An Algorithmic Approach for Stability Verification of Hybrid Systems

PhD Thesis

Miriam García Soto, MSc

July 2017
An Algorithmic Approach for Stability Verification of Hybrid Systems

Submitted in partial fulfillment of the requirements for the degree of
Doctor en Software y Sistemas

Author: Miriam García Soto, MSc
Advisor: Pavithra Prabhakar, PhD

July 2017
Abstract:
Cyber-physical systems (CPS) are engineered systems which integrate computer-based algorithms and physical components. One of the main features of CPS is the interaction between the discrete algorithmic behaviour and the continuous physical behaviour, which results in a more intricate performance. This mixed behaviour can be modelled by hybrid systems.

A critical property in hybrid systems design is stability, which ensures robustness of the system with respect to small perturbations of the input to the system. The state-of-the-art methodology for providing stability proofs lies on searching for Lyapunov functions. This thesis proposes an alternative and novel algorithmic approach to stability verification of hybrid systems, which uses formal methods.

The aim of formal verification is to automatically provide strong guarantees of correctness. This thesis introduces the development and implementation of a formal verification method for analysing stability of hybrid systems. Such a stability verification approach consists of an automated proof which relies on abstraction and model checking techniques. The abstraction technique is a modified predicate abstraction that constructs a finite weighted graph from a hybrid system by preserving stability. The evaluation of an abstract weighted graph, which is a finite state system, is more efficient than evaluating a hybrid system with infinite states. Therefore, the abstract weighted graph is model checked to either infer stability of the former system or output a potential instability reason namely counterexample.

The counterexample obtained through model checking can either correspond to an unstable behaviour of the hybrid system or be spurious. This thesis includes a validation algorithm to differentiate between the two possibilities by solving a non-bounded model checking problem, and it also introduces two refinement techniques for the case that the counterexample is spurious. These refinement techniques provide additional predicates, deducted from the counterexample, to construct a more accurate abstract weighted graph.

All the developed techniques (abstraction, model checking, validation and refinement) are articulated to obtain a counterexample guided refinement abstraction (CEGAR) framework. This CEGAR framework allows to iterate over the abstract systems, returns a counterexample if the abstraction fails and uses such a counterexample for guiding the choice of the next abstraction.

The introduced CEGAR algorithm is illustrated by verifying stability of an automatic gearbox and implemented in a software tool called AVERIST. Such an implementation provides empirical evidence of the correctness of the theoretical techniques. Some advantages of using AVERIST are that the input data can be easily defined by people with no experience in hybrid systems and running AVERIST does not require a formal knowledge on stability analysis.
Resumen:
Los sistemas ciberfísicos son sistemas tecnológicos interconectados que integran componentes de índole física y elementos software. Los elementos software evolucionan de forma discreta, mientras las características físicas son de naturaleza continua; la interacción entre ambos comportamientos da lugar a un comportamiento mucho más complejo, que es característico de los sistemas ciberfísicos. Una herramienta clave para modelar este tipo de comportamientos son los sistemas híbridos.

Una propiedad relevante en el diseño de sistemas híbridos es la estabilidad, puesto que garantiza solidez en el estado del sistema a pesar de posibles pequeñas modificaciones en las condiciones iniciales del sistema. El estado del arte en cuanto a la metodología que se emplea en el análisis de estabilidad radica en la búsqueda de funciones de Lyapunov, puesto que la existencia de dicha función supone una demostración de estabilidad. Esta tesis propone un método alternativo y novedoso en el análisis de estabilidad de sistemas híbridos, que consiste en una técnica algorítmica que hace uso de métodos formales.

El objetivo de las técnicas de verificación formal es proporcionar de forma automática garantías sólidas del correcto funcionamiento de un sistema. En esta tesis se desarrolla e implementa un método de verificación formal que analiza la estabilidad en sistemas híbridos. Este método analítico consiste en un proceso automático basado en técnicas de abstracción y verificación de modelos (o model checking). El método de abstracción es una modificación de las abstracciones basadas en predicados, que a partir de un sistema híbrido construye un grafo ponderado finito que conserva la estabilidad. El análisis sobre el grafo ponderado resulta más eficiente que el que se realizaría directamente sobre el sistema híbrido, puesto que este último contiene infinitos posibles estados. Por lo tanto, el grafo abstracto ponderado se analiza para inferir estabilidad del sistema híbrido u obtener una causa potencial de inestabilidad del sistema, denominada contraejemplo.

El contraejemplo, obtenido mediante el análisis aplicado al grafo ponderado, se corresponde a un comportamiento inestable del sistema híbrido o es falso. Esta tesis incluye un algoritmo de validación que distingue entre estas dos posibilidades y que implica verificar una ejecución del modelo con infinitas posiciones. También se incluyen dos técnicas de refinamiento de la abstracción, para el caso en que el contraejemplo sea falso. Las técnicas de refinamiento proveen nuevos predicados, que se añaden en la construcción del grafo ponderado para que este resulte más preciso.

La articulación de todas las técnicas desarrolladas en esta tesis (abstracción, verificación de modelos, validación y refinamiento) conforman una técnica algorítmica conocida como refinamiento de abstracción guiado por contraejemplo (cegar). Este enfoque CEGAR permite iterar sistemáticamente sobre el sistema
abstracto, devolviendo un contraejemplo en caso de que la abstracción no resulte informativa, y usando este contraejemplo como guía para la elección de la siguiente abstracción.

También se presenta el desarrollo de una herramienta software, llamada AVERIST, que incluye la implementación del algoritmo CEGAR y proporciona evidencia empírica de los resultados teóricos. Los datos de entrada de este software se pueden definir fácilmente sin necesidad de experiencia en sistemas híbridos, y su ejecución no requiere conocimientos formales en cuanto al análisis de estabilidad.
## Contents

1 Introduction ................................................................. 1
  1.1 Motivation .......................................................... 2
  1.2 Formal methods ................................................... 3
  1.3 CPS modelling and stability specifications .................. 4
  1.4 Stability verification ............................................. 7
    1.4.1 CEGAR ..................................................... 8
  1.5 Contributions and thesis outline ............................ 10
    1.5.1 CEGAR for stability ..................................... 10
    1.5.2 Hybridization ........................................... 13
    1.5.3 Global asymptotic stability verification ............. 14

2 Preliminaries ............................................................... 15
  2.1 Hybrid automaton: syntax and semantics .................... 19
  2.2 Stability notions ............................................... 22
    2.2.1 Preliminary properties of stability .................. 24

3 Quantitative Predicate Abstraction .................................. 27
  3.1 Polyhedral switched systems ................................ 29
    3.1.1 Functions on switched systems ....................... 31
    3.1.2 Piecewise linear executions and splitting .......... 32
  3.2 Stability verification procedure ............................. 34
    3.2.1 Formal definition of the graph ...................... 34
    3.2.2 Stability Criteria ..................................... 37
    3.2.3 Correctness of the stability criteria ............ 37
    3.2.4 Verifying Conditions [P1-P4] ....................... 39
  3.3 Weighted graph construction ................................ 40
    3.3.1 Algorithm 1 - Abstraction construction ............ 40
    3.3.2 Algorithm 2 - Scaling computation ................ 42
    3.3.3 Reach Relation polyhedron .......................... 44
  3.4 Implementation and experimental results ................... 45
  3.5 Summary .......................................................... 47

4 Counterexample Guided Abstraction Refinement .................. 49
  4.1 Reachability relations ........................................ 51
  4.2 CEGAR algorithm ............................................... 52
  4.3 Validation ........................................................ 55
    4.3.1 Validation Problem .................................... 55
    4.3.2 Validation procedure ................................. 57
  4.4 Refinement ........................................................ 60
    4.4.1 Refinement problem ................................... 60
INTRODUCTION

This chapter introduces cyber-physical systems, the challenges related to their design and analysis, and highlights the motivation of the research conducted in the thesis.

Contents

1.1 Motivation ........................................... 2
1.2 Formal methods ................................... 3
1.3 CPS modelling and stability specifications .......... 4
1.4 Stability verification .................................. 7
  1.4.1 CEGAR ........................................... 8
1.5 Contributions and thesis outline ...................... 10
  1.5.1 CEGAR for stability .............................. 10
  1.5.2 Hybridization .................................... 13
  1.5.3 Global asymptotic stability verification .......... 14
Cyber-physical systems (CPS) are a generation of innovative systems which are controlled by computer-based algorithms while integrated in a physical environment. The interaction of software components with physical processes is one of the main features of this kind of systems. The software exhibits discrete behaviour while the physical component behaves in a continuous manner.

Cyber-physical systems appear widely in the fields of automotive systems, energy, assisted living, robotics and avionics. A cruise control interacting with an automatic gearbox [PL00], a smart energy grid [KM13], a pacemaker [BJM13], a healthcare monitoring system [HGY+14] and a brain-controlled prosthesis [ZLL14] are some of the examples which surround us. For instance, a pacemaker is required to control the continuous heart rate in order to keep the ratio of venous oxygen under a predefined value. This is achieved by applying discrete electrical impulses to the heart. Another example is a cruise control in a vehicle which consists of software that measures the vehicle velocity and appropriately sets the engine throttle and the corresponding gear position.

Critical aspects to be considered in the CPS design are security, reliability and safety. In the case of a pacemaker, a missed electrical impulse can result in the body not getting oxygenated blood in time. A failure in the speed regulation of an autonomous car could provoke a car crash. The prevention of such incidents requires the use of a rigorous methodology to validate and verify pivotal properties in the CPS.

The state of the art for verifying and validating software and systems relies on testing and simulation. These solutions are not in general cost-efficient and they do not provide a proof of correctness for the system performance. In order to build high-confidence systems, solid modelling formalisms along with verification and validation algorithms have been developed in the area of computer science. These techniques fall into the discipline known as “formal methods”, which has its foundations in rigorous mathematical modelling and analysis.
Formal methods are mathematical techniques applied to the specification and verification of hardware and software systems. The goal of formal methods consists of providing rigorous guarantees about the correctness of a system. The use of formal methods in the early stages of system design can provide either earlier detection of failures or a certificate of correct performance. The latter will provide strong guarantees of reliability of the system. More precisely, given a mathematical model of the system and a specification, formal verification techniques demonstrate correctness of the system behaviour with respect to its specification. A general framework for verification is shown in Figure 1.1, where the input is a model of the system and a formal specification of correctness, and it outputs either that the system is correct with respect to the specification along with a correctness certificate or that it is incorrect along with a counterexample.

![Figure 1.1: Formal verification](image)

Many approaches in industrial environments rely on simulation and testing to analyse the correctness of the performance of the systems. Both techniques consist of observing the behaviour of the system with different inputs and checking if this behaviour satisfies the specified requirements. Even though these techniques are useful for bug detection, they cannot consider every potential input to the system. This shortcoming renders it impossible to provide a certificate of correctness. In addition, considering every input that should be tested leads to an unaffordable cost.

Trying to overcome the cost-inefficiency and lack of reliability guarantees of simulation and testing, alternative and complementary verification methods have been proposed in the area of formal methods for the last forty years [BH14]. These approaches have gained attention from the industrial sectors in the last decade as evidenced in the following examples: The development of a smartcard product for electronic commerce by the National Westminster Bank (NatWest) [SCW00], the development of Tokeneer system for controlling the
physical access to a domain by the NSA (National Security Agency) [RL98] and the fully formally verified general-purpose operating system micro kernel seL4 [KAE +14].

Two extensively studied techniques in formal verification are theorem proving [Rob01, NPW02] and model checking [BK08, CGP99]. Theorem proving is a deductive verification technique where the system and the design specification, are expressed as formulas in a logic. These formulas are then derived using the axioms and inference rules of the formal logic in order to obtain a proof of the property. This technique has the advantage of tackling complex system designs, but often requires manual intervention. On the contrary, model checking is an algorithmic technique which consists of an exhaustive state space exploration to infer the correctness of the system. While model checking is fully automated, it is restricted in the class of systems to which it can be directed applied, such as finite state systems. To apply to more general class of systems, symbolic state space exploration and abstraction methods have been developed.

1.3 CPS modelling and stability specifications

The goal of this thesis is to develop model checking techniques for the stability verification of CPS. Stability verification refers to analyse whether the behaviour of a system approaches to a predefined behaviour. This section provides a brief overview of hybrid systems as a model for capturing the intricate behaviours of CPS and discusses formal specifications for stability.

System model: hybrid system Hybrid systems are systems exhibiting mixed discrete and continuous behaviours. This feature manifests naturally in CPS as the result of the interaction between software components, which execute in discrete steps, and the physical components, which evolve continuously in real time. Discrete systems have been investigated in depth in computer science to model software and hardware systems, while continuous dynamical systems have been studied in modelling and analysing engineering applications, in particular in the domain of control systems. Hence, a challenging issue in the analysis of hybrid systems is to develop novel techniques which borrow insights from the existing techniques in the two areas.

Example 1. An illustrative example of a hybrid system is a cruise controller which interacts with an automatic gearbox to set the velocity of a vehicle to a desired velocity [PL00]. The gearbox receives the torque from the engine and appropriately sets the velocity depending on the current gear position, while
The cruise control receives the current velocity and, depending on the difference between the specified velocity and the current velocity, sets the gear. The automatic gearbox has four positions and at each position, the velocity of the vehicle evolves according to different continuous dynamics. When some constraint over the velocity is satisfied, a change in the gear position occurs, which corresponds to the discrete behaviour of the system.

Figure 1.2: Model of the gearbox

The cruise controller and automatic gearbox is modelled as a hybrid automaton, as shown in Figure 1.2. Here, the locations/modes (circles) represent the set of gear positions, \( Q = \{1, 2, 3, 4\} \). The continuous dynamics of the system describes the behaviour of two variables over time, \( E \) and \( T_I \), where \( E \) is the difference between the desired velocity \( v_d \) and the current velocity \( v \) of the vehicle and \( T_I \) is the integral part of the torque from the engine. The dynamics are determined by the vector field function, which in this case corresponds to a linear differential equation of the form \( \dot{x} = A_q x \) (where \( \dot{x} \) refers to \( \frac{dx}{dt} \)), for each gear position \( q \) and with \( x = (E, T_I)^T \). A mode switching corresponds to a change of gear position (edges), either to the next position, from \( q \) to \( q + 1 \), or to the previous position, from \( q \) to \( q - 1 \), for \( q \in Q \), when some constraint over the velocity is satisfied.

Figure 1.3 shows some sample executions of the automatic gearbox for a particular choice of \( v_d \). The horizontal axis represents the velocity difference \( E \) and the vertical axis represents the integrator torque \( T_I \). Since the value \( T_I \) is set to 0 every time the driver inputs a new desired velocity, every execution starts at the horizontal axis. The origin corresponds to \( v_d = v \), that is, the state where the target velocity is achieved. Vertical red and blue lines correspond respectively to upward and downward gear position changes.

The characteristic behaviour of hybrid systems is clearly shown in the gearbox example, where the continuous evolution is defined by the linear dynamics over \( E \) and \( T_I \), and the discrete evolution corresponds to the gear position switchings. Further details about this gearbox model will be introduced in Chapter 6.
6 1. Introduction

Figure 1.3: Gearbox sample executions

**Design specification: stability** A specification describes operational and performance requirements of a system. The target of this dissertation is stability verification of hybrid systems, where stability is a fundamental performance property required in control system design. This property captures the notion that small perturbations in the initial state or input to the system will result only in small variations of the behaviour of the system. This leads to the system executions starting at any state to remain close or to converge to a desired behaviour.

This thesis considers three classical notions of stability, known as Lyapunov stability, asymptotic stability and global asymptotic stability. A system is Lyapunov stable if the executions of the system starting close to a predefined state, remain close to this state. More strongly, if the executions starting close to a predefined state not only remain close to the predefined state but also converge to it, the system is asymptotically stable. Lastly, an even stronger notion of stability is global asymptotic stability, which states that the system is Lyapunov stable and every execution of the system, starting from any state, converges to a predefined state.

Consider the previous hybrid system example, namely, the cruise control interacting with an automatic gearbox which appears in Figure 1.2. The desired behaviour of the system consists of achieving a velocity set by the user starting from any possible initial velocity of the vehicle and despite small velocity perturbations due to, for instance, uphill or downhill motions of the vehicle. This implies that every execution of the system converges to $E = 0$, that is $v = v_d$. Therefore, establishing that the system satisfies the desired behaviour is equivalent to verifying that it is globally asymptotically stable with respect to the origin.
1.4. Stability verification

The next section will provide an overview of the state of the art for stability verification and the novel approach developed in this thesis with ideas from formal methods.

1.4 Stability verification

The classical approach to stability analysis in control theory is based on Lyapunov functions [Kha02]. A Lyapunov function for an $n$-dimensional dynamical system $\dot{x} = f(x)$, is a continuously differentiable function of the form $V : \mathbb{R}^n \to \mathbb{R}$, such that it is positive definite, $V(x) > 0$, and its value along any solution of the dynamical system is decreasing, which is captured by the algebraic expression $\nabla V(x) \cdot f(x) < 0$ where $\nabla V(x) = \frac{\partial V(x)}{\partial x}$; for every $x \in \mathbb{R}^n \setminus \{0\}$.

For the case of a continuous dynamical system, the existence of a Lyapunov function corresponds to a stability certificate. The approach has been extended to hybrid systems in the form of common and multiple Lyapunov functions [Son06, GST09, LA09, GC06, Hes04, MBC06]. Automated analysis involves starting with a template which serves as a candidate Lyapunov function and then encoding the conditions of the Lyapunov function in form of a sum-of-squares programming problem over the template. The sum-of-squares programming problem can be efficiently solved using tools such as sostools [Par00, PP02, MT13].

**Example 2.** Consider for instance a two dimensional linear dynamical system of the form $\dot{x} = Ax$, where $x = (x_1, x_2) \in \mathbb{R}^2$ and the dynamics is defined by the matrix

$$A = \begin{pmatrix} -1 & -1 \\ 1 & -1 \end{pmatrix}.$$ 

A candidate Lyapunov function is instantiated as $V(x_1, x_2) = ax_1^2 + bx_1x_2 + cx_2^2$ where $a, b, c \in \mathbb{R}$ are unknown parameters. The candidate function needs to satisfy two conditions to be a Lyapunov function, which are to be positive definite, $V(x) > 0$, and to decrease along the solutions of the system, $\nabla V(x) \cdot Ax < 0$, for every $x \in \mathbb{R}^2 \setminus \{(0, 0)\}$. These two conditions result in the following inequalities:

$$ax_1^2 + bx_1x_2 + cx_2^2 > 0 \quad \text{and} \quad (2ax_1 + bx_2, bx_1 + 2cx_2) \begin{pmatrix} -x_1 - x_2 \\ x_1 - x_2 \end{pmatrix} < 0 \quad \text{for} \quad x_1, x_2 \neq 0.$$ 

A successful triple of parameter values as $a = 1$, $b = 0$ and $c = 1$ can be obtained by means of a sum-of-squares software tool. The existence
of such parameters ensures the existence of a Lyapunov function, which in addition ensures stability of the original linear dynamical system.

The major difficulty with this approach is to determine the right template for the candidate Lyapunov function. Furthermore, with the exception of [KDSA14], automatically learning the templates is a problem which has not been adequately addressed. In summary, the main drawbacks in this template-based search methodology are that the success of the approach depends on a smart choice of the template, and that a template failure does not provide insights into the reason for the failure or into the next choice of a template.

Motivated by these issues, this dissertation introduces an algorithmic approach, not based on Lyapunov functions search, which iterates over abstractions of the original hybrid system, in a guided and automatic fashion. In particular, the algorithmic approach is a novel instantiation of the so-called Counterexample Guided Abstraction Refinement (CEGAR) [CGJ+00] method, oriented to obtain a stability certificate.

\section{CEGAR}

In general, a direct analysis of the correctness of a system is not cost efficient due to the complexity of the system. A standard procedure to facilitate the analysis involves constructing a simplification, also known as abstraction, of the system and evaluating over it correctness conditions. A too coarse simplification leads to obtain an inconclusive analysis result, while a too tight simplification causes a cost inefficient analysis, as in the case of the original system. The difficulty here lies on constructing an enough accurate simplification of the system which results in an affordable cost for analysis. The CEGAR method consists of an algorithmic approach which searches for this accurate simplified system by iteratively refining an initial abstract system such that the refinement procedure is based on the property analysis output.

Formally, a CEGAR approach is a verification algorithm which integrates abstraction and model checking techniques as shown in Figure 1.4. The input to the algorithm is a concrete system and a property, and it outputs if the system either satisfies or does not satisfy the property. The concrete system is simplified into a finite system which over-approximates the behaviour of the former one. This finite system is obtained by an abstraction procedure, and it is explored by a model checking algorithm which establishes correctness of the concrete system with respect to the property, or outputs an abstract
1.4. Stability verification

The abstract counterexample is a potential reason for violation of the property and a validation function evaluates if it corresponds to a concrete counterexample or not. In case of existence of a concrete counterexample, the CEGAR approach outputs that the property is violated, while if the abstract counterexample is spurious, the validation result will be input to a refinement procedure. The refinement procedure evaluates the abstract counterexample and provides additional constraints to the abstraction process, resulting in the construction of a more accurate abstract system, where the abstract counterexample is eliminated.

Initially, the CEGAR paradigm was proposed for discrete systems in the context of hardware systems [CGJ+00, Kur94], and applied to software systems [BPR02]. Subsequently, several CEGAR approaches have been extended to more complex systems such as hybrid systems. These particular approaches attack mainly safety verification problems [ADI03, CFH+03, PDMV13], but not stability verification problems, except for the attempt proposed in [DM11], which considers a CEGAR application to region stability.

The application of the CEGAR algorithm to the classical notions of stability that this thesis considers is not straightforward, unlike for the case of region stability. Bisimulation relations [Mil89] are the foundational basis for the abstraction procedure in the case of region stability, since these relations preserve properties expressible in linear temporal logics and region stability is one of those. The considered notions of stability cannot be expressed in linear temporal logics, hence standard abstractions are not useful for them. Results in [Cui07, PDV12, PLM13] show that stability is not bisimulation invariant and that bisimulation relations need to be reinforced with continuity conditions in order to define abstractions preserving the stability property.
1.5 Contributions and thesis outline

This dissertation provides the first CEGAR framework to apply abstraction and model checking algorithms for the verification of Lyapunov and asymptotic stability of hybrid systems.

1.5.1 CEGAR for stability

The CEGAR approach for stability introduces a novel quantitative predicate abstraction which outputs a weighted graph, and provides a set of verifiable conditions over the abstract weighted graph for evaluating stability satisfaction. The stability analysis returns either a sound stability certificate or an abstract counterexample. In case of a counterexample detection, this will provide insights about the reason for instability or to select a better abstraction.

The CEGAR approach systematically iterates over the abstract systems, returns a counterexample in the case that the abstraction fails and the counterexample is used to guide the choice of the next abstraction; in contrast to the template-based search illustrated in Example 2, where the success of the methodology requires the ingenuity of the user in the choice of the initial template, and the failure of the template does not provide insights in either the reason for the violation of the property or the choice of the next template.

The contributions of this thesis are the development and implementation of the building blocks involved in the CEGAR algorithm. They are listed and detailed in the following paragraphs.

Quantitative predicate abstraction The core contribution is the development of a quantitative predicate abstraction (QPA) technique over hybrid systems with polyhedral inclusion dynamics. Various methods include predicate and hybrid abstractions [DKL07, PDMV13, ADI03, CFH+03] for computing abstract models oriented to safety verification. These standard methods do not suffice for stability analysis, so a novel predicate abstraction technique is introduced for stability preservation.

The novel abstraction technique is a modified predicate abstraction which constructs a finite weighted graph from a polyhedral hybrid system and from a finite partition of the state space. The output weighted graph is a conservative approximation.
Figure 1.5: Local quantitative predicate abstraction of gearbox

A graph refers to a collection of nodes and a collection of edges connecting the nodes. The nodes of the abstract graph correspond to a pair of a mode and an element. The mode belongs to the hybrid system and the element belongs to the state space partition. An edge between two mode-element pairs indicates the existence of an execution starting from a state in the first element to a state in the second element such that it remains in a single element at all the intermediate states. The initial and final states belong to the first and the second modes, respectively. Therefore, existence of an edge corresponds to satisfiability of a non trivial reachability predicate.

The additional element of the QPA with respect to standard predicate abstractions is the quantization over the edges. The weight on the edges traces the distance of the executions from the origin, and its computation is a crucial part of the approach. Weight computation corresponds to solving an optimization problem over the reachability predicate. A formula equivalent to the predicate is constructed. This formula is a boolean combination of linear constraints. Hence, the weight is computed by solving a bunch of linear programming problems.

In the abstracted polyhedral hybrid system from the gearbox in Example 1.3, the behaviour close to the origin, so the one illustrated by the executions in the most left region, is abstracted into a weighted graph as shown in Figure 1.5, after hybridization.

The QPA provides a finite weighted graph, preserving stability, from a polyhedral hybrid system. The theoretical foundations and results related to the abstraction technique are presented in Chapter 3. Next, efficiently verifiable conditions on the graph are presented such that their satisfaction implies Lyapunov and/or asymptotic stability. Hence, the verification problem is efficient when reduced to that over a finite state system as compared to that over an infinite state system corresponding to the original hybrid system.
Model checking technique  The weighted graph is analysed for the absence of cycles with weight greater than 1, which can be solved by dynamic programming. The absence of these cycles corresponds to a stability certificate for the polyhedral hybrid system. While the existence of a cycle of this kind corresponds to a potential reason for instability, and it is referred to as abstract counterexample. The theoretical foundations and results related to the conditions for stability over the weighted graph are presented in Chapter 3.

Figure 1.5 shows a weighted graph in the right hand side with a single cycle where the weight of the cycle, that is the product of the weights, is approximately 0.04. Hence, stability of the system in the left hand side of Figure 1.5 is established.

The existence of an abstract counterexample is due to two different reasons. One reason is the existence of an infinite divergent execution in the hybrid system, which hence refutes stability of this system and is abstracted by the counterexample. The other reason is the fact that the finite weighted graph abstracted from the hybrid system is too coarse. In this second case the abstract counterexample is spurious. A validation algorithm identifies the reason for the presence of a counterexample.

Validation  The validation procedure consists of checking the existence of a stability-refuting execution which is expressed in the weighted graph by following the abstract counterexample. In the context of safety analysis [CGJ+00], validation corresponds to checking if there exists a finite execution of the system which follows a sequence of abstract states. However, the validation problem in the context of stability analysis is not a bounded model checking problem as in safety analysis. Instead, it consists of checking if there exists an infinite diverging trajectory that follows the counterexample cycle infinitely many times. This property cannot, as is, be encoded as the satisfiability of a formula in a finitary logic. This thesis introduces a novel 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 by means of SMT solvers. The validation technique based on such characterization is presented in Chapter 4.

Refinement  The validation procedure can determine that an infinite diverging execution does not exist in the polyhedral hybrid system. In this case, the finite abstract graph is too coarse. Hence, a tighter abstraction needs to be constructed such that the abstract counterexample is eliminated. 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 uniformly the state space. A contribution in this thesis is the other refinement strategy that is an algorithmic technique which constructs predicates based on the counterexample. The extra predicates are added in the quantitative predicate abstraction process to refine the state space partition in order to obtain a more accurate weighted graph. The theoretical foundations and results related to the refinement techniques are presented in Chapter 4.

