Carl von Ossietzky Universität Oldenburg
Fakultät II – Informatik, Wirtschafts- und Rechtswissenschaften
Department für Informatik

# Automatic Stability Verification via Lyapunov Functions

## Representations, Transformations, and Practical Issues

Von der Fakultät für Informatik, Wirtschafts- und Rechtswissenschaften der Carl von Ossietzky Universität Oldenburg zur Erlangung des Grades und Titels eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

angenommene Dissertation

von Herrn Eike Möhlmann

geboren am 01.04.1985 in Oldenburg

Gutachter:
Prof. Dr.-Ing. Oliver Theel

Weiterer Gutachter:
Prof. Dr. Martin Fränzle

Tag der Disputation:
05.12.2018

## Acknowledgements

Many people helped and supported me while I was working on this topic and writing this thesis. I am grateful for all of them and I am sorry for everyone I do not mention here.

First of all, I thank my wife, Anke, my kids, Oskar and Ida, and the rest of my family for always being a good backup, motivating me on my long journey, and having understanding for all the time I could not spent with you and all the activities I could not join. I thank my parents, Heidi Möhlmann and Udo Hilberts, for enabling and supporting my studies.

I thank my scientific supervisors, Oliver Theel, Martin Fränzle, Werner Damm, and Ernst-Rüdiger Olderog for inspiring ideas, the shared extraordinary amount of experience and knowledge as well as the patience not only during this work but also during my studies towards the diploma degree.

I thank my colleges, especially Jens Oehlerking for introducing me to this topic, the long and interesting discussion about fruitful and not so fruitful ideas, Oday Jubran for all the fun, brain teasers, heated debates, and the paper-deadline-pizza-evenings, Willem Hagemann and Astrid Rakow for the great cooperation and scientific support, Robert Schadek for being the best travel-mate on the unbelievable exploration of South America, and Sebastian Gerwinn for pushing me over the finishing line.

I also thank the many interns and students that I was allowed to meet and supervise, especially, Dhanya Sridhar who supported me in the development of the spidercam example and the dense to sparse relaxation, Birte Kramer who supported me during the development of the unrolling relaxation, Adam Wheeler with whom I was working on another (yet unpublished) relaxation idea.

Further, I thank my whole former group at the university for the delicious cake and cinema visits that truly added joy to life, the whole team at AVACS for creating such a vivid and highly inspiring atmosphere, and my current group at OFFIS for the affectionate welcome and introduction to further research directions.

And finally, this work would not exist without financial support for which I am also thankful. Hence, I would like to mention that my studies were supported by the Hans-Böckler-Stiftung and this work has been supported by the German Research Council DFG within SFB/TR 14 “Automatic Verification and Analysis of Complex Systems”.

# Abstract

Starting with the industrial revolution, more and more tasks in our every-day life are taken over by machines. Since then, the standard of living has improved for the general population. Besides the evident improvements, a major challenge was to assure flawless functionality of the machines. Mathematical modeling and analysis are key enabler for this validation. In the beginning the machines were mechanical systems, large but simple compared to the machines build to satisfy modern requirements. Nowadays, these machines combine mechanical and electronic systems and have a hybrid (discrete-continuous) state space. The discrete components correspond to an embedded controller and the continuous components often correspond to a physical plant. Interconnected, such mixed systems are usually called cyber-physical systems and their application range from assistance functions to fully autonomous functions. Especially in the field of autonomous systems, it is of utter importance, that disturbances will be counterbalanced (e. g., a satellite needs to compensate forces induced by radiation, gravity, and the reorientation of the solar panels). Such a robustness against disturbances should be ensured before launch, e. g., via a formal proof of stability.

In this thesis, we focus on the automatic verification of global asymptotic stability of such hybrid systems modeled as hybrid automata. Global asymptotic stability ensures that the state of the system converges to a predefined desired state independently of the initial state. A standard tool to derive a proof of global asymptotic stability is Lyapunov theory. Lyapunov theory requires us to find a so-called Lyapunov function. A Lyapunov function assigns each state of the system an energy value such that the value decreases while the system evolves. The existence of such a function does not only prove stability but also robustness against external disturbances. Even though the search is usually manual, constraint systems whose solution yield Lyapunov functions can be formulated and numerical methods for solving exists.

The contribution of this thesis is three-fold: (1) the automatic derivation of Lyapunov functions (2) the systematic transformation of hybrid systems in order to simplify the computation of Lyapunov functions, and (3) the extension of a framework for component-based design of safe and stable hybrid systems.

With respect to automatic derivation of Lyapunov functions, we have fully automatized all steps from deriving the constraints over solving them to interpreting the results. Our implementation includes proof schemes for finding common Lyapunov functions, piecewise Lyapunov functions as well as a compositional Lyapunov functions. Additionally, we have developed and implemented numerical and symbolic techniques to double-check the results of the numerical solver. Without such checks a compositional Lyapunov functions might be invalid. Further, heuristics to simplify obtaining valid results are presented.

We have developed transformation techniques that simplify the search for Lyapunov functions. One can improve the applicability of cycle-based decomposition approaches by relaxing the automata's underlying structure. This – in the best case – requires a linear number of cycles to be investigate but – in the worst case – successively reverts to the original structure. The other technique carefully combines reachability and stability analysis by iteratively unrolling the underlying automaton and mutually uses results

from the analysis for simplification.

## Zusammenfassung

Seit der industriellen Revolution werden fortlaufend weitere Aufgaben durch Maschinen übernommen. Seit dieser Zeit hat sich der Lebensstandard für die breite Bevölkerung weitergehend verbessert. Jedoch war es von Anfang an eine große Herausforderung die einwandfreie Funktionalität der Maschinen nachzuweisen. Das Hauptwerkzeuge für die Validierung sind mathematische Modellierung und Analyse. Zu Beginn waren die Maschinen rein mechanisch und groß, jedoch im Vergleich einfacher als solche Maschinen, die gebaut werden um aktuelle Anforderungen zu erfüllen. Moderne Maschinen kombinieren mechanische und elektronische Systeme und haben einen hybriden (diskret-kontinuierlichen) Zustandsraum. Die diskreten Komponenten im Zustandsraum entsprechen oft digitalen Kontrollern und die kontinuierlichen Komponenten einer physikalischen Anlage. Vernetzt werden solche Systeme als Cyber-physikalische Systeme bezeichnet und finden Anwendung als Assistenzsysteme sowie als voll autonome Systeme. Gerade im Bereich der autonomen Systeme ist es erforderlich, dass Störungen automatisch ausgeglichen werden (z.B. müssen Satelliten Strahlungs-, Gravitationskräfte und Bewegungen der Sonnenkollektoren ausgleichen). Schon vor der Inbetriebnahme sollte die Robustheit gegenüber Störungen, z.B. durch den formalen Nachweis von Stabilität, sichergestellt werden.

Diese Arbeit widmet sich der automatischen Nachweisführung globaler asymptotischer Stabilität für Systeme welche als Hybride Automaten gegeben sind. Globale asymptotische Stabilität stellt sicher, dass der Systemzustand unabhängig vom initialen Zustand zu einem vorgegebenen Zustand, dem Equilibrium, konvergiert. Typischerweise wird der Nachweis mit Hilfe von Ljapunow-Theorie erbracht. Dazu wird eine sogenannte Ljapunow-Funktion gesucht, welche jedem Systemzustand einen nicht-negativen “Energiewert” zuordnet, so dass dieser Wert beim Voranschreiten des Systems abfällt und das Minimum im Equilibrium erreicht. Obwohl die Suche meist manuell ist, können Gleichungssysteme aufgestellt werden deren Lösung die Existenz einer solche Funktion implizieren.

Die Beiträge dieser Arbeit fallen in drei Bereiche: (1) die automatische Bestimmung von Ljapunow-Funktionen, (2) die Transformation der Systembeschreibung zur Vereinfachung des Stabilitätsnachweises und (3) der kompositionelle Entwurfsprozess von sicheren und stabilen hybriden Systemen.

Hinsichtlich des automatischen Nachweises von Stabilität wurden alle Schritte von der Herleitung von Gleichungssystemen über das Lösen dieser bis hin zur Überprüfung einer möglichen Lösung automatisiert und prototypisch implementiert. Der Prototyp ermöglicht das Finden von gemeinsamen, stückweise-definierten und kompositionellen Ljapunow-Funktionen. Weiterhin wurden numerische und symbolische Methoden zur Überprüfung der numerischen Lösung sowie zur Vereinfachung des Gleichungssystem entwickelt und implementiert. Durch die Überprüfung kann verhindert werden, dass ungültige kompositionelle Ljapunow-Funktionen erzeugt werden.

Hierzu werden zwei Transformationstechniken entwickelt und implementiert. Die erste verbessert die Anwendbarkeit eines existieren Zyklen-basierten Dekompositionsansatzes auf Automaten mit einer dichten Graphstruktur. Die zweite Technik integriert Erreichbarkeits- und Stabilitätsanalyse durch das iterative Abrollen des Automaten, wo-

bei wechselseitig Analyseergebnisse zur Vereinfachung verwendet werden.

# Contents

# CHAPTER 1

# Introduction

It is a fact that our every day's life relies more and more on machines that are taking over an increasing number of tasks. Among these, there are critical tasks where failures result in irreversible harm to humans and nature. One example is the push towards the application of advanced driver assistance systems (ADASs) and autonomous driving functions (ADFs) in the automotive domain. An advanced driver assistance system is a system that assists a human driver in sensing the environment and controlling the vehicle. And an autonomous driving function is a function that takes over all steps from the perception over maneuver planning to maneuver execution. In both applications, the quality of the continuous monitoring of the environment and the actions of the vehicle as well as a continuous re-planning is crucial for avoiding critical situations in which humans, nature, or property is put at risk. Several methodologies and techniques have been invented, standardized, and successfully applied to reduce the risk of such critical situations in practice. However, the more complex the tasks become – like fully autonomous driving of vehicles, airplanes, and vessels – the more complicated is the analysis of the system preforming the tasks. Taking into account the number of involved components and the associated physical processes, it is hardly possible for a single engineer to get an overview of all the interrelations, and it has been agreed that (semi-)automatic methods are essential for analyzing complete systems and even subsystems. For some of the so called cyber-physical systems, it is sufficient to address *safety*, i. e., the absence of (potentially) harming "bad" events, but for others, it is mandatory to address *liveness*, i. e., the existence of "good" events. In particular, an interesting and common special cases of liveness properties are convergence properties like *stability*. An example of an autonomous operation, where stability is key, is an orbit control system of a satellite whose task is to keep the satellite in a geostationary orbit. The satellite has only limited resources (e. g., fuel) for adjusting the orbit. If the satellite veers away too much from earth, then more resources are needed to prevent the satellite from getting lost; if the satellite comes too close to earth, then more resources are needed to prevent the satellite from crashing onto earth. In such a case, optimal control of the impulses towards a stable orbit maximize the life-time of the satellite and stability is the concept to apply.

Consider the basic feedback loop given in Figure 1.1. It consists of a *controller* and a *plant*. For many real systems the controller is an embedded system and the plant is its (partial) surrounding environment. Examples of such systems are orbit control systems of satellites, automatic cruise controllers, engine control units, or unmanned powerhouses. In all these examples, certain operating conditions have to be maintained, and it is the

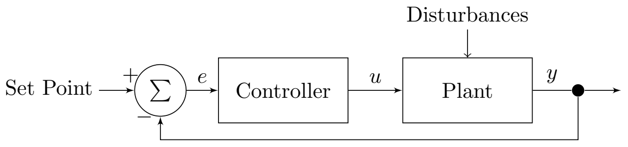


Figure 1.1: A basic feedback loop.

task of the controller to drive the output $y$ of the plant into the desired operating range. This desired operating range is often a single point called the *set point* or *equilibrium point*. Given such a set point and the output of the plant $y$, an error value $e$ can be computed. The controller then decides for appropriate values of the *actuators* $u$ based on this error value. In practice, such a control task has to be performed in the presence of external disturbances which makes it more complicated – if possible at all – to ever achieve no error. However, a controller that achieves a vanishing error value is called an *asymptotically stabilizing controller* and the overall system is said to be *asymptotically stable*.

