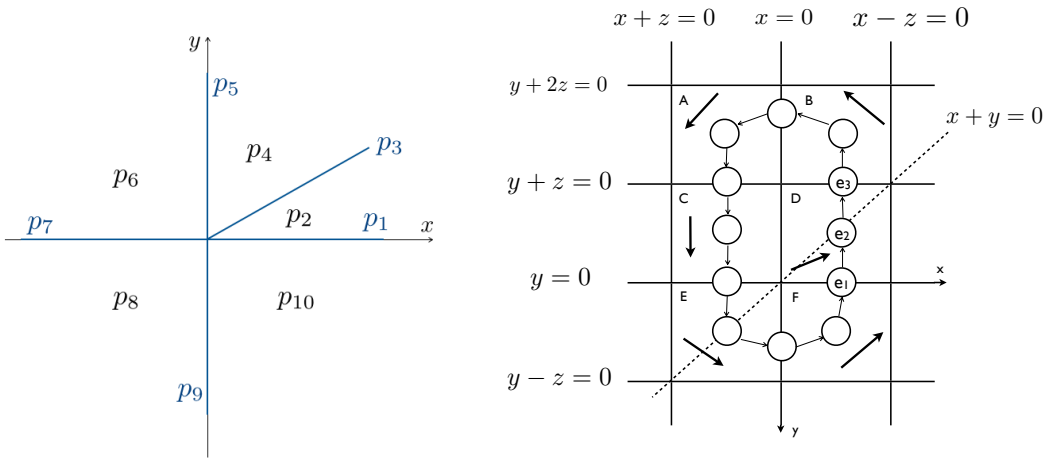


Figure 4: Switched System Example

dim to the dimension of the concrete *PSS*. *Answer* is the verification answer, which can be stable (S), unstable (NS) or inconclusive (IN). *Ref* refers to the number of abstraction refinements performed before termination, *Size* refers to the number of nodes of the final weighted graph. The time for constructing the final weighted graph $\mathcal{G}$ *time* and verification *Ver time* is shown along with the total time *Time*. All the times are in seconds.

Observe that the increment of the time with respect to the dimension of the *PSS* for constructing the weighted graph increments polynomially with respect to the number of abstraction refinements. The time increment with respect to the size of the abstraction is sharper but still not exponential. When comparing the time for the weighted graph construction with the verification time, it is recognizable that the weighted graph construction is a crucial step in terms of computability. There is a remarkable total time increment in the case of increasing the number of refinements when considering the same dimension of the *PSS*. This points to the need of decreasing the number of refinements. In the future, we will develop heuristics to address this.



(a) Planar polyhedral partition (b) Abstraction-Refinement

Figure 5

8. Conclusions

We investigated an algorithmic approach to stability verification of polyhedral switched systems. Our method relies on constructing a finite weighted graph abstracting the system using a finite partition of the state-space as a parameter. In the event of a failure to infer stability from the graph, our algorithm provides insights into the reasons for the failure in terms of a counterexample. In addition, a qualitatively better approximation of the system can be constructed by feeding a finer partition as input to the procedure. The benefits of the approach are illustrated through the analysis of an example using the tool implementing the approach.

Our method can be extended to more complex dynamics provided that the reachability predicate, involved in deducing the existence of edges and computing the weights, can be effectively computed. In the case of linear and non-linear dynamics, the reachability predicate cannot be computed exactly, however, over-approximations using tools such as SpaceEx [32] can be computed. Another direction for future work includes exploring the possibility of automatically refining the partitions by analyzing the counterexample returned by the procedure [33].

Exp	*Dim*	*Answer*	*Ref*	*Size*	$\mathcal{G}$ *time (s)*	*Ver time (s)*	*Time (s)*
1	3	S	1	60	5.56	0.001	6.05
2	3	S	3	83	5.09	0.006	23.6
3	3	S	3	82	7.43	0.001	19.85
4	3	S	7	122	28.62	0.003	98.71
5	3	S	9	142	24.83	0.003	135.39
6	3	S	11	166	37.45	0.009	208.47
7	3	NS	1	–	0	0	0.27
8	3	S	10	143	32.27	0.006	206.04
9	3	S	11	157	42.07	0.009	239.39
10	3	S	16	202	138.08	0.049	1256.9
11	4	S	3	341	285.94	0.019	708.23
12	4	S	7	601	819.87	0.037	3154.2
13	5	S	3	1365	4288.73	0.147	11939.06
14	3	IN	10	115	20.46	0.04	111.43
15	3	IN	10	65	7.71	0.01	226.98
16	3	S	3	74	7.94	0.001	14.33
17	3	S	3	74	9.74	0.001	17.72
18	3	S	3	74	7.7	0.001	14.57
20	3	S	3	74	7.66	0.001	14.5
21	3	S	3	74	7.53	0.001	13.66

Table 1: Experimental results

References

[1] D. Liberzon, Switching in Systems and Control, Boston : Birkhäuser, 2003.

[2] T. A. Henzinger, The Theory of Hybrid Automata, in: Proceedings of the IEEE Symposium on Logic in Computer Science, 1996, pp. 278–292.

[3] H. Lin, P. J. Antsaklis, Stability and stabilizability of switched linear systems: A survey of recent results, IEEE Trans. Automat. Contr. 54 (2) (2009) 308–322.