The QPA, validation and refinement algorithms are integrated to describe the CEGAR framework for stability analysis of polyhedral hybrid systems.

The previous results are further developed in two directions. On one side, a more complex class of hybrid systems is considered, and on the other side, a new stability property is introduced. The stability notions considered in the previous framework are Lyapunov and asymptotic stability, which are local properties. Local properties refer to a particular region in the state space, unlike the new one, global asymptotic stability (GAS), which refers to the full state space. An extension of the previous algorithms is introduced in this thesis to obtain GAS verification of linear hybrid systems.

1.5.2 Hybridization

The continuous behaviour of linear hybrid systems is determined by linear dynamics, and the computation of the weights in the QPA procedure for this kind of dynamics is of high complexity. An efficient approach to overcome such drawback consists of defining a simpler system based on a linear hybrid system and that is sound with respect to stability satisfaction (stability of the simpler system implies stability of the linear hybrid system). The contribution to this end is to define an abstraction technique for over approximating the behaviour of a linear hybrid system by a polyhedral hybrid system. The abstraction technique is called hybridization and consists of splitting the state space into a finite number of regions and approximating the linear dynamics in each region by a polyhedral inclusion dynamics. Hybridization approaches for safety analysis already exist [ADG07, DMT10], but they are not appropriate for stability purposes. We have introduced a novel hybridization procedure, where the state space splitting is specifically tuned for stability verification. The computed polyhedral hybrid system enables to define scalable stability verification techniques in order to verify stability of the linear hybrid system. The theoretical foundations and the results related to this hybridization technique are presented in Chapter 5.
For instance, the gearbox introduced in Example 1 is modelled as a linear hybrid system for stability analysis, the specific hybridization needs to be applied in order to construct a polyhedral hybrid system.

**Algorithmic verifier of stability** A software tool called AVERIST has been built to implement the *cegar* approach for Lyapunov stability verification of linear hybrid systems. Its flowchart is shown in Figure 1.6. Details on the architecture and the implementation of AVERIST are presented in Chapter 7.

### 1.5.3 Global asymptotic stability verification

The GAS verification is built up in the above-mentioned algorithmic approaches. The main result consists of a decomposition theorem that reduces the GAS verification problem into a region stability (RS) analysis problem and an asymptotic stability (AS) analysis problem. The developed QPA technique is used for AS analysis and also for extracting a stability zone required for the RS analysis. In addition to the decomposition result, the fundamental contributions of the extended procedure are the algorithmic computation of the stability zone and the model checking algorithm for RS verification.

These results are presented in Chapter 6 along with a semi-automated proof of GAS for the cruise controller interacting with the automatic gearbox presented in Example 1. This proof shows in detail the local AS stated by evaluating the quantitative predicate abstraction illustrated in Figure 1.5, in addition to the RS verification problem.
2

Preliminaries

This chapter provides fundamental definitions and preliminary results to be used throughout the thesis dissertation.

Contents

2.1 Hybrid automaton: syntax and semantics . . . . . . . . . . 19
2.2 Stability notions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
  2.2.1 Preliminary properties of stability . . . . . . . . . . . . . . . . . . 24
Numbers  Let \( \mathbb{R}, \mathbb{R}_{\geq 0}, \) and \( \mathbb{N} \) denote the set of reals, non-negative reals and natural numbers, respectively.

Functions  Given a function \( F \), we use \( \text{dom}(F) \) to denote the domain of \( F \), and \( \text{img}(F) \) to denote the image of \( F \). Given a set \( A \subseteq \text{dom}(F) \), we denote by \( F|_A \), 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 \( \text{len}(S) \), is \( n+1 \) if \( D = \{0, 1, \ldots, n\} \) and \( \infty \) if \( D = \mathbb{N} \). \( S \) is also represented by enumerating its elements as in \( S(0)S(1) \ldots \).

Euclidean space  The \( n \)-dimensional Euclidean space is given by \( \mathbb{R}^n \). A point in the continuous space \( \mathbb{R}^n \) is represented by \( x \) which is of the form \((x_1, \ldots, x_n)\). The addition of two points, \( x + y \), is defined as \((x_1 + y_1, \ldots, x_n + y_n)\), and the product of a constant \( c \in \mathbb{R} \) and \( x \in \mathbb{R}^n \), denoted \( cx \), is equal to the point \((cx_1, \ldots, cx_n)\). 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 \( \varepsilon \in \mathbb{R}_{\geq 0} \), we use \( B_{\varepsilon}(x) \) to denote an open ball around \( x \) of radius \( \varepsilon \), that is, \( B_{\varepsilon}(x) = \{y : |x - y| < \varepsilon\} \).

A set \( X \subseteq \mathbb{R}^n \) is open if for every \( x \in X \), there exists a \( \delta > 0 \) such that \( B_{\delta}(x) \subseteq S \); a set \( X \subseteq \mathbb{R}^n \) is closed if \( \mathbb{R}^n \setminus X \) is open; and a set \( X \subseteq \mathbb{R}^n \) is convex if for every pair of points \( x, y \in \mathbb{R}^n \) and every value \( \alpha \in [0, 1] \), the point \( \alpha x + (1 - \alpha)y \) belongs to \( X \). A set \( X \) is closed under positive scaling if for every \( x \in X \) and \( \alpha > 0 \), \( \alpha x \in X \). Note that open sets are closed under arbitrary union and closed sets are closed under arbitrary intersection. Hence, given any set \( X \subseteq \mathbb{R}^n \), the interior of \( X \), denoted \( \check{X} \), is the largest open subset of \( X \); the closure of \( X \), denoted \( \overline{X} \) is the smallest closed set containing \( X \), and the boundary of \( X \), denoted \( \partial(X) \), is the set \( \overline{X} \setminus \check{X} \). Given sets \( X, Y \subseteq \mathbb{R}^n \), \( X \times Y \) denotes the Cartesian product \( \{(x, y) : x \in X, y \in Y\} \) and \( X + Y \) denotes the Minkowski sum \( \{x + y : x \in X, y \in Y\} \). Given a set \( X \subseteq \mathbb{R}^n \), we use \( \text{chull}(X) \) to denote the smallest convex set containing \( X \). The Hausdorff distance between two sets \( X, Y \subseteq \mathbb{R}^n \) is defined as \( d_H(P, Q) = \max\{\sup_{x \in X} \inf_{y \in Y} |x - y|, \sup_{y \in Y} \inf_{x \in X} |x - y|\} \).

Intervals, time domain and interval domain  An interval is a closed convex subset of \( \mathbb{R} \). Given an interval \( I \), \( \text{first}(I) \) denotes the greatest lower bound of \( I \) and \( \text{last}(I) \) denotes the least upper bound of \( I \). A time domain is an interval \( I \) such that \( \text{first}(I) = 0 \). 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
An interval domain is a finite or infinite sequence of intervals \( i = I_0I_1 \cdots \) such that \( \text{first}(I_0) = 0, \text{last}(I_i) = \text{first}(I_{i+1}) \) for \( 0 \leq i < \text{len}(i) \), and if \( \text{dom}(i) \) is infinite, then for every \( n \in \mathbb{N} \), there exists \( m \in \mathbb{N} \) such that \( n \in I_m \). We denote by \([i]\) the interval \( \bigcup_{i \in \text{dom}(i)} i(i) \). The concatenation of an interval domain \( i_2 \) to a finite interval domain \( i_1 \), denoted \( i_1 \circ i_2 \), is the interval domain \( i \) whose domain is \( \text{dom}(i_1) \cup (\text{dom}(i_2) + \text{len}(i_1)) \), and \( i(i) = i_1(i) \) if \( i \in \text{dom}(i_1) \), and is \( i_2(i - \text{len}(i_1)) \) 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 \( S_1(\text{len}(S_1)) = S_2(0), \text{len}(S) = \text{len}(S_1) + \text{len}(S_2) \), and \( S(i) = S_1(i) \) if \( i \in \text{dom}(S_1) \), and \( S_2(i - \text{len}(S_1)) \), 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 \( \text{last}(I_1) = \text{first}(I_2) \), and \( I = I_1 \cup I_2 \). Interval domains \( i_1 \) and \( i_2 \) form a splitting of an interval domain \( i \), denoted \( i = i_1 \circ i_2 \), if \( \text{len}(i) = \text{len}(i_1) + \text{len}(i_2) \), and \( i(i) = i_1(i) \) if \( i < \text{len}(i_1) \) and \( i(i) = i_2(i - \text{len}(i_1)) \) otherwise.

Convex polyhedral sets A linear expression over \( \mathbb{R}^n \) is a function \( p : \mathbb{R}^n \to \mathbb{R} \) where \( p(x) = a \cdot x + b \) with \( a \in \mathbb{R}^n \) and \( b \in \mathbb{R} \) being constant values, and \( x = (x_1, \cdots, x_n) \) being a tuple of variables. A linear constraint is an expression of the form \( p(x) \sim 0 \), where the relation \( \sim \) belongs to \( \{<, \leq, =\} \). 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 \( \text{hyper}(n) \). Given a set \( P \subseteq \mathbb{R}^n \) of dimension lower than \( n \), \( \text{plane}(P) \) is the intersection of all hyperplanes which contain \( P \),

\[
\text{plane}(P) = \bigcap_{h \in \text{hyper}(n), P \subseteq h} h,
\]

while in case of dimension equal to \( n \), \( \text{plane}(P) = \mathbb{R}^n \). 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 homogeneous linear constraints and elementary if the linear constraints defining the half-spaces only use the relation \( \{<\} \). A minimal set of linear constraints that define a convex
polyhedral set $P$ is denoted as $\text{lc}(P)$. A cone generated by a polyhedral set $P$ is defined as $\text{cone}(P) = \{ \alpha x \in \mathbb{R}^n : x \in P, \alpha \in \mathbb{R}_{\geq 0} \}$. We will use $\text{polysets}(X)$ to denote the set of all convex polyhedral subsets of $X$, and $\text{cpolysets}(X)$ to denote the set of compact convex polyhedral sets. In case of $X = \mathbb{R}^n$ we denote $\text{polysets}(n)$ and $\text{cpolysets}(n)$.

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 $X = \bigcup_{i=1}^k P_i$ and $\hat{P}_i \cap \hat{P}_j = \emptyset$ for $1 \leq i, j \leq k$. The elements of a polyhedral partition are indistinctly referred to as regions or elements. An elementary partition is a partition in which all the convex polyhedral sets are elementary.

Linear, piecewise-linear and differentiable functions A trajectory $\eta$ is a function from a time domain $D$ to $\mathbb{R}^n$ for some $n$. A trajectory $\eta$ is said to be linear if there exist $a, b \in \mathbb{R}^n$, such that for every $t \in \text{dom}(\eta)$, $\eta(t) = at + b$; and piecewise-linear if there exists an interval domain $\iota$ such that $[\iota] = \text{dom}(\eta)$ and $\eta|_{\iota(i)}$ is linear for every $i \in \text{dom}(\iota)$. A trajectory $\eta$ is said to be differentiable if its derivative exists at all points of the domain, and the derivative of $\eta$ 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 $\pi = v_0v_1 \ldots$ such that $(v_i, v_{i+1})$ is an edge for each $i < \text{len}(\pi)$. Let 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 $\mathcal{G} = (V, E, w)$ where $(V, E)$ is a graph and $w : E \to \mathbb{R}_{\geq 0} \cup \{+\infty\}$ is a weighting function on the edges. The weight of a finite path $\pi$ is $\Pi_{i \in \text{dom}(\pi)} w(\pi(i))$. Given a set of vertices $V' \subseteq V$, we use $\mathcal{G}[V']$ to represent $(V', E \cap V' \times V', w|_{V' \times V'})$.

A strongly connected component $\text{scc}$ of a graph $\mathcal{G}$ is a set of vertices $V'$ of $\mathcal{G}$ such that for every $v_1, v_2 \in V'$, there is a path in $\mathcal{G}[V']$ from $v_1$ to $v_2$. Let $\text{scc}(\mathcal{G})$ denote the set of maximal strongly connected components of $\mathcal{G}$. Note that $\text{scc}(\mathcal{G})$ is a partition of the vertices of $\mathcal{G}$. The quotient graph $\mathcal{G}/\text{scc}$ is a graph where the set of vertices is $\text{scc}(\mathcal{G})$ and the edges correspond to pairs $(C_1, C_2) \in \text{scc}(\mathcal{G})$ with $C_1 \neq C_2$ such that there exist $c_1 \in C_1$ and $c_2 \in C_2$ with $(c_1, c_2)$ being 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 to compute the same [MB08].

2.1 Hybrid automaton: syntax and semantics

A hybrid automaton [Hen00, ACHH92] is a popular formalism to model mixed discrete and continuous behaviours, and hence hybrid systems. It extends the finite state automaton model for discrete dynamics by annotating the modes with differential equations or differential inclusions for modelling the physical systems. In addition, invariants on the modes and guards on the edges provide constraints that need to be satisfied during evolution and mode switching, respectively.

Definition 1. A $n$-dimensional hybrid automaton (HA) 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;
- $F : Q \rightarrow (\mathbb{R}^n \rightarrow 2^{\mathbb{R}^n})$ is the vector field function;
- $I : Q \rightarrow \text{polysets}(n)$ is the invariant function; and
- $G : E \rightarrow \text{polysets}(n)$ is the guard function.

Notation. We will denote each of the elements in a HA $\mathcal{H}$ with $\mathcal{H}$ as a subscript to avoid confusion when necessary. For instance, the invariant function will be referred to as $I_\mathcal{H}$.

A polyhedral partition $\mathcal{P}$ is said to respect a HA $\mathcal{H}$ if for every $P \in \mathcal{P}$, $q \in Q_\mathcal{H}$ and $e \in E_\mathcal{H}$, either $P \subseteq I_\mathcal{H}(q)$ or $P \cap I_\mathcal{H}(q) = \emptyset$ and either $P \subseteq G_\mathcal{H}(e)$ or $P \cap G_\mathcal{H}(e) = \emptyset$.

The hybrid automaton evolves continuously by following the vector field function at each location. It executes starting in a mode $q$ and a continuous state $x$. 

For every location \( q \in Q \) the state follows the vector field \( F(q) \) while being inside the invariant \( I(q) \). The continuous state switches from one location to other when it satisfies the guard between these locations. The vector field function is typically represented as a differential inclusion \( \dot{x} \in F(x) \) or a linear differential equation \( \dot{x} = Ax \), where \( F(x) \) represents a polyhedron and \( A \) is a real matrix.

A hybrid automaton is linear if in every mode \( q \) the behaviour of the system is determined by a linear dynamical system of the form \( \dot{x} = A_q x \). Hence, for every \( q \in Q \) and \( x \in I(q) \), the flow in the hybrid automaton is given by \( F(q)(x) = \{ A_q x \} \). And we say that a hybrid automaton is polyhedral, when, for every mode \( q \), the dynamics is defined by a polyhedral inclusion of the form \( \dot{x} \in P_q \) where \( P_q \in \text{cpolysets}(n) \). Here, the flow is specified as \( F(q)(x) = P_q \) for every \( q \in Q \) and \( x \in I(q) \).

In particular, the system model to consider in the dissertation is the switched hybrid system \[Lib03\], a special kind of hybrid automaton in which the continuous state does not change during a mode switch. Concretely, the focus is on polyhedral switched systems (pss) and linear switched systems (lss).

**Example 3.** Figure 2.1 shows an example of linear switched systems. The continuous evolution of these systems is determined by linear dynamical systems. Let us consider two linear dynamical systems defined by matrices \( A \) and \( B \). The phase portraits for both systems are shown in Figure 2.1a and Figure 2.1b respectively. We define two linear switched systems \( \mathcal{H}_1 \) and \( \mathcal{H}_2 \), by switching between these linear dynamical systems. \( \mathcal{H}_1 \) follows the dynamics determined by matrix \( B \) in the first and third quadrants while follows \( A \) in the other two quadrants. \( \mathcal{H}_2 \) follows \( A \) in the first and third quadrants and follows \( B \) in the second and fourth quadrants. Sample executions for both systems are shown in Figure 2.1c and Figure 2.1d. These executions capture the evolution of the system in a particular quadrant, and the switching transitions, that occur at the boundaries of the quadrants. A formal definition of executions is introduced.
right after this example.

The two linear switched systems in Figure 2.1c and Figure 2.1d are represented as the hybrid automata shown in Figure 2.2a and in Figure 2.2b, respectively. Every location corresponds to a quadrant, namely $q_1$, $q_2$, $q_3$ and $q_4$ for the first, second, third and fourth quadrant, respectively. The vector field function at each location is determined by the matrices $A$ or $B$, such that, for instance, $F(q_i)(x) = Bx$ for $i = 2, 4$ and $F(q_i)(x) = Ax$ for $i = 1, 3$ for the hybrid automaton in Figure 2.2a. The invariants are the polyhedral sets which describe the quadrants, and the invariant function is defined as $I(q_i) = I_i$ for $i = 1, 2, 3, 4$, where $I_i$ corresponds to the $i$-th quadrant. The switching between dynamics is represented by the edge connections between locations, and the switching conditions are expressed as linear constraints tagged on the edges.

**Definition 2.** An execution $\sigma$ of a hybrid automaton $H = (Q, E, X, F, I, G)$ of dimension $n$ is a triple $(\iota, \eta, \gamma)$, such that:

- $\iota$ is an interval domain;
- $\eta : [\iota] \rightarrow \mathbb{R}^n$ such that for each $i \in \text{dom}(\iota)$, $\eta|_{\iota(i)}$ is a differentiable function;
- $\gamma : \text{dom}(\iota) \rightarrow Q$ such that:
  - for all $i \in \text{dom}(\iota)$, for all $t \in \iota(i)$, $\eta(t) \in I(\gamma(i))$ and $\dot{\eta}(t) \in F(\gamma(i))$;
  - for all $0 \leq i < \text{len}(\iota)$, $(\gamma(i), \gamma(i+1)) \in E$ and $\eta(\text{last}(\iota(i))) \in G((\gamma(i), \gamma(i+1)))$.

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

The set of all executions of $H$ will be denoted by $\text{exec}(H)$, and the set of all complete executions by $\text{cexec}(H)$. The restriction of a set of executions $\Sigma$ to a
set $X \in \mathbb{R}^n$, denoted $\Sigma|_X$, corresponds to the executions $\sigma \in \Sigma$ such that for all $i \in \text{dom}(\iota)$ and for all $t \in \iota(i)$, $\eta(t) \in X$.

**Notation.** It is used $\text{fs}(\sigma)$ to refer to the first state $(\gamma(0), \eta(0))$. If $\sigma$ is finite, it is used $\text{ls}(\sigma)$ to refer to the last state $(\gamma(\text{len}(\gamma)), \eta(\text{last}(\iota)))$. Given an execution $\sigma$, its components will be referred to using appropriate subscripts, for example, the components of $\sigma_i$ are denoted using $\iota_i$, $\eta_i$, and $\gamma_i$, respectively.

**Definition 4.** An execution $\sigma' = (\iota', \eta', \gamma')$ is said to extend $\sigma = (\iota, \eta, \gamma)$ if
- $\text{dom}(\iota)$ is a strict subset of $\text{dom}(\iota')$,
- $\text{dom}(\eta)$ is a strict subset of $\text{dom}(\eta')$,
- for all $i \in \text{dom}(\iota)$, $\iota(i) = \iota'(i)$ and $\gamma(i) = \gamma'(i)$, and
- for all $t \in \text{dom}(\eta)$, $\eta(t) = \eta'(t)$.

An execution $\sigma$ is maximal if it cannot be extended.

Note that complete executions are maximal.

We define the scaling of an execution by a positive real number.

**Definition 5.** Given an execution $\sigma = (\iota, \eta, \gamma)$ of $\mathcal{H}$ and a real number $\alpha > 0$, $\alpha \sigma$ is the entity $(\iota', \eta', \gamma')$, where:
- $\text{dom}(\iota') = \text{dom}(\iota)$ and if $\iota(i) = [t, t']$, then $\iota'(i) = [\alpha t, \alpha t']$;
- $\eta'(t) = \alpha \eta(t/\alpha)$ and
- $\gamma' = \gamma$.

Note that $\alpha \sigma$ may not in general be an execution, since, it may not satisfy the invariant and guard conditions in the definition of an execution. Note, however, that it satisfies the flow conditions, since, $\frac{d}{dt} \eta'(t) = \alpha \frac{d}{dt} \eta(t/\alpha) = \alpha \frac{d}{dt} \eta(t/\alpha) = \alpha (\frac{d}{dt} \eta(t/\alpha)) = (\frac{d}{dt} \eta(t/\alpha)) \in F(\gamma(i))$, where $t/\alpha \in [t_i, t_{i+1}]$. Then $t \in [\alpha t_i, \alpha t_{i+1}]$ and $\frac{d}{dt} \eta'(t) \in F(\gamma(i))$.

## 2.2 Stability notions

In this section, classical notions of stability in control theory are defined, and preliminary results about the stability of polyhedral switched systems are stated. The notions of stability are considered with respect to an equilibrium point.
2.2. Stability notions

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

Without loss of generality, the origin $\bar{0}$ is assumed to be the equilibrium point.

**Remark.** The stability verification algorithms described in this dissertation do not require the input system to have $\bar{0}$ as an equilibrium point. However, if the algorithms deduce that the system is stable, then it also implies that $\bar{0}$ is an equilibrium point.

**Lyapunov stability** 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 neighbourhood around the equilibrium point converge to it.

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

**Definition 7.** A set of executions $\Sigma$ of a hybrid system $H$ is said to be Lyapunov stable (LS) with respect to $\bar{0}$, if for every real $\varepsilon > 0$, there exists a real $\delta > 0$ such that $\text{lyap}(\Sigma, \varepsilon, \delta)$.

**Asymptotic stability** Given an execution $\sigma$, it is said to converge to $\bar{0}$, denoted $\text{conv}(\sigma, \bar{0})$, if for every real $\varepsilon > 0$, there exists a $T \in \text{dom}(\eta)$, such that $\eta(t) \in B_\varepsilon(\bar{0})$ for every $t \geq T$. Further, $\text{asymp}(\Sigma, \delta)$ denotes the fact that every complete execution $\sigma = (\iota, \eta, \gamma) \in \Sigma$ with $\eta(0) \in B_\delta(\bar{0})$ satisfies $\text{conv}(\sigma, \bar{0})$.

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

A hybrid system $H$ is said to be Lyapunov (asymptotically) stable if $\text{exec}(H)$ is Lyapunov (asymptotically) stable with respect to $\bar{0}$.

**Example 4.** Observe in Figures 2.1a and 2.1b that the executions of these linear dynamical systems remain close to the origin, which implies Lyapunov stability of both systems. In the case of Figure 2.1c, the execution from system $H_1$ converges to the origin, implying, therefore, asymptotic stability; while in the case of system $H_2$, the execution of Figure 2.1d diverges, hence the system is unstable.
Global asymptotic stability  Lyapunov and asymptotic stability are local notions of stability. An universal and stronger notion is global asymptotic stability. It assures that all the executions of the system converge to a desired equilibrium state and any perturbations to this state do not drive the system too far wherever the system execution starts from.

Definition 9. A set of executions $\Sigma$ of a hybrid system $H$ is said to be globally asymptotically stable ($\text{gas}$) with respect to $\bar{0}$ if it is Lyapunov stable and every execution $\sigma = (i, \eta, \gamma) \in \text{exec}(H)$ satisfies $\text{conv}(\sigma, \delta)$.

A hybrid system $H$ is said to be globally asymptotically stable if $\text{exec}(H)$ is globally asymptotically stable with respect to $\bar{0}$.

Region stability  Here, it is formalized the notion that captures the fact that all executions of a predefined set of executions reach a region.

Definition 10. A set of executions $\Sigma$ of a hybrid system $H$ is said to be region stable ($\text{rs}$) with respect to a set $R \subseteq X$ if for every maximal execution $\sigma = (i, \eta, \gamma) \in \text{exec}(H)$ there exists a value $T \geq 0$ such that $\eta(T) \in R$.

A hybrid system $H$ is said to be region stable with respect to $R$ if $\text{exec}(H)$ is region stable with respect to $R$. Often, region stability not only requires that the executions reach $R$, but also requires the executions to remain within $R$ always after some time [PW07]. Note that considering a weaker notion than the standard region stability suffices for this thesis purpose.

2.2.1  Preliminary properties of stability

Next, some properties about stability are proved. 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 1. A set of executions $\Sigma$ is Lyapunov stable if and only if there exists an $\varepsilon' > 0$ such that for every $0 < \varepsilon < \varepsilon'$, there exists a $\delta > 0$ for which $\text{lyap}(\Sigma, \varepsilon, \delta)$ holds.

Proof. Suppose $\Sigma$ is Lyapunov stable. Then, by definition of Lyapunov stability, it is known that for every $\varepsilon > 0$, $\text{lyap}(\Sigma, \varepsilon, \delta)$ holds. Therefore, it is clear that for a subset of values $0 < \varepsilon < \varepsilon'$, with a fixed $\varepsilon' > 0$, $\text{lyap}(\Sigma, \varepsilon, \delta)$ will hold.

In the other direction, suppose that there exists $\varepsilon' > 0$ such that $\forall \ 0 < \varepsilon < \varepsilon'$ there exists $\delta_\varepsilon > 0$ such that $\text{lyap}(\Sigma, \varepsilon, \delta_\varepsilon)$ holds. Consider $\varepsilon \geq \varepsilon'$ and choose
the value $\delta_{\varepsilon'/2}$. Since $0 < \varepsilon'/2 < \varepsilon'$, it is clear that $\text{lyap}(\Sigma, \varepsilon, \delta_{\varepsilon'/2})$ holds. Therefore $\Sigma$ is Lyapunov stable.

In a small enough neighbourhood around 0, the executions of a switched system are determined by the homogeneous linear constraints of the guards and the invariants that contains 0 in their closures. Hence, one can reduce the system to a system capturing only the behaviours close to the origin, for the case of the local stability analysis.

**Definition 11.** A polyhedral switched system is in normal form if the invariants and the guards of the system are pointed.

Given a polyhedral switched system $\mathcal{H}$, we construct a polyhedral switched system $\mathcal{H}'$ in normal form based on $\mathcal{H}$. First, remove from $\mathcal{H}$ the modes and edges whose invariant and guard closures do not contain the equilibrium. Then, replace every guard and invariant, say $P$, with $P' := \{\alpha x : x \in P, \alpha > 0\}$. Observe that the local behaviour close to 0 of both systems is analogous, that is the same as the behaviour of $\mathcal{H}$ restricted to $B_\varepsilon(0)$ for a small enough $\varepsilon$ value.