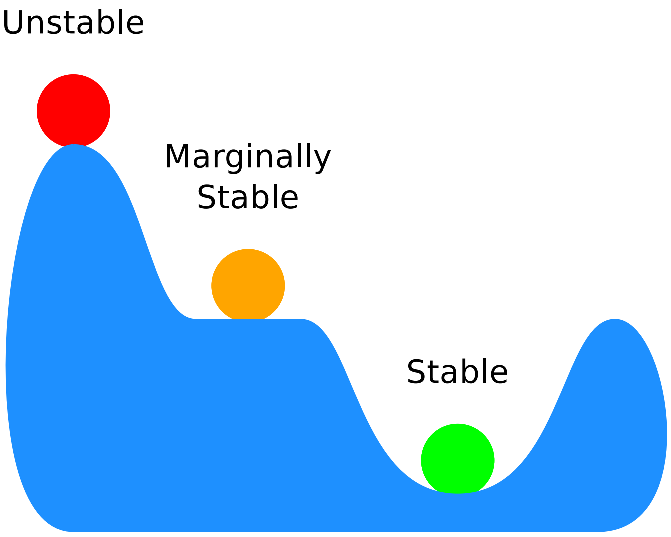
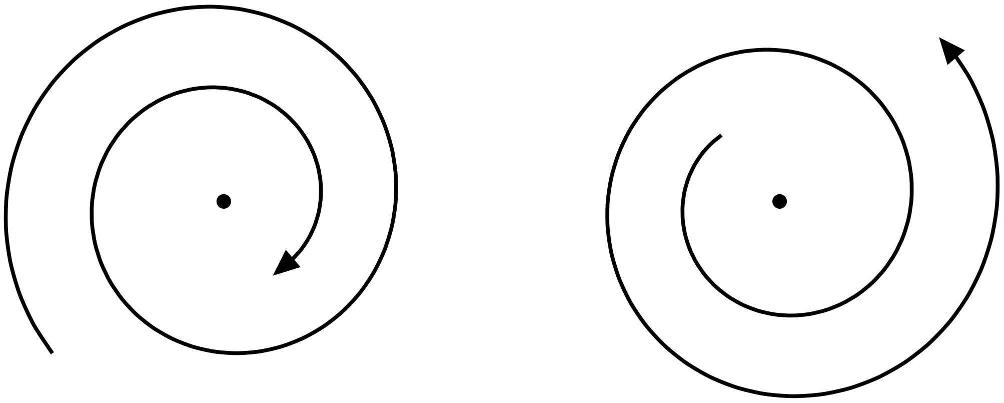


Figure 1.2: Visualization of an unstable (left, red), a marginally stable (middle, orange), and a stable (right, green) system.

Stability is a very desirable property, since stable systems are inherently fault-tolerant: after the occurrence of faults leading to, for example, a changed environment[1], the system will automatically "drive back" to the set of desired (i.e., stable) states. Stable systems are therefore particularly suited for contexts where autonomy is important, such as dependable assistance systems, or in contexts where properties have to be assured in an adverse environment. A system that is not stable could be

[1] Such a changed environment can be regarded as disturbances, e. g., a drop of the temperature due to an open door. In which case a controller could increase the heating effort.



(a) Visualization of a trajectory of a stable (convergent) system.

(b) Visualization of a trajectory of an unstable (divergent) system.

Figure 1.3: Trajectories of a stable and an unstable system.

- *unstable*: the system's output grows unboundedly, or
- *marginally stable*: starting at the equilibrium point, the system stays at this point and small perturbations lead to small changes of the output without the system returning to the equilibrium point.

These very basic properties are visualized in Figure 1.2. The figure shows three balls. The red ball on the top of the hill is in an unstable situation, because a slight impulse makes the ball roll arbitrary far away. The orange ball on the plateau is in a marginal stable situation, because a slight impulse results in a slightly different position. The green ball in the valley is in a stable position because a slight impulse makes the ball return to its initial position. These properties and other related properties will be discussed in more detail in Section 2.3. The continuous evolution of a system over time is called a *trajectory*, and Figure 1.3 visualizes two trajectories in the phase space. Figure 1.3a shows a converging trajectory corresponding to a stable system and Figure 1.3b shows a diverging trajectory corresponding to an unstable system. Both trajectories can be described by the system $\dot{x}(t) = K_p(x_0 - x(t))$ which is a classical proportional controller with only one parameter $K_p$. A pure proportional controller is a special case of a proportional-integral-derivative (PID) controller which has two additional parameters $(K_i, K_d)$ for the integral and derivative parts. Such PID controllers can be seen as a Swiss army knife for control engineers and are broadly applicable due to the fact that only the error $x_0 - x(t)$ between the desired value $x_0$ and the actual value $x(t)$ needs to be measured. However, stability depends on the choice of tuning parameters (i. e., $K_p, K_i, K_d$). In that sense, for a proportional-only controller, a negative value for $K_p$ leads to Figure 1.3a whereas a positive value for $K_p$ leads to Figure 1.3b. This can be easily seen as the solution of the above differential equation is $x(t) = x(0)e^{K_p t}$ which obviously diverges for $K_p > 0$ (unstable), stays constant for $K_p = 0$ (marginally stable), and converges for $K_p < 0$ (stable). Note that this illustration does not include a plant. Instead, we simply closed the loop by feeding the output of the controller to the input. Therefore, we are able to explicitly determine the solution of the differential equation. Usually, we are not able to do this, since for a more realistic system, we often have

- several variables that depend on each other, for example, consider a vector-valued system $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}$,
- several different dynamics that depend on the mode of the system, for example, a vector-valued system $\dot{\mathbf{x}} = \mathbf{A}_i\mathbf{x}$, where $i$ depends on systems state (i. e., the value of $\mathbf{x}$), and
- several different dynamics that depend on the common state of the plant and the controller, for example, a vector-valued system $\dot{\mathbf{x}} = \mathbf{A}_i\mathbf{x}$, where the actual $i$ is selected by algorithms implemented in an embedded system.

While all these systems are dynamic systems, the last two are hybrid systems as they exhibit, both, discrete and continuous behaviors. In the control theory domain, the continuous behavior – described by differential equations – is called the *flow* and the discrete behavior – often described by an automaton – is called the *jump*. A powerful model for such hybrid systems are *hybrid automata* which have been introduced by Alur et al. in [Alu+92]. A hybrid automaton modeling a simple temperature control unit is given in Figure 1.4. The implemented control law is essentially the proportional controller from above where, additionally, the change of the temperature is saturated from below and above, i. e., $\dot{T} \in [-1, 1]$. As depicted, hybrid automata combine both, finite automata and differential equations. The discrete states are modeled as *locations* (or sometimes *modes*), the discrete behavior is modeled as *transitions* (or sometimes *jumps*), and the continuous behavior is modeled as differential equations over real valued *data variables* (or short *variables*) that are annotated to the locations. Additionally, locations are annotated with *invariants* that correspond to the values of the variables that are allowed while the location is active and transitions are labeled with *guards* and *updates*. The guards correspond to the values of the variables for which a transition can be taken and the update describes the effect on the valuation of the variables, e. g., setting a variable to some specific value. That way, hybrid automata are in particular well-suited to model complex systems where physical processes interact with embedded controllers. Often, we find the physical process described by differential equations and the controller by a

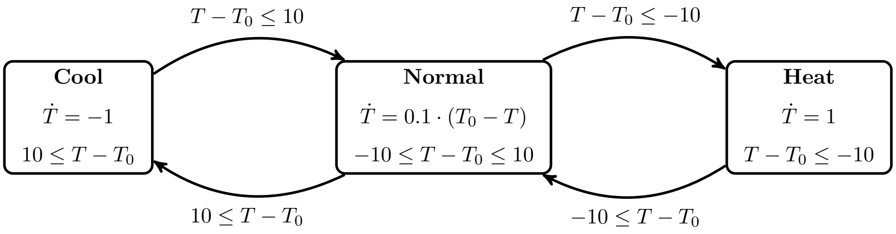


Figure 1.4: Hybrid automaton modeling a saturated temperature control loop.

finite automaton. Although it is sometimes possible to discretize physical relations (using quantization) or to fluidize discrete steps (having a real-valued count of objects), it is more natural and less error-prone to use hybrid automata for modeling and verification.

## 1.1 Stability Verification

Consider the vector-valued system $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}$. For this purely continuous linear system, the evolution of the trajectory depends only on the transition matrix $\mathbf{A}$. For such first-order (and even for second-order) systems, the set of trajectories can be derived by solving the equation explicitly. However, when proving stability for such systems one can simply check whether the matrix $\mathbf{A}$ is Hurwitz, without solving the equation. This is called the Routh-Hurwitz criterion and roughly corresponds to checking whether all roots of the characteristic polynomial have negative real parts. However, in general and in particular for hybrid systems, asymptotic stability properties are undecidable [BT99; Blo+00]. Nevertheless, indirect methods that establish properties on the states are quite successful in proving asymptotic stability. The idea is to find a function that maps every state of the system to a value such that the value decreases while the system evolves. These functions are called *Lyapunov functions* [Lya07] and the value can be seen as the energy stored in the system which dissipates over time. Such a Lyapunov function has to satisfy three properties: (1) the minimum is at the equilibrium, (2) the derivative of the function in the direction of the evolution of the system is (strictly) negative everywhere but at the equilibrium, and (3) the value of the function grows unboundedly with the distance to the equilibrium. This criterion is well-known and used in control theory but was originally developed for purely continuous systems only. Throughout the 1990s, the use of Lyapunov functions was lifted to the field of hybrid systems [Bra94; LSW94; Bra98; Pet99]. The key idea was to search for multiple Lyapunov functions that are compatible with the above properties, i. e., one assigns each location (or mode) of the hybrid system a separate Lyapunov function. These local Lyapunov functions have to decrease while the corresponding location is active and, additionally, when taking a transition (switching to another mode), the value must not increase.

Further, it has been found that when using templates for the Lyapunov functions, one can construct a constraint system whose solution yields a set of Lyapunov functions for the hybrid system. The most popular method for solving such a constraint system is called semidefinite programming (SDP) [Pet99; PL03; PP03]. Using sum-of-squares decomposition, it is even possible to apply the search for Lyapunov functions for non-linear hybrid system or rather hybrid systems whose dynamics, guards, and invariants are described by polynomials. This can even be automatized, e. g., Oehlerking et al. implemented a tool that combines state space partitioning with the search for Lyapunov functions for linear hybrid systems [OBT07].

## 1.2 Contributions

In this section, we state the basic topics and issues tackled in this thesis. The topic we address is the automatic verification of *global asymptotic stability* in the context of *hybrid systems*. To this end, we focus on techniques to algorithmically derive a mathematical proof of global asymptotic stability for a given hybrid system modeled as a hybrid automaton. The basic techniques we employ are semidefinite programming, sum-of-squares and Lyapunov functions. These techniques are not new in particular but have not yet been integrated as an automatic push-of-a-button verification tool for hybrid systems. The automatization of these techniques and the integration is the central contribution of this work. During the work, we identify several issues:

**(Issue 1)** During the computation of Lyapunov functions *numerical issues* arise easily for automatic generated constraint system. As a result, solutions produced by a numerical solver might not yield valid Lyapunov functions although the feasibility information might be correct. Hence, it must be taken care when reusing such a computed Lyapunov function in further proofs.