[4] E. D. Sontag, Input to state stability: Basic concepts and results, in: Nonlinear and Optimal Control Theory, Springer, 2006, pp. 163–220.

[5] R. Goebel, R. Sanfelice, A. Teel, Hybrid dynamical systems, IEEE Control Systems, Control Systems Magazine 29 (2009) 28–93.

[6] J. C. Geromel, P. Colaneri, Stability and stabilization of continuous-time switched linear systems, SIAM J. Control Optim. 45 (5) (2006) 1915–1930.

[7] J. P. Hespanha, Uniform stability of switched linear systems: extensions of lasalle's invariance principle., IEEE Trans. Automat. Contr. 49 (4) (2004) 470–482.

[8] P. Mason, U. V. Boscain, Y. Chitour, Common polynomial lyapunov functions for linear switched systems., SIAM J. Control and Optimization 45 (1) (2006) 226–245.

[9] P. A. Parrilo, Structure Semidefinite Programs and Semialgebraic Geometry Methods in Robustness and Optimization, PhD thesis, California Institute of Technology, Pasadena, CA, May, 2000.

[10] A. Papachristodoulou, S. Prajna, On the construction of Lyapunov functions using the sum of squares decomposition, in: Conference on Decision and Control, 2002.

[11] E. Möhlmann, O. 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, ACM, New York, NY, USA, 2013, pp. 107–112.

[12] J. Kapinski, J. V. Deshmukh, S. Sankaranarayanan, N. Arechiga, Simulation-guided Lyapunov Analysis for Hybrid Dynamical Systems, in: Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control, HSCC '14, ACM, 2014, pp. 133–142.

[13] H. Dierks, S. Kupferschmid, K. Larsen, Automatic Abstraction Refinement for Timed Automata, in: Proceedings of Formal Modeling and Analysis of Timed Systems, 2007, pp. 114–129.

[14] P. Prabhakar, S. Duggirala, S. Mitra, M. Viswanathan, Hybrid automata-based cegar for rectangular hybrid automata, in: Proceedings of the International Conference on Verification, Model Checking, and Abstract Interpretation, 2013.

[15] R. Alur, T. Dang, 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, 2003, pp. 208–223.

[16] E. Clarke, A. Fehnker, Z. Han, B. Krogh, J. Ouaknine, O. Stursberg, M. Theobald, Abstraction and Counterexample-Guided Refinement in Model Checking of Hybrid Systems, International Journal on Foundations of Computer Science 14 (4) (2003) 583–604.

[17] P. Prabhakar, M. G. Soto, AVERIST: An Algorithmic Verifier for Stability, Electronic Notes in Theoretical Computer Science 317 (2015) 133–139.

[18] P. Prabhakar, M. G. Soto, Counterexample Guided Abstraction Refinement for Stability Analysis, in: Proceedings of the International Conference on Computer Aided Verification, 2016.

[19] A. Puri, P. Varaiya, Verification of hybrid systems using abstractions, in: Hybrid Systems II, 1994, pp. 359–369.

[20] E. Asarin, T. Dang, A. Girard, Hybridization methods for the analysis of nonlinear systems, Acta Informatica 43 (7) (2007) 451–476.

[21] P. Prabhakar, M. G. Soto, An algorithmic approach to stability verification of polyhedral switched systems, in: American Control Conference, 2014.

[22] L. de Moura, N. Bjørner, Z3: An efficient SMT solver, in: Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Vol. 4963 of Lecture Notes in Computer Science, Springer, 2008, pp. 337–340.

[23] P.-H. Ho, Automatic Analysis of Hybrid Systems, Ph.D. thesis, Cornell University (1995).

[24] P. Prabhakar, M. Viswanathan, On the decidability of stability of hybrid systems, in: Proceedings of the International Conference on Hybrid Systems: Computation and Control, 2013, pp. 53–62.

[25] P. Prabhakar, M. G. Soto, Abstraction based model-checking of stability of hybrid systems, in: Proceedings of the International Conference on Computer Aided Verification, 2013, pp. 280–295.

[26] R. Bellman, On a Routing Problem, Quarterly of Applied Mathematics 16 (1958) 87–90.

[27] E. Nuutila, E. Soisalon-Soininen, On Finding the Strongly Connected Components in a Directed Graph, Inf. Process. Lett. 49 (1) (1994) 9–14.

[28] R. Bagnara, P. M. Hill, E. Zaffanella, The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems, Science of Computer Programming 72 (1–2) (2008) 3–21.

[29] A. A. Hagberg, D. A. Schult, P. J. Swart, Exploring network structure, dynamics, and function using NetworkX, in: Proceedings of the Python in Science Conference, 2008, pp. 11–15.

[30] A. Makhorin, Glpk (GNU Linear Programming Kit),
Available at `http://www.gnu.org/software/glpk/glpk.html`.

[31] E. Möhlmann, O. Theel, Stabhyli: A Tool for Automatic Stability Verification of Non-linear Hybrid Systems, in: Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control, HSCC '13, ACM, 2013, pp. 107–112.

[32] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, O. Maler, Spaceex: Scalable verification of hybrid systems, in: Proceedings of the International Conference on Computer Aided Verification, 2011.

[33] E. Clarke, O. Grumberg, S. Jha, Y. Lu, H. Veith, Counterexample-Guided Abstraction Refinement, in: Proceedings of the International Conference on Computer Aided Verification, 2000, pp. 154–169.