**Proposition 2.** 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 $\varepsilon > 0$ such that

$$\text{exec}(\mathcal{H})|_{B_\varepsilon(0)} = \text{exec}(\mathcal{H}')|_{B_\varepsilon(0)}.$$  

**Proof.** It is enough to fix an $\varepsilon$--ball around the origin such that the invariants and guards of the PSS $\mathcal{H}$ not containing 0 are out of the ball. Fix $\varepsilon > 0$ such that $B_\varepsilon(0) \cap I_{\mathcal{H}}(q) = \emptyset$ for every $q \in Q_{\mathcal{H}}$ with $I_{\mathcal{H}}(q)$ not pointed and $B_\varepsilon(0) \cap G_{\mathcal{H}}(q_1, q_2) = \emptyset$ for every pair $(q_1, q_2) \in E_{\mathcal{H}}$ such that $G_{\mathcal{H}}(q_1, q_2)$ not pointed. Consider $\sigma \in \text{exec}(\mathcal{H})|_{B_\varepsilon(0)}$. Then, $\sigma = (\iota, \eta, \gamma)$ where $\iota$ is an interval domain; $\eta : \mathbb{I} \rightarrow \mathbb{R}^n$ such that for each $i \in \text{dom}(\iota)$, $\eta|_i$ is a differentiable function. $\gamma : \text{dom}(\iota) \rightarrow Q_{\mathcal{H}}$ is such that for all $i \in \text{dom}(\iota)$, $I_{\mathcal{H}}(\gamma(i))$ is pointed. Then, $\gamma : \text{dom}(\iota) \rightarrow Q_{\mathcal{H}'}$. For all $i \in \text{dom}(\iota)$, for all $t \in \iota(i)$, $\eta(t) \in I_{\mathcal{H}}(\gamma(i)) \cap B_\varepsilon(0)$ and $\dot{\eta}(t) \in F_{\mathcal{H}}(\gamma(i))$. By construction of the normal PSS $\mathcal{H}'$ based on $\mathcal{H}$ and definition of $\varepsilon$, $I_{\mathcal{H}}(\gamma(i)) \cap B_\varepsilon(0) = I_{\mathcal{H}'}(\gamma(i)) \cap B_\varepsilon(0)$. By construction of $\mathcal{H}'$, $F_{\mathcal{H}}(\gamma(i)) = F_{\mathcal{H}'}(\gamma(i))$. For all $0 \leq i < \text{len}(\iota)$, $(\gamma(i), \gamma(i+1)) \in E_{\mathcal{H}}$ and $\eta(\text{last}(i(i))) \in G_{\mathcal{H}}((\gamma(i), \gamma(i+1))) \cap B_\varepsilon(0)$, where $G_{\mathcal{H}}((\gamma(i), \gamma(i+1)))$ is pointed. Then, $G_{\mathcal{H}'}((\gamma(i), \gamma(i+1))) \cap B_\varepsilon(0) = G_{\mathcal{H}'}((\gamma(i), \gamma(i+1))) \cap B_\varepsilon(0)$, by construction of the normal PSS $\mathcal{H}'$ based on $\mathcal{H}$. Hence, $\sigma \in \text{exec}(\mathcal{H}')|_{B_\varepsilon(0)}$. On the other hand, consider $\sigma \in \text{exec}(\mathcal{H}) \cap B_\varepsilon(0)$, then by an analogous reasoning, $\sigma \in \text{exec}(\mathcal{H}) \cap B_\varepsilon(0)$ is obtained.

**Proposition 3.** 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 neighbourhood to the origin [PV13]. Suppose $\mathcal{H}$ is Lyapunov stable. By Proposition 2, it is fixed $\varepsilon > 0$ such that $\text{exec}(\mathcal{H}) \cap B_\varepsilon(\bar{0}) \subseteq \text{exec}(\mathcal{H}')$. It is known that there exists $\delta > 0$ such that $\text{lyap}(\text{exec}(\mathcal{H}), \varepsilon, \delta)$. Then, by construction of $\mathcal{H}'$, it is clear that $\text{lyap}(\text{exec}(\mathcal{H}'), \varepsilon, \delta')$. Therefore, it is inferred that $\mathcal{H}'$ is Lyapunov stable. Now, suppose $\mathcal{H}'$ is Lyapunov stable. Choose $\varepsilon' > 0$ such that $\text{exec}(\mathcal{H}) \cap B_{\varepsilon'}(\bar{0}) \subseteq \text{exec}(\mathcal{H}')$. For every $\varepsilon < \varepsilon'$ it is known that there exists $\delta > 0$ such that $\text{lyap}(\text{exec}(\mathcal{H}'), \varepsilon, \delta)$. Since $\text{exec}(\mathcal{H}) \cap B_{\varepsilon'}(\bar{0}) \subseteq \text{exec}(\mathcal{H}')$, for every $\varepsilon < \varepsilon'$, $\text{lyap}(\text{exec}(\mathcal{H}), \varepsilon, \delta)$ holds. Then, by Proposition 1, $\text{exec}(\mathcal{H})$ is Lyapunov stable.

Next, it is needed to prove the second point, about asymptotic stability. Suppose first $\mathcal{H}'$ is asymptotically stable. Then, by definition, there exists $\delta > 0$ such that $\text{asymp}(\text{exec}(\mathcal{H}'), \delta)$ is satisfied. Also, by Proposition 2, there exists $\varepsilon > 0$ such that $\text{exec}(\mathcal{H}) \cap B_\varepsilon(\bar{0}) \subseteq \text{exec}(\mathcal{H}')$. In addition, since Lyapunov stability of $\mathcal{H}'$ implies Lyapunov stability of $\mathcal{H}$, it is known that there exists $\delta_\varepsilon > 0$ such that $\text{lyap}(\text{exec}(\mathcal{H}), \varepsilon, \delta_\varepsilon)$. Therefore, every execution $\sigma \in \text{exec}(\mathcal{H})$ starting from $B_{\delta_\varepsilon}(\bar{0})$ remains in $B_\varepsilon(\bar{0})$, which implies that $\sigma \in \text{exec}(\mathcal{H}')$. Therefore, in the case of $\delta_\varepsilon \leq \delta$, $\sigma$ satisfies $\text{conv}(\sigma, \bar{0})$ since $B_\delta(\bar{0}) \subseteq B_\varepsilon(\bar{0})$, which implies that $\text{asymp}(\text{exec}(\mathcal{H}), \delta_\varepsilon)$. While in the case of $\delta_\varepsilon > \delta$, $\sigma$ does not satisfy $\text{conv}(\sigma, \bar{0})$. Then, consider $\sigma$ that, in addition, starts from $B_\delta(\bar{0})$. These kind of executions satisfy $\text{conv}(\sigma, \bar{0})$ and consequently $\text{asymp}(\text{exec}(\mathcal{H}), \delta)$. Now, for the case of $\mathcal{H}$ is asymptotically stable, the proof is analogous.

From now on, assume that the polyhedral switched system is in normal form.
Quantitative Predicate Abstraction

This chapter introduces a modified predicate abstraction technique suitable for stability verification, which is published in “Abstraction Based Model-Checking of Stability of Hybrid Systems”, CAV’13 [PS13] and in “An algorithmic approach to stability verification of polyhedral switched systems”, ACC’14 [PS14]

Contents

3.1 Polyhedral switched systems ...................... 29
  3.1.1 Functions on switched systems .................. 31
  3.1.2 Piecewise linear executions and splitting ........ 32

3.2 Stability verification procedure .................... 34
  3.2.1 Formal definition of the graph ................... 34
  3.2.2 Stability Criteria ................................. 37
  3.2.3 Correctness of the stability criteria ................. 37
  3.2.4 Verifying Conditions [P1-P4] ................. 39

3.3 Weighted graph construction ....................... 40
  3.3.1 Algorithm 1 - Abstraction construction .......... 40
  3.3.2 Algorithm 2 - Scaling computation ................. 42
  3.3.3 Reach Relation polyhedron ....................... 44

3.4 Implementation and experimental results ........... 45

3.5 Summary ............................................. 47
In this chapter, we focus on the class of polyhedral switched systems (PSS). These are hybrid automata 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 quantitative predicate abstraction approach consists of constructing a finite weighted graph which represents a conservative approximation of the switched system in order to infer stability by analysing certain properties of the graph. The algorithm takes as input a PSS \( H \) and a finite partition of the state space into convex polyhedral sets \( \mathcal{P} \), and outputs a finite weighted graph \( G \). 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. Criteria based on graph analysis is provided, which ensure 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. It is shown that a formula equivalent
A polyhedral switched system is an $n$-dimensional hybrid automaton $\mathcal{H}$ (remind Definition 1) where the vector field function is $F : Q \rightarrow \text{cpolysets}(n)$ and in which the continuous state is not modified when the mode switches. Here, $\text{cpolysets}(n)$ is interpreted as a constant function mapping every $x \in \mathbb{R}^n$ to a polyhedron belonging to $\text{cpolysets}(n)$.

**Example 5.** Figure 3.1 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 3.2a and Figure 3.2b are two graphical instances of the PSS in Figure 3.1. 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 3.2. Observe that in Figure 3.2 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 $G_3 = \{(x, y) : x - 2y \geq 0\}$.

Every execution of the PSS in Figure 3.1 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 $q_5$ overlap 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 3.2, two sample finite executions are shown. They correspond to the dotted lines.
Let us fix a pss \( H = (Q, E, X, F, I, G) \) of dimension \( n \). The underlying graph of a hybrid system \( H \), denoted \( UG(H) \), is the pair of the set of locations and the set of edges \((Q, E)\).

Given a set \( P \subseteq \mathbb{R}^n \), define \( H \cap P \) to be the pss \( (Q_P, E_P, X_P, F_P, I_P, G_P) \), where:

- \( Q_P = \{ q \in Q | P \subseteq I(q) \} \),
- \( E_P = \{ (q_1, q_2) \in (Q_P \times Q_P) \cap E | P \subseteq G((q_1, q_2)) \} \),
- \( F_P(q) = F(q) \cap \text{plane}(P) \) 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 \( \text{reach}(H, P) \) denote the set of pairs \((q_1, q_2) \in Q_P \times Q_P \) such that there exists a path in \( UG(H \cap P) \) from \( q_1 \) to \( q_2 \).

By considering the polyhedral switched system in Figure 3.1, \( H \), and the polyhedral set \( P_2 = \{ (x, y) : y > 0, x - 2y > 0 \} \), observe the pss \( H \cap P_2 \) enclosed by a dotted line.

The stability verification of pss’s do not require an analysis over all the possible executions of these systems but only over a class of executions, called piecewise linear executions. Next, this particular class of executions for polyhedral switched systems and the basis for reducing the stability verification to this class of executions, are introduced.
### 3.1.2 Piecewise linear executions and splitting

**Definition 12.** An execution $\sigma = (t, \eta, \gamma)$ is said to be piecewise linear if $\eta$ is piecewise linear.

We denote the set of all piecewise linear executions of $\mathcal{H}$ by $\text{pwexec}(\mathcal{H})$. 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 4.** Let $\mathcal{H}$ be a polyhedral switched system. Given any finite execution $\sigma \in \text{exec}(\mathcal{H})$, there exists a piecewise linear execution $\sigma' \in \text{pwexec}(\mathcal{H})$ such that $\text{fs}(\sigma) = \text{fs}(\sigma')$ and $\text{ls}(\sigma) = \text{ls}(\sigma')$.

**Proof.** Let $\sigma = (t, \eta, \gamma)$ be an execution in $\text{exec}(\mathcal{H})$. We define $\sigma' = (t', \eta', \gamma')$ where $t' = t$, $\gamma' = \gamma$ and

$$
\eta'_{i(t)}(t) = \eta(\text{first}(i(t))) + \frac{\eta(\text{last}(i(t))) - \eta(\text{first}(i(t)))}{\text{len}(i(t))} t
$$

for every $i \in \text{dom}(\sigma)$ and $t \in i(t)$. Next, we prove that $\sigma'$ is a piecewise linear execution of $\mathcal{H}$. By definition, it is clear that $\sigma'$ is piecewise linear, that its first state $\text{fs}(\sigma')$ and last state $\text{ls}(\sigma')$ coincide with the first state $\text{fs}(\sigma)$ and last state $\text{ls}(\sigma)$, respectively, that $t'$ is an interval domain and that $\eta'$ is a differentiable function. Recall that $\gamma : \text{dom}(\sigma) \rightarrow Q$, by definition of execution. For every $i \in \text{dom}(\sigma)$, we know $I(\gamma(i)) \in \text{polysets}(n)$, so it is a convex set. Since $\eta(\text{first}(i(t)))$ and $\eta(\text{last}(i(t)))$ belong to the convex set $I(\gamma(i))$, we obtain $\eta'(t) \in I(\gamma(i))$ for all $t \in i(t)$. We also know that $F(\gamma(i)) \in \text{cpolysets}(n)$, so it is determined by a finite set of linear constraints of the form $a \cdot x + b \sim 0$, where $x \in \mathbb{R}^n$ is a variable, $a \in \mathbb{R}^n$ and $b \in \mathbb{R}$ are the constant coefficients and relation $\sim$ refers to $\leq$ or $\geq$. Since $\dot{\eta}(t) \in F(\gamma(i))$ for every $t \in i(t)$, it satisfies all the linear constraints, which means that $a \cdot \dot{\eta}(t) + b \sim 0$. This implies

$$
\int_{\text{first}(i(t))}^{\text{last}(i(t))} a \cdot \dot{\eta}(t) + b \, dt \sim 0,
$$

and dividing by $\text{last}(i(t)) - \text{first}(i(t))$,

$$
a \cdot \frac{\eta(\text{last}(i(t))) - \eta(\text{first}(i(t)))}{\text{last}(i(t)) - \text{first}(i(t))} + b \sim 0
$$

is obtained. Therefore

$$
\frac{\eta(\text{last}(i(t))) - \eta(\text{first}(i(t)))}{\text{last}(i(t)) - \text{first}(i(t))}
$$
satisfies every linear constraint. Since

\[ \eta'(t) = \frac{\eta(\text{last}(\iota(i))) - \eta(\text{first}(\iota(i)))}{\text{last}(\iota(i)) - \text{first}(\iota(i))} \]

we conclude that \( \eta'(t) \in F(\gamma(i)) \). Therefore, all the terms in \( (\iota', \eta', \gamma') \) are well defined and determine a piecewise linear execution \( \sigma' \).

**Lemma 1.** Let \( \sigma_1 = (\iota_1, \eta_1, \gamma_1) \) 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 \( t \in \text{dom}(\eta_1) \), and \( \eta_2(t) = \eta(t + \text{last}(\text{dom}(\eta_1))) \) for all \( t \in \text{dom}(\eta_2) \).
- \( \gamma_1(i) = \gamma(i) \) for all \( i \in \text{dom}(\gamma_1) \), and \( \gamma_2(i) = \gamma(i + \text{len}(\gamma_1)) \) for all \( i \in \text{dom}(\gamma_2) \).

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 an execution \( \sigma \) 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 \( \sigma_1 \sigma_2 \ldots \) is a splitting of a complete execution \( \sigma \), 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 \( i \geq 2 \).

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 1.** Let \( \mathcal{H} \) be a polyhedral switched system. Then:

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

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

Next, suppose that \( \text{pwexec}(\mathcal{H}) \) is Lyapunov stable. We prove that this implies \( \text{exec}(\mathcal{H}) \) is Lyapunov stable. Suppose that \( \text{exec}(\mathcal{H}) \) is not Lyapunov stable, then there exists an \( \varepsilon > 0 \) such that for every value \( \delta > 0 \) there exists an execution \( \sigma \in \text{exec}(\mathcal{H}) \) starting from \( B_\delta(\bar{0}) \) which at some time \( T > 0 \) goes out from \( B_\varepsilon(\bar{0}) \), that is, it is such \( \eta(0) \in B_\delta(\bar{0}) \) and \( \eta(T) \notin B_\varepsilon(\bar{0}) \). The restriction of \( \sigma \) to the time interval \([0, T]\) is a finite execution in \( \mathcal{H} \). By applying Proposition 4,
construct a piecewise linear execution $\sigma' \in \text{pwexec}(\mathcal{H})$ such that $fs(\sigma) = fs(\sigma')$ and $ls(\sigma) = ls(\sigma')$. This means that for every $\delta > 0$, there exists a piecewise linear execution $\sigma' \in \text{pwexec}(\mathcal{H})$ starting from $B_\delta(\bar{0})$ and at time $T > 0$ goes out of $B_\varepsilon(\bar{0})$, which contradicts the fact that $\text{pwexec}(\mathcal{H})$ is Lyapunov stable.

Finally, we prove that $\text{pwexec}(\mathcal{H})$ is asymptotically stable implies $\text{exec}(\mathcal{H})$ is asymptotically stable. Suppose that $\text{asym}(\text{pwexec}(\mathcal{H}), \delta)$ holds. We will show that $\text{asym}(\text{exec}(\mathcal{H}), \delta)$ holds. Let $\sigma$ be an execution in $\mathcal{H}$ with $\eta(0) \in B_\delta(\bar{0})$. Suppose that $\text{conv}(\sigma, \bar{0})$ does not hold. Then there exists $\delta' > 0$, such that the trajectory $\eta$ 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 $\sigma'$ of $\mathcal{H}$ using Proposition 4 by replacing $\sigma$ in each of the intervals $[t_i, t_{i+1}]$ by a piecewise linear execution. Then $\text{conv}(\sigma', \bar{0})$ does not hold, which contradicts $\text{asym}(\text{pwexec}(\mathcal{H}), \delta)$. ■

3.2 Stability verification procedure

In this section, we present an algorithmic approach for verifying stability of polyhedral switched systems. This is an extension of our algorithms in [PS13] for piecewise constant derivative systems and in [PV13] 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. Analysing 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.

3.2.1 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. Hence, we define element execution which corresponds to an execution evolving inside a particular element except at the end points.
3.2. Stability verification procedure

**Definition 14.** 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 $\sigma$ such that $\eta(t) \in P$ for every $0 < t < \text{last}(\text{dom}(\eta))$. An element execution is an execution which is a $P$-execution for some element $P$ of $\mathcal{P}$.

The set of all element executions of $\mathcal{H}$ is denoted by $\text{elemexec}(\mathcal{H})$, and the set of all $P$-executions for a particular element $P \in \mathcal{P}$ by $P\text{-exec}(\mathcal{H})$.

**Proposition 5.** $\sigma$ is a $P$-execution of $\mathcal{H}$ if and only if $\sigma$ is an execution of $\mathcal{H} \cap P$.

*Proof.* If $\sigma$ is an execution of $\mathcal{H} \cap P$, then it is an execution of $\mathcal{H}$ which always remains within $P$ except possibly at end times. Hence, it is a $P$-execution of $\mathcal{H}$.

Suppose $\sigma$ is a $P$-execution of $\mathcal{H}$. Then it always remains within $P$ except possibly at end times. We need to show that the derivative at all times belongs to $\text{plane}(P)$. We observe that each of the hyper-planes containing $P$ are defined by homogenous linear constraints, since, $P$ itself is defined by homogeneous linear constraints. Suppose that the derivative of $\sigma$ does not belong to $\text{plane}(P)$ at some point $t$, then $\sigma$ necessarily goes out of $\text{plane}(P)$ at $t$ for sometime. Since $P \subseteq \text{plane}(P)$, this implies that $\sigma$ is not a $P$-execution. Hence, the derivative belongs to $\text{plane}(P)$.

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 $\sigma$, its scaling, denoted $\text{scaling}(\sigma)$ is given by:

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

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

The weighted graph $\mathcal{G} = (V, E, w)$ is determined by $\mathcal{H}$ and $\mathcal{P}$, denoted as $\text{abs}(\mathcal{H}, \mathcal{P})$ and it is defined as follows:
3. Quantitative Predicate Abstraction

- The set of vertices $V$ is given by $Q \times P$;

- The set of edges $E \subseteq V \times V$ is given by $\{(q_1, P_1), (q_2, P_2)\} | \exists \sigma \in \text{elemexec}(H), \text{fs}(\sigma) \in q_1 \times P_1, \text{ls}(\sigma) \in q_2 \times P_2$;

- The weighting function $w$ is given by the supremum of a set of scaling values as follows, $w((q_1, P_1), (q_2, P_2)) = \sup\{\text{scaling}(\sigma) | \exists \sigma \in \text{elemexec}(H), \text{fs}(\sigma) \in q_1 \times P_1, \text{ls}(\sigma) \in q_2 \times P_2\}$.

A weighted graph is represented in Figure 3.3b. It is constructed by considering the pss $H$ in Figure 3.2a and an element partition $P$. The element partition, depicted in Figure 3.3a, 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 $\text{abs}(H, P)$ 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 $\{(q_1, P_1), (q_5, P_1)\}$, $\{(q_2, P_1), (q_1, P_1)\}$, $\{(q_5, P_1), (q_5, P_3)\}$, $\{(q_1, P_1), (q_1, P_3)\}$, $\{(q_5, P_1), (q_1, P_3)\}$, $\{(q_5, P_3), (q_1, P_3)\}$, $\{(q_5, P_3), (q_1, P_3)\}$, $\{(q_5, P_3), (q_1, P_3)\}$, $\{(q_5, P_3), (q_1, P_3)\}$, and $\{(q_5, P_3), (q_1, P_3)\}$. The computation of the weights associated with the edges will be presented later.
3.2. Stability verification procedure

3.2.2 Stability Criteria

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

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

P2 Every simple cycle $\pi$ of $G$ is such that $w(\pi) \leq 1$.

P3 Every simple cycle $\pi$ of $G$ is such that $w(\pi) < 1$.

In addition, we need to verify the following property:

P4 For every element $P \in \mathcal{P}$, every complete execution $\sigma$ such that $\eta(t) \in P$ for all $t \in [0, \infty)$, satisfies $\text{conv}(\sigma, \bar{0})$.

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

**Theorem 1.** (Lyapunov stability) If Conditions [P1] and [P2] hold for $G$, then $H$ is Lyapunov stable.

**Theorem 2.** (Asymptotic stability) If Conditions [P1], [P3] and [P4] hold for $G$, then $H$ is asymptotically stable.

3.2.3 Correctness of the stability criteria

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

**Lemma 2.** Let $H$ be a polyhedral switched system and $\mathcal{P}$ an elementary partition. For every execution $\sigma \in \text{exec}(H)$, there exists a finite or infinite sequence $\sigma_0 \sigma_1 \ldots$ of executions in $\text{exec}(H)$ such that for each $i$, $\sigma_i$ is a $P$-execution for some $P \in \mathcal{P}$, and $\sigma = \sigma_0 \circ \sigma_1 \circ \ldots$.

**Proof.** Consider $\sigma \in \text{exec}(H)$. We know $\sigma = (\iota, \eta, \gamma)$ and $\forall i \in \text{dom}(\iota)$ 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 $\text{last}(\text{dom}(\iota))$ 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 \( t \in \text{dom}(\eta_j) \), and \( \eta_{j+1}(t) = \eta(t + \text{last(dom}(\eta_j))) \) for all \( t \in \text{dom}(\eta_{j+1}) \).

- \( \gamma_j(1) \) 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 \( \gamma(\text{dom}(\iota)) \) if \( k_{l-1} < j \leq k_l \).

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

**Proof of Theorem 1** 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 a \( P \)-execution \( \sigma \in \text{exec}(\mathcal{H}) \). Hence, there exist \( q_0, q_1, q_2 \in Q \) such that \( fs(\sigma) \in q_1 \times P \) and \( ls(\sigma) \in q_2 \times P \) with \( \text{scaling}(\sigma) \leq w((q_1, P), (q_2, P)) \). \( w((q_1, P), (q_2, P)) \in \mathbb{R}_{\geq 0} \) because of condition [P1], and corresponds to an edge in \( G \) which is finite; therefore we can define

\[
b_1 = \max_{e \in E} w(e)
\]

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

\[
b_2 = \max_{\sigma \in \text{exec}(\mathcal{H}), \sigma \text{ finite}} w(\sigma).
\]

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

\[
b_3 = \max_{\sigma \in \text{exec}(\mathcal{H}), \sigma \text{ infinite}} w(\sigma) \in \mathbb{R}_{\geq 0}.
\]
Finally, by choosing $B = \max\{b_1, b_2, b_3\}$, we claim that for every $\sigma \in \text{exec}(\mathcal{H})$, $w(\sigma) \leq B$, it means that $|\eta(t)| \leq B \cdot |\eta(0)|$ for all $t \in \text{dom}(\eta)$, hence $\forall \varepsilon \geq 0$, $\text{lyap}(\text{exec}(\mathcal{H}), \varepsilon, \frac{\varepsilon}{B})$ holds.

**Proof of Theorem 2** 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 $\sigma$, can be split into an infinite sequence of element executions, that is, corresponds to an infinite path $\pi$ in the graph. An infinite path $\pi$ 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 $G$. Let $b$ be an upper bound on the weight of all simple paths, it is

$$b = \max_{\pi \in \text{paths}(G)} \max_{1 \leq \text{len}(\pi) < \infty} w(\sigma).$$

From Condition [P3], we know that the weight of each simple cycle is $< 1$. Since the number of simple cycles in the graph $G$ is finite, there exists $\varepsilon > 0$ such that the weight of each simple cycle is $< 1 - \varepsilon$. Every prefix of the infinite path $\pi$ 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 $b^k(1 - \varepsilon)^m$. The weight goes to 0 as $m$ increases. Hence, the scaling of $\sigma$ converges to $0$. Therefore, $\sigma$ converges to $\bar{0}$.

### 3.2.4 Verifying Conditions [P1-P4]

Given the graph $G$, conditions [P1-P3] can be efficiently verified in $O(|V|\|E|)$ complexity where $|V|$ and $|E|$ are the number of vertices and edges in $G$, respectively. Verifying Condition [P1] consists of iterating over the edges and checking if the values of the weights is $< +\infty$. 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 $-\log(w(e))$. Negative cycles can be detected, for example, by Bellman-Ford algorithm.

Note that any complete execution which remains within $P$ 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
3. Quantitative Predicate Abstraction

Algorithm 1 Weighted graph construction

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

1: for each location $q$ in $Q$ and element $P$ in $\mathcal{P}$ do
2:   if $p \subseteq l(q)$ then
3:      AddNode($(q, P), \mathcal{G})$
4:   for element $P$ in $\mathcal{P}$ do
5:      $\mathcal{G}' :=$ QUOTIENTGRAPH($\mathcal{H}, P$)
6:      for $(q_1, P_1)$ and $(q_2, P_2)$ in $\mathcal{G}$ do
7:        if ADJACENT($P_1, P$) and ADJACENT($P_2, P$) then
8:          $w := -\infty$
9:          for each pair of nodes $C_1, C_2$ in $\mathcal{G}'$ do
10:             $\Pi :=$ SIMPLEPATHS($C_1, C_2, \mathcal{G}'$)
11:             for $\pi$ in $\Pi$ do
12:                $w_\pi :=$ SCALING($P_1, P_2, P, \pi, \mathcal{H}$)
13:               $w :=$ MAX$(w, w_\pi)$
14:          if $q_1$ in ENTER($P_1, C_1$) and $q_2$ in EXIT($P_2, C_2$) then
15:             AddEdge($((q_1, P_1), (q_2, P_2), P, w), \mathcal{G})$

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 check:

$$P4' \forall C \in \text{scc}(\mathcal{H} \cap P), \emptyset \notin \text{chull}(\bigcup_{q \in C} F(q)) \text{ and } \text{chull}(\bigcup_{q \in C} F(q)) \cap P = \emptyset.$$

3.3 Weighted graph construction

Here, the weighted graph construction is presented in detail. Algorithm 1 and Algorithm 2 describe the full procedure. The main part of the graph construction involves determining the existence of edges between vertices and computing the weights associated with them. These computations are captured in Algorithm 2.