**(Issue 2)** Success and failure of automatically computing Lyapunov functions depend on the *representation of the system*. For example, it can be crucial to make implicit knowledge explicit (e. g. by tightening guards and invariants) or choosing a certain shape for the hybrid automaton's underlying graph.

**(Issue 3)** There is *no automatic support in the design* of stable hybrid systems. Even worse, the established methods give no hint about the reason when failing to prove stability. Moreover, slight modifications of the system require to restart proving stability.

In other works, Issue 1 is not relevant because the feasibility information of the solver is usually reliable – e. g., this is the case for well-conditioned constraint systems. When performing a monolithic proof, feasibility is all we are interested in because that already certifies stability. However, in the light of compositional approaches, we reuse and recombine obtained numerical results and slight errors could render the whole proof invalid. Issue 2 and Issue 3 arises due to the automatization. Otherwise, when deriving the constraints manually, then this task includes finding the best representation and deciding how later modifications in the design affect the proof obligations.

In this thesis we address these issues by the following contributions:

**(Contribution 1)** We have implemented a tool to automatically derive a certificate of stability for non-linear hybrid systems, called Stabhyli [MT13a]. Certificates are obtained by different proof schemes such as computing common and piecewise Lyapunov functions as well as the decomposition by Oehlerking and Theel [Oeh11] and the component based approach by Damm et al. [Dam+10]. Details are given in Section 3.2. Further, numerical issues have been addressed and the tool includes techniques to detect and handle implicit equalities [MT13b] and validate candidate solutions obtained from numerical solvers [MT14]. These techniques are described in detail in Section 3.3 and Section 3.4.

**(Contribution 2)** To address Issue 2, we have developed and implemented transformation techniques that simplify the computation of Lyapunov functions. One technique is the relaxation of the graph structure [MT15] and the other is the unrolling of the hybrid automaton [HMT15; HM15; MHT15]. The relaxation technique takes as input a hybrid automaton whose underlying graph is dense and returns a hybrid automaton whose underlying graph is sparse. The benefit is that the latter is better-suited for the decomposition approach of Oehlerking and Theel but may yield additional behavior which could render the automaton unstable. The unrolling technique is presented in more detail in Section 4.3. The unrolling technique takes a hybrid automaton as input and – in case it terminates – returns an equivalent hybrid automaton that is acyclic. The benefit is that for acyclic hybrid automata proving stability can be done by successively inspecting all locations in isolation and then all transitions in isolation. Furthermore, since this technique interleaves reachability and stability proving steps, safety properties can be decided with few overhead. Details on this interleaving are given in Section 4.3.

**(Contribution 3)** In addition to the component-based approach developed by Damm et al. [Dam+10] which addresses sequential (or rather transition) composition, we have built a supplementary framework for the parallel composition of hybrid automata also preserving safety and stability [MHR17; Dam+14; DMR14; DMR16]. Details on the underlying theory focusing on stability and obtaining Lyapunov functions for parallel composed controllers are given in Section 5.3. An application to an advanced driver assistance system (ADAS) in form of a case study is given in Section 5.4.

Most of the contributions have already been published in peer-reviewed proceedings and some additionally as technical reports which then also included supplementary material. Only for Contribution 3, the formal background for composing stable hybrid systems and their Lyapunov functions is published in this thesis for the first time.

## 1.3 Outline of the Thesis

The outline of this thesis is as follows: Chapter 2 introduces some basic notations and defines the main system model, hybrid automata, and the stability notion that this thesis focuses on. Equipped with this knowledge, we look at existing techniques for stability verification and collect the ingredients for automating stability verification based on Lyapunov functions.

In Chapter 3, we focus on Contribution 1 and present Stabhyli, which is our prototype tool for automatic stability verification and investigate details and background. Further, we explore practical issues in stability verification and propose solutions for these.

The focus of Chapter 4 is Contribution 2, where we present the graph relaxation and unrolling technique. Applicability is shown by analyzing several examples for each technique.

Contribution 3, which is the component based design framework or more specific, its extension for safety and stability preserving parallel composition, is given in Chapter 5. Here, we introduce basic concepts like controllers, plants, and communication channels. As the background, we summarize the stability-related aspects behind the sequential composition and, then, introduce adapted versions of the stability notation and proofs suitable for capturing parallel controllers in the presence of a shared plant. We show how to apply this new stability notion in a case study of an advanced driver assistance system. In the case study the task is to control a vehicle using two loosely-coupled concurrent controllers. Each controller has its specific goals such as (1) keeping the vehicle on the lane and (2) keeping a user-chosen velocity if applicable and together they have to (3) maintain a comfortable centrifugal force.

In Chapter 6, we conclude this thesis and give a short outlook on open questions and possible future extensions of this work.

CHAPTER 2

# Preliminaries and Related Work

In this chapter, we will introduce the theoretic foundations, draw the state-of-the-art, and discuss related work.

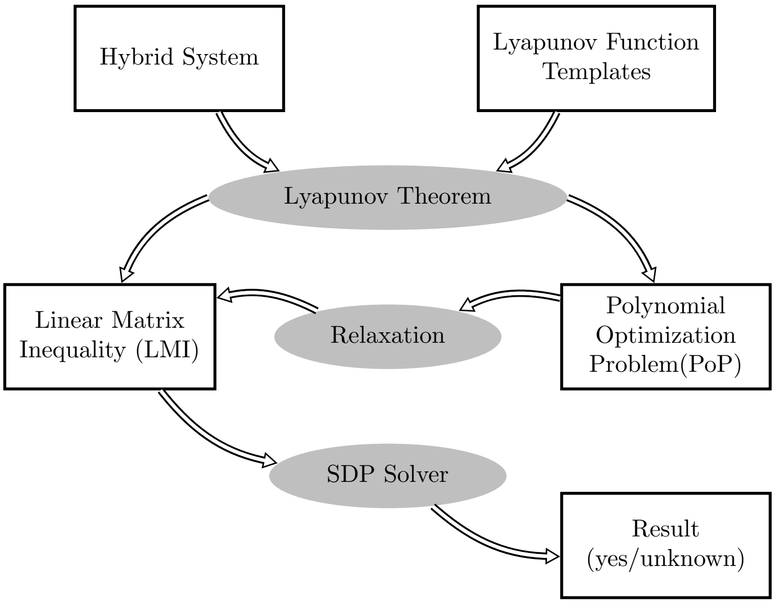

Hybrid System
Lyapunov Function Templates
Lyapunov Theorem
Linear Matrix Inequality (LMI)
Relaxation
Polynomial Optimization Problem(PoP)
SDP Solver
Result (yes/unknown)

Figure 2.1: Sketch of the Basic Steps and Artefacts in Automated Stability Verification via Lyapunov Theory.

Figure 2.1 sketches the basic steps behind the automated verification of stability of hybrid systems. The inputs to that procedure are (1) the hybrid system for which we want to prove stability, and (2) a set of so-called Lyapunov Function Templates. The Lyapunov theorem is then used to generate a set of constraints that, if solvable, prove that the hybrid system is indeed stable. Where, we can handle two cases:

Case 1: if a hybrid system's description involves linear expressions only, then the derived set of constraints consists of linear matrix inequalities (LMIs), or

Case 2: if a hybrid system's description involves higher order polynomials, then the derived set of constraints involves polynomial constraints.

In the second case, a series of relaxations can be applied such that, again, we obtain an LMI. This relaxtation introduces additional parametes and, hence, teh resulting

LMI is higher dimension. Such an LMI can be feed to a, so-called semidefinite program (SDP) solver, which is finally used to check whether a solution exists. An SDP solver optimizes a linear objective function under additional semidefiniteness constraints on symmetric matricies. Such a constraint requires that, for a matrix $M$, the scalar $\mathbf{x}^T M \mathbf{x}$ is non-negative for every column vector $\mathbf{x}$.

The outline of this section is as follows. In Section 2.1 we introduce some basic notations. We will then introduce hybrid automata in Section 2.2. Hybrid automata is the formalism we use to describe hybrid systems throughout this thesis. Various stability notions will be introduced and compared in Section 2.3, among them global asymptotic stability which is the stability notion we focus on. Using Lyapunov theory, we are able to prove (global) asymptotic stability which is shown in Section 2.4. This section also formally introduces LMIs and shows how to relax constraints obtained from the Lyapunov theorem via a sum-of-squares (SOS) decomposition to LMIs. This relaxation allows to handle systems whose system description, in form of a hybrid automaton, involves polynomials. The resulting linear matrix inequalities can be solved automatically and existing tools are presented in Section 2.4.5. To increase self-containment of this thesis, we briefly sketch the decomposition as proposed by Oehlerking and Theel [OT09; Oeh11] in Section 2.5. Here, we start with basic graph notions and then, present the two main decomposition levels

Level 1: decomposition into strongly connected components and

Level 2: decomposition into (overlapping) cycles.

This decomposition technique is presented in more detail, since is serves as the main preliminary work for this thesis. Other related work is presented in the following chapters.

## 2.1 Basic Notations

In this section, we introduce some basic notations. Let us first introduce some common sets, we often refer to.

**Definition 2.1 (Common Sets)**
*By* $\mathbb{N}, \mathbb{N}_{>0}, \mathbb{R}, \mathbb{R}_{\geq 0}, \mathbb{R}_{>0}$*, we denote the* non-negative integers*, the* positive integers (without 0)*, the* real numbers*, the* non-negative real numbers*, and the* positive real numbers (without 0)*, respectively. We use* $\mathit{Time} := \mathbb{R}_{\geq 0}$ *to denote the set of* timepoints *and* $\mathit{Time}_\infty := \mathit{Time} \cup \{\infty\}$ *to denote the set of* timepoints including infinity*. Let* $X$ *be an arbitrary set. By* $\mathcal{P}(X) := 2^X$ *we denote the* power set *of* $X$*, i. e., the set of all subsets of* $X$*.* ♢

We deal with hybrid systems in this thesis and often, we have to handle runs in forms of trajectory segments. These trajectory segments describe the evolution of the continuous variables over time and can be either finite or infinite. As we are concerned with stability – or rather asymptotic stability, – we will usually look at infinite trajectories and therefore the last segment of a run continues to evolve ad infinitum. Hence, we use two time points $t_0 \in \mathit{Time}$ and $t_1 \in \mathit{Time}_\infty$ to denote the interval $[t_0, t_1]$ of time points of a trajectory segment. Note that $[t_0, \infty)$ denotes the right-open interval of all time points $t$ with $t \geq t_0$ of an infinite trajectory segment. To describe the evolution of the continuous variables, we use vector-valued functions over time, e. g., $\mathbf{x}(t)$. To distinguish vector and scalars we will use boldface letters to denote vectors, e. g., $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{R}^n$ and upper case letter for matrices, e. g., $M, F$, where $F$ is used when the matrix relates to a function $f$.

**Definition 2.2 (Vector and Matrix Notations)**
*Let $M \in \mathbb{R}^{n\times m}$ be a real-valued matrix and let $\mathbf{x} \in \mathbb{R}^n = \mathbb{R}^{n\times 1}$ be a column vector. We denote by $M^T$ the* transpose *of matrix $M$ and analogously by $\mathbf{x}^T \in \mathbb{R}^{1\times n}$, we denote the result of the transposition which is a* row vector*. We access the elements of a matrix or vector by a subscript in parentheses, e. g., $\mathbf{x}_{(i)}$ is the i-th element of vector $\mathbf{x}$ and $M_{(i,j)}$ is the element in the i-th row and j-th column of matrix $M$. For two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, we denote by*

$$\langle \mathbf{x} \mid \mathbf{y} \rangle = \mathbf{x}^T \mathbf{y} = \sum_{i=1}^{n} \mathbf{x}_{(i)} \mathbf{y}_{(i)}$$

*the* inner product*. For a squared matrix $M \in \mathbb{R}^{n\times n}$*

$$\mathrm{tr}(M) = \sum_{i=1}^{n} M_{(i,i)}$$

*defines the* trace of the matrix *$M$.* ♢

Often, each element of a vector represents the value of a certain variable $v \in \mathit{Var}$, where $\mathit{Var}$ is a finite set of variables. We refer to $\mathbf{x}_{\downarrow \mathit{Var}'} \in \mathbb{R}^{|\mathit{Var}'|}$ as the *sub-vector* of a vector $\mathbf{x} \in \mathbb{R}^{|\mathit{Var}|}$ containing only values of variables in $\mathit{Var}' \subseteq \mathit{Var}$. We use the shorthand notation $\mathbf{x}_{\downarrow v}$ to denote $\mathbf{x}_{\downarrow \{v\}}$ where $\{v\} \subseteq \mathit{Var}$ is a singleton set.

**Definition 2.3 (Vector Order)**
*For two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, we denote by*

$$\begin{bmatrix} \mathbf{x}_{(1)} \\ \vdots \\ \mathbf{x}_{(n)} \end{bmatrix} \leq \begin{bmatrix} \mathbf{y}_{(1)} \\ \vdots \\ \mathbf{y}_{(n)} \end{bmatrix} := \bigwedge_{i=0}^{n} \mathbf{x}_i \leq \mathbf{y}_i$$

*the* vector order *which is a partial order.* ♢

Given a function $f$, we can use the gradient to describe direction and steepness of the tangent of the graph of a function. The direction is always the direction of the greatest increase.

**Definition 2.4 (Gradient)**
*For a real-valued function $f : \mathbb{R}^n \mapsto \mathbb{R}$, we denote by*

$$\nabla f = (\nabla f)_{\mathbf{x}} = \frac{\partial f}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$$

*the* gradient *or* directional derivative *of $f(\mathbf{x})$ which is a column vector of* partial derivatives $\frac{\partial f}{\partial x_i}$ *in all directions of $\mathbf{x} = \begin{bmatrix} x_1 \dots x_n \end{bmatrix}^T$. If clear from the context, we omit $\mathbf{x}$.* ♢

We will use this property to formulate the requirement that the value of a Lyapunov function decreases in the direction of the flow. As we will see later, this requirement among others results in properties of the level sets of Lyapunov functions that can be further exploited. A level set of a Lyapunov function contains all states (i. e., elements of the domain) that have the same Lyapunov function value and the sublevel set includes all level sets with a lower Lyapunov function value. Furthermore, the sublevel sets of a Lyapunov function are invariant.

**Definition 2.5 (Level Set)**
*Let $f : \mathbb{R}^n \mapsto \mathbb{R}$ be a real-valued function. The* level set of a function $f$ at $c$ *is defined as*

$$L_{f,c} = \{\, \mathbf{x} \mid f(\mathbf{x}) = c \,\}.$$

*Likewise $L^+_{f,c} = \{\, \mathbf{x} \mid f(\mathbf{x}) \geq c \,\}$ and $L^-_{f,c} = \{\, \mathbf{x} \mid f(\mathbf{x}) \leq c \,\}$ are the* superlevel set *and* sublevel set*, respectively.* ♢

**Proposition 2.6**
*If a function $f$ is differentiable, then its gradient at an arbitrary point $\mathbf{x}$ is either $\mathbf{0}$, or perpendicular to the level set of $f$ at that point.*

Proposition 2.6 is exploited in the famous Lyapunov theorem (Theorem 2.31) in the sense that, for a stable system, the angle between gradient and the vectors describing the flow of the system is obtuse (more than right-angled) and thus pointing into the level set. This relation is explained in more detail in Section 2.4.1 which also contains a visualization (Figure 2.7b).

Another property of Lyapunov functions is *radial unboundedness*, i. e., $||\mathbf{x}|| \to \infty \Rightarrow f(\mathbf{x}) \to \infty$ which is achieved by stipulating that a Lyapunov function $V$ is bounded from below by the norm of the argument, i. e.,

$$\forall \mathbf{x} \in \mathbb{R}^n \bullet ||\mathbf{x}|| \leq aV(\mathbf{x}) \text{ for some } a \in \mathbb{R}_{>0}.$$

**Definition 2.7 (Euclidean Norm [Oeh11, Definition 3.3])**
*Let $\mathbf{x} = \begin{bmatrix} x_1 \dots x_n \end{bmatrix} \in \mathbb{R}^n$ be a real-valued vector. The norm*

$$||\mathbf{x}|| = \sqrt{\sum_{i=1}^{n} x_i^2}$$

*is the* euclidean norm *of* $\mathbf{x}$. ♢

Sometimes, we are only interested in a certain neighborhood of the origin. For example a hybrid system stabilizes only for values close to the origin. This property is called local asymptotic stability and can be verified by requiring the Lyapunov function to be valid only in an $\epsilon$ ball around the origin (see Section 2.3).

**Definition 2.8 (Closed Ball)**
*For a vector $\mathbf{x} \in \mathbb{R}^n$, we denote by*

$$\mathcal{B}_\varepsilon(\mathbf{x}) = \{\, \mathbf{y} \in \mathbb{R}^n \mid ||\mathbf{x} - \mathbf{y}|| \leq \varepsilon \,\}$$

*the* closed ball *around* $\mathbf{x}$ *with radius $\varepsilon \in \mathbb{R}_{\geq 0}$. As a shorthand notation, we use $\mathcal{B}_\epsilon$ to denote the closed ball $\mathcal{B}_\epsilon(\mathbf{0})$ around* $\mathbf{0}$. ♢

As already mentioned, a run of a hybrid system has both, continuous and discrete behavior. We use sequences of trajectory segments to describe the evolution of the continuous variables over time.

**Definition 2.9 (Sequence)**
*By $(s_i)$, we denote a* sequence *which can be either finite or infinite. The $i$-th element of $(s_i)$ is denoted by $s_i$ without parentheses.* ♢

In Chapter 3, we will focus on the automatic computation of Lyapunov functions and address some practical issues with the proposed method. To algebraically handle polynomials, let us introduce the following.

**Definition 2.10 (Polynomial Ring)**
$\mathbb{R}[\mathbf{x}]$, $\mathbf{x} = \begin{bmatrix} x_1 \dots x_n \end{bmatrix}$ *is the ring of polynomials in $n$ variables $x_1, \dots, x_n$ with real coefficients with proper definition of addition and multiplication.* ♢

**Definition 2.11 (Monomial)**
*A* monomial *$m(x) \in \mathbb{R}[\mathbf{x}], \mathbf{x} = \begin{bmatrix} x_1 \dots x_n \end{bmatrix}$, is a function $m : \mathbb{R}^n \mapsto \mathbb{R}$ of the form*

$$m(x_1, \dots, x_n) = \mathbf{x}^{\mathbf{d}} = x_1^{d_1} \cdots x_n^{d_n},$$

*where $\mathbf{d} = \begin{bmatrix} d_1 \dots d_n \end{bmatrix} \in \mathbb{N}^n_{\geq 0}$ is a vector of exponents. The* degree of *$m(\cdot)$ is defined* *as* degree$(m(\cdot)) = \sum_i d_i$. $\diamondsuit$

Now, we can define a polynomial as a weighted sum of monomials.

**Definition 2.12 (Polynomial)**
*A* polynomial $p(\mathbf{x}) \in \mathbb{R}[\mathbf{x}], \mathbf{x} = \begin{bmatrix} x_1 \dots x_n \end{bmatrix}$ *is a function* $p : \mathbb{R}^n \mapsto \mathbb{R}$ *of the form*

$$p(\mathbf{x}) = \sum_i^k a_i \cdot m_i(\mathbf{x}),$$

*where* $a_i \in \mathbb{R}$ *is a coefficient and* $m_i(\mathbf{x})$ *are monomials for all* $i = 1, \dots, k$. *The* degree of $p(\cdot)$ *is defined as* degree$(p(\cdot)) = \max_i(\{\text{degree}(m_i(\cdot))\})$. $\diamondsuit$

One of the key tools to efficiently compute Lyapunov functions is a relaxation to positive semidefinite programming. This relaxation will be introduced in Section 2.4.5. Positive semidefiniteness can be defined for functions and for matrices. With the help the Lyapunov theorem as introduced in Section 2.4, we obtain semidefiniteness conditions on functions which we will then relax to semidefiniteness conditions on matrices.

**Definition 2.13 (Definiteness)**
*Let* $f : D \rightarrow R$ *be a function, we define the following relations:*

- $f(\mathbf{x}) \succ 0$ *if and only if* $(\mathbf{x} = \mathbf{0} \implies f(\mathbf{x}) = 0) \wedge (\mathbf{x} \neq \mathbf{0} \implies f(\mathbf{x}) > 0)$,
- $f(\mathbf{x}) \succeq 0$ *if and only if* $(\mathbf{x} = \mathbf{0} \implies f(\mathbf{x}) = 0) \wedge (\mathbf{x} \neq \mathbf{0} \implies f(\mathbf{x}) \geq 0)$,
- $f(\mathbf{x}) \prec 0$ *if and only if* $(\mathbf{x} = \mathbf{0} \implies f(\mathbf{x}) = 0) \wedge (\mathbf{x} \neq \mathbf{0} \implies f(\mathbf{x}) < 0)$,
- $f(\mathbf{x}) \preceq 0$ *if and only if* $(\mathbf{x} = \mathbf{0} \implies f(\mathbf{x}) = 0) \wedge (\mathbf{x} \neq \mathbf{0} \implies f(\mathbf{x}) \leq 0)$.

*The function* $f$ *is called*

- *positive definite (PD) if and only if* $f(\mathbf{x}) \succ 0$ *for all* $\mathbf{x} \in D$,
- *positive semidefinite (PSD) if and only if* $f(\mathbf{x}) \succeq 0$ *for all* $\mathbf{x} \in D$,
- *negative definite if and only if* $f(\mathbf{x}) \prec 0$ *for all* $\mathbf{x} \in D$,
- *negative semidefinite if and only if* $f(\mathbf{x}) \preceq 0$ *for all* $\mathbf{x} \in D$.

*Let* $f$ *and* $g$ *be two functions. We write* $f(\mathbf{x}) \succeq g(\mathbf{x})$ *as a shorthand for* $f(\mathbf{x}) - g(\mathbf{x}) \succeq 0$. *The abbreviation* $f \succeq 0$ *is short for* $f(\mathbf{x}) \succeq 0$ *for all* $\mathbf{x}$. *Similar, a real symmetric matrix* $M : \mathbb{R}^{n \times n}$ *is called positive semidefinite, symbolically* $M \succeq 0$, *if and only if* $\mathbf{x}^T M \mathbf{x} \geq 0$ *for all* $\mathbf{x}$. *We stipulate to use analog abbreviations for* $\succ$, $\preceq$, *and* $\prec$. $\diamondsuit$

**Note 2.14**
*Positive (semi-)definiteness allows us to "hide" one universal quantifier. Instead of checking every non-zero argument (resp. non-zero column vector) by e. g. performing a costly quantifier elimination, we can check a property of the function (resp. matrix). As we will see later in Theorem 2.47, this can be done, for example, by checking the eigenvalues, which is very fast if done numerically.* ◁

As mentioned above, we require a Lyapunov function to be bounded from below by the norm of its argument. Actually, Lyapunov functions are bounded from below and from above by so-called $\mathcal{K}^\infty$ functions.