3.3.1 Algorithm 1 - Abstraction construction

Given a pss $\mathcal{H}$ and an elementary partition $\mathcal{P}$, Algorithm 1 computes the graph $\text{abs}(\mathcal{H}, \mathcal{P})$, defined in Subsection 3.2.1. The functions involved in Algorithm 1 are enumerated.
3.3. Weighted graph construction

1. **AddNode**((q, P), G) adds the node (q, P) to the graph G.

2. **QuotientGraph** (H, P) constructs the quotient graph UG(H \cap P)/scc, where UG(H \cap P) corresponds to the underlying graph of the pss H \cap P as defined in Subsection 3.1.1.

3. **Adjacent** (P', P) takes as inputs two elements P', P ∈ P and returns True if they are adjacent and False otherwise. An element P' is adjacent to P if P' ⊂ P and P' \ \overset{\circ}{\cap} P = ∅.

4. **SimplePaths** (C_1, C_2, G') provides all the simple paths in the graph G' starting from the node C_1 and ending at the node C_2.

5. **Scaling** (P_1, P_2, P, π, H) will be explained further on.

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_2 of UG(H \cap P)/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 I(q). These locations are stored into a list l_1. The second function lists into l_2 the set of locations q ∈ l_1 such that the vector field F(q) can define an execution which enters in P. Finally, the third function outputs the locations in l_2 such that there exists an edge in the subgraph H \cap P_1 to them.

8. **Exit** (P_2, C_2) takes as input an element P and a strongly connected component C of UG(H \cap P)/scc. It requires three steps. The first one identifies the locations q in the strongly connected component C_2 such that P_2 ⊆ \overline{I(q)}. They are stored into a list l_1. The second step lists into l_2 the locations q ∈ 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 H \cap P_2 from them.

The input to Algorithm 1 is a pss H and an elementary partition 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 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 10 in Algorithm 1). For each simple path in Π, the scaling is computed. Such computation is performed by the function **Scaling**, whose inputs are a pair of elements P_1
3.3.2 Algorithm 2 - Scaling computation

Next, given three elements $P_1, P_2, P \in \mathcal{P}$ and the polyhedral switched system $\mathcal{H}$, the function $\text{SCALING}(P_1, P_2, P, \pi, \mathcal{H})$, described in Algorithm 2, is explained in
3.3. Weighted graph construction

detail. This function 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 $\pi$. It consists of five functions:

1. $\text{CollectDynamics}(C, H)$ takes as input the pss $H$ and a component $C$ and outputs the collection of polyhedral images of vector fields $\text{img}(F(q))$ such that $q \in C$.

2. $\text{ConvexHull}(C)$ takes as input a list of polyhedra, say $C$, and computes the convex hull polyhedron of these polyhedra, $\text{chull}(\bigcup_{C \in C} C)$.

3. $\text{ReachRelation}(P_1, P, D)$ returns a reach relation 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 reach relation polyhedron construction is explained later.

4. $\text{Compose}(R, Q)$ composes two given reach relations, $R$ and $Q$ by returning a new reach relation, which contains all pairs and only those pairs of the form $(x, z)$ such that there exist $(x, y) \in R$ and $(y, z) \in Q$.

5. $\text{Optimization}(R)$ 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 $H$, $\pi = (C_0, \ldots, C_k)$. 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 $C_i$, the flows determined by them are collected into a list $C$. 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 $\text{Scaling}(P_1, P_2, P, \pi, H)$.

We illustrate the main steps of Algorithm 1. Consider the polyhedral switched system $H$ depicted in Figure 3.1, and the partition $P$ defined at the end of Section 3.2.1. The weighted graph $G(H, P)$ is represented in Figure 3.3b. 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 $\text{QuotientGraph}(H, P_2)$ constructs first $UG(H \cap P_2)$ as defined in subsection 3.1.1. This subgraph is circled by a dotted line in
Figure 3.1. Then, the quotient graph obtained from $U\mathcal{G}(\mathcal{H} \cap P_2)$ 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 3.2.1, are $P_1$ and $P_3$. Let us consider $P_1$ as initial adjacent element and $P_3$ as the final adjacent element. We explain in the following subsection how to construct 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 $P_2$.

### 3.3.3 Reach Relation polyhedron

$\text{REACHRELATION}(P_1, P_2, D)$ 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 vector dynamics contained 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:

1. 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 are equal 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 $\{ (x, y) \in \mathbb{R}^{2n} : a_1 \cdot x \sim 0, \ldots, a_m \cdot x \sim 0, a_1 \cdot y \sim 0, \ldots, a_m \cdot y \sim 0, x = y \}$.

2. 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 that the first half variables are 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 $\{ (x, y) \in \mathbb{R}^{2n} : a_1 \cdot y \sim 0, \ldots, a_m \cdot y \sim 0, x = 0 \}$.

3. 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, $\{ (x, y) \in \mathbb{R}^{2n} : a_1 \cdot y \sim 0, \ldots, a_m \cdot y \sim 0 \}$.

Let us consider the polyhedral partition in Figure 3.3a. We construct a reach relation polyhedron for the elements $P_1$ and $P_3$, and the dynamics determining $P_2$-executions. The $P_2$ element is contained in the invariants of locations $q_1$ and
3.4. Implementation and experimental results

$q_5$, then a $P_2$-execution evolves following any of the dynamics in the strongly connected component formed by locations $q_1$ and $q_5$ in the pss of Figure 3.1. The involved dynamics are determined by the sets $F_1$ and $F_5$. These sets are defined for the stable system in Figure 3.2a, $F_1 = \{(x, y) : x + y < 0, x + 2y > 0\}$ and $F_5 = \{(x, y) : x < 0, x + 2y > 0\}$, so $d = \{F_1, F_5\}$. Then, all the ingredients for constructing the relation polyhedron $\text{REACHRELATION}(P_1, P_3, D)$ are here. First, the duplicated polyhedron for $P_1$ is constructed. It is defined as

$$D_1 = \{(x, y, z, t) : x > 0, y = 0, z > 0, t = 0, z = x, t = y\}.$$

Then, the duplicated polyhedron for the final element $P_3$ is

$$D_2 = \{(x, y, z, t) : z - 2t = 0, t > 0\}.$$

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 $\text{REACHRELATION}(P_1, P_3, D)$ will be the one constructed by considering the initial polyhedron $D_1$, the final polyhedron $D_2$ and the dynamics polyhedron $\{(x, y, z, t) : z - 2t = 0, y = 0, x - z >= 0, z > 0, -x + 2z >= 0\}$.

3.4 Implementation and experimental results

The stability verification procedure for polyhedral switched systems has been integrated in the software tool AVERIST [PS15] (Chapter 7). The experimental results 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.

The algorithm is illustrated on an example which has been verified using AVERIST. The switched system is shown in Figure 3.4a and a pictorial illustration of the system when the value of the variable $z = 1$ is shown in Figure 3.4b. The system executions do not diverge while remaining in any single cell, and as seen 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 3.4b. 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. Add the constraint \( x + y = 0 \) shown by the dotted line in Figure 3.4b 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 3.4a, by considering different vector fields and by extending the state space to 4 and 5 dimensions. AVERIST has been run for every PSS many times until obtaining a conclusive answer while increasing the number of facets involved in the QPA procedure. The results are summarized in Table 4.1. Stability refers to Lyapunov stability. Here, Exp-Dim refers to the experiment number along with the state space dimension of the concrete PSS. Answer is the verification answer, which can be stable (S), unstable (NS) or inconclusive (IN). Elements refers to the number of elements in the state space partition. \( G \) size refers to the number of nodes of the abstract weighted graph. The time for constructing the weighted graph \( G \) time and the time for verifying stability conditions Ver time are shown along with the total verification time Ver time. All the times are in seconds.

Observe that most of the verification time is devoted to construct the abstract weighted graph. This time overhead is because the weighted graph construction
requires solving a bunch of linear programming problems in order to compute weights. Observe also that there is a time overhead with respect to the number of elements when fixing the dimension of the PSS. Both observations points to the need of considering as less number of elements as possible while obtaining an enough accurate weighted graph. In Chapter 4, we will develop heuristics to address this.

3.5 Summary

This chapter has introduced results related to a novel algorithmic approach for verifying stability of polyhedral switched systems. The innovation relies on the use of formal methods for inferring stability, unlike the traditional approach, which is based on Lyapunov function search. The introduced approach relies on constructing a finite weighted graph abstracting the polyhedral hybrid system, which is model checked in order to deduce stability. In the event of a failure to infer stability from the graph, the algorithm provides an abstract counterexample.

Next chapter exploits the output counterexample in order to either state instability of the polyhedral switched system or use it to construct a qualitatively better weighted graph. This enables the development of a counterexample guided abstraction refinement framework.
<table>
<thead>
<tr>
<th>Exp-Dim</th>
<th>Answer</th>
<th>Elements</th>
<th>$G$ size</th>
<th>$G$ time</th>
<th>Ver time</th>
<th>Total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 – 3D</td>
<td>$S$</td>
<td>135</td>
<td>60</td>
<td>5,56</td>
<td>0.001</td>
<td>6.02</td>
</tr>
<tr>
<td>2 – 3D</td>
<td>$IN$</td>
<td>135</td>
<td>60</td>
<td>5,09</td>
<td>0.006</td>
<td>5.58</td>
</tr>
<tr>
<td></td>
<td>$S$</td>
<td>163</td>
<td>83</td>
<td>9,19</td>
<td>0.001</td>
<td>9.74</td>
</tr>
<tr>
<td>3 – 3D</td>
<td>$IN$</td>
<td>135</td>
<td>60</td>
<td>5,04</td>
<td>0.005</td>
<td>5.46</td>
</tr>
<tr>
<td></td>
<td>$S$</td>
<td>163</td>
<td>75</td>
<td>7,43</td>
<td>0.001</td>
<td>7.81</td>
</tr>
<tr>
<td>4 – 3D</td>
<td>$IN$</td>
<td>135</td>
<td>60</td>
<td>6,18</td>
<td>0.006</td>
<td>6.95</td>
</tr>
<tr>
<td></td>
<td>$IN$</td>
<td>163</td>
<td>82</td>
<td>8,41</td>
<td>0.012</td>
<td>8.97</td>
</tr>
<tr>
<td></td>
<td>$IN$</td>
<td>191</td>
<td>100</td>
<td>13,13</td>
<td>0.017</td>
<td>13.80</td>
</tr>
<tr>
<td></td>
<td>$S$</td>
<td>223</td>
<td>115</td>
<td>28,62</td>
<td>0.003</td>
<td>29.35</td>
</tr>
<tr>
<td>5 – 3D</td>
<td>$IN$</td>
<td>135</td>
<td>60</td>
<td>4,88</td>
<td>0.015</td>
<td>5.50</td>
</tr>
<tr>
<td></td>
<td>$IN$</td>
<td>171</td>
<td>81</td>
<td>8,19</td>
<td>0.012</td>
<td>8.73</td>
</tr>
<tr>
<td></td>
<td>$IN$</td>
<td>199</td>
<td>98</td>
<td>14,33</td>
<td>0.019</td>
<td>15.08</td>
</tr>
<tr>
<td></td>
<td>$IN$</td>
<td>227</td>
<td>116</td>
<td>22,11</td>
<td>0.026</td>
<td>22.84</td>
</tr>
<tr>
<td></td>
<td>$S$</td>
<td>259</td>
<td>131</td>
<td>24,83</td>
<td>0.003</td>
<td>25.46</td>
</tr>
<tr>
<td>6 – 3D</td>
<td>$IN$</td>
<td>135</td>
<td>60</td>
<td>6,66</td>
<td>0.029</td>
<td>7.77</td>
</tr>
<tr>
<td></td>
<td>$IN$</td>
<td>171</td>
<td>78</td>
<td>7,78</td>
<td>0.011</td>
<td>8.32</td>
</tr>
<tr>
<td></td>
<td>$IN$</td>
<td>199</td>
<td>99</td>
<td>12,63</td>
<td>0.016</td>
<td>13.35</td>
</tr>
<tr>
<td></td>
<td>$IN$</td>
<td>227</td>
<td>118</td>
<td>21,62</td>
<td>0.026</td>
<td>22.35</td>
</tr>
<tr>
<td></td>
<td>$IN$</td>
<td>255</td>
<td>139</td>
<td>26,84</td>
<td>0.035</td>
<td>27.78</td>
</tr>
<tr>
<td></td>
<td>$S$</td>
<td>287</td>
<td>153</td>
<td>37,45</td>
<td>0.009</td>
<td>39.85</td>
</tr>
<tr>
<td>7 – 3D</td>
<td>$NS$</td>
<td>135</td>
<td>–</td>
<td>0</td>
<td>0</td>
<td>0.21</td>
</tr>
<tr>
<td>8 – 3D</td>
<td>$IN$</td>
<td>135</td>
<td>60</td>
<td>5,67</td>
<td>0.008</td>
<td>6.85</td>
</tr>
<tr>
<td></td>
<td>$IN$</td>
<td>163</td>
<td>82</td>
<td>11,02</td>
<td>0.018</td>
<td>12.24</td>
</tr>
<tr>
<td></td>
<td>$IN$</td>
<td>191</td>
<td>104</td>
<td>15,54</td>
<td>0.032</td>
<td>17.06</td>
</tr>
<tr>
<td></td>
<td>$IN$</td>
<td>231</td>
<td>128</td>
<td>29,43</td>
<td>0.056</td>
<td>31.77</td>
</tr>
<tr>
<td></td>
<td>$S$</td>
<td>259</td>
<td>143</td>
<td>32,27</td>
<td>0.006</td>
<td>35.14</td>
</tr>
<tr>
<td>9 – 4D</td>
<td>$IN$</td>
<td>405</td>
<td>250</td>
<td>76,31</td>
<td>0.114</td>
<td>78.81</td>
</tr>
<tr>
<td></td>
<td>$S$</td>
<td>537</td>
<td>341</td>
<td>132,09</td>
<td>0.008</td>
<td>134.49</td>
</tr>
<tr>
<td>10 – 4D</td>
<td>$IN$</td>
<td>405</td>
<td>240</td>
<td>78,21</td>
<td>0.100</td>
<td>80.88</td>
</tr>
<tr>
<td></td>
<td>$IN$</td>
<td>473</td>
<td>307</td>
<td>108,09</td>
<td>0.175</td>
<td>111.20</td>
</tr>
<tr>
<td></td>
<td>$IN$</td>
<td>533</td>
<td>355</td>
<td>163,52</td>
<td>0.278</td>
<td>169.39</td>
</tr>
<tr>
<td></td>
<td>$IN$</td>
<td>593</td>
<td>399</td>
<td>219,06</td>
<td>0.393</td>
<td>223.93</td>
</tr>
<tr>
<td></td>
<td>$IN$</td>
<td>677</td>
<td>477</td>
<td>269,44</td>
<td>0.603</td>
<td>275.11</td>
</tr>
<tr>
<td></td>
<td>$IN$</td>
<td>745</td>
<td>537</td>
<td>329,25</td>
<td>0.688</td>
<td>336.16</td>
</tr>
<tr>
<td></td>
<td>$S$</td>
<td>865</td>
<td>601</td>
<td>441,87</td>
<td>0.018</td>
<td>447.96</td>
</tr>
<tr>
<td>11 – 5D</td>
<td>$IN$</td>
<td>1215</td>
<td>960</td>
<td>1079,93</td>
<td>1.745</td>
<td>1096.58</td>
</tr>
<tr>
<td></td>
<td>$IN$</td>
<td>1403</td>
<td>1147</td>
<td>5268,03</td>
<td>5.600</td>
<td>5310.95</td>
</tr>
<tr>
<td></td>
<td>$S$</td>
<td>1707</td>
<td>1365</td>
<td>4288,72</td>
<td>0.147</td>
<td>4322.27</td>
</tr>
</tbody>
</table>

Table 3.1: Experimental results for stability verification of PSS
This chapter introduces a counterexample guided abstraction refinement (cegar) algorithm for stability analysis of polyhedral hybrid systems, which is published in “Counterexample Guided Abstraction Refinement for Stability Analysis”, CAV’16 [PS16b]

Contents

4.1 Reachability relations .......................... 51
4.2 cegar algorithm ................................. 52
4.3 Validation ....................................... 55
  4.3.1 Validation Problem ............................ 55
  4.3.2 Validation procedure ......................... 57
4.4 Refinement ..................................... 60
  4.4.1 Refinement problem ........................... 60
  4.4.2 Refinement procedure ......................... 61
4.5 Implementation and experimental results .......... 63
4.6 Summary ....................................... 65
In this chapter, we present a counterexample guided abstraction refinement (CEGAR) algorithm for stability analysis of polyhedral switched systems. The results are built upon the quantitative predicate abstraction and model checking algorithm for stability analysis introduced in Chapter 3. The model checking algorithm returns a counterexample indicating a potential reason for instability. The main contributions include the validation of the counterexample and the 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 consists of checking if an abstract counterexample actually corresponds to a concrete counterexample. In the context of safety verification, it consists of checking if there exists a finite execution of the system which follows the sequence of abstract states in the abstract counterexample. However, the validation problem we encounter is not a bounded model checking problem as for the case of safety validation. 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. Using novel insights, we present a simple 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 $y = mx$ for some $m > 1$. 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.

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. Two refinement strategies are proposed, both based on the insights from the validation step. One of them is applicable always, however, does not eliminate a large fraction of counterexamples. The other is applicable only in certain cases, but it 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 the refinement algorithm ignores the edge weights and aims to “eliminate” the cycle. Otherwise, the refinement algorithm considers the weights and only aims to reduce the weights on the cycle.

The validation and refinement algorithms are integrated into AVERIST to implement the CEGAR algorithm. An experimental comparison between the CEGAR algorithm and the previous abstraction and model checking algorithms is reported. For the latter algorithms, the refinement is based on the naive strategy of uniformly adding new predicates. Such comparison demonstrates
the benefits of CEGAR algorithm both in terms of reduced computation time and smaller abstractions that result from the careful refinement in each iteration.

There is some work on automated verification of stability of linear systems by iteratively refining partitions [OBT07, MT13, YS04], however, it is not an abstraction based approach and the refinements are not guided by counterexample analyses. Counterexample guided abstraction refinement technique has been explored for safety verification of hybrid systems [ADI03, CFH+03, PDMV13] and for region stability analysis [DM11].

\section{Reachability relations}

Certain predicates related to reachability, necessary in this chapter, are introduced. Initially, fix an \( n \)-dimensional pss \( \mathcal{H} \), two locations \( q_1 \) and \( q_2 \) in \( Q_{\mathcal{H}} \) and three polyhedral sets \( P_1 \), \( P_2 \) and \( P \) over \( \mathbb{R}^n \) for the rest of the section.

\[\text{reachrel}_{\mathcal{H}}((q_1, P_1), P, (q_2, P_2)) = \{ (x, y) \in \mathbb{R}^n \times \mathbb{R}^n \ | \ \exists \ \text{finite } P\text{-execution } \sigma = (\iota, \eta, \gamma) \in \text{P-exec}(\mathcal{H}), [\iota] = [0, T], x = \eta(0) \in P_1, y = \eta(T) \in P_2, \gamma(0) = q_1 \text{ and } \gamma(\text{last(dom}(\iota))) = q_2 \}\]

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 Section 3.3.3 that the \text{reachrel}_{\mathcal{H}} \) is computable and can be represented as a 2\( n \)-dimensional polyhedral set. Next, the predecessor and successor operators denoted by \text{pre}_{\mathcal{H}} \) and \text{post}_{\mathcal{H}} \), and their weighted counterparts \text{wpre}_{\mathcal{H}} \) and \text{wpost}_{\mathcal{H}} \), respectively, are defined.

\begin{itemize}
  \item \text{pre}_{\mathcal{H}}((q_1, P_1), P, (q_2, P_2)) = \\
      = \{ x \in P_1 \ | \ \exists y \in P_2 : (x, y) \in \text{reachrel}_{\mathcal{H}}((q_1, P_1), P, (q_2, P_2)) \}
  \item \text{post}_{\mathcal{H}}((q_1, P_1), P, (q_2, P_2)) = \\
      = \{ y \in P_2 \ | \ \exists x \in P_1 : (x, y) \in \text{reachrel}_{\mathcal{H}}((q_1, P_1), P, (q_2, P_2)) \}
  \item \text{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 \text{reachrel}_{\mathcal{H}}((q_1, P_1), P, (q_2, P_2)), |y| = w \}
  \item \text{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 \text{reachrel}_{\mathcal{H}}((q_1, P_1), P, (q_2, P_2)), |y| = w \}
\end{itemize}

Here, \( w \) is a positive real number and \(| \cdot |\) denotes the infinity norm of an element in \( \mathbb{R}^n \).
Algorithm 3 CEGAR for stability analysis

Require: $H, P, F$
Ensure: Stable/Unstable
1: if CHECK-EXPLOSION($H, P$) then
2: return Unstable
3: $G :=$ ABSTRACTION($H, P, F$)
4: while true do
5: $\pi :=$ MODEL-CHECKING($G$)
6: if $\pi$ not counterexample then
7: return Stable
8: $\psi :=$ ENCODE-$\psi_\pi (m > 1)$
9: if CHECK-SATISFIABILITY($\psi$) then
10: return Unstable
11: else
12: $\psi :=$ ENCODE-$\psi_\pi (m \leq 1)$
13: if CHECK-SATISFIABILITY($\psi$) then
14: $G :=$ WEIGHTED-REFINEMENT($G, \pi$)
15: else
16: $G :=$ REFINEMENT($G, \pi$)

4.2 CEGAR algorithm

In this section, we present the CEGAR framework for Lyapunov stability verification. The algorithm is summarized in Algorithm 3. First, minor modifications in the abstraction and model checking algorithms from Chapter 3 are introduced. Then, the new validation and refinement algorithms are presented.

Abstraction The abstraction procedure is the quantitative predicate abstraction introduced in Chapter 3 which constructs a finite state system using a finite set of predicates. This finite state system simulates the concrete system. The abstraction procedure constructs a finite weighted graph as illustrated in Figure 4.1b. A small modification to simplify the output weighted graph is applied. Precisely, this modification is that the vertices of the graph correspond to pairs of location and facet of the partition, instead to pairs of location and element of the partition as previously considered. 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 element of the facets. A facet is an element with affine dimension smaller than the space dimension. Next, the formal construction of this modified abstract system is presented, for which some auxiliary definitions need to be introduced.
4.2. CEGAR algorithm

Definition 15. A facet partition \( \mathcal{F} \) of a polyhedral partition \( \mathcal{P} \) is a polyhedral partition of \( \bigcup_{P \in \mathcal{P}} \partial(P) \), where \( \partial(P) \) is the boundary of \( P \).

Definition 16. Let \( \mathcal{H} \) be a PSS, \( \mathcal{P} \) be a polyhedral partition of \( X \) and \( \mathcal{F} \) be a facet partition of \( \mathcal{P} \). The abstract system is the finite weighted graph \( \text{abs}(\mathcal{H}, \mathcal{P}, \mathcal{F}) = (V, E, w) \) defined as follows.

- \( V = Q \times F \).
- \( E \subseteq V \times P \times V \) is \( \{(q_1, f_1), (q_2, f_2) \mid \text{reachrel}(q_1, f_1, (q_2, f_2)) \neq \emptyset\} \).
- \( w : E \rightarrow \mathbb{R}_{\geq 0} \cup \{\infty\} \), such that for \( e = ((q_1, f_1), (q_2, f_2)) \in E \),
  \[
  w(e) = \sup\{|y|/|x| \mid (x, y) \in \text{reachrel}(q_1, f_1, (q_2, f_2))\}
  \]

The weight computation on the edges of the abstract graph is performed by solving an optimization problem over the reachability relation polyhedral as explained in 3.3.2.

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. Therefore, Theorem 1 provides sufficient conditions on the finite weighted graph which imply Lyapunov 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 \( q \in Q_{\mathcal{H}} \) such that \( P \subseteq I_{\mathcal{H}}(q) \). The region \( P \) is exploding in \( \mathcal{H} \) in the case of \( P \cap F_{\mathcal{H}}(q) \neq \emptyset \). Given a path \( \pi \), let \( w(\pi) \) denote the product of the weights on the edges of \( \pi \).
According to Theorem 1, stability can be verified by checking Conditions [P1] and [P2]. These conditions can be efficiently checked as explained in Subsection 3.2.4. Such an evaluation corresponds to the model checking procedure for Lyapunov stability of polyhedral switched systems. The model checking procedure analyses if the abstract system satisfies Condition [P1] and Condition [P2]. In case of both conditions hold the procedure returns that $\mathcal{H}$ is stable. In case of Condition [P1] does not hold, the concrete system has an exploding element, hence the model checking procedure returns instability. And in case of Condition [P2] does not hold, the procedure returns an abstract counterexample in the form of a simple cycle with weight $> 1$. For this last case, the CEGAR algorithm proceeds to the validation phase.

**Example 6.** Consider the 3-dimensional PSS shown in Figure 4.1. The Figure 4.1b 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, 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.** The conditions in Theorem 1 are, in fact, both necessary and sufficient in the case of 2-dimensional PSS’s [PV13], however, they are 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 $w_2$.

**Validation** is the technique to analyse the abstract counterexample in case of being output by the model checking procedure. The analysis will determine if the abstract counterexample corresponds to a violation of stability in the input polyhedral switched system or is a spurious counterexample.

**Refinement** is the technique applied in case the abstract counterexample is spurious. It consists of adding new predicates for the state space partition in
order to obtain a more accurate weighted abstract graph. The refinement can be chosen as uniform or selected one. In case of uniform refinement, the added predicates will divide the state space uniformly; while in the case of selected refinement the new predicates are constructed by considering the abstract counterexample. The selected refinement can be of two types – one only based on the abstract counterexample or one based also on the weights associated to the abstract counterexample.

The validation and refinement procedures and their theoretical basis are presented in detail in subsequent sections. Therefore, some of the functions integrated in Algorithm 3 are also explained in these sections.

### 4.3 Validation

Some preliminaries are presented and the validation problem is defined. Next, the validation procedure and its theoretical basis are presented.

**Definition 17.** A simple cycle $\pi$ in $\text{abs}(\mathcal{H}, \mathcal{P}, \mathcal{F})$ is an abstract counterexample if $w(\pi) > 1$.

### 4.3.1 Validation Problem

Validation consists of checking if the abstract counterexample corresponds to a violation of stability in the concrete system. 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 $\mathcal{G} = \text{abs}(\mathcal{H}, \mathcal{P}, \mathcal{F})$. The following definition states a connection between the abstract counterexample and executions in the concrete system.