**Definition 2.15 (Monotonic Function [TT13])**
*Let $f : D \to \mathbb{R}$ be a continuous function. $f$ is called* (strictly) monotonically increasing *if and only if for each $\mathbf{x}_1, \mathbf{x}_2 \in D$, whenever $\mathbf{x}_1 < \mathbf{x}_2$, then $f(\mathbf{x}_1) \leq f(\mathbf{x}_2)$ (resp. $f(\mathbf{x}_1) < f(\mathbf{x}_2)$). The definition for a decreasing function can be obtained by changing the direction of the order.* ♢

**Definition 2.16 (Class $\mathcal{K}, \mathcal{K}^\infty$ Function [Kha00, Definition 4.2])**
*A continuous function $\alpha : [0, a) \to [0, \infty)$ is said to belong to class $\mathcal{K}$ if it is strictly monotonically increasing and $\alpha(0) = 0$. It is said to belong to class $\mathcal{K}^\infty$ if $a = \infty$ and $\alpha(x) \to \infty$ as $x \to \infty$.* ♢

Finally, we introduce convex sets which are used, for example to represent reachsets as in Section 4.3 and conic sets which are key to the so-called decompositional stability proofs by Oehlerking and Theel, which we present in Section 2.5.

**Definition 2.17 (Convex and Conic Sets and Hulls [Oeh11])**
*A set $X \subseteq \mathbb{R}^n$ is called* convex *if for every two points that lie in $X$, also the connecting line segment lies in $X$. Or formally,*

$$\mathbf{x}, \mathbf{y} \in X \Rightarrow \forall \lambda \in [0, 1] : \lambda(\mathbf{x} - \mathbf{y}) + \mathbf{y} \in X.$$

*If the cone spanned by two points in $X$ lies in $X$, then $X$ is called* conic. *Or formally,*

$$\mathbf{x}, \mathbf{y} \in X \Rightarrow \forall \lambda_1, \lambda_2 \geq 0 : \lambda_1 \mathbf{x} + \lambda_2 \mathbf{y} \in X.$$

*We can additionally exclude the* tip *of the cone by requiring $\lambda_1 + \lambda_2 > 0$. The hull is an operation on a set of elements $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^n$ that yields a set with the above properties. The* convex hull *is defined as*

$$\operatorname{conv}(\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}) := \left\{ \sum_i \lambda_i \mathbf{x}_i \;\middle|\; (\forall i : \lambda_i \geq 0) \wedge \sum_i \lambda_i = 1 \right\}$$

*and the* conic hull *is defined as*

$$\mathrm{cone}(\{\mathbf{x}_1,\dots,\mathbf{x}_n\}) := \left\{ \sum_i \lambda_i \mathbf{x}_i \,\middle|\, \forall i : \lambda_i \geq 0 \right\}.$$

*Additionally, we define*

$$\mathrm{cone}^+(\{\mathbf{x}_1,\dots,\mathbf{x}_n\}) := \left\{ \sum_i \lambda_i \mathbf{x}_i \,\middle|\, (\forall i : \lambda_i \geq 0) \wedge \sum_i \lambda_i > 0 \right\}.$$

*which, again, excludes the "tip". We define similar operators for functions. Let $f_i : \mathbb{R}^n \to \mathbb{R}^m, 1 \leq i \leq k$, be a familiy of real-valued functions. The* convex hull *is defined as*

$$\mathrm{conv}(\{f_1,\dots,f_n\}) := \left\{ \sum_i \lambda_i f_i \,\middle|\, (\forall i : \lambda_i \geq 0) \wedge \sum_i \lambda_i = 1 \right\},$$

*the* conic hull *is defined as*

$$\mathrm{cone}(\{f_1,\dots,f_n\}) := \left\{ \sum_i \lambda_i f_i \,\middle|\, \forall i : \lambda_i \geq 0 \right\},$$

*and*

$$\mathrm{cone}^+(\{f_1,\dots,f_n\}) := \left\{ \sum_i \lambda_i f_i \,\middle|\, (\forall i : \lambda_i \geq 0) \wedge \sum_i \lambda_i > 0 \right\}$$

*is the conic hull of functions excluding the "tip".* ◇

## 2.2 Hybrid Systems

After introducing these basic notations, we can focus on the system model which we use to describe dynamical systems throughout this thesis. We introduce hybrid automata as a means to capture the behavior of hybrid systems, that is, systems involving discrete as well as continuous components. A discrete component can be an embedded system that is controlling a physical (continuously evolving) environment. While the state of the environment might change continuously, the logical state of the embedded system might only changed at certain time-points—for example, dictated by a clock. To this end, we are concerned with deriving stability certificates for these kind of systems and analyze the hybrid automata that describe them.

A *hybrid system (HS)* is a system exhibiting, both, continuous evolution and discrete actions over time. Different ways of modeling have been proposed throughout the literature. Two, which are very common modeling concepts, are switched systems [Lib03] and hybrid automata [Alu+92]. Switched systems are often used in physics and control theory as they focus more on continuous evolution while hybrid automaton are more often found in computer science due to their relationship to state-machines. State-Machines allow capturing discrete behavior more easily.

**Definition 2.18 (Switched System ([cf. Lib03]))**
*A* switched system *is a hybrid system where the dynamics are represented as a set of differential equations* $\dot{\mathbf{x}} = f_{\sigma(t)}(\mathbf{x})$ *in the continuous time case* $t \in \mathit{Time}$ *or as a set of difference equations* $\mathbf{x}_{t+1} = f_{\sigma(t)}(\mathbf{x}_t)$ *in the discrete time case* $t \in \mathbb{N}$ *where* $f_i : \mathbb{R}^n \mapsto \mathbb{R}^n$, $1 \leq i \leq n$ *and* $\sigma(\cdot) : \mathit{Time} \mapsto [1, n]$ *is a* switching signal ♢

In the literature, it is sometimes distinguished between *time-dependent switching* where the value of $\sigma(\cdot)$ depends only on the time and *state-dependent switching* where the value $\sigma(\cdot)$ depends only on the state. However, every time-dependent switching signal can be transformed into a state-dependent switching signal by introducing an extra variable encoding the time but not vice versa. Another interesting special case is when the switching signal is not constraint at all, which we call *arbitrary switching*.

**Note 2.19**
*Stability of a system under arbitrary switching implies stability of a system with the same dynamics but constraint switching.* ◁

Another famous model for hybrid system are hybrid automata which were introduced by Alur et al. in [Alu+92; Alu+95]. Undecidability of the reachability problem in various aspects and decidability of the problem for very restrictive classes have been shown by Henzinger et al. in [Hen96; Hen+98]. Asarin et al. also gave a good summary and some more recent results in [Asa+12]. A hybrid automaton is often represented as a multigraph[1] made of vertices (called *modes* or *locations*) and edges (called *jumps* or *transitions*). The discrete actions (i.e. *guards* and *updates*) are annotated at the transitions between the locations and the continuous evolution is given in form of *invariants*, *differential equations*, or *differential inclusions* and annotated at the locations.

A hybrid automaton is defined as follows.

**Definition 2.20 (Hybrid Automaton (HA) [Oeh11, p.35])**
*A* hybrid automaton (HA) *is a sextuple*

$$\mathcal{H} = (\mathit{Var}, \mathit{Loc}, \mathit{Trans}, \mathit{Flow}, \mathit{Inv}, \mathit{Inits}) \text{ where}$$

- *Var is a finite set of* variables *and* $\mathcal{X} = \mathbb{R}^{|\mathit{Var}|}$ *is the corresponding* continuous state space*,*
- *Loc is a finite set of* locations *spanning the* discrete state space*,*

[1] A multigraph — in contrast to a simple graph — might have multiple edges between any two nodes. Thus, the set of all edges yields a multiset.

- *Trans is a finite set of* transitions $(l_1, G, U, l_2)$ *where*
  - $l_1, l_2 \in \mathit{Loc}$ *are the* source and target location *of the transition, respectively,*
  - $G \subseteq \mathcal{X}$ *is a* guard *which restricts the valuations of the variables for which this transition can be taken,*
  - $U : \mathcal{X} \to \mathcal{X}$ *is the* update function *which might update some valuations of the variables,*
- $\mathit{Flow} : \mathit{Loc} \to [\mathcal{X} \to \mathcal{P}(\mathcal{X})]$ *is the* flow function *which assigns a* flow *to every* location*. A flow* $f : \mathcal{X} \to \mathcal{P}(\mathcal{X})$ *in turn assigns a closed subset of* $\mathcal{X}$ *to each* $\mathbf{x} \in \mathcal{X}$*, which can be seen as the right-hand side of a differential inclusion* $\dot{\mathbf{x}} \in f(\mathbf{x})$*,*
- $\mathit{Inv} : \mathit{Loc} \to \mathcal{P}(\mathcal{X})$ *is the* invariant function *which assigns a closed subset of the continuous state space to each* location $l \in \mathit{Loc}$*, and therefore restricts valuations of the variables for which this* location *can be active,*
- $(l, \mathit{Init}) \in \mathit{Inits} \subseteq \mathit{Loc} \times \mathcal{X}$ *is a closed set of* initial (hybrid) states *where* $l$ *is the* initial discrete state *and Init is the* initial continuous state.

◇

The above definition is also sometimes called an *initialized hybrid automaton*, because *Inits* restricts the initial states of the system. The special case where *Inits* does not restrict the initial states, is called an *uninitialized hybrid automaton.* In this case, we might omit *Inits* completely and denote $\mathcal{H}$ by the quintuple $(\mathit{Var}, \mathit{Loc}, \mathit{Trans}, \mathit{Flow}, \mathit{Inv})$. For an uninitialized hybrid automaton, the initial stats correspond to the union of all invariants, i. e.,

$$\mathit{Inits} = \{\, (l, \mathit{Inv}(l)) \mid l \in \mathit{Loc} \,\}.$$

Throughout the thesis, we will focus on *polynomial hybrid system* as this is the system class that allows us to use automated stability verification.

**Definition 2.21**
*A hybrid automaton* $\mathcal{H} = (\mathit{Var}, \mathit{Loc}, \mathit{Trans}, \mathit{Flow}, \mathit{Inv}, \mathit{Inits})$ *is called*

- linear *if and only if flows, invariants, guards, and updates are described by linear expressions,*
- polynomial *if and only if flows, invariants, guards, and updates are described by polynomials,*
- update-free *if and only if all updates are the identity function.*

◇

**Remark 2.22**
*Often, we describe a hybrid automaton graphically. The representation of a hybrid automaton is similar to that of a finite-state machine with additional annotations. The underlying graph consists of vertices, one for each location or mode, and edges, one for each transition. Vertices are labeled by a name, ordinary differential equations (ODEs) or inclusions defining the continuous flow, and an invariant. Edges are labeled with guards and updates. Guards and invariants are represented by predicates over the continuous variables and resets are described by assignments* $v := f(\mathbf{x})$ *where* $v \in \mathit{Var}$ *is the continuous variable to be updated and* $f : \mathcal{X} \to \mathbb{R}$ *is a function taking the old valuation — before taking the transition — as its argument and computes the new value. To increase readability, we allow to omit updates for variables whose value does not change due to the transition. Initial states are described by arrows having a target but no source location where the target location denotes the initial location and an annotated predicate defines the set of valid initial valuations for the continuous variables like a guard.* ◁

Finally, let us formally define the execution semantics of a hybrid automaton which describes the evolution of the state of an hybrid automaton.

**Definition 2.23 (Runs of a Hybrid Automaton [Oeh11, Definition 3.13])**
*Let* $\mathcal{H} = (\mathit{Var}, \mathit{Loc}, \mathit{Trans}, \mathit{Flow}, \mathit{Inv}, \mathit{Inits})$ *be a hybrid automaton and let* $(t_i)$ *be a — possibly infinite — sequence of* switching times *for which holds*

- $t_i \in \mathit{Time}_\infty$,
- $t_0 = 0$,
- $\forall i > 0 : t_{i-1} \leq t_i$,
- *at most the last element of the sequence is* $\infty$.