**Definition 18.** An execution $\sigma = (i, \eta, \gamma)$ of $\mathcal{H}$ is said to follow the abstract counterexample $\pi$ of $\mathcal{G} = \text{abs}(\mathcal{H}, \mathcal{P}, \mathcal{F})$, denoted $\sigma \sim \pi$, if there exists a non-decreasing sequence of times $0 = t_0, t_1, t_2, \ldots$ such that:

- $\eta(t_i) \in f_i \mod k$,
- $\eta(t) \in P_i$ for $t \in [t_j, t_i]$ where $j \mod k = i, l \mod k = i + 1$ and
- $(\eta(t_i), \eta(t_{i+1})) \in \text{reachrel}((q_i, f_i), P_i, (q_{i+1}, f_{i+1}))$. 
Further, $\sigma$ is said to follow $\pi$ respecting the weights, denoted $\sigma \sim^{w} \pi$, if in addition
\[
\left| \eta(t_{i+1}) \right| = W((q_{i \mod k}, f_{i \mod k}), P_{i \mod k}, (q_{(i+1) \mod k}, f_{(i+1) \mod k})).
\]

The following notion captures the violation of Lyapunov stability along $\pi$. The abstract counterexample $\pi$ 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.

\[(C1) \exists \varepsilon > 0, \forall \delta > 0, \exists \sigma \in \text{exec}(\mathcal{H}) \text{ such that} \]
\[
\sigma \sim^{w} \pi, \eta(0) \in B_{\delta}(\bar{0}), \exists t \in [\tau], \eta(t) \not\in B_{\varepsilon}(\bar{0}).
\]

The next propositions state that the above condition in fact implies that there is a complete execution along $\pi$.

Previously, some definitions need to be introduced. Let $\text{prereach}^i(S_0)$, for some $S_0 \subseteq f_0$, denote the set of points from which there is a sequence of length $i$ following $\pi$ which starts at $S_0$. Similarly, let $\text{wprereach}^i(S_0)$ denote the points from which the executions also respect the weights. The formal definitions are introduced below.

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

In addition to $\text{wprereach}$, $\text{wpostreach}$ is also defined.

- $\text{wprereach}^0(S_0) = S_0$.
- $\text{wprereach}^i(S_0) = \text{wpre}_{\mathcal{H}}((q_j, f_j), P_j, w_j, (q_{j+1}, \text{wprereach}^{i-1}(S_0)))$, where $w_j = W((q_{j-1}, f_{j-1})(q_j, f_j)), i > 0$ and $j = k - (i \mod k)$.
- $\text{wpostreach}^0(S_0) = S_0$.
- $\text{wpostreach}^i(S_0) = \text{post}_{\mathcal{H}}((q_j, f_j), P_j, w_j, (q_{j+1}, \text{wpostreach}^{i-1}(S_0)))$, where $w_j = W((q_{j-1}, f_{j-1})(q_j, f_j)), i > 0$ and $j = i \mod k$.

**Proposition 6.** If $S_0 \subseteq f_0$ is a closed set, then $\text{prereach}^i(S_0)$ is a closed set.

**Proof.** It follows from the fact that the mapping from a facet $f_{(i+1) \mod k}$ to the facet $f_i$ is following the polyhedral set $-\text{chull}(P_i)$. Since $-\text{chull}(P_i)$ is a closed set, the image of the mapping is closed. By induction on $i$, one can deduce that $\text{prereach}^i(S_0)$ is a closed set. \[\blacksquare\]
Proposition 7. Condition (C1) is equivalent to the existence of a complete execution $\sigma$ of $\mathcal{H}$ such that $\sigma \xrightarrow{w} \pi$.

Proof. Note that for every $\varepsilon > 0$, there is an execution following $\pi$ through the weights whose scaling is $> \varepsilon$, and it reaches one of the facets. Define $S_i$ as the set of points in $f_{i \mod k}$ such that there exist executions with $i$ iterations starting at the points and following $\hat{\pi}$ through the weights. Then $S_0 = f_0$ and $S_i = \text{wprereach}^i(S_0)$, which are closed sets. We know by Proposition 6 that $S = \bigcap_{i \geq 0} S_i$ is not empty. Therefore, there exists $x \in S$ such that there exists executions following $\pi$ starting from $x$ with $i$ iterations for every $i \geq 0$. Without loss of generality, assume $x = x_0 \in f_0$. Hence, $\text{wpost}_{\mathcal{H}}((q_0, f_0), P_0, w_0, (q_1, x)) \neq \emptyset$ and it is a subset of $f_1$. In this successor reach set there exists at least one point $x_1$ with executions with $i$ iterations for every $i \geq 0$ starting from it. By iterating, construct an infinite sequence of points $\{x_0, x_1, x_2, \ldots\}$ and with them construct easily an infinite execution following $\pi$ respecting the weights since $(x_i, x_{i+1}) \in \text{reachrel}((q_i, f_i), P_i, (q_{i+1}, f_{i+1}))$ and $|x_{i+1}| = W((q_i, f_i), P_i, (q_{i+1}, f_{i+1}))$. ■

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 $\pi$ exactly. In order to accelerate the progress in the CEGAR iterations, a stronger validation problem is considered, where the execution is not required to follow the weights, but still be diverging.

Definition 19. An abstract counterexample $\pi$ is said to be spurious if there does not exist a divergent complete execution $\sigma$ such that $\sigma \xrightarrow{w} \pi$.

Validation problem Given an abstract counterexample $\pi$, is $\pi$ spurious?

4.3.2 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 $x_k = mx_0$.

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

$$
x_i \in f_i, (x_i, x_{i+1}) \in \text{reachrel}((q_i, f_i), P_i, (q_{i+1}, f_{i+1})).
$$
Next, the main theorem for validation is stated.

**Theorem 3.** The following holds for the abstract counterexample \( \pi \):

\[ V1 \exists m > 1 : \psi_\pi(m) \Rightarrow \exists \sigma \in \text{cexec}(H) : \sigma \sim \pi \land \sigma \text{ diverges}. \]
\[ V2 \nexists m : \psi_\pi(m) \Rightarrow \nexists \sigma \in \text{cexec}(H) : \sigma \sim \pi. \]
\[ V3 \exists m : \psi_\pi(m) \land \nexists m > 1 : \psi_\pi(m) \Rightarrow \exists \sigma \in \text{cexec}(H) : \sigma \sim \pi \text{ and } \nexists \sigma \in \text{cexec}(H) : \sigma \sim w_\pi. \]

**Remark.** 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 \( \pi \) which respects the weight, however, there is some complete execution (diverging or not).

Before proving the above result, some definitions and a fixpoint theorem are required 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 \( G \).

**Theorem 4** (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 a strategy for proving the next result.

**Proposition 8.** If there exists \( \sigma \in \text{cexec}(H) \) such that \( \sigma \sim w_\pi \), then there exists a value \( m \) greater than 1 such that \( \psi_\pi(m) \) holds.

**Proof.** Suppose \( \sigma \in \text{cexec}(H) \) such that \( \sigma \sim w_\pi \). Define a set of starting points for divergent executions following \( \pi \) respecting the weights. \( \text{kernel}(\pi) = \{ x \in f_0 | \exists \sigma = (i, \eta, \gamma) \in \text{cexec}(H) : \eta(0) = x, \sigma \sim w_\pi \}. \)

\( \text{kernel}(\pi) \) 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 \( H \) in normal form. Next, the set \( \text{kernel}(\pi) \) is the intersection of \( \text{wprereach}^i(f_0) \) for \( i \geq 0 \) which is a multiple of \( k \), the length of the counterexample \( \pi \). (This depends on the fact that the set \( Z = \bigcap_{i \mod k = 0} \text{wprereach}^i(f_0) \) has the property that \( Z \subseteq \text{prereach}^k(Z) \)). Finally, the \( \text{wprereach} \) and intersection operation preserve closedness, 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 $\text{wpostreach}^k(x_0)$. Define $S = \{x \mid |x| \leq 1, x \in \text{kernel}(\pi)\}$. Since $\text{kernel}(\pi)$ is non-empty, convex, closed and closed under positive scaling, we can conclude that $S$ is non-empty, compact and convex. Compactness follows from the assumption that $\text{kernel}(\pi)$ 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 $\text{kernel}(\pi)$ and the set $|x| \leq 1$, both of which are convex.

Define $K$ as an upper bound for the scaling of executions following $\pi$ for one iteration and respecting the weights, so the executions starting from $f_0$ and reaching to $\text{wpostreach}^k(f_0)$, being $k$ the length of $\pi$. 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 $F(q)$ for $q \in Q$ which represents the dynamics.

Next, it is shown that $H$ has a fixed point. For this, the Kakutani’s fixed point theorem is applied. 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 $\pi$ is a counterexample, and $K^{j+1}s^* \in \text{wpostreach}^k(K^j s^*)$ for every $j \geq 0$. ■

Next, Theorem 3 is proven. Suppose $\pi$ 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 $(x_i,x_{i+1}) \in \text{reachrel}((q_i,f_i),P_i,(q_{i+1},f_{i+1}))$, and $x_k = mx_0$. Then consider the infinite execution $\nu = x_0,\ldots,x_{k-1},mx_0,\ldots,mx_{k-1},m^2x_0,\ldots,m^jx_{k-1},\ldots$ such that $(m^jx_i,m^jx_{i+1}) \in \text{reachrel}((q_i,f_i),P_i,(q_{i+1},f_{i+1}))$ for every $j \geq 0$ because of linearity of the flows. Construct with such points and $\pi$ an execution $\sigma$ such that $\sigma \sim \pi$. Note that $\sigma$ diverges, since $m > 1$.

V2 It can be shown by using a similar argument to that in the proof of Proposition 8 but defining $\text{kernel}(\pi)$ as $\{x \in f_0 \mid \exists \sigma = (\iota,\eta,\gamma) \in \text{ceexec}(\mathcal{H}) : \eta(0) = x, \sigma \sim \pi\}$.

V3 Suppose there exists $0 < m \leq 1$ and $x_0,\ldots,x_k \in \mathbb{R}^n$ such that for all $0 \leq i < k$, $x_i \in f_i$ and $(x_i,x_{i+1}) \in \text{reachrel}((q_i,f_i),P_i,(q_{i+1},f_{i+1}))$, and $x_k = mx_0$. Then consider the infinite execution $\nu = x_0,\ldots,x_{k-1},mx_0,\ldots,mx_{k-1},m^2x_0,\ldots,m^jx_{k-1},\ldots$ Construct with
such points and π an execution σ such that σ ∼ π. Note that there
does not exist σ respecting the weights in π because existence of σ would
contradict Proposition 8.

4.4 Refinement

First, the refinement problem is formalized. Then, different strategies for
refinement are presented by considering the reason for the spuriousness of the
abstract counterexample.

4.4.1 Refinement problem

The formal notion of refinement is introduced.

**Definition 20.** Given two abstract systems for a pss \( \mathcal{H} \), \( \mathcal{G} = \text{abs}(\mathcal{H}, \mathcal{P}, \mathcal{F}) = (V, E, w) \) and \( \mathcal{G}' = \text{abs}(\mathcal{H}, \mathcal{P}, \mathcal{F}') = (V', E', w') \), \( \mathcal{G}' \) is said to be a refinement of \( \mathcal{G} \), if there exists a mapping \( \alpha : V' \rightarrow V \) 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 a set of triples is associated with an abstract system which captures the
potential executions and scalings along the edges.

**Definition 21.** Given an abstract system \( \mathcal{G} = (V, E, w) \) of \( \mathcal{H} \),

\[
\text{pot}(\mathcal{G}) = \{((q_1, x), w, (q_2, y)) | \exists ((q_1, f_1), P, (q_2, f_2)) \in E, x \in f_1, y \in f_2, \ |y|/|x| = w((q_1, f_1), P, (q_2, f_2)) \}.
\]

**Definition 22.** An abstract system \( \mathcal{G}' \) of \( \mathcal{H} \) is a strict refinement of an abstract
system \( \mathcal{G} \) of \( \mathcal{H} \), if \( \mathcal{G}' \) is a refinement of \( \mathcal{G} \) and \( \text{pot}(\mathcal{G'}) \) is a strict subset of \( \text{pot}(\mathcal{G}) \).
4.4. Refinement

Refinement problem Given the concrete system $H$ and an abstract system $G$ of $H$, find a strict refinement $G'$ of $G$.

We propose a solution to the refinement problem which depends on a given spurious abstract counterexample $\pi$ of $G$. Consider $\alpha$ to be the mapping associated with the strict refinement $G'$ of $G$. Then, we construct the strict refinement $G'$ in such a way that it will have no cycle whose nodes will be mapped by $\alpha$ to the cycle $\pi$ in $G$.

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

4.4.2 Refinement procedure

Two different strategies for refinement are presented based on the reason for the spuriousness. First, it is shown that non-existence of a complete execution along $\pi$ (respecting the weights) implies that the $\text{prereach}$ ($\text{wprereach}$) computation terminates.

Theorem 5. Consider a pss $H$, an abstract system $G$ of $H$ and a counterexample $\pi$. Then

R1 If $\nexists \sigma \in \text{exec}(H)$ such that $\sigma \sim \pi \Rightarrow \text{prereach}^i(f_0) = \emptyset$ for some $i$.

R2 If $\nexists \sigma \in \text{exec}(H)$ such that $\sigma \overset{w}{\sim} \pi \Rightarrow \text{wprereach}^i(f_0) = \emptyset$ for some $i$.

From Theorem 3, 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 $\iota$ be the smallest index such that $\text{prereach}'(f_0)$ empty. Note that $S_1 = \text{prereach}^{-1}(f_0)$ is not empty. Let the value $\hat{k} = \iota - (\iota \mod k)$. Also, $(\hat{k} + 1) \mod k$ is the index of the facet which contains $S_1$. It also implies that the set $S_2 = \text{post}_{H}((q_{\hat{k}}, f_{\hat{k}}), (g_{(\hat{k}+1) \mod k}, f_{(\hat{k}+1) \mod k}))$ which is also a subset of $f_{(\hat{k}+1) \mod k}$ has an empty intersection with $S_1$. Refinement corresponds to refining the facet $f_{(\hat{k}+1) \mod k}$ 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 $S_1 \cap f^2 = \emptyset$ and $S_2 \cap f^1 = \emptyset$. 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 4.2. 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 $f_3$, we obtain $S_2$, which is a subset of $f_2$, and predecessor reach set of $S_2$ has empty intersection with $f_1$. 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 $3x + 2z = 0$.

**Proposition 9.** The abstract system $\text{abs}(\mathcal{H}, \mathcal{P}, F')$ is a strict refinement of the abstract system $\text{abs}(\mathcal{H}, \mathcal{P}, F)$, where $F' = (\mathcal{F} \setminus \{f_{(k+1) \mod k}\}) \cup \{f^1, f^2\}$.

**Proof.** It follows from the fact that there is an edge from $(q_k, f_k)$ to $(q_{(k+1) \mod k}, f^2)$, but no edge from $(q_k, f_k)$ to $(q_{(k+1) \mod k}, f^1)$. ■

**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, $\text{prereach}$ is replaced by $\text{wprereach}$ and $\text{post}$ by $\text{wpost}$. The following proposition implying progress is similar to Proposition 9, however, the proof relies on the reduction of the weight rather than the removal of an edge.

**Proposition 10.** The abstract system $\text{abs}(\mathcal{H}, \mathcal{P}, F')$ is a strict refinement of the abstract system $\text{abs}(\mathcal{H}, \mathcal{P}, F)$, where $F' = (\mathcal{F} \setminus \{f_{(k+1) \mod k}\}) \cup \{f^1, f^2\}$.

**Proof.** Note that the weight of the edge $((q_k, f_k), P_k, (q_{(k+1) \mod k}, f^1))$, if it exists, is less than the weight $w_k$, the weight of the edge $((q_k, f_k), P_k, (q_{(k+1) \mod k}, f_{(k+1) \mod k}))$. ■
Algorithm 3 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 therefore a non-weighted refinement is performed. However, if $\psi_\pi (m \leq 1)$ is satisfiable, an infinite execution exists, but it is not possible to conclude that it is diverging, hence, Line 14 proceeds with a weighted refinement.

### 4.5 Implementation and experimental results

The validation procedure and the refinement strategies have been integrated into AVERIST, together with the abstraction and model checking algorithms. This integration provides a CEGAR algorithm for polyhedral switched systems. Performance of CEGAR algorithm is tested and also compared with iterating over the abstraction and model checking techniques while adding uniformly new predicates to refine the abstract system.

The experiments are inspired by the example described in Figure 3.4b. The 3-dimensional experiments consist of the same locations as the one in the example where the configurations of the flow function $F$ are modified. The 4 and 5-dimensional experiments are obtained by extending every element of the example to higher dimensions.

<table>
<thead>
<tr>
<th>Exp-dim</th>
<th>Stab</th>
<th>Ans</th>
<th>Elem</th>
<th>IT</th>
<th>Ref</th>
<th>Size</th>
<th>Abs time</th>
<th>MC time</th>
<th>Val time</th>
<th>Ref time</th>
<th>Total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 − 3D</td>
<td>Y</td>
<td>S</td>
<td>163</td>
<td>3</td>
<td>Pre</td>
<td>75</td>
<td>18.75</td>
<td>0.01</td>
<td>0.03</td>
<td>0.03</td>
<td>20.02</td>
</tr>
<tr>
<td>2 − 3D</td>
<td>Y</td>
<td>S</td>
<td>287</td>
<td>11</td>
<td>Pre</td>
<td>153</td>
<td>196</td>
<td>0.23</td>
<td>0.84</td>
<td>0.30</td>
<td>204.50</td>
</tr>
<tr>
<td>3 − 3D</td>
<td>N</td>
<td>NS</td>
<td>135</td>
<td>1</td>
<td>−</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>0.00</td>
<td>0.15</td>
<td>0.15</td>
</tr>
<tr>
<td>4 − 3D</td>
<td>Y</td>
<td>NA</td>
<td>59</td>
<td>11</td>
<td>WPre</td>
<td>123</td>
<td>119.32</td>
<td>0.25</td>
<td>0.94</td>
<td>2.44</td>
<td>130.91</td>
</tr>
<tr>
<td>5 − 3D</td>
<td>N</td>
<td>NS</td>
<td>9</td>
<td>1</td>
<td>−</td>
<td>16</td>
<td>0.30</td>
<td>ε</td>
<td>0.13</td>
<td>0.55</td>
<td>0.55</td>
</tr>
<tr>
<td>6 − 3D</td>
<td>Y</td>
<td>S</td>
<td>151</td>
<td>2</td>
<td>WPre</td>
<td>74</td>
<td>12.55</td>
<td>ε</td>
<td>0.17</td>
<td>0.07</td>
<td>5.72</td>
</tr>
<tr>
<td>7 − 3D</td>
<td>Y</td>
<td>S</td>
<td>179</td>
<td>4</td>
<td>Pre</td>
<td>87</td>
<td>31.63</td>
<td>0.02</td>
<td>0.03</td>
<td>0.04</td>
<td>33.62</td>
</tr>
<tr>
<td>8 − 3D</td>
<td>Y</td>
<td>S</td>
<td>291</td>
<td>11</td>
<td>Pre</td>
<td>157</td>
<td>249.40</td>
<td>0.38</td>
<td>0.57</td>
<td>0.39</td>
<td>269.11</td>
</tr>
<tr>
<td>9 − 4D</td>
<td>Y</td>
<td>S</td>
<td>537</td>
<td>3</td>
<td>Pre</td>
<td>341</td>
<td>312</td>
<td>0.32</td>
<td>0.32</td>
<td>0.10</td>
<td>319.42</td>
</tr>
<tr>
<td>10 − 4D</td>
<td>Y</td>
<td>S</td>
<td>865</td>
<td>7</td>
<td>Pre</td>
<td>601</td>
<td>1543</td>
<td>2.15</td>
<td>1.18</td>
<td>0.28</td>
<td>1592.33</td>
</tr>
<tr>
<td>11 − 5D</td>
<td>Y</td>
<td>S</td>
<td>1706</td>
<td>3</td>
<td>Pre</td>
<td>1365</td>
<td>4208</td>
<td>4.32</td>
<td>0.51</td>
<td>0.12</td>
<td>4252</td>
</tr>
</tbody>
</table>

Table 4.1: Experimental results for CEGAR algorithm

Some of the results are summarized in Table 4.1. Here, Exp-dim refers to the experiment number and to the dimension of the concrete system (number of continuous variables) and Stab states whether the concrete system is Lyapunov
Table 4.2: Comparison of AVERIST and CEGAR technique

<table>
<thead>
<tr>
<th>Exp-dim</th>
<th>Stab</th>
<th>Ans</th>
<th>Elem</th>
<th>Runs</th>
<th>Time</th>
<th>Ans</th>
<th>Elem</th>
<th>IT</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2–3D</td>
<td>Y</td>
<td>85250</td>
<td>6</td>
<td>10658.26</td>
<td>S</td>
<td>287</td>
<td>11</td>
<td>204.50</td>
<td></td>
</tr>
<tr>
<td>2–3D</td>
<td>Y</td>
<td>4034</td>
<td>4</td>
<td>857.23</td>
<td>NA</td>
<td>59</td>
<td>11</td>
<td>130.91</td>
<td></td>
</tr>
<tr>
<td>3–3D</td>
<td>Y</td>
<td>4035</td>
<td>4</td>
<td>181.32</td>
<td>S</td>
<td>151</td>
<td>2</td>
<td>5.72</td>
<td></td>
</tr>
<tr>
<td>4–3D</td>
<td>Y</td>
<td>4035</td>
<td>4</td>
<td>187.24</td>
<td>S</td>
<td>179</td>
<td>4</td>
<td>33.62</td>
<td></td>
</tr>
<tr>
<td>5–4D</td>
<td>Y</td>
<td>27201</td>
<td>3</td>
<td>4728.61</td>
<td>S</td>
<td>537</td>
<td>3</td>
<td>319.42</td>
<td></td>
</tr>
</tbody>
</table>

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). Elem is the number of elements 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, Abs time, model-checking, MC time, validation, Val time and refinement, Ref time, are shown along with the total time Total time. All the times are in seconds, and \( \varepsilon \) 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 are reported. In the case of experiment 4, no answer is obtained, while in the case of experiments 3 and 5 instability is obtained as answer. In experiment 3, instability is due to an exploding region, therefore all the times are zero. In experiment 5, the refinement is not performed because the validation algorithm returns the existence of a concrete counterexample. The experiments illustrate that the CEGAR framework is practically feasible, since validation and refinement overheads are low compared to the total time.

Next, the CEGAR algorithm is compared with AVERIST applying an uniform refinement. AVERIST allows specification of predicates as well as built-in automated methods for generating uniform predicates based on an input granularity value. In the comparison, the examples on AVERIST are run by iteratively increasing the number of predicates using this feature. The presented CEGAR algorithm on the other hand applies the new refinement strategies based on the returned counterexamples for adding the predicates. The chosen termination criterion for CEGAR is a bound of 11 on the number of iterations, and for AVERIST, the process is stopped when the running time is more than 5 times the total time taken by the CEGAR algorithm. All the examples in Table 4.1 are compared with AVERIST plus uniform refinement, however, only a
4.6. Summary

A representative subset of them in Table 4.2 are presented. In Table 4.2, most of the headers refer to the same values that those in Table 4.1. In addition, Runs refers to the number of times AVERIST is run with an incremented number of uniform predicates. As you can 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.

4.6 Summary

This chapter has developed a counterexample guided abstraction refinement framework for the stability analysis of polyhedral switched systems. The developed 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 than in the case of safety analysis.

Next chapter extends the presented results to linear hybrid systems by using a hybridization technique specialized for stability. This technique abstracts a polyhedral hybrid system from a linear hybrid system.
This chapter introduces a dynamic simplification technique, namely hybridization, to abstract a linear hybrid system to a switched system with polyhedral inclusion dynamics, and proves completeness of the hybridization technique for a subclass of linear hybrid systems, which is published in “Hybridization for Stability Analysis of Switched Linear Systems”, HSCC’17 [PS16c]

Contents

5.1 Hybridization for stability analysis .................. 69
  5.1.1 Hybridization construction for linear hybrid automata 69
  5.1.2 Stability preservation theorem ..................... 71
  5.1.3 Completeness results for hybridization ............. 72
5.2 Heuristics for non uniform predicates ............... 76
5.3 Experimental results ................................. 79
  5.3.1 AVERIST evaluation ............................. 80
  5.3.2 Comparison with Stabbyli ........................ 81
5.4 Summary ............................................. 82
The main goal in this chapter is to extend the quantitative predicate abstraction from Chapter 3 to linear hybrid systems. One of the difficulties in extending QPA to linear hybrid systems is the computation of the edges and their weights. This computation requires the construction of a reachability relation in the context of linear hybrid systems, which becomes especially challenging when the number of mode switches is not bounded. A way of tackling the problem consists of abstracting the linear hybrid system (LHS) to a polyhedral hybrid system (PHS), where reachability relations can be efficiently computed as explained in Chapter 3.

The main result is a novel application of the hybridization approach [ADG07, DMT10] for stability analysis. Broadly, hybridization consists of splitting the state space into a finite number of regions and approximating the dynamics in each of the regions by a simpler dynamics. Our hybridization is specialized for stability analysis, and differs from those used for safety analysis, in the class of partitions. We use partitions that split the state space into conic polyhedral sets. The linear dynamics in each of the regions is over approximated by a polyhedral inclusion dynamics. The conic partitioning is fine tuned for stability analysis, since, it guarantees a bound on the “scaling” in a region. The scaling refers to an execution, and is the ratio of the distance between the origin and the end of the execution to the distance between the origin and the beginning of the execution. The scaling in a region is an upper bound of the scalings for all the executions in such region. Hybridization techniques for safety analysis instead seek a bound on the approximation error in the reach set [ADG07, DMT10].

The main theoretical result in this chapter is the completeness of the hybridization technique for the class of asymptotically stable switched linear systems that are uniformly converging in time and Lipschitz continuous. For this class of systems, a determined increment of the granularity for the element partition guarantees obtaining an abstract polyhedral switched system that is asymptotically stable.

On the practical side, this hybridization approach is integrated in the software tool AVERIST. Then, the new version of AVERIST is experimentally compared with Stabhyli. Stabhyli is a state-of-the-art tool for stability analysis of hybrid systems which is based on Lyapunov functions. The experimental results illustrate that the presented method is less prone to numerical errors and scales better than the traditional approaches. In addition, AVERIST returns a counterexample in the event that it fails to prove stability, providing feedback regarding the potential reason for instability. This chapter also investigates some heuristics for partitioning the state space in a more efficient manner.
5.1. Hybridization for stability analysis

This section formally presents the hybridization procedure for constructing a polyhedral hybrid automaton from a linear hybrid automaton, and provides soundness and completeness results.

5.1.1 Hybridization construction for linear hybrid automata

The basic construct in the hybridization procedure is the abstraction of the linear vector field in a region by a polyhedron. This is done by the function $\text{cpoly}$, which given $F : \mathbb{R}^n \to 2^{\mathbb{R}^n}$ and $P \subseteq \mathbb{R}^n$ is defined as

$$\text{cpoly}(F, P) = \bigcup_{x \in P} F(x).$$

Next, we present the formal definition of hybridizing a system $H$ with respect to a partition $\mathcal{P}$ and an auxiliary definition.

**Definition 23.** Two polyhedra $P, Q \subseteq \mathbb{R}^n$ are adjacent in case of $P \cap Q$ has affine dimension $n - 1$, and we denote as $P \adj Q$.

**Definition 24 (Hybridization).** Given an $n$-dimensional linear hybrid system $H = (Q, E, X, F, I, G)$ and a polyhedral partition of $\mathbb{R}^n$, $\mathcal{P} = \{R_1, \ldots, R_k\}$, the hybridization of $H$ with respect to $\mathcal{P}$ is $\text{polyh}(H, \mathcal{P}) = (\hat{Q}, \hat{E}, X, \hat{F}, \hat{I}, \hat{G})$, where:

- $\hat{Q} = \{(q, R) : q \in Q, R \in \mathcal{P} \text{ and } R \subseteq I(q)\}$
- $\hat{E} = \{((q_i, R_i), (q_j, R_j)) \in \hat{Q} \times \hat{Q} : q_i = q_j \text{ or } (q_i, q_j) \in E \text{ and } R_i \adj R_j\}$
- $\hat{F}((q, R))(x) = \text{cpoly}(F(q), R)$.
- $\hat{I}(q, R) = I(q) \cap R$.
- $\hat{G}((q_i, R_i), (q_j, R_j)) = \mathbb{R}^n$ if $q_i = q_j$ and $G((q_i, q_j))$, otherwise.

Note that for every $q \in Q$ and $R \in \mathcal{P}$, $\text{cpoly}(F(q), R)$ is a polyhedral set. Hence, the polyhedral differential inclusion $\dot{x} \in \text{cpoly}(F(q), R)$ over approximates the dynamical system $\dot{x} \in F(q)(x)$ in the polyhedral region $R$. Then, it follows that $\text{polyh}(H, \mathcal{P})$ is a polyhedral hybrid automaton.
Remark. Assume from now on that the partition $\mathcal{P}$ used in the hybridization only consists of conic polyhedral sets, that is, those that are positive scaling closed. The proposed hybridization is similar to that in the literature [PBV95, DHR05], where rectangular/polyhedral abstractions are considered, except that the partition of the state space is specific to stability analysis.

5.1.1.1 Illustration

Here, we explain in detail the hybridization process for a particular 2-dimensional linear hybrid system example. The dynamical behaviour of the example is described below.

The evolution of a linear dynamical system is determined by the equation $\dot{x} = Ax$ where $A$ is a real square matrix. Consider two linear dynamical systems of the form $\dot{x} = A_1 x$ and $\dot{x} = A_2 x$, where the matrices are given as follows

$$A_1 = \begin{pmatrix} 0 & 1 \\ -4 & 0 \end{pmatrix} \quad A_2 = \begin{pmatrix} 0 & 1 \\ -0.1 & 0 \end{pmatrix}$$

The phase portraits showing the solutions for each of these systems are depicted in Figure 5.1. A linear dynamical system defines the evolution of a linear hybrid automaton at each location. A linear hybrid automaton model is depicted in Figure 5.2a. Observe that this system consists of four modes, $q_1, q_2, q_3$ and $q_4$, with associated invariants $R_1, R_2, R_3$ and $R_4$, which correspond to the first, second, third and fourth quadrants of the two dimensional plane, respectively. Hence, the system evolves in the first and third quadrants following a solution of the dynamical system $\dot{x} = A_1 x$ and in the other two quadrants following one of $\dot{x} = A_2 x$. The switching condition is along the common boundaries, and is
5.1. Hybridization for stability analysis

specified by the guard annotations on the edges. A sample execution of the linear hybrid system is given in Figure 5.2b.

\[
\begin{align*}
\dot{x} &= A_2 x \\
x &\in R_2 \\
y &= 0 \\
q_2 &
\end{align*}
\begin{align*}
\dot{x} &= A_1 x \\
x &\in R_1 \\
y &= 0 \\
q_1 &
\end{align*}
\begin{align*}
\dot{x} &= A_1 x \\
x &\in R_3 \\
q_3 &
\end{align*}
\begin{align*}
\dot{x} &= A_2 x \\
x &\in R_4 \\
q_4 &
\end{align*}

(a) Hybrid automaton $\mathcal{H}$

(b) Sample execution

Figure 5.2: Linear hybrid automaton $\mathcal{H}$

The first step to perform hybridization is to choose a polyhedral partition of the continuous state space. Consider the polyhedral partition $P = \{R_1, R_2, R_3, R_4\}$, formed by the planar quadrants, where $R_i$ corresponds to the $i$-th one. This choice determines a polyhedral hybrid system with the same number of modes as in the linear automaton, with the same associated invariants, and the same edges labelled with the same guards. The polyhedral dynamics in the four modes are given by $\dot{x} \in P_1$, $\dot{x} \in P_2$, $\dot{x} \in P_3$ and $\dot{x} \in P_4$, respectively, where $P_1 = \text{cpoly}(A_1, R_1)$, $P_2 = \text{cpoly}(A_2, R_2)$, $P_3 = \text{cpoly}(A_1, R_3)$ and $P_4 = \text{cpoly}(A_2, R_4)$. For instance, in the first quadrant all the flow vectors of the linear hybrid system are between $(1,0)$ and $(0,-1)$, therefore $P_1$ is determined by the constraints $x > 0$ and $y < 0$, as observed in Figure 5.3b. The polyhedral hybrid system is shown in Figure 5.3a and it is analogous to the linear one but with dynamics determined by the differential inclusion $\dot{x} \in P_i$ for each mode $q_i$. A sample execution of the polyhedral hybrid system is depicted in Figure 5.3b. However, this polyhedral hybrid automaton is not stable. To obtain a stable one, we need to use a finer partition.

5.1.2 Stability preservation theorem

The proposed hybridization procedure is sound; the stability of the polyhedral hybrid system output by the hybridization procedure implies the stability of the linear hybrid system. This principle is formalized below.

**Theorem 6.** Given a linear hybrid automaton $\mathcal{H}$ and a polyhedral partition $P$, the following hold:

1. $\text{polyh}(\mathcal{H}, P)$ is Lyapunov stable implies $\mathcal{H}$ is Lyapunov stable.
2. \( \text{polyh}(\mathcal{H}, \mathcal{P}) \) is asymptotically stable implies \( \mathcal{H} \) is asymptotically stable.

Proof. This follows from the fact that \( \text{exec}(\mathcal{H}) \subseteq \text{exec}(\text{polyh}(\mathcal{H}, \mathcal{P})) \), which in turn follows from the fact that the polyhedral differential inclusion \( \dot{x} \in \text{cpoly}(F, R) \) over approximates the dynamical system \( \dot{x} = F(x) \) in the polyhedral region \( R \).

Remark. It has been observed in [PDV12] that simulations do not preserve stability and the stronger notion of uniformly continuous simulations is required. Note that in Theorem 6, the simulation relation between the two systems is the identity relation which trivially satisfies the additional constraints of uniform continuity imposed by [PDV12].

5.1.3 Completeness results for hybridization

This section identifies a subclass of switched linear hybrid systems for which the hybridization procedure is complete, that is, for every asymptotically stable system in this class, by iteratively choosing finer and finer partitions for the hybridization, the procedure will provide an abstract polyhedral hybrid system that is asymptotically stable.

Before identifying the subclass of systems referred above, we will provide a detailed proof for the class of linear dynamical systems. Then, this proof will be extended for the defined subclass of switched linear hybrid systems. Let \( \mathcal{H}_A \) denote the linear hybrid automaton (with one mode) corresponding to the dynamical system \( \dot{x} = Ax \).

Theorem 7. Let \( \mathcal{H}_A \) be a linear hybrid automaton corresponding to the dynamical system \( \dot{x} = Ax \) which is asymptotically stable. Then, there exists a
polyhedral partition $\mathcal{P}$ such that the polyhedral hybrid system $\text{polyh}(\mathcal{H}_A, \mathcal{P})$ is asymptotically stable with respect to $\bar{0}$.

The hybridization techniques for safety analysis ensure a bounded error in the executions of the original and approximate systems over a bounded time. They rely on the fact that error between the concrete and the abstract vector fields is bounded. However, this is no longer true in our setting. More precisely, the Hausdorff distance between $Ax$ (the value of the concrete vector field) and $P = \{Ax \mid x \in R\}$, the abstract vector field in a region $R$ (positively closed) is infinity. Hence, our proof of completeness is bit more involved. We construct an intermediate system which is equivalent to the abstract system. In this new system, though we cannot bound the distance between the vector fields of the concrete system and that of the intermediate system at state $x$ by some $\varepsilon$, we can bound it by $\varepsilon|x|$.

Observe that the scaling in a region $R$ is determined by the vector field restricted to the state space with norm 1. Let $\text{cupo}(A, R) = \{Ax : x \in R, |x| = 1\}$.

**Definition 25.** Given a polyhedral set $P$ and a scalar $a$, let $Pa = \{xa : x \in P\}$. The norm of a polyhedral set $P$ is $|P| = \max\{|p| : p \in P\}$.

**Definition 26.** The scaled hybridization of $\mathcal{H}$ with respect to $\mathcal{P}$ is defined as $\text{spolyh}(\mathcal{H}, \mathcal{P}) = (\hat{Q}, \hat{E}, \bar{X}, \hat{F}, \hat{I}, \hat{G})$, where all the elements in the tuple are the same as for the hybrid system $\text{polyh}(\mathcal{H}, \mathcal{P})$ but for the vector field function, which for every $(q, R) \in \hat{Q}$, $\hat{F}((q, R))(x) = \text{cupo}(F(q), R)|x|$.

The following lemmas comprise the proof for Theorem 7.

**Lemma 3.** $\text{spolyh}(\mathcal{H}, \mathcal{P})$ is asymptotically stable if and only if $\text{polyh}(\mathcal{H}, \mathcal{P})$ is asymptotically stable.

**Lemma 4.** Let $A$ be an $n \times n$ matrix and let $\mathcal{H}_A$ be the hybrid system associated with the matrix. Let $\mathcal{H}_A$ be asymptotically stable with respect to $\bar{0}$. Then, there exists a partition $\mathcal{P}$ such that the polyhedral system $\text{spolyh}(\mathcal{H}_A, \mathcal{P})$ is asymptotically stable with respect to $\bar{0}$.

The Lemma 4 depends on the following two propositions.

**Proposition 11.** Consider a polyhedral partition $\mathcal{P} = \{R_1, \ldots, R_k\}$ and a finite set of bounded polyhedral sets $\{P_1, \ldots, P_k\}$. Consider two differential inclusions,

\[ \dot{x} \in \{Ax\} \quad (5.1) \]
\[ \dot{x} \in F(x) \quad (5.2) \]
where $F(x) = P_i |x|$ for $x \in R_i$ and for every $0 \leq i \leq k$. Let $d_H(Ax, F(x)) \leq \varepsilon |x|$ for every $x$. Let $x$ and $y$ be solutions of 5.1 and 5.2 respectively, such that $x(0) = y(0)$. Then,

$$|x(t) - y(t)| \leq \varepsilon |y(0)| \frac{e^{mt} - e^{|A|t}}{m - |A|}$$

for every $t \geq 0$ with $m = \max_{0 \leq i \leq k} |P_i|$. 

Proof. Consider $x$ and $y$ be solutions of 5.1 and 5.2 respectively, such that $x(0) = y(0)$. Define the error function $z(t) = |x(t) - y(t)|$ for $t \in [0, \infty)$. For infinity norm it is known that $\frac{d}{dt}|f(t)| \leq |\frac{d}{dt} f(t)|$. Then, $\dot{z} \leq |\dot{x} - \dot{y}|$, which is equal to $|Ax - \dot{y}| = |Ax - Ay + Ay - \dot{y}| \leq |Ax - Ay| + |Ay - \dot{y}|$, and since $\dot{y} \in F(y), |Ay - \dot{y}| \leq \varepsilon |y|$. Then, the following differential system is obtained

$$\begin{cases} \dot{z}(t) \leq |A|z(t) + \varepsilon |y| \\ \dot{y} \in F(y) \end{cases}$$

Let us first solve the inclusion in the system. Consider $y \in R_i$, then $\dot{y} \in P_i |y|$. Therefore $|\dot{y}| \leq |P_i| |y| = |P_i| |y|$. Define $m = \max_{0 \leq i \leq k} |P_i|$. Therefore, $|\dot{y}| \leq m|y|$. Denote $|y(t)| = r(t)$. Since $\frac{d}{dt}|y| \leq |\dot{y}|$, the following inequality holds $r(t) \leq m r(t)$. Since infinity norm is differentiable everywhere except at the origin, and there is no time such that zero norm is obtained, Grönwall’s lemma is applied and states that the inequality solution is bounded by the correspondent differential equation. The solution is $r(t) \leq r(0) e^{mt}$. Plug the solution to the first equation in the system, $\dot{z}(t) \leq |A|z(t) + \varepsilon e^{mt} r(0)$ where $z(t)$ is differentiable everywhere but in the origin and by applying also the Grönwall’s lemma and solving the differential equation, we obtain

$$z(t) \leq e^{|A|t} z(0) + \frac{\varepsilon r(0)}{m - |A|} (e^{mt} - e^{|A|t})$$

(5.3)

Since $z(0) = 0$, the inequality is just

$$z(t) \leq \frac{\varepsilon r(0)}{m - |A|} (e^{mt} - e^{|A|t})$$

(5.4)

Proposition 12. Let $A$ be an $n \times n$ matrix and $H_A$ be the hybrid system associated with the matrix. Consider $\varepsilon > 0$. Then, there exists a partition $P = \{ R_1, \ldots, R_k \}$ such that $d_H(A, P_i |x|) \leq \varepsilon |x|$ for every $x \in R_i$ where $P_i = \text{cupoly}(A, R_i)$. 

Proof. $Ax$ is a continuous function and the state space with norm 1 is compact.
Hence, $Ax$ restricted to the state space of norm 1 is a uniformly continuous function. Therefore, given any $\varepsilon$, there exists a $\delta$, such that for every $x, y$ with $|x - y| \leq \delta$, $|Ax - Ay| \leq \varepsilon$. Choose a polyhedral partition such that each of the elements restricted to norm 1 is contained in a $\delta$ ball. Consider an element $R_i$, one wants to prove that $d_H(Ax, P_i|x|) \leq \varepsilon|x|$. It is known $d_H(Ax, P_i|x|) = |Ax - Ay|x|$ for some $y \in R_i$ such that $|y| = 1$. Then, $|Ax - Ay|x| = |Ax - Ay'|$ where $y' \in R_i$ and $|y'| = |x|$, so it can be rewritten as $|(A \frac{x}{|x|} - A \frac{y'}{|y'|})|x|$. Since $|\frac{x}{|x|} - \frac{y'}{|y'|}| \leq \delta$ by the partition construction, we get $|A\frac{x}{|x|} - A\frac{y'}{|y'|}| \leq \varepsilon$ and finally, by substituting, $d_H(Ax, P_i|x|) \leq \varepsilon|x|$.

Theorem 6 states that our hybridization based analysis approach is sound in that if the polyhedral abstraction is stable, then so is the linear hybrid system. Here, we show that our analysis method is complete for linear dynamical systems with respect to asymptotic stability, that is, given a linear dynamical system which is asymptotically stable, one can construct a polyhedral abstraction which is stable.

**Proposition 13.** Let $\dot{x} = Ax$ be a dynamical system in $\mathbb{R}^n$ which is asymptotically stable. Then, for every $\gamma > 0$ there exists a value $T_\gamma \geq 0$ such that for every solution $x(t)$ with $|x(0)| = \gamma$, $|x(t)| \leq \frac{\gamma}{2}$ holds for every $t \geq T_\gamma$.

**Proof.** Let $\dot{x} = Ax$ be asymptotically stable. Then $A$ can be written as $PJP^{-1}$, where $P$ is a real invertible matrix and $J$ is the real Jordan matrix [HSD12]. A solution $x(t)$ of the system can be expressed as $e^{At}x(0) = e^{PAP^{-1}t}x(0)$, which is equal to $Pe^{Jt}P^{-1}x(0)$ by definition of matrix exponential. Because of norm matrix properties, we get $|x(t)| \leq |P| |e^{Jt}| |P^{-1}| |\gamma|$. The asymptotic stability implies that $A$ is Hurwitz, and therefore the matrix $e^{Jt}$ is also Hurwitz, which means that every term in it is of the form $e^{\lambda t}p(t)$ where $\lambda < 0$ and $p(t)$ is a polynomial function. Then, every term in $Pe^{Jt}P^{-1}x(0)$ is equal to $e^{\lambda t}p'(t)$ where $p'(t)$ is a different polynomial and the values will be dominated by the exponentials. Hence, for every row, $x_i(t)$, there exists $T_i^\gamma > 0$ such that $|x_i(t)| \leq \frac{\gamma}{2}$ for every $t \geq T_i^\gamma$. So it is enough to choose $T_\gamma = \max_{0 \leq i < n}(T_i^\gamma)$. ■

**Proof of Lemma 4** Suppose $H_A$ is asymptotically stable. Fix $\gamma \geq 0$. Proposition 13 states that there exists $T_\gamma \geq 0$ such that for every $\sigma \in \text{exec}(H_A)$ with $|\sigma(0)| = \gamma$, $|\sigma(t)| \leq \frac{\gamma}{2}$ for every $t \geq T_\gamma$. Fix

$$
\varepsilon = \frac{\gamma(m - |A|)}{4|\sigma'(0)|(e^{mT_\gamma} - e^{|A|T_\gamma})}
$$

By Proposition 12, there exists a partition $P$ such that $d_H(Ax, \text{cupoly}(A, P)) \leq \varepsilon|x|$. Consider $\sigma \in \text{exec}(H_A)$ and $\sigma' \in \text{exec}(\text{spoly}(H_A, P))$ with $\sigma(0) = \sigma'(0)$. 

**5.1. Hybridization for stability analysis**

75
Then, by Proposition 11 \(|\sigma(t) - \sigma'(t)| \leq \frac{\gamma}{2}\). Since it is also known that \(|\sigma(T_\gamma)| \leq \frac{\gamma}{2}\), we get \(\sigma'(T_\gamma) \leq \frac{3\gamma}{4}\). It is possible to iterate such construction for values of \(\gamma\) smaller and smaller, obtaining \(\sigma' \in \text{spolyh}(H_A, P)\) closer and closer to zero for some time values.

**Definition 27.** A linear hybrid system \(H\) is Lipschitz continuous if there exists a constant \(L\) such that for every \(x, y \in X_H\) with \(x \in I(q_1)\) and \(y \in I(q_2)\) the following holds \(|A_i x - A_j y| \leq L|x - y|\). We say that \(H\) is uniformly converging if for every \(\gamma > 0\), there exists \(T_\gamma \geq 0\) such that for every execution \(\sigma\) of \(H\), \(|\sigma(0)| = \gamma\) implies \(|\sigma(t)| \leq \gamma/2\) for all \(t \geq T_\gamma\).

**Theorem 8.** Let \(H\) be a linear hybrid system that is Lipschitz continuous and uniformly converging. Then, there exists a polyhedral partition \(P\) such that the polyhedral hybrid system \(\text{polyh}(H, P)\) is asymptotically stable with respect to \(\bar{0}\).

**Proof.** Here, the proof of the theorem is sketched and is similar to that of the proof of Theorem 7. Proposition 11 can be extended to linear switched systems which are Lipschitz continuous. The error between the solution of \(H\) and some approximation whose vector field is at most \(\varepsilon\) far from \(H\) can be analyzed as follows. \(\dot{z} \leq |\dot{x} - \dot{y}| = |A_i x - A_j y| = |A_i x - A_j y + A_j y - y'| \leq |A_i x - A_j y| + |A_j y - y'|\) for some \(y' \in F_j(y)\). Then, a new differential system comes out

\[
\begin{align*}
\dot{z}(t) & \leq Lz(t) + \varepsilon|y'| \\
\dot{y} & \in F_j(y)
\end{align*}
\]

The rest of the proof is similar to that of Proposition 11. Propostion 12 requires to choose a partition such that the difference in the vector fields between \(H\) and the polyhedral inclusion induced by it is bounded by \(\varepsilon\). This can be done by choosing a partition which ensures the \(\varepsilon\) bound for every \(A_i\) in \(H\). Proposition 13 is in fact captured using the uniformly converging condition in the theorem.

### 5.2 Heuristics for non uniform predicates

Next, this section explores certain heuristics to construct the polyhedral partition for the hybridization procedure. A naive algorithm is to uniformly partition the state space and increase the granularity of the partitions. This partitioning approach ignores the dynamics of the system in the selection of the partition. Instead, a new heuristic to refine the partition is proposed, such that the size of the resulting polyhedral inclusions are similar.
The contribution in this section is to explain heuristics for defining a non uniform predicate construction. Consider a polyhedral switched system $\text{polyh}(\mathcal{H}, \mathcal{P})$, obtained by hybridization of the linear switched system $\mathcal{H}$ with an initial set of predicates. The state space will be partitioned in order to obtain a more accurate polyhedral hybrid system. For constructing a finer partition, choose a location, $(q, R)$, in $\text{polyh}(\mathcal{H}, R)$ and the polyhedral inclusion associated to such location, $P = \text{cpoly}(F(q), R)$. The target is to split region $R$. To split this region, first compute the diameter of $P$, $d_P$, which corresponds to the maximum of $|x - y|$ while $x, y \in P$ and $|x| = |y| = 1$. The solution to the optimization problem returns the diameter value $d_P$ and the two points $x_P, y_P$. Then, construct a vector $v_P$ starting at $x_P$ and finishing at $y_P$. This vector determines the direction in the dynamical polyhedron $P$ where the longest error on approximating the linear dynamics arises. To overcome this problem, a hyperplane is constructed, which splits the dynamical polyhedron and it is perpendicular to that direction. The hyperplane is the set $\{x \in \mathbb{R}^n : v_P \cdot x = 0\}$. Observe that this hyperplane is not partitioning the state space but the polyhedral dynamics. Then, the hyperplane is transformed into a hyperplane in region $R$ which will result in the polyhedral dynamics splitting. This last hyperplane is the set $\{x \in R : v_P A x = 0\}$, where $A$ is the matrix for the linear dynamical system defining the vector field of location $q$ in the initial linear hybrid system.

One needs to split just regions that generate a coarse polyhedral inclusion dynamics. For choosing such regions we compare the diameter of each dynamical polyhedra with a diameter bound. This diameter bound is computed by considering every location $(q, R)$ in $\text{polyh}(\mathcal{H}, R)$ and the polyhedra $\text{cupoly}(F(q), R)$, and it is defined as

$$D = \max_{(q, R) \in \hat{Q}} d_{\text{cupoly}}(F(q), R)$$

The regions chosen are those with polyhedral inclusion dynamics whose polyhedral diameter is greater or equal to $D$ or to some percentage of $D$. The higher the percentage, the less predicates will be added for partitioning the state space.

**Example 7.** A 2-dimensional linear hybrid system $\mathcal{H}$ illustrates the hybridization process. The dynamical behaviour of the system is determined by two linear dynamical systems, $\dot{x} = A_1 x$ and $\dot{x} = A_2 x$, defined by the following matrices:

$$A_1 = \begin{pmatrix} 0 & 1 \\ -4 & 2 \end{pmatrix}, \quad A_2 = \begin{pmatrix} 0 & 1 \\ -3 & 2 \end{pmatrix}$$

The goal is to evaluate the difference between uniform and non uniform predicate construction methods. In the case of the uniform method, 64 predicates are
needed to decide stability of the polyhedral hybrid system $\text{polyh}(\mathcal{H})$, while with the non uniform method, stability is verified by considering 40 predicates for the polyhedral hybrid system construction. We observe that the non uniform method provides a less number of predicates for verifying stability. The reduction in number of predicates when compared with the uniform predicate construction is because the choice for new predicates in the non uniform methods implies a homogeneous division of the dynamical polyhedra.

Consider the linear dynamical system defined by the matrix $A_1$ and restrict it to the polyhedral region $R = \{(x, y) \in \mathbb{R} : x \geq 0, y \geq 0\}$ which coincides with the first quadrant. Observe in Figure 5.4 a sample execution in $R$ of the linear system $\dot{x} = A_1x$, and the dynamical polyhedron defined for such region and linear dynamics, denoted as $P$. In the case of uniform predicate construction, observe in Figure 5.4a the addition of the hyperplane $\{(x, y) \in \mathbb{R} : x - y = 0\}$, which results in a new partition with two regions, $P \cap \{(x, y) \in \mathbb{R} : x - y \leq 0\}$ and $P \cap \{(x, y) \in \mathbb{R} : x - y >= 0\}$. In the first region we add a new dynamical polyhedron $P_1$ with diameter 2 while in the second region the new dynamical polyhedron $P_2$ has diameter 0.05. In the case of applying the heuristics explained above, the hyperplane $\{(x, y) \in \mathbb{R} : 2x - y = 0\}$ partitions $R$ into two new subregions in an analogous manner to the uniform case. These subregions determine two new dynamical polyhedra, $P_1$ and $P_2$ with 1 as diameter for both as shown in Figure 5.4b. Observe that the case of non uniform partition results in a more homogeneous distribution of the error between linear and polyhedral dynamics.
5.3. Experimental results

This section provides details regarding the implementation of the presented hybridization method, its integration into an abstraction based stability analysis approach for polyhedral hybrid systems, experimental set up, experimental evaluation and comparison with another stability verification tool based on Lyapunov function proofs.

The hybridization procedure is implemented in AVERIST. The experiments have been performed on a virtual machine with OS Ubuntu 14.04, processor equivalent to 4 virtual CPUs and 4.68 GB of memory, installed on Mac OS X 10.10 with processor 2.8 GHz Intel Core i5 and 8GB 1600 MHz DDR3 memory.

These experiments are performed, first, to evaluate the scalability of AVERIST,
in terms of state space dimension, number of modes in the input hybrid automaton and predicates involved in hybridization, and second, to compare the implemented approach with a stability verification tool based on searching for Lyapunov functions.

5.3.1 AVERIST evaluation

The evaluation is performed over two kinds of input linear hybrid systems, those with arbitrary switching represented as $AS_i$ and those with state based switching represented as $SS_i j$. The $AS_i$ examples are chosen from [PP03] (Example 2) and the $SS_i j$ examples are constructed from $AS_i$ examples by restricting the switchings to happen at the boundaries of a partition with $j$ regions. In particular, the set of executions of $SS_8 i$ is a subset of the executions of $SS_4 i$ which is a subset of the executions of $AS_i$. The $4$ and $8$ refer to the number of locations in the hybrid system.

The initial linear predicates used in partitioning the state space for hybridization are taken to be the linear expressions extracted from the input system definition, the ones delimiting the invariants and guards. AVERIST runs iteratively with increasing number of linear predicates. We have implemented a function to choose the subsequent predicates, such that these new predicates split the state space into uniform-sized chunks. For instance, for a two dimensional system, in addition to the constraints in the hybrid system, after the first iteration the predicates $x = 0$ and $y = 0$ are added, after the second iteration $x = y$ and $x = -y$ are added and so on.

A summary of experimental results is presented in Table 5.1. Every experiment has been run with time-out of 25 minutes. The first 6 columns consist of the AVERIST experiments. In addition to the dimension (number of continuous variables) and the experiment name, the table reports the number of iterations of the hybridization procedure that were run until stability was proven or the tool timed out, the number of linear expressions and the regions in the corresponding partition in the final hybridization, and the total time taken to infer Lyapunov stability. The total time includes the time taken by the underlying quantitative predicate abstraction technique to analyse the polyhedral hybrid system as well.