*A* run *of* $\mathcal{H}$ *is a — possibly infinite — sequence of tuples* $(\pi_i) = ((l_i, \mathbf{x}_i))$ *for which holds*

- $l_i \in \mathit{Loc}$ *is a location of* $\mathcal{H}$,
- $\mathbf{x}_i : [t_i, t_{i+1}] \mapsto \mathcal{X}$ *(or* $\mathbf{x}_i : [t_i, t_{i+1}) \mapsto \mathcal{X}$*, in case of* $t_{i+1} = \infty$*), is a function which is absolutely continuous on* $[t_i, t_{i+1}]$ *(or* $[t_i, t_{i+1})$ *in case of* $t_{i+1} = \infty$*),*
- $(t_i)$ *is the sequence of switching times where either both sequences* $(t_i)$ *and* $(\pi_i)$ *are infinite or the sequence* $(t_i)$ *is one element longer than the sequence* $(\pi_i)$,

*such that*

1. $\pi_0 = (l_0, \mathbf{x}_0) \in \mathit{Inits}$,

2. *for all* $t_i \leq t < t_{i+1}$ *holds* $\mathbf{x}_i(t) \in \mathit{Inv}(l_i)$,

3. *for almost all* $t_i \leq t < t_{i+1}$ *holds* $\dot{\mathbf{x}}_i(t) \in \mathit{Flow}(l_i)(\mathbf{x}_i(t))$

4. *for all* $t_{i+1}$, $i > 0$, *but the last element of* $(t_i)$, *there exists a transition* $(m_i, G, U, m_{i+1}) \in \mathit{Trans}$ *with*
   a) $\lim_{t \nearrow t_{i+1}} \mathbf{x}_i(t) \in G$ *and*
   b) $\mathbf{x}_{i+1}(t_i) = U(\mathbf{x}_i(t_i))$.

5. *if the sequence* $(t_i)$ *is infinite, then* $\lim_{i \to \infty} t_i = \infty$.

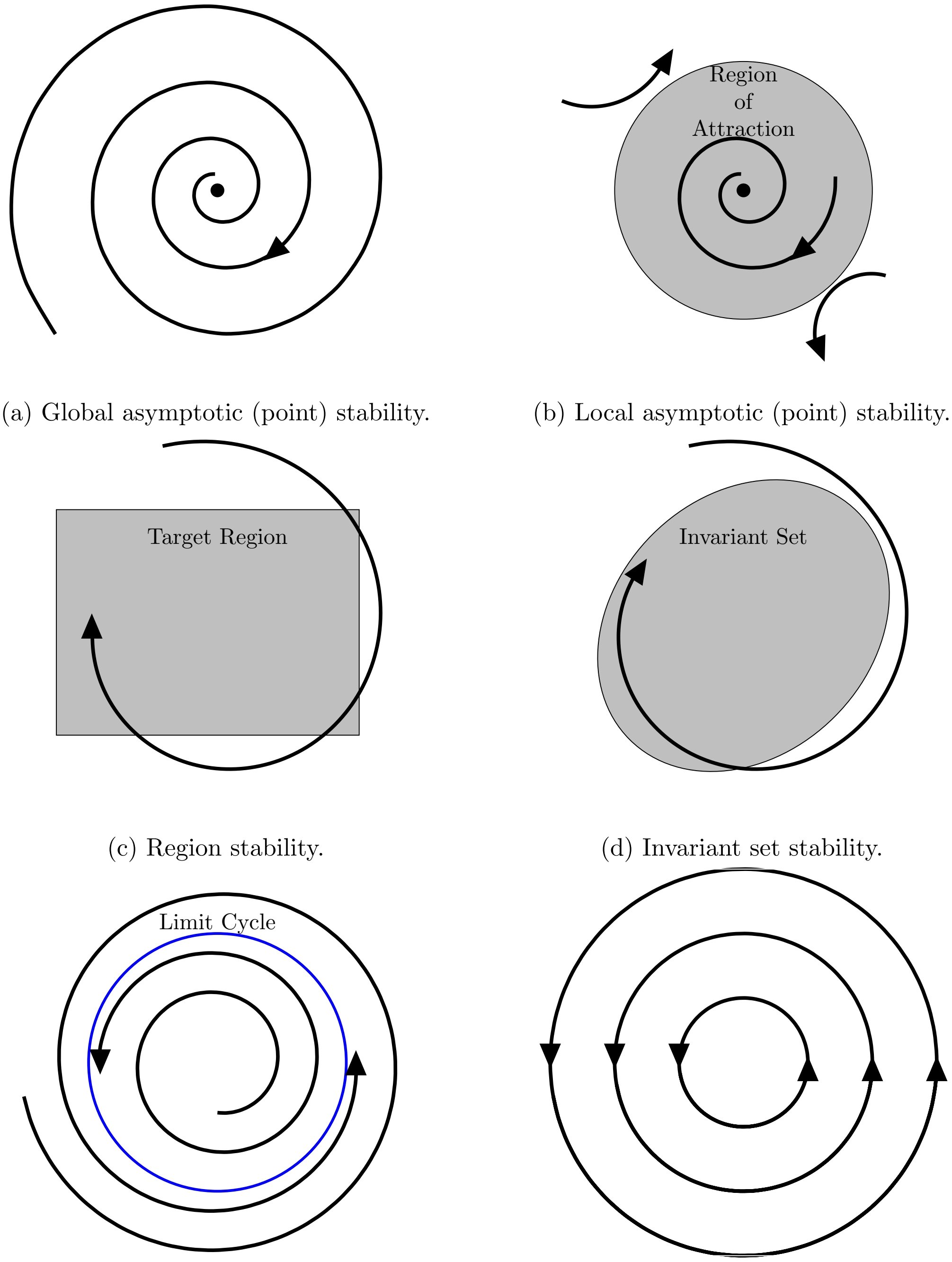
*We call the run* infinite *if* $(t_i)$ *is infinite and diverges to infinity or* $(t_i)$ *is finite and its last element is* $\infty$. *We call the run* finite *otherwise. The sequence* $(m_i)$ *is called the* location sequence. ♢

**Definition 2.24 (Discrete, Continuous, and Hybrid State [Oeh11])**
*Let* $\mathcal{H} = (\mathit{Var}, \mathit{Loc}, \mathit{Trans}, \mathit{Flow}, \mathit{Inv}, \mathit{Inits})$, *let* $(\pi_i) = (l_i, \mathbf{x}_i)$ *be a run of* $\mathcal{H}$. *For any time* $t \in \mathit{Time}$, *the value* $l_i$ *with uniquely determined index* $i$ *such that* $t_i \leq t < t_{i+1}$ *is the* discrete state *and denoted by* $l(t) : \mathit{Time} \mapsto \mathit{Loc}$. *Analogously, the value of the function* $\mathbf{x}_i(t)$ *such that* $t_i \leq t < t_{i+1}$ *is the* continuous state *at time* $t$ *and is denoted by* $l(t)$. *The continuous state before taking a transition is defined as* $\mathbf{x}(t^-_{i+1}) := \mathit{lim}_{t \to t_{i+1}} \mathbf{x}(t)$. *Any function* $\mathbf{x} : D \mapsto \mathcal{X}$ *with* $D = [0, T]$, $T \in \mathit{Time}$ *or* $D = \mathit{Time}$ *such that for all* $t'$, *the vector* $\mathbf{x}(t')$ *is the continuous state at time* $t'$, *is called a* trajectory *of the system. By* $\pi(t) = (l(t), \mathbf{x}(t))$ *we denote the* hybrid trajectory *of* $\mathcal{H}$ *describing the* hybrid state *at time* $t$. *We call a hybrid trajectory* extendable, *if it is a prefix of another hybrid trajectory and* non-extendable *otherwise.* ♢

## 2.3 Stability

Having introduced hybrid automata as the system model of our choice, we can now focus on the property we are interested in. This property is called stability and basically expresses that all trajectories of the system eventually reach an equilibrium point (or region) of the sub-state space and stay in that point (or region) forever. There are many different forms of stability that can be distinguished. The six most common ones are visualized in Figure 2.2 and explained in more detail in this section.

**Global Asymptotic (Point) Stability** [cf. Kha00] is the stability notation we focus on in this thesis. It will be formally defined in Definition 2.26. All trajectories, no matter where they start will, eventually approach an equilibrium point, which is usually assumed to be the origin. A trajectory is allowed veer away from the equilibrium point but not arbitrarily far and eventually has to converge to the equilibrium point. Neither does the trajectory need to finally reach the equilibrium point nor does it need to maintain a



(a) Global asymptotic (point) stability.

(b) Local asymptotic (point) stability.

(c) Region stability.

(d) Invariant set stability.

(e) Limit cycle stability.

(f) Orbital stability.

Figure 2.2: The six common stability notions.

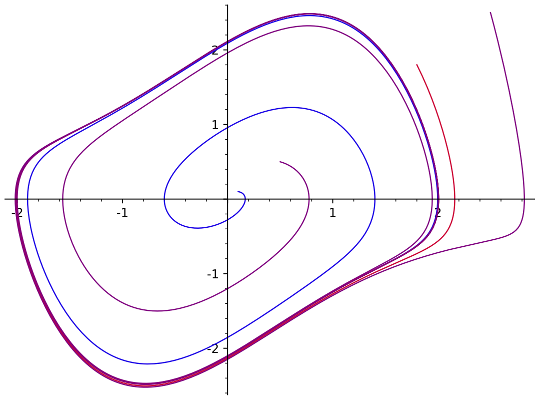


Figure 2.3: Phase Portrait of the Van der Pol Equation.

minimal convergence rate. In fact, the time a trajectory needs for decreasing the distance to the equilibrium point by a fixed amount may grow arbitrarily. If the decay is bounded by an exponential function, then we call the system globally exponentially stable (GES).

**Local Asymptotic (Point) Stability** is obtained by restricting global asymptotic (point) stability to those trajectories that start in a neighborhood around the equilibrium point [Kha00]. A similar restriction can be applied to global exponential (point) stability in order to obtain local exponential (point) stability.

Global asymptotic stability and its local counterpart are the two most popular stability notions and are part of most lectures and books on stability.

**Region Stability** replaces the equilibrium point by a region that the system's trajectories are allowed to enter and leave finitely often and ultimately have to stay within that region. This notion has been studied by Podelski and Wagner in [PW06; PW07a; PW07c].

**Invariant Set Stability** requires every trajectory to ultimately enter and never leave a certain region or set of states. This kind of stability notation has been studied in [Kha96; YMH98; CGT08; RS10].

**Limit Cycle Stability** denotes that every trajectory of the system ultimately converges towards a single periodic behavior, called a *limit cycle* or *orbit* [Kha00]. In that sense, trajectories do not converge towards a point or a region or set but to a behavior that can be described as a trajectory on its own. An well-known system that exhibits such a behavior can be found in the electrical circuits employing vacuum tubes and can be described by the Van der Pol equation depicted in Figure 2.3. The figure shows several trajectories each converging to the same (infinitely repeating) periodic behavior.

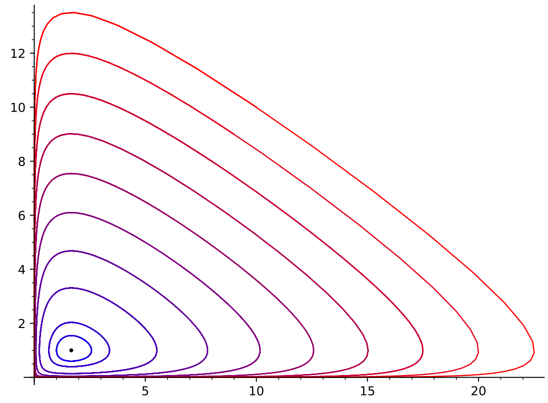


Figure 2.4: Phase Portrait of the Lotka-Volterra Equation.

**Orbital Stability** allows the trajectories to have different periodic behaviors and we have that a trajectory starting $\epsilon$-close to an orbit will remain $\epsilon$-close to that orbit [Che05]. One example of such a system can be found in biology where two dependent populations (predator and prey) change over time. This can be modeled by the Lotka-Volterra equations and this is depicted in Figure 2.4. The figure shows trajectories of the evolution of the size of the populations that depending on the initial state follow a different periodic behavior.

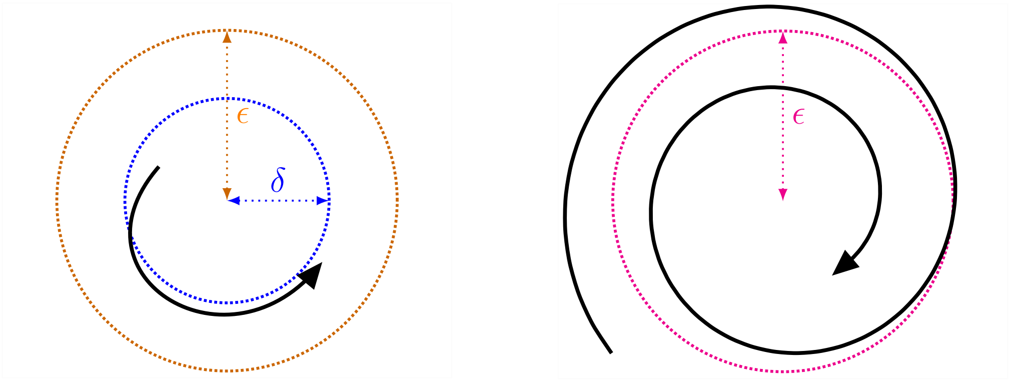
### Global Asymptotic Stability

Usually, for technical reasons, the equilibrium point is assumed to be the origin $\mathbf{0}$ of the continuous state space. This is not a restriction since a system can always be shifted by $x' := x - x_e$.

**Proposition 2.25**
*For every hybrid automaton $\mathcal{H}$ with equilibrium point $x_e \neq 0$ one can construct a hybrid automaton $\mathcal{H}'$ with equilibrium point $x_e = 0$ via a coordinate transformation.*

In the sequel, we focus on *asymptotic stability* which does not require the equilibrium point to be reached in finite time, but only requires every trajectory to converge. This property is weaker than *exponential stability* where the existence of an exponential convergence rate is additionally required.

**Definition 2.26 (Global asymptotic stability with respect to a subset of variables [Oeh11])**
*Let $\mathcal{H}$ be a continuous-time dynamical system with variables $Var$ and let $Var^S \subseteq Var$ be a subset of variables that are required to converge to the equilibrium point $\mathbf{0}$. The*



(a) Visualization of Lyapunov stability.

(b) Visualization of global attractivity.

Figure 2.5: Visualization of global asymptotic stability.

*system* $\mathcal{H}$ *is called* Lyapunov stable (LS) with respect to $Var^S$ *if for all trajectories* $\mathbf{x}(\cdot)$ *of* $\mathcal{H}$*,*

$$\forall\epsilon>0 \bullet \exists\delta>0 \bullet \forall t \geq 0 \bullet ||\mathbf{x}(0)|| < \delta \Rightarrow \left|\left|\mathbf{x}_{\downarrow Var^S}(t)\right|\right| < \epsilon.$$

$\mathcal{H}$ *is called* globally attractive (GA) with respect to $Var^S$ *if for all trajectories* $\mathbf{x}(\cdot)$ *of* $\mathcal{H}$*,*

$$\lim_{t\to\infty} \mathbf{x}_{\downarrow Var^S}(t) = \mathbf{0},\ i.\,e.,\forall\epsilon>0 \bullet \exists t_0 \geq 0 \bullet \forall t > t_0 \bullet \left|\left|\mathbf{x}_{\downarrow Var^S}(t)\right|\right| < \epsilon,$$

*where* $\mathbf{0}$ *is the origin of* $\mathbb{R}^{|Var^S|}$*. If a system is both, globally stable with respect to* $Var^S$ *and globally attractive with respect to* $Var^S$*, then it is called* globally asymptotically stable (GAS) with respect to $Var^S$. ◇

Intuitively, LS means that trajectories starting $\delta$-close to the origin remain $\epsilon$-close to the origin. GA means that for each $\epsilon$-distance to the origin, there exists a point in time $t_0$ such that a trajectory always remains within this distance. It follows that each trajectory is eventually always approaching the origin. This property can be proven using Lyapunov theory [Lya07] and is discussed in more detail in Section 2.4.1.

There is also a local version of asymptotic stability called *local asymptotic stability*. The definition can be obtained from Definition 2.26 by considering only those $\mathbf{x}(\cdot)$ which start in a certain neighborhood $\mathcal{B}_\epsilon$ of $\mathbf{0}$, i. e., $\mathbf{x}(0) \in \mathcal{B}_\epsilon$ for some $\varepsilon$.

Sometimes, we are not only interested in convergence but we would also like to have the trajectories converge as fast as a certain fixed rate. This can be achieved by adding one extra requirement to Definition 2.26.

**Definition 2.27 (Global Exponential Stability [Pet99])**
*A continuous-time dynamical system* $\mathcal{H}$ *is* globally exponentially stable *if*

- *$\mathcal{H}$ is globally asymptotically stable and*
- *there exists $c > 0, \mathsf{r} > 0$ such that for all trajectories $\mathbf{x}(\cdot)$ of $\mathcal{H}$, it holds that*

$$||\mathbf{x}(t)|| \leq c \cdot ||\mathbf{x}(0)|| \cdot e^{-\mathsf{r} \cdot t}.$$

$\mathsf{r}$ *is also called the* exponential rate*.* ♢

## 2.4 Stability Verification

In the last sections, we have introduced the system model under consideration and the property we are interested in. The question remains how to show that all (potentially infinitely many) trajectories (of potentially infinite length) satisfy this property. One way of proving satisfaction of the property is given by Lyapunov theory and we will show its application in the following.

In this section, we will incrementally construct a hybrid automaton modeling a simple velocity controller. The purpose of the velocity controller is to keep the velocity $v$ of a vehicle at a desired, user-chosen, velocity $v_e$. We consider the unit of the velocity $v, v_e$ to be m/s and the desired velocity to be arbitrary but fixed during our analysis. For figures, we will pick $v_e = 25$m/s.

**Example 2.28 (Running Example of Chapter 2)**
*We start with a very common and simple but yet powerful dynamical system: a PI-controller with the following differential equation:*

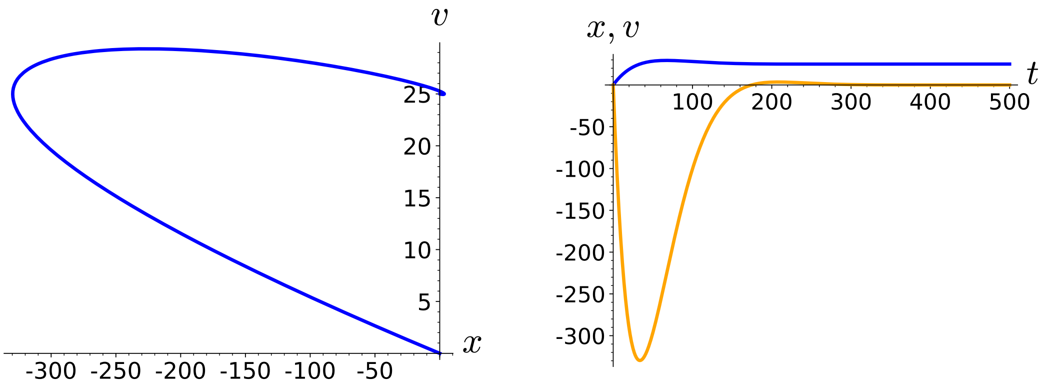
$$\begin{aligned}\dot{v} &= -0.001x - 0.052 \cdot (v - v_e)\\ \dot{x} &= v - v_e\end{aligned}$$

*$x$ is the integrator variable, $v$ is the current velocity, and $v_e$ is the desired velocity (equilibrium point). Figure 2.6 shows a trajectory of the system as a phase plot and as a plot over time.* ◁

### 2.4.1 Lyapunov Theorem

Stability can be proven using Lyapunov Theory [Lya07]. Lyapunov Theory was originally restricted to continuous systems but has been lifted to hybrid systems.

**Theorem 2.29 (Lyapunov Functions [Lya07].)**
*Let $\dot{\mathbf{x}} = f(\mathbf{x})$ with $f(\mathbf{x}) : \mathbb{R}^n \mapsto \mathbb{R}^n$ be a dynamical system with the equilibrium $x_e$. If there exists three $\mathcal{K}^\infty$ functions $\alpha, \beta, \gamma$ and a continuously differentiable function $V : \mathbb{R}^n \to \mathbb{R}$ such that*



(a) Phase plot (x over v).

(b) Time plot of $v$ (blue) and $x$ (orange).

Figure 2.6: Example trajectory of the dynamical system from Example 2.28.

*(C1)*

$$\forall \mathbf{x} \bullet \alpha(||\mathbf{x}||) \leq V(\mathbf{x}) \leq \beta(||\mathbf{x}||) \text{ and}$$

*(C2)*

$$\forall \mathbf{x} \bullet \langle \nabla V(\mathbf{x}) \mid f(\mathbf{x}) \rangle \leq -\gamma(||\mathbf{x}||),$$

*then the dynamical system $\dot{\mathbf{x}} = f(\mathbf{x})$ is global asymptotic stability (GAS) and $V$ is called a* Lyapunov function (LF).

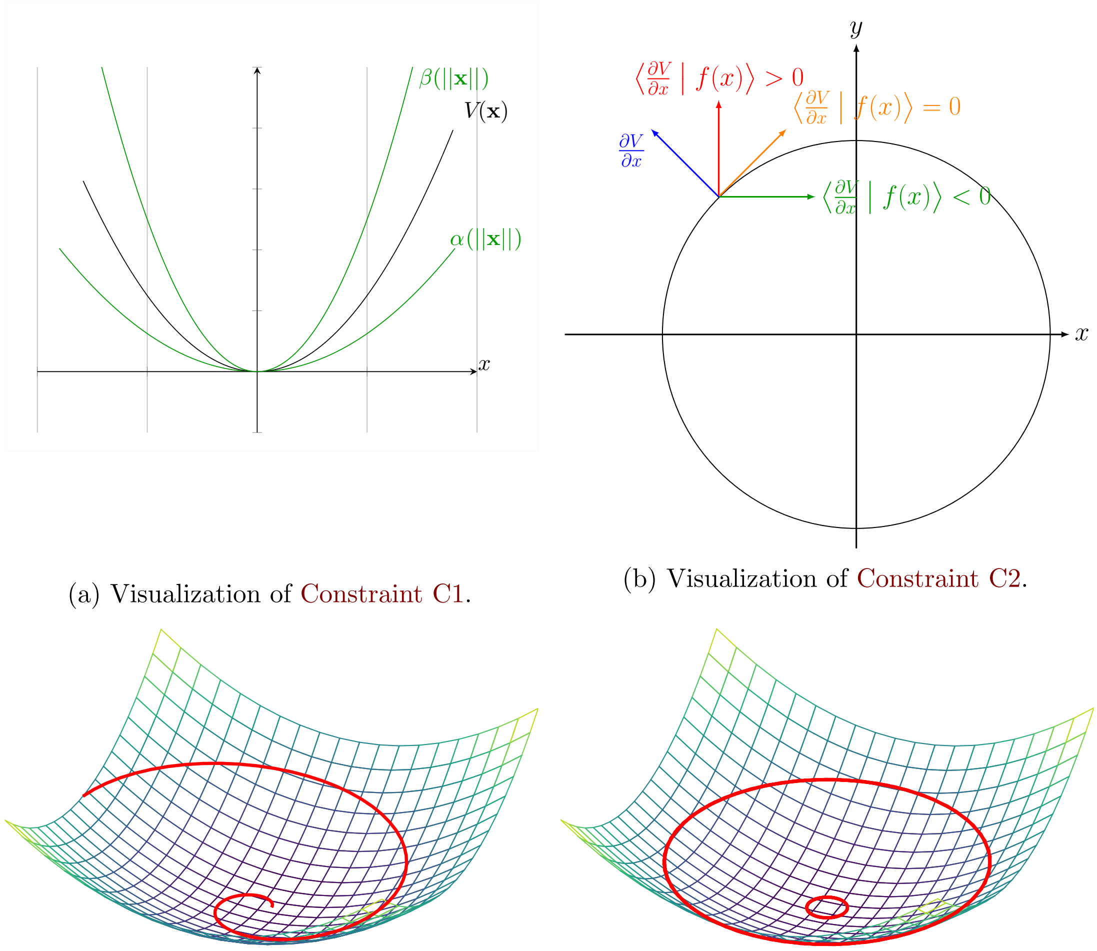
In particular interesting is that the Lyapunov theorem does not require to explicitly investigate a single run of a hybrid automaton. Instead, it establishes relations on all hybrid states and their successor states such that every run approaches the origin of the state space.