The experimental results in Table 5.1 demonstrate the feasibility of our approach for stability analysis of linear hybrid systems. Note that AVERIST was able to prove stability on almost all the 2D, 3D and 4D considered examples. The time for analysis increases reasonably slowly with the increase in the number of modes, for instance, consider the 4D systems $SS_4_7$ and $SS_8_7$ with 4 and 8
modes and with similar dynamics. However, the time increase with respect to increase in dimensions is steeper. This is expected because of the large number of regions in the partition as the dimension increases. Compositional analysis methods need to be explored for dealing with the curse of dimensionality.

Also, note that since the running time depends on the number of regions, which in turn depends in an exponential manner on the number of linear expressions used in the partitioning, it is crucial to develop methods which are clever in the choice of the linear expressions. We will explore techniques which dynamically partition the state space in a non-uniform manner to circumvent this problem. Some ideas are presented in Section 5.2.

### 5.3.2 Comparison with Stabhyli

This section compares the method in AVERIST with Stabhyli [MT13], which is a tool for global stability verification based on Lyapunov function search that can handle the class of switched linear systems. Broadly, it searches iteratively with increasing degree of polynomial templates until it finds a solution. It only iterates over even degree polynomials, since the Lyapunov functions are required to be non-negative everywhere. In addition, it has the option for finding either common or multiple Lyapunov functions. We ran Stabhyli on our example set with both common or piecewise Lyapunov function options, in turn, and set a time out of 25 minutes as before. The experimental results of Stabhyli are summarized in the last three columns of Table 5.1. The second column from the end indicates whether the system proved stability (Y) or that it timed out (-) with the common Lyapunov function option. The third column from the end shows the maximum degree of the Lyapunov function in the common Lyapunov function case that Stabhyli successfully analyzed before the time out in the case of (-), or the degree of the polynomial Lyapunov function returned in the case of (Y). The last column shows the total time taken for solving the templates up to the degree indicated in the first column. AVERIST was not able to prove stability on any of the examples with the multiple Lyapunov function option, hence, the table does not report the details.

First, note that Stabhyli fails to prove stability on several examples, on which AVERIST succeeded. On AS_9, Stabhyli succeeded, where as AVERIST timed out. While we do not claim that the set of benchmarks here is representative, we observed several numerical issues with Stabhyli with respect to which we claim that our method and tool are well behaved. For instance, in AS_1, Stabhyli returned with a 6th degree Lyapunov function. We ran Stabhyli again with the same example by fixing the degree of the polynomial to be 8,
and it returned with an answer that stability could not be proved. This is surprising, and we expect it to be a result of numerical issues which cannot set the coefficient of degree 8 term in the polynomial to 0 exactly. Note that this implies that the iteration over the templates needs to be done in a linear fashion. Next, while it found a common Lyapunov function, it did not succeed in finding a multiple Lyapunov function (note that the same Lyapunov function in each mode is a multiple Lyapunov function). In addition, on the systems SS$_4$ and SS$_8$, Stabhyli did not succeed in proving stability which consist of fewer executions than AS$_1$.

On the examples, where Stabhyli finds a Lyapunov function, it does so quickly. On all examples, except SS$_4$-3, Stabhyli performs better. However, the time required in checking whether a template is a Lyapunov function, increases much faster with the increase in the degree, as compared to, for instance, with the increase in the number of predicates for AVERIST. For instance, in the 4D example AS$_7$, it took 12 seconds to check the polynomial of degree 2, but timed out for the polynomial with degree 4.

### 5.4 Summary

This chapter has proposed a new hybridization approach for stability verification of linear hybrid automata. The main theoretical result refers to completeness of the hybridization technique for the class of asymptotically stable switched linear systems that are uniformly converging in time and Lipschitz continuous. Furthermore, the experimental results show the practical feasibility of such an approach and exhibit the merits in comparison to existing approaches.

Next chapter extends the CEGAR and hybridization approaches to a stronger notion of stability, global asymptotic stability, which states that every execution of a system starting at any state will converge to an equilibrium point.
6

GLOBAL ASYMPTOTIC STABILITY

This chapter extends the algorithmic approach presented in previous chapters to a stronger notion of stability, namely global asymptotic stability (GAS), and illustrates the GAS verification for a cruise control interacting with an automatic gearbox. These results are published in “An algorithmic approach to global asymptotic stability verification of hybrid systems”, EMSOFT’16 [PS16a]

Contents

6.1 GAS verification .............................. 84
6.2 AS verification ............................ 88
6.3 Stability zone .............................. 90
6.4 RS verification ............................ 91
6.5 Extension to other dynamics ............ 92
6.6 An automatic gearbox ................... 93
   6.6.1 Model .................................. 93
   6.6.2 Analysis .............................. 95
6.7 Summary .................................. 97
This chapter presents a novel algorithmic approach to global asymptotic stability (GAS) verification of hybrid systems. The broad approach consists of reducing the GAS verification to the verification of a region stability (RS) analysis problem and an asymptotic stability (AS) analysis problem. The chapter develops the theory for polyhedral hybrid systems but the broad approach of decomposing GAS analysis to RS and AS analysis can be applied to more general class of systems including linear hybrid systems.

Global asymptotic stability captures the property that a system behaviour starting at any state of the system will converge to a desired state and can be maintained there under small perturbations. For instance, one expects a cruise control to drive the vehicle velocity to a set velocity and also to maintain the velocity at the set velocity in the presence of disturbances that arise due to uphills and downhills on the road.

We build up on the algorithmic approaches presented in the previous chapters, to obtain a new algorithmic method for GAS verification. The main contribution consists of a decomposition theorem that reduces the GAS verification problem into a AS verification problem and a RS verification problem. More precisely, the GAS verification of a polyhedral hybrid system $\mathcal{H}$, is reduced to checking

- the asymptotic stability of a system $\mathcal{H}'$ that captures the dynamics of $\mathcal{H}$ close to the origin, and

- the region stability of $\mathcal{H}$ with respect to a zone $Z$ which ensures that the execution starting from $Z$ remains within the centre of $\mathcal{H}$.

The first step is verified using the QPA method introduced in Chapter 3. The weighted graph which exhibits the proof of AS of $\mathcal{H}'$ is used to compute the stability zone $Z$. A new algorithm for RS verification is provided. Except for the use of QPA for AS verification, all the other algorithms in the chapter are new including the computation of $Z$ and the integration of RS and AS verification procedures. As a proof of concept, we apply the GAS verification algorithm to a linear hybrid system model of a cruise control for an automatic gearbox to analyse the convergence of the vehicle velocity to a set velocity.

### 6.1 GAS verification

This section presents the algorithm for global asymptotic stability verification, where the main result is a decomposition theorem that states that global asymptotic stability verification can be reduced to an asymptotic stability verification problem and a region stability verification problem.
We cannot conclude the GAS of the system in Figure 6.1a by showing that it is asymptotically stable and it is region stable with respect to the region $\mathcal{R}$. This is because asymptotic stability only ensures convergence of executions starting close to the origin, and the executions that enter the region $\mathcal{R}$ from outside may not reach the points close to the origin, and hence, again leave the region. For instance, in Figure 6.1a, if we start at $\left(3M/2, 0\right)$, we reach $(M, M/2)$ by following $F_3$, and further, reach $(3M/4, M)$ by following $F_8$, then potentially switch to $q_2$ and reach $q_1$ outside of $\mathcal{R}$.

However, if it is possible to construct a zone $\mathcal{Z}$ within $\mathcal{R}$ from which all the executions are guaranteed to remain within $\mathcal{R}$, and to have previously established asymptotic stability, then one can conclude that the executions starting from $\mathcal{Z}$ also converge to the origin. Here, we use downward closedness
of the executions of the polyhedral switched systems. The intuition is that every execution starting at point $z$ in $\mathcal{Z}$ is similar to an execution that starts at $z/\alpha$, for any $\alpha \geq 1$. Note that it is crucial that the execution from $z$ remains within $\mathcal{Z}$. The zone $\mathcal{Z}$ is called a stability zone. Global asymptotic stability can then be deduced by showing region stability with respect to $\mathcal{Z}$. Theorem 9 captures this intuition. Below, we formalize the concepts required in stating the theorem. Let us fix a polyhedral switched system $\mathcal{H} = (Q, E, X, F, I, G)$ for the rest of the section.

A centre zone consists of a set of states of the system adjacent to the origin which contain downward closed subsets of invariants and guards. A set $S \subseteq Q \times X$ is downward closed if for every $(q, x) \in S$, $(q, \alpha x) \in S$ for every $0 < \alpha < 1$.

**Definition 28.** A centre zone of $\mathcal{H}$ is a downward closed set $\mathcal{R}$ such that

$$\mathcal{R} \subseteq \{(q, x) \in Q \times X : x \in \overline{I(q)}, \bar{0} \in \overline{I(q)}\}$$

and for every $(q, x), (q', x') \in \mathcal{R}$ with $(q, q') \in E$, $G(q, q')$ is downward closed.

Note that if $\bar{0} \in \overline{I(q)}$, then $I(q)$ is necessarily a downward closed set. In Figure 6.1a, the region $\mathcal{R}$ given by $q_7, q_8, q_9$ and $q_{10}$ is a centre zone. In the sequel, we use the symbol $\mathcal{R}$ to always refer to the centre zone.

**Definition 29.** Given a centre zone $\mathcal{R}$, a set $Z \subseteq \mathcal{R}$ is called a stability zone of $\mathcal{H}$ with respect to $\mathcal{R}$ if for every $\sigma \in \text{cexec}(\mathcal{H})$ such that $\eta(0) \in Z$, $\eta(t) \in \mathcal{R}$ for every $t \geq 0$.

**Theorem 9.** Let $\mathcal{H}$ be a pss, $\mathcal{R}$ be a centre zone of $\mathcal{H}$ and $Z$ be a corresponding stability zone of $\mathcal{H}$. If $\mathcal{H}$ is asymptotically stable with respect to $\bar{0}$ and $\mathcal{H}$ is region stable with respect to $Z$, then $\mathcal{H}$ is globally asymptotically stable with respect to $\bar{0}$.

**Proof.** Since $\mathcal{H}$ is asymptotically stable, to show that $\mathcal{H}$ is GAS, we need to show that every complete execution of $\mathcal{H}$ converges to the origin. However, since, all complete executions are maximal, by region stability with respect to $Z$, they reach $Z$. Hence, it remains to show that all complete executions that start from $Z$ converge to the origin. From asymptotic stability of $\mathcal{H}$, it is known that there is some $\delta > 0$, such that all executions that start in $B_\delta(\bar{0})$ converge to the origin. Consider an execution $\sigma$ that starts in $Z$, which implies that $\sigma$ always remains in $\mathcal{R}$. There is some $0 < \alpha < 1$ such that $\alpha \sigma$ starts in $B_\delta(\bar{0})$. Note that $\alpha \sigma$ is an execution of $\mathcal{H}$, since, it satisfies the corresponding invariants and guards, which follows from the fact that the invariants and guards restricted to $\mathcal{R}$ are downward closed sets. Since $\alpha \sigma$ converges to the origin, $\sigma$ also converges to the origin. $\blacksquare$
The technique to analyse GAS of $\mathcal{H}$ with respect to the origin is described in Algorithm 4. Given a pss $\mathcal{H}$ the GAS verification process returns either a “yes”, “no” or “non-established” answer. A “yes” answer implies GAS of the system $\mathcal{H}$, a “no” answer implies instability and a “non-established” answer means that the system $\mathcal{H}$ could not be deduced to be either GAS or otherwise, and hence, may or may not be stable. This verification process consists of the following main steps.

First, the asymptotic stability of $\mathcal{H}$ is analysed by calling the function $\text{as-verification}(\mathcal{H})$. It returns a “yes”, “no” or “non-established” answer. A “yes” answer means that AS was deduced, and in this case a weighted graph $\mathcal{G}$ that serves as the proof of AS is returned. A “no” answer states that $\mathcal{H}$ is not asymptotically stable and “non-established” answer implies that AS or otherwise could not be deduced for the system $\mathcal{H}$. These two last answers are accompanied by an empty graph $\mathcal{G}$. In the case that the asymptotic stability of the system $\mathcal{H}$ is deduced, the algorithm proceeds to construct the centre zone $\mathcal{R}$, a stability zone $\mathcal{Z}$ and to verify the region stability of $\mathcal{H}$ with respect to the stability zone $\mathcal{Z}$. A “no” or “non-established” answer to AS verification of $\mathcal{H}$ results in the same answer for GAS of $\mathcal{H}$, since global asymptotic stability requires asymptotic stability.

The centre zone of $\mathcal{H}$ is computed by the function $\text{Centre}(\mathcal{H})$ based on Definition 28. Our stability zone computation algorithm takes as input the centre zone $\mathcal{R}$ as well as the graph $\mathcal{G}$ returned from the asymptotic stability analysis.

Region stability verification is performed by the function $\text{rs-verification}(\mathcal{H}, \mathcal{Z})$ over the system $\mathcal{H}$ by considering the stability zone $\mathcal{Z}$. Again it returns either “yes”, “no” or “non-established”. If region stability of $\mathcal{H}$ is established, then $\mathcal{H}$ is concluded to be GAS in accordance with Theorem 9. If the output to region stability analysis is either “no” or “non-established”, then in general we cannot conclude that GAS cannot hold. Firstly, we do not have a converse of Theorem 9, and secondly, the region stability could be violated by the existence of maximal finite executions that do not reach the zone, but which can be
safely ignored for GAS deduction. Next, we explain the different procedures in detail.

6.2 AS verification

This section describes the asymptotic stability verification method for polyhedral switched systems based on the QPA. It is important to understand this construction, since the implementation of our integration algorithm (Theorem 9) relies crucially on the foundations of this construction.

Observe that since asymptotic stability is a local property, it suffices to restrict the system to a small neighbourhood around the origin for the purpose of asymptotic stability analysis. Hence, we define a PSS extracted from $\mathcal{H}$ that uses the dynamics in a small neighbourhood for the whole state space.

**Definition 30.** Given a PSS $\mathcal{H}$ and a centre zone $\mathcal{R}$ of $\mathcal{H}$, the extracted PSS from $\mathcal{H}$ is a hybrid automaton $\mathcal{H}^e = (\text{Loc}^e, \text{Edges}^e, X, \text{Flow}^e, \text{Inv}^e, \text{Guard}^e)$ such that

- $\text{Loc}^e = \{q \in Q : \mathcal{R} \cap (q \times I(q)) \neq \emptyset\}$,
- $\text{Edges}^e = \{(q_1, q_2) \in E : q_1, q_2 \in \text{Loc}^e\}$,
- $\text{Flow}^e = F|_{\text{Loc}^e}$,
- $\text{Inv}^e : \text{Loc}^e \rightarrow \text{polysets}(X)$ with $\text{Inv}^e(q) = \text{cone}(I(q))$,
- $\text{Guard}^e = G|_{\text{Edges}^e}$.

Note that the extracted PSS $\mathcal{H}^e$ does not depend on the centre zone $\mathcal{R}$, since the set $\text{Loc}^e$ will be the same for every centre zone $\mathcal{R}$ of $\mathcal{H}$.

**Proposition 14.** Given a PSS $\mathcal{H}$ and the extracted PSS $\mathcal{H}^e$ from $\mathcal{H}$, $\mathcal{H}$ is asymptotically stable if and only if $\mathcal{H}^e$ is asymptotically stable.

Proposition 14 states that it is enough to analyse AS of the extracted PSS $\mathcal{H}^e$. The full process for AS analysis of system $\mathcal{H}$ is described in Algorithm 5. First, the extracted PSS from $\mathcal{H}$ based on Definition 30 is constructed by the function $\text{Extract}(\mathcal{H})$. It requires the computation of $\text{cone}(P)$ for a polyhedral set $P$. It is computed as follows:

$$\text{cone}(P) = \begin{cases} \bigcap_{p_i(x)\sim 0 \in 1c(P)} \{x \in \mathbb{R}^n : p_i(x) \sim 0\} & \text{if } \bar{0} \in \partial(P) \\ \mathbb{R}^n & \text{if } \bar{0} \in \hat{P} \end{cases}$$
Algorithm 5 Asymptotic stability verification

Require: $\mathcal{H}$

Ensure: AS

1: $\mathcal{H}' := \text{Extract}(\mathcal{H})$
2: $\mathcal{P} := \text{Generate-partition}(\mathcal{H}')$
3: $\mathcal{G} := \text{Abstraction}(\mathcal{H}', \mathcal{P})$
4: return $\text{AS-satisfaction}(\mathcal{G})$

Essentially, $\text{cone}(\mathcal{P})$ is represented by the constraints in $\text{lc}(\mathcal{P})$ that contain $\bar{0}$ in their closure. Note that every $\mathcal{P}$ corresponds to an invariant whose closure contains the origin by Definition 30.

Next, it is required to analyse the asymptotic stability of PSS $\mathcal{H}^e$. The broad approach is to construct an abstraction (a simplification) such that certain analysis on the abstract system yields stability of $\mathcal{H}^e$. However, standard finite state abstractions do not preserve stability and hence, the QPA that construct finite weighted graphs has been proposed in Chapter 3. The system is deduced to be asymptotically stable, if the following two properties hold for the weighted graph $\mathcal{G}$ abstracting the PSS $\mathcal{H}^e$:

$A1$ $\mathbf{w}(e) < +\infty$ for every edge $e$ in $\mathcal{G}$.

$A2$ Every simple cycle $\pi$ of $\mathcal{G}$ satisfies $\mathbf{w}(\pi) < 1$.

Condition $A2$ ensures that the executions that cross infinitely many facets converge to the origin, since, the scalings associated with such executions is provided by the corresponding paths in the graph. Condition $A1$ ensures that the executions do not diverge while remaining within a single region of the partition.

In particular, the product of the weights on a path provide an upper bound on the ratio of the norm of the end to that of the starting point of all executions that follow the sequence of facets specified by the path, as seen in Chapter 3.

Hence, the algorithm proceeds with the generation of the partition using the function $\text{Generate-partition}(\mathcal{H}')$. The function either automatically generates the partition from the PSS $\mathcal{H}'$ or seeks the user’s input in computing the partition. The user’s input can be taken in the form of a set of linear expressions that are then used to construct the partition. A useful set of linear expressions to add either automatically or manually are those that define the invariants and the guards in $\mathcal{H}'$. The PSS $\mathcal{H}'$ and the state space partition $\mathcal{P}$ are the input to the function $\text{Abstraction}(\mathcal{H}', \mathcal{P})$, which outputs the weighted graph abstraction $\mathcal{G}$. The function $\text{AS-satisfaction}(\mathcal{G})$ verifies that the Conditions $A1$ and $A2$ hold. The function $\text{AS-satisfaction}(\mathcal{G})$ returns $\text{AS}$ of $\mathcal{H}'$ if both $A1$ and $A2$ are
satisfied. If $A_1$ is violated, the algorithm will output instability, and if $A_1$ is satisfied but $A_2$ is not satisfied, the output will be a “do not know” answer.

Figure 6.3 shows the quantitative predicate abstraction over the extracted PSS from the one in Figure 6.3a. It is a simplified version, just to give a flavour of the construction procedure. Observe that every execution starting from $f_4$ reaching $f_1$ by following the vector associated with $q_8$ has as maximum scaling $2/1$, that is, 2. The other weights are computed in an analogous manner.

### 6.3 Stability zone

This section introduces the technique to construct a stability zone when given a weighted graph $G$ abstracting the system $H^e$, and a centre zone $R$. Let $R$ be a ball of radius $M$ for some $M$ (note that when we use the infinity norm, a ball provides a rectangular set). The detailed algorithm is given in Algorithm 6, in which the stability zone is defined as a scaled version of the centre zone $R$.

**Algorithm 6** Stability zone construction

<table>
<thead>
<tr>
<th>Require: $G, R$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ensure: $Z$</td>
</tr>
<tr>
<td>1: $m := \text{Maximum-scaling}(G, R)$</td>
</tr>
<tr>
<td>2: $m' := 2 \cdot m$</td>
</tr>
<tr>
<td>3: return $\frac{1}{m'} R$</td>
</tr>
</tbody>
</table>

First, the maximum weight of a path in $G$ is computed with the function $\text{Maximum-scaling}(G, R)$, which returns $\max\{1, \max\{w(\pi) : \pi \text{ is a path in } G\}\}$. 
This value represents an upper bound on the variation between the distances to the origin of a pair of states belonging to the same execution of the system $H^e$ whose abstraction is represented by $G$. Denote this bound as $m$ and observe that an execution in $H^e$ starting at a distance $d$ from $\bar{0}$ can only reach as far as $md$ from $\bar{0}$. The stability zone $Z$ is taken to be $\frac{1}{m'}R$ for some $m' > m$. Algorithm 6 considers $m'$ to be $2m$. Observe that in Figure 6.1 the stability zone $Z$ coincides with a scaled version of $R$.

6.4 RS verification

This section introduces the novel abstraction based algorithm for region stability analysis. Given a pss $H$ and a stability zone $Z$, region stability of $H$ with respect to $Z$ is verified. An algorithmic procedure for rs verification is shown in Algorithm 7. It consists of the following steps.

Initially, a state partition $P$ is generated as explained in the previous section, by the function $\text{Generate-partition}(H)$. This partition $P$ is refined into a partition $\text{Partition}(P, Z)$ using the stability zone $Z$. If $P = \{P_1, \ldots, P_k\}$, then $P' = \text{Partition}(P, Z)$ is defined as $\{P_1 \setminus Z, \ldots, P_k \setminus Z, P_1 \cap Z, \ldots, P_k \cap Z\}$. The new partition ensures that the zone $Z$ is a union of certain regions of the partition, and hence, the boundary of $Z$ can be captured by the facets of the partition. Let us call these facets $F_Z$.

The pss $H$ and the state space partition $P'$ are input to $\text{Abstraction}(H, P')$, which constructs a weighted graph $G_1$ by applying the abstraction over the system $H$ explained in Section 6.2. This is slightly modified to capture those executions which remain within a region (may not necessarily diverge), such executions are captured using infinite weight edges as well. In order to establish region stability, we need to show that all executions that start outside the zone $Z$ will eventually reach its boundary. Hence, we can ignore the subgraph that corresponds to the interior of $Z$. This is achieved by the function $\text{Delete-inner-nodes}(G_1, Z)$ that deletes all nodes that correspond to the interior of $Z$ and the edges out of them, and outputs $G_2$. Then we remove all nodes that are not reachable from the initial nodes which capture all the points from which we want to reach $Z$. $\text{Delete-non-reachable-nodes}(G_2, N_1)$ removes all non-reachable nodes resulting in the graph $G_3$. From the modified graph construction, this also detects any executions that remain in a region and do not move to the next facet. Next, we ensure that there are no executions that diverge while remaining in a single region, this is ensured by checking that there are no edges with infinite weight in $G_3$. Note that a cycle can represent an infinite execution that follows the cycle infinitely many times and hence,
Algorithm 7 Region stability verification

Require: $\mathcal{H}, \mathcal{Z}$
Ensure: $\mathcal{R}_s$
1: $\mathcal{P} := \text{Generate-partition}(\mathcal{H})$
2: $\mathcal{P}' := \text{Partition}(\mathcal{P}, \mathcal{Z})$
3: $\mathcal{G}_1 := \text{Abstraction}(\mathcal{H}, \mathcal{P}')$
4: $\mathcal{G}_2 := \text{Delete-inner-nodes}(\mathcal{G}_1)$
5: $\mathcal{G}_3 := \text{Delete-non-reachable-nodes}(\mathcal{G}_2)$
6: if edge in $\mathcal{G}_3$ with $w(\text{edge}) = \infty$ then
7: return false
8: $\mathcal{C} := \text{Cycles}(\mathcal{G}_3)$
9: if $\mathcal{C}$ not empty then
10: return non-established
11: $\mathcal{N}_2 := \text{Not-outgoing-edges}(\mathcal{G}_3)$
12: $\mathcal{N}_3 := \text{Nodes}(\mathcal{G}_3, \text{Facets}(\mathcal{Z}))$
13: if node in $\mathcal{N}_2$ and not in $\mathcal{N}_3$ then
14: return non-established
15: return true

does not reach $\mathcal{Z}$. The function $\text{Cycles}(\mathcal{G}_3)$ returns the simple cycles in the graph; if it is empty, the analysis will proceed. Finally, it is needed to ensure that none of the executions get stuck (terminate) at any of facets. The function $\text{Not-outgoing-edges}(\mathcal{G})$ computes all the nodes which have outgoing edges. The only nodes that are allowed not to be in this set are those corresponding to $\mathcal{Z}_F$.

6.5 Extension to other dynamics

This chapter is focused on the analysis of global asymptotic stability of polyhedral switched systems. Theorem 9 provides a decomposition theorem for GAS verification in terms of asymptotic stability and region stability verification. It uses certain properties about the polyhedral inclusion dynamics, namely, that the solutions are scaling invariant in the sense that if $\sigma$ is a solution of a polyhedral inclusion dynamics specified by the polyhedron $P$, then $\alpha \sigma$ is a solution as well. A similar property can be established for linear dynamics, however, here, in the definition of $\alpha \sigma$ we do not need to scale the time. More precisely, if $x(t)$ is a solution of $\dot{x} = Ax$, then $y(t)$ given by $\alpha x(t)$ is also its solution. Hence, the decomposition theorem can be extended to linear hybrid systems. However, such properties do not hold for general non-linear dynamics.
6.6. An automatic gearbox

An alternate approach to deal with the general class of hybrid systems is to use hybridization to reduce them to polyhedral hybrid systems [PBV95, ADG07, DMT10]. Hybridization constructs a partition of the state space into a finite number of regions and over-approximates a given dynamics by a simpler one. Hybridization results in an over-approximate system and hence, it suffices to establish the gas of the over-approximate system to deduce the same for the given system. The hybridization approach is used in the sequel to analyse a linear hybrid system modelling an automatic gearbox. It will be constructed a polyhedral switched system that over-approximates the linear hybrid system and analysed the pss for gas.

6.6 An automatic gearbox

This section presents an automatic gearbox example and analyses global asymptotic stability. The gearbox example is a linear hybrid automaton. We construct a polyhedral hybrid automaton and then apply the developed gas verification algorithm on that. The overall process is illustrated through the automatic gearbox model presented in [PL00].

6.6.1 Model

An automatic gearbox is a device for controlling the gear changes in a vehicle. It consists of four different positions and at each position the dynamics is governed by a linear system. The dynamics describe the evolution of two variables, $v$ and $T_I$, where $v$ is the difference between the desired velocity $v_d$ and the current velocity $v$ and $T_I$ corresponds to the integrator torque state $\Delta T_I$. Every switching is performed when some constraint over the velocity is satisfied. The target is to achieve a desired velocity $v_d$, for which we need to verify global asymptotic stability of the system with respect to $v = 0$ and $T_I = 0$. The value $T_I$ is set to 0 every time the driver inputs a new desired velocity.

The linear switched system modelling the system is shown in Figure 6.4, where the state is defined by $x = (v, T_I)^\top$ and the closed-loop dynamics are determined at each gear by the matrix

$$A_q = \begin{pmatrix} -p_q k_q / M & -p_q / M \\ k_q / T & 0 \end{pmatrix}$$
with $q$ referring to each of the modes $q_1, q_2, q_3$ and $q_4$. The invariant intervals are $I_1 = [v_d - \frac{1}{p_1}\omega_{\text{high}}, \infty)$, $I_2 = [v_d - \frac{1}{p_2}\omega_{\text{low}}, v_d - \frac{1}{p_2}\omega_{\text{high}}]$, $I_3 = [v_d - \frac{1}{p_3}\omega_{\text{low}}, v_d - \frac{1}{p_3}\omega_{\text{high}}]$ and $I_4 = (-\infty, v_d - \frac{1}{p_4}\omega_{\text{low}}]$. The values for each parameter in the linear switched system definition appear in Table 6.1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M$</td>
<td>1500</td>
</tr>
<tr>
<td>$T$</td>
<td>40</td>
</tr>
<tr>
<td>$\omega_{\text{low}}$</td>
<td>500</td>
</tr>
<tr>
<td>$\omega_{\text{high}}$</td>
<td>230</td>
</tr>
<tr>
<td>$p_1$</td>
<td>50</td>
</tr>
<tr>
<td>$p_2$</td>
<td>32</td>
</tr>
<tr>
<td>$p_3$</td>
<td>20</td>
</tr>
<tr>
<td>$p_4$</td>
<td>14</td>
</tr>
<tr>
<td>$k_1$</td>
<td>$15/4$</td>
</tr>
<tr>
<td>$k_2$</td>
<td>$375/64$</td>
</tr>
<tr>
<td>$k_3$</td>
<td>$75/8$</td>
</tr>
<tr>
<td>$k_4$</td>
<td>$375/28$</td>
</tr>
</tbody>
</table>

Table 6.1: Parameter values

Figure 6.5 shows some representative sample executions of the automatic gearbox for $v_d = 30$. The horizontal axis represents the velocity difference $v$ and the vertical axis represents the integrator torque $T_I$. Since the value $T_I$ is set to 0 every time the driver inputs a new desired velocity, every execution starts at the horizontal axis. The initial state depends on the current velocity of the vehicle. The origin is the state such that $v_d = v$, that is, the state where the target velocity is achieved. An execution starting from the negative side of the axis, that is $v < 0$, means that the vehicle needs to slow down, because the current velocity is higher than $v_d$. The gear is in the 4th position, which corresponds to mode $q_4$. Observe that the execution evolves continuously in location $q_4$ until reaching the origin. In case of starting from a positive value of $v$, the current velocity is lower than $v_d$ and the gear position will depend on this value. We depicted two different executions, one starting from location $q_1$ and another one starting from location $q_2$. The execution starting from $q_1$ switches three times. The current velocity increases continuously in $q_1$ until reaching the value $\frac{1}{p_1}\omega_{\text{high}}$ (involved in the guard constraint from $q_1$ to $q_2$ in Figure 6.4), represented by the rightmost solid red line. Then, it switches from location $q_1$ to location $q_2$. The switching implies a dynamics change but no jump on the state. The execution evolves by following the new dynamics until reaching a new value $\frac{1}{p_2}\omega_{\text{high}}$ for the current velocity and it switches from location $q_2$ to location $q_3$. The next switching is from location $q_3$ to $q_4$, where the execution converges to the origin. An analogous behaviour is observed in
6.6. An automatic gearbox

the case of the execution starting from location \( q_2 \).

6.6.2 Analysis

Consider the gearbox hybrid automaton in Figure 6.4 where the cruise controller sets the desired velocity \( v_d \) to be 30. Let the corresponding linear switched system be \( \mathcal{H}_{gb} \). Next, this section enumerates the different GAS verification steps performed for the linear switched system \( \mathcal{H}_{gb} \).

**Step 1.**- All the constants in the system description of \( \mathcal{H}_{gb} \) are transformed into integers while maintaining the system behaviour. This is necessary since the different used tools require such property. Every polyhedral set in PPL needs to be defined by linear expressions with integer parameters. Therefore, all the constant values in \( \mathcal{H}_{gb} \) are expressed as rational numbers and after multiplied by 1344000, which is the least common multiple of all the denominator values.

**Step 2.**- A polyhedral switched system \( \mathcal{H}_1 \) is constructed by applying the hybridization technique in Chapter 5 over the linear switched system \( \mathcal{H}_{gb} \). The hybridization technique returns in addition to \( \mathcal{H}_1 \), the polyhedral state space partition used for construction. Denote the state space partition as \( \mathcal{P} \). It is the one shown in Figure 6.5, where the state space is divided by the linear constraints which describe the invariants of the original gearbox, the linear constraint \( T_I = 0 \) which defines the initial states and the linear equation \( 375E + 28T_I = 0 \) which equalizes to zero the velocity coordinate of the flow vectors defining executions close to the origin.

**Step 3.**- From the pss \( \mathcal{H}_1 \) is extracted a pss \( \mathcal{H}_2 \) which consists of only one location, where the invariant is the full planar continuous space and the flow is defined by the polyhedral approximation of the linear dynamics in the \( q_4 \) mode of the original system. Then asymptotic stability proof reduces to that of a single mode system.

**Step 4.**- Asymptotic stability of \( \mathcal{H}_2 \) is analysed by using the tool AVERIST. The hybrid system \( \mathcal{H}_2 \) and the state space partition \( \mathcal{P} \) are the inputs to the software, which returns an answer and a weighted abstract graph \( \mathcal{G} \). It is obtained a positive answer about asymptotic stability. The abstraction process results in the weighted graph shown in Figure 6.6, where the weights appear truncated for simplicity.

Since the asymptotic stability proof consists of a proof for a single mode system, the asymptotic stability could have been verified by eigenvalue analysis over the linear dynamics in \( q_4 \), so over a linear version of the extracted \( \mathcal{H}_2 \) shown in the left side of Figure 6.6, but we require extra information for the stability
zone construction which is provided by the abstract weighted graph $\mathcal{G}$, shown on the right in Figure 6.6.

**Step 5.** The stability zone $\mathcal{Z}$ is computed. The centre zone is $q_4 \times (\langle -\infty, 18, 240, 000 \rangle \times \mathbb{R})$ that is shown in Figure 6.5. The extracted system that is used in the asymptotic stability analysis and subsequently in the computation of $\mathcal{Z}$, is depicted on the left in Figure 6.6. The abstract weighted graph $\mathcal{G}$ is shown on the right in Figure 6.6. Observe that in Figure 6.6 the maximum value of any path in the graph is in fact the value of the edge with weight 2.678. Denote this maximum value as $m$. Then, the stability zone defined as in Algorithm 6 corresponds to $\alpha(\langle -\infty, 18, 240, 000 \rangle \times \mathbb{R})$ where $\alpha \simeq 0.1867$. We are going to be more restrictive and instead of considering such stability region, we construct a smaller one, contained in the previous one. This new region is a ball centred on the origin with radius $1,250,000$ (this is a not automatic choice), which is less than $\alpha \cdot 18,240,000 = 3,405,408$. Let us denote this ball as $\mathcal{Z}$. The predicates defining the ball $\mathcal{Z}$ are added manually in the implementation.

**Step 6.** A weighted graph is automatically constructed to verify region stability. A new partition $\mathcal{P}'$ is automatically constructed by intersecting the polyhedral...
6.7. Summary

This chapter presented an algorithmic approach to global asymptotic stability verification that relies on a decomposition result. This principle reduces the global asymptotic stability problem into a local asymptotic stability problem and a region stability problem. The algorithm has been applied to a case study involving a cruise control which interacts with an automatic gearbox.

Next chapter presents a software tool which implements the theoretical results developed in previous chapters and integrates the stability verification techniques presented so far.
This chapter introduces the software tool AVERIST which implements the algorithmic approach for stability verification of linear hybrid systems and the details of its implementation, which are partially published in “AVERIST: An Algorithmic Verifier for Stability”, Electr. Notes Theor. Comput. Sci.’15 [PS15]

Contents

7.1 AVERIST architecture and data structures .... 100
7.2 Detailed flowchart .................................. 104
  7.2.1 CEGAR for PHS .............................. 105
  7.2.2 CEGAR for LHS .............................. 105
7.3 Summary ........................................... 106
This chapter presents the architecture and implementation of the tool AVERIST that performs stability verification for linear hybrid systems. The tool implements the hybridization method for approximating linear hybrid systems by polyhedral hybrid systems, introduced in Chapter 5. It also implements the novel CEGAR framework in Chapter 4 for analysing the polyhedral hybrid systems that are generated as a result of the hybridization. The main features of the tool are as follows:

- it is based on algorithmic techniques that do not rely on the computation of Lyapunov functions,
- it returns a counterexample when it fails to establish stability and
- it is less prone to numerical instability issues as compared to Lyapunov function based tools as is demonstrated by the experimental results in Section 5.3.2.

The next sections present a general description of AVERIST and describe how to use it.

### 7.1 AVERIST architecture and data structures

This section explains some of the main building blocks of the tool, which are shown in the software architecture diagram of AVERIST in Figure 7.1a. These blocks correspond to the hybridization, quantitative predicate abstraction, model checking, validation and refinement techniques. They are implemented in Python 2.7. A short overview provides the functionalities of these blocks along with some details of their implementation and the software packages they depend on. In particular, the software dependencies of AVERIST are the following:

- Parma Polyhedra Library (ppl) [BHZ08] provides routines for representation and manipulation of convex polyhedra as well as a solver for mixed integer linear programming problem, which are required for the hybridization, abstraction and refinement procedures.

- NetworkX [nx] is a Python package intended to create and manipulate graphs, which is required in the abstraction and model checking procedures. The PPL and NetworkX packages are included in the free open-source mathematics software system sage [S+16].
• The GLPK (GNU Linear Programming Kit) [glp] package contains functionalities for solving large-scale linear programming, mixed integer programming and other related problems, which are required in the abstraction construction.

• The Z3 software [MB08] is a high-performance theorem prover required in the validation procedure.

The main data structures are the base class DiGraph defined in the NetworkX package and the class NNC_Polyhedron in PPL which represent respectively a directed weighted graph and a not necessarily closed (NNC) convex polyhedron in the state space.

The next paragraphs provide a brief reminder of the main implemented blocks in the tool and highlight the use of the software dependencies.

**Hybridization** The hybridization module over-approximates the linear dynamics in every mode of a linear hybrid automaton by a set of polyhedral inclusion dynamics, as detailed in Chapter 5. More precisely, given a linear dynamics \( \dot{x} = Ax \), the continuous state space is divided into a set of polyhedral regions and an (over-approximated) polyhedral inclusion dynamics \( \dot{x} \in P \) is constructed for each region \( R \), where \( P \) is a polyhedral set obtained by collecting the vectors in the vector field restricted to \( R \), that is, \( P = \{Ax : x \in R\} \). The state space partition is constructed using a set of linear predicates. An edge is added between the adjacent regions of the partition, however, some of these edges are not necessary if the vector fields at the boundaries of these regions are such that they do not allow the continuous trajectory to evolve from one to the
other. The hybridization block uses the software package PPL to represent and manipulate polyhedral sets that arise in the state space partition, as well as, in the over-approximated polyhedral inclusion dynamics. The pruning of edges based on vector fields is done by solving some constraints using the Z3 SMT solver. This module essentially constructs a hybrid system with polyhedral inclusion dynamics (PHS) from a linear hybrid system (LHS).

**Abstraction** Quantitative predicate abstraction (QPA) module takes as input a PHS and outputs a weighted graph that over-approximates the behavior of the PHS. It is a modified version of the standard predicate abstraction [GS97, ADI06], wherein the nodes in the graphs correspond to certain facets of the boundaries of the regions in a state space partition, and the edges correspond to the existence of an execution between the facets corresponding to the nodes. In addition, each edge is associated with a weight that provides an upper bound on the scaling factor associated with the executions between the facets, where the scaling corresponds to the ratio of the distance to a fixed equilibrium point (typically the origin) at the end of the execution as compared to that at the beginning of the execution. The solutions of the polyhedral inclusion dynamics are linear trajectories, hence, the existence of an edge between two nodes (which correspond to polyhedral facets), can be encoded as a set of satisfiability problems with linear constraints. The computation of the scaling factor corresponds to solving an optimization over these linear constraints. Again, this block uses PPL to represent polyhedral sets and the NetworkX package to represent the graphs that arise in the description of the PHS as well as in the weighted graph constructed using QPA. The user has the choice of using either the GLPK linear programming solver or the MIP_problem class in PPL for the computation of edges and their weights. In addition, a method to prune redundant nodes and edges in the graph has been implemented. This method removes self-cycles in the graph as well as nodes \( v \) such that paths through \( v \) are covered by alternate edges.

**Model checking** The model checking module takes a weighted graph corresponding to a QPA as input and checks structural conditions corresponding to the existence of certain kinds of cycles to either deduce stability, or output a counterexample showing a potential reason for instability. The model checking algorithm uses the NetworkX package and consists of a modification of the Bellman-Ford algorithm [Bel58] for shortest path computation. In particular, the weight of a path in our problem is the product of all the weights on the edges in the path rather than their sum, as is the case in the standard setting of shortest path computation. In particular, if there is no cycle \( \pi \) with weight...
7.1. Averist architecture and data structures

If \( w(\pi) \) greater than one, then the concrete system is stable. On the contrary, a cycle with weight greater than one represents an abstract counterexample.

**Validation** The validation module takes as input an abstract counterexample from the weighted graph analysis of a QPA, and checks if it corresponds to an actual execution that exhibits instability. In particular, the analysis determines if the abstract counterexample \( \pi \) is spurious or on the contrary corresponds to an infinite divergent execution in the concrete hybrid system. The counterexample \( \pi \) consists of a finite set of nodes connected by edges in the abstract weighted graph, namely, \( R_0, R_1, \ldots, R_k \). The algorithm in Chapter 4 reduces the problem of checking the existence of an infinite execution that cycles through the facets \( R_0, R_1, \ldots, R_k \) infinitely and diverges, to just checking a condition \( \psi_\pi(m) \) that encodes the existence of an execution which cycles through the \( k \) regions only once, that is, it encodes a sequence of states \( s_0, \ldots, s_k \) such that \( s_k \) can be reached from \( s_0 \) by following the cycle \( \pi \) once, and \( s_k = ms_0 \).

\[
\psi_\pi(m) := \exists x_0, x_1, \ldots, x_k \in \mathbb{R}^n : s_k = ms_0, \forall 0 \leq i < k, s_i \in R_i, (s_i, s_{i+1}) \in \text{reach}_{R_i, R_i+1} \text{ for some } R.
\]

In particular, if there is no \( m \geq 0 \) such that \( \psi_\pi(m) \) is satisfied, then we deduce that in fact there is no infinite execution corresponding to \( \pi \) (not even a converging one). If there exists some \( m \) such that \( \psi_\pi(m) \) is satisfied, but no \( m > 1 \) such that \( \psi_\pi(m) \) is satisfied, then we deduce that there is no infinite diverging execution corresponding to \( \pi \), but there could be a non-diverging execution. The above two cases correspond to a spurious abstract counterexample. If there is an \( m > 1 \) such that \( \psi_\pi(m) \) is satisfied, then we deduce that there is an infinite diverging execution corresponding to \( \pi \), and hence, the PHS which is being analysed is unstable. The formulae corresponding to \( \psi_\pi(m) \) are linear except for the condition \( s_k = ms_0 \), which contains the multiplication of two variables, \( m \) and \( s_0 \). We use the Z3 SMT solver to check for satisfiability of these formulae, since, it supports non-linear arithmetic.

**Refinement** The refinement module takes as input an abstract counterexample of a QPA that has been determined spurious in the validation module and identifies predicates to be added to the state space partition of the QPA which will eliminate the abstract counterexample. We refer to these as “separation predicates”. Depending on the reason for spuriousness, namely, if there exists no infinite execution or if there exists infinite execution, but no diverging ones, AVERIST performs a regular refinement or a weighted refinement, respectively. The regular refinement attempts to reduce the potential infinite executions along the cycle, where as the weighted refinement attempts to reduce the
potential infinite executions whose weights follow those of the edges in the cycle. The determination of a separation predicate requires computing either forward reachability or a weighted version of it along the facets of the cycle, until an empty set is reached for some facet $f$. The tool needs to compute a predicate that separates the last non empty reach set, say, $P_1$ in this computation and the one step backward reach set of $f$, say, $P_2$. Both $P_1$ and $P_2$ are polyhedral sets, and the hyperplanes defining the boundaries of these sets are considered as potential separating planes, which is then verified by checking inclusions of $P_1$ and $P_2$ in the two half spaces corresponding to this hyperplane. This computation involves using the PPL library for computing the reachable sets and checking for polyhedral containment.

### 7.2 Detailed flowchart

The general workflow of AVERIST is shown in Figure 7.1b. The input to AVERIST is a polyhedral hybrid system (PHS) or a linear hybrid system (LHS), which will be analysed for stability. In addition, the user needs to set several parameters and provide a set of predicates, which will be used for state space partition during the abstraction construction. The Create predicates module
7.2. Detailed flowchart

events predicates from the input hybrid automaton description as well as
additional predicates provided as input, depending on parameters set by the
user. Depending on whether the input is a PHS or a LHS, the flow charts in
Figure 7.2a or Figure 7.2b, are called, respectively. The tool returns three
possible outputs, (1) the input system is stable, (2) the input system is unstable
or (3) the stability of the input system could not be concluded, inconclusive.

7.2.1 CEGAR for PHS

The algorithm first checks if the input PHS exhibits unstable behavior in any
individual mode of the system using the procedure Explosion. If it can, then it
outputs that the system is unstable. Otherwise, it uses the predicates computed
in the Create predicates module to construct a weighted graph using the
QPA procedure of Section 7.1, which is model checked using the model checking
algorithm in Section 7.1. The model checking algorithm either returns that the
PHS is stable or returns an abstract counterexample. The counterexample is
validated using the validation algorithm in Section 7.1 and it either concludes
that the system is unstable or proceeds to the refinement step. Depending on
the reason for spuriousness concluded by the validation module, the appropriate
refinement algorithm is called. The refinement algorithm outputs a separation
predicate. This concludes one iteration of the CEGAR loop for PHS. The next
iteration starts with a weighted graph construction using predicates that include
the separation predicate computed in the previous iteration. In fact, in the
implementation, if the counterexample is determined to be spurious, the tool
proceeds in two ways. It proceeds with the refinement step and determines a
separating predicate to be added to the set of predicates for the next iteration,
or it uses a set of uniform predicates to be added that is independent of the
counterexample analysis. Uniform predicates split symmetrically the state
space and are generated by setting a granularity factor (0, 1, 2, etc).

7.2.2 CEGAR for LHS

In the first iteration, the predicates from the Create predicates procedure
are used to perform hybridization on the input LHS and output a PHS. One
iteration of the CEGAR for PHS from Section 7.2.1 is run on the PHS. The
output of this is either stable (because the model checking succeeds), unstable
(because either the explosion succeeds or the validation succeeds) or a separation
predicate (as a result of the refinement algorithms). If the output of CEGAR
for PHS is stable, then we conclude that the LHS is also stable (because the PHS over-approximates the LHS). If the output is a set of separation predicates, AVERIST uses them to compute the hybridization in the next iteration (this is reflected by setting the refinement type to selected). If the output is unstable, it only indicates an unstable behaviour in the abstract PHS, which might not be realizable in the original LHS. Hence, it is needed to refine and the tool uses uniform predicates to construct the next state space partition for the refinement (this is achieved by setting the refinement type to uniform).

7.3 Summary

This chapter has presented AVERIST, a software tool for the verification of stability of hybrid systems. AVERIST implements the hybridization and CEGAR techniques presented in Chapter 5 and Chapter 4 respectively.

Additionally, this chapter has presented the main features of AVERIST, its integration with additional software packages and the user instructions. The modular design of AVERIST facilitates the integration of new implemented extensions for the treatment of new classes of systems and the development of more intricate techniques.
8

Conclusions

This chapter summarizes the work in the thesis and discusses future research directions in stability verification of hybrid systems

Contents

8.1 Summary ........................................... 108
8.2 Future work ......................................... 109
  8.2.1 Short term research ......................... 109
  8.2.2 Long term research ......................... 110
8. Conclusions

8.1 Summary

In this thesis, we presented a novel algorithmic technique for the verification of stability of hybrid systems. The novelty of the algorithmic approach is that it is based on abstraction and model checking techniques, unlike the state-of-the-art approaches. The overall process is automated for the classes of linear and polyhedral hybrid systems, but the framework can be extended to support a more complex class of hybrid systems.

The state-of-the-art approaches for stability verification are based on searching for Lyapunov functions. This search relies on the user expertise to obtain good templates for the Lyapunov function. Further, an unsuccessful attempt at instantiating the templates does not provide insights into the choice of better templates. Therefore, the advantages of the CEGAR approach rely on the fact that the abstraction refinement framework systematically searches for a proof and provides insights to prove stability in the case of a failure.

The abstraction procedure constructs a finite weighted graph which over-approximates the initial hybrid system, by using a finite partition of the state space as a parameter. Then, the obtained weighted graph is model checked to determine whether the hybrid system is stable. In the event of a failure to infer stability from the graph, the novel algorithm provides insights into the reasons for the failure in terms of an abstract counterexample.

The abstract counterexample is validated to check if it corresponds to a concrete counterexample. Hence, the validation technique either returns a concrete counterexample or infers that the abstract counterexample is spurious. The concrete counterexample refutes stability of the hybrid system, while the spurious abstract counterexample highlights the need of a tighter abstract weighted graph. In the second case, the counterexample provides information that can be used to construct a qualitatively better approximation of the hybrid system by considering a finer partition as input to the abstraction procedure. The validation technique is restricted to the polyhedral hybrid systems. Therefore, an useful research direction would be to study validation techniques for the class of linear hybrid systems.

Despite the lack of a validation technique for linear hybrid systems in this work, these systems can be analysed by the CEGAR approach. In Chapter 5, we presented a new hybridization technique specialized for stability verification, which constructs a polyhedral hybrid system over-approximating the linear system behaviour. In case of the existence of an abstract counterexample, a
finer partition will be fed to the abstract procedure, which for the class of linear hybrid systems is unrelated to the counterexample.

In addition to the local stability notions, Lyapunov and asymptotic stability, we have considered a stronger notion of stability, global asymptotic stability. The verification problem for GAS has been attacked in Chapter 6 by reducing it to an AS verification problem and an RS verification problem. Also, Chapter 6 presents a semi-automated proof for proving GAS of an automatic gearbox. However, the steps requiring manual intervention are easy to implement, so the overall GAS procedure can be automated.

In order to show the feasibility of the proposed CEGAR technique and the integrated procedures, Chapter 7 presented AVERIST, a software tool which implements the different algorithms explained throughout this thesis. The tool was used to verify stability for several polyhedral and linear hybrid input systems, obtaining promising results, as shown in Section 3.4, Section 4.5 and Section 5.3.

8.2 Future work

8.2.1 Short term research

Throughout this thesis, several challenges with respect to automating the stability analysis of hybrid systems have been answered, but open questions still remain.

**Non-linear hybrid systems** An immediate direction for future research is to extend the algorithmic framework to non-linear hybrid systems, which are more apt for modelling realistic behaviours. The challenge for this kind of systems, as for linear hybrid systems, relies on computing reachability predicates. These predicates are involved in deducing the existence of edges and in computing the weights associated with them for the weighted graph construction in the QPA procedure. In the case of linear and non-linear dynamics, the reachability predicates cannot be computed exactly, however, over-approximations using tools such as SpaceEx [FGD+11] can be computed. As was discussed in Chapter 5, these approximation techniques also fall short, especially, when the number of discrete switches is unbounded. Hence, hybridization was introduced.
8. Conclusions

We will explore extensions of the hybridization technique for abstracting non-linear dynamics to linear or polyhedral inclusion dynamics.

**Validation and refinement for linear hybrid systems** In Chapter 5, linear hybrid systems are abstracted into polyhedral hybrid systems by a hybridization technique appropriate for stability. The abstract polyhedral system can be processed by the QPA and model checking algorithms, but when these algorithms output an abstract counterexample, there is no technique to validate the counterexample, that is, to evaluate if it corresponds to an explanation for instability of the linear hybrid system. This lack of validation method shows the need to develop validation and refinement techniques specific to linear hybrid systems.

**Completeness of hybridization** Chapter 5 proved completeness of the hybridization technique for the class of asymptotically stable switched linear systems that are uniformly converging in time and Lipschitz continuous. One interesting direction for future research is to perform an in-depth analysis of the class of systems for which the method is complete, that is, will eventually prove stability if the system is stable. While theoretically, extending the approach to the general class of hybrid systems is straightforward, the practical challenge is in finding the right class of partitions to consider for hybridization.

**Averist** The implemented software tool is under continuous development, so there are numerous possible extensions. For instance, a tool improvement is to apply memoisation, which consists of storing solutions for the input instances in order to avoid recomputation when the input instances repeat.

8.2.2 Long term research

Considering stability verification of hybrid systems, there are two lines of work that naturally arise. One is the use of compositional analysis to tackle the state space explosion problem, which is an essential issue with model checking techniques in practice. The other research line points to the development of techniques for automatically designing stable hybrid systems.
8.2. Future work

**Compositional analysis** A fundamental approach successfully applied to verification of large-scale hybrid systems is compositional reasoning. A compositional analysis of a system consists of decomposing the system into subsystems and independently analysing the subsystems in such a way that a specified property of the system is proven to be either correct or not. This approach reduces the complexity of the original verification problem, making therefore feasible to verify large-scale hybrid systems.

Hybrid systems have complex dynamics, so in general, formal verification of them relies on constructing abstractions of the systems and analysing such abstractions to deduce correctness of the hybrid systems. The abstraction techniques can be integrated into a compositional analysis framework for hybrid systems to reduce the analysis complexity. The resulting compositional approach analyses separately the abstracted subsystems to infer correctness of the hybrid system.

In particular, for stability verification of hybrid systems, we can investigate rules for decomposing a given hybrid system into constitutive hybrid systems that can be stability-analysed by applying techniques inspired by the QPA and model checking techniques explained in this thesis. Moreover, the compositional framework requires defining interacting constraints on the systems and formalising modified stability specifications.

**Synthesis** Stability verification consists of checking if an input hybrid system is stable. An alternate approach to verification is to automatically synthesise the design of a stable hybrid system. In this regard, a research direction is to investigate construction algorithms that output stable hybrid systems by considering behavioural constraints on the hybrid systems.

In terms of computational cost, synthesis approaches are usually more expensive than verification techniques. Therefore, synthesis is useful when focused on partial aspects of the design. Inspired by the algorithmic approach to stability verification in this thesis, an algorithmic technique for the synthesis of stable hybrid systems is presented in [PS17]. Concretely, this technique synthesizes a state-based switching control for stabilizing a family of linear dynamical systems. Considering the family of linear dynamical systems and a set of polyhedral switching surfaces, an abstract weighted graph is constructed, in such a way that the stability synthesis problem is reduced to a strategy synthesis for an energy game.

This research has been partially supported by BES-2013-065076 grant from the Spanish Ministry of Economy and Competitiveness.