In the following we give a short intuition for the individual constraints of the Lyapunov theorem. Figure 2.7a visualizes Constraint C1. It shows that the two $\mathcal{K}^\infty$ functions $\alpha, \beta$ bound the Lyapunov function $V(\mathbf{x})$ from below and from above, respectively. $\alpha$ ensures that $V$ is positive everywhere and radially unbounded, i. e., $\mathbf{x} \to \infty \Rightarrow V(\mathbf{x}) \to \infty$ and $\beta$ ensures that $\mathbf{x} \to \mathbf{0} \Rightarrow V(\mathbf{x}) \to 0$. Figure 2.7b visualizes Constraint C2. First, recall the definition of the angle $\measuredangle(\mathbf{x}, \mathbf{y})$ between two non-zero vectors $\mathbf{x}, \mathbf{y}$:

$$\langle \mathbf{x} \mid \mathbf{y} \rangle := \cos \measuredangle(\mathbf{x}, \mathbf{y}) \cdot ||\mathbf{x}|| \cdot ||\mathbf{y}||.$$

Considering the relation of the angle and the sign of the inner product gives the following important property

$$\langle \mathbf{x} \mid \mathbf{y} \rangle \begin{cases} > 0 & \text{if } \measuredangle(\mathbf{x}, \mathbf{y}) \in [0, \frac{\pi}{2}) \\ = 0 & \text{if } \measuredangle(\mathbf{x}, \mathbf{y}) = \frac{\pi}{2} \\ < 0 & \text{if } \measuredangle(\mathbf{x}, \mathbf{y}) \in (\frac{\pi}{2}, \pi] \end{cases}$$


$\beta(||\mathbf{x}||)$
$V(\mathbf{x})$
$\alpha(||\mathbf{x}||)$
$x$
$y$
$\langle \frac{\partial V}{\partial x} \mid f(x) \rangle > 0$
$\langle \frac{\partial V}{\partial x} \mid f(x) \rangle = 0$
$\frac{\partial V}{\partial x}$
$\langle \frac{\partial V}{\partial x} \mid f(x) \rangle < 0$
$x$

(a) Visualization of Constraint C1.

(b) Visualization of Constraint C2.

(c) Strictly decreasing Lyapunov function and a corresponding trajectory.

(d) Non-increasing Lyapunov function and two corresponding trajectories.

Figure 2.7: Visualization of the Lyapunov theorem (continuous case).

which is exploited in Constraint C2 as it requires the inner product of the gradient of the Lyapunov function and the vector-valued flow function $f(\mathbf{x})$ to be negative. Since — by Constraint C1 — the Lyapunov function is $\mathbf{0}$ at the origin and non-negative everywhere, the gradient of the Lyapunov function points outwards (see also Proposition 2.6). Now, additionally requiring negativity of the inner product between gradient and flow function implies that the flow function points inwards as visualized in Figure 2.7b on any orbit of the Lyapunov function $V(\mathbf{x}) = c$.

Figure 2.7c visualizes the relation between the Lyapunov function and a trajectory of a system. The $x$-$y$ axes span the system's state space while the $z$-axis corresponds ot the Lyapunov function's value. The Lyapunov function is drawn in blue and the trajectory in red. It can be seen that the Lyapunov function decreases while the trajectory converges towards the equilibrium point $\mathbf{0}$. Modifying the constraint of Constraint C2 such that the value of the Lyapunov function is required to be non-increasing instead of decreasing, we can have the situation shown in Figure 2.7d. The figure shows two trajectories on a stable orbit and the inner product is 0.

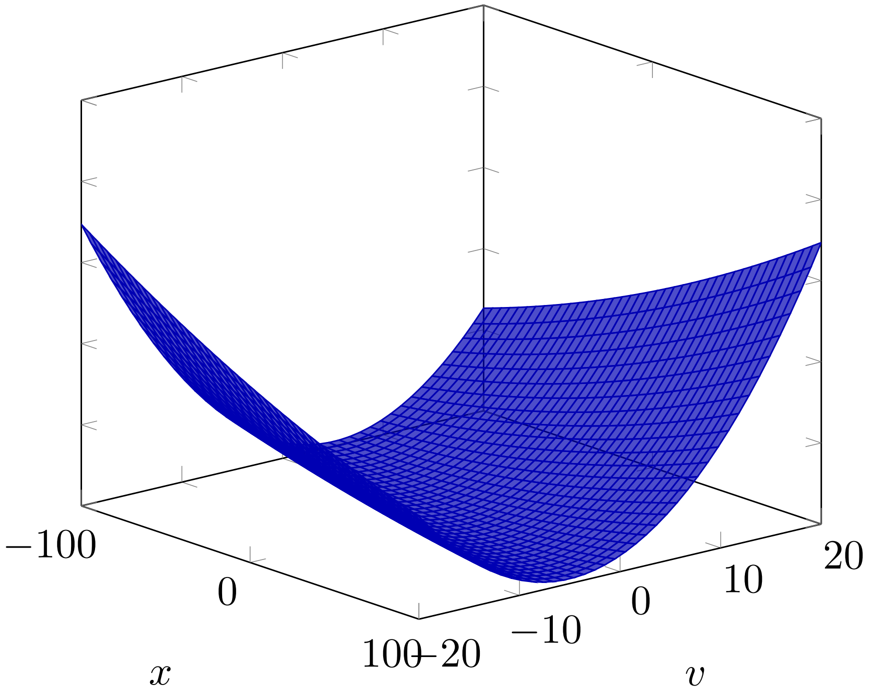


Figure 2.8: A Lyapunov Function for the Dynamical System from Example 2.28.

Let us return to our running example. Using Theorem 2.29, we can prove that the dynamical system from Example 2.28 is stable and

$$11.004257v \cdot x + 2.862176x^2 + 10.630175v^2$$

is a valid Lyapunov function[2]. The Lyapunov function is shown in Figure 2.8.

[2] automatically obtained by STABHYLI, scaled and rounded to six digits

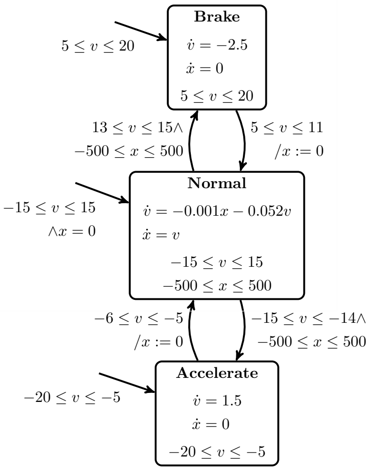


Figure 2.9: Hybrid Automaton Modeling a Saturated Velocity Controller.

**Example 2.30 (Continuation of Example 2.28)**
*Now, suppose we want to saturate the acceleration by at most* $a_{\mathtt{max}} = 1.5\mathrm{m}/\mathrm{s}^2$ *and the deceleration by at most* $a_{\mathtt{min}} = -2.5\mathrm{m}/\mathrm{s}^2$. *We could do this by introducing two new operation modes covering the operation ranges for which the differential equation of* $v$ *would exceed the given limits. This is shown in Figure 2.9. The previous behavior of the dynamical system is captured by the location* `Normal` *and the saturation is achieved by the two extra locations* `Brake` *and* `Accelerate` *where in location* `Brake` *we set* $\dot{v} = a_{\mathtt{max}}$ *and in location* `Accelerate` *we set* $\dot{v} = a_{\mathtt{min}}$. *We have also introduced initial conditions on the locations represented as transitions without a source location and transitions between locations that define proper switching. The incoming transitions of the location* `Normal` *additionally resets the integrator variable, i. e.,* $x := 0$.

*To achieve more convenient driving experience, one would, however, introduce more locations guaranteeing smother behavior, i. e., reducing the jerk.* ◁

Since our running example is hybrid, we need a version of the Lyapunov theorem, that additionally covers switching between operation modes. The straightforward version is to introduce a third constraint that ensures that the Lyapunov function decreases whenever a run takes a transition: for each $(l_1, G, U, l_2) \in \mathit{Trans}$,

$$\forall \mathbf{x} \in G \bullet V(U(\mathbf{x})) \leq V(\mathbf{x}).$$

This additional constraint basically relates the Lyapunov function of different regions of the state space. This mechanism can easily be extended to allow each location to have its own Lyapunov function and have an additional constraint type relating the individual Lyapunov functions. This leads to the following theorem:

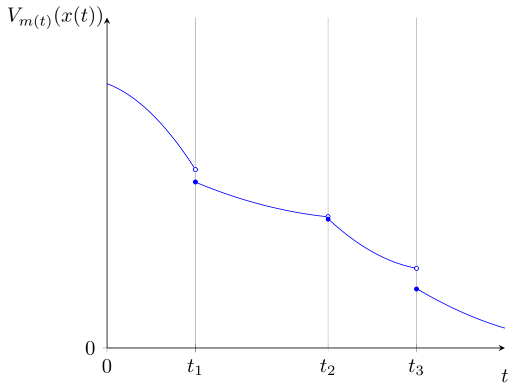


Figure 2.10: Visualization of Constraint C3 of the Lyapunov Theorem (Hybrid Extension).

**Theorem 2.31 (Discontinuous Lyapunov functions for a subset of variables [Oeh11])**
*Let $\mathcal{H} = (Var, Loc, Trans, Flow, Inv)$, be a hybrid automaton and let $Var^S \subseteq Var$ be a set of variables that are required to converge. If for each $l \in Loc$, there exists a set of variables $Var_l$ with $Var^S \subseteq Var_l \subseteq Var$ and a continuously differentiable function $V_l : \mathcal{X} \rightarrow \mathbb{R}$ such that*

*(C1) for each $l \in Loc$, there exist two class $\mathcal{K}^\infty$ functions $\alpha$ and $\beta$ such that*

$$\forall \mathbf{x} \in Inv(l) \bullet \alpha\big(||\mathbf{x}_{\downarrow Var_l}||\big) \leq V_l(\mathbf{x}) \leq \beta\big(||\mathbf{x}_{\downarrow Var_l}||\big),$$

*(C2) for each $l \in Loc$, there exists a class $\mathcal{K}^\infty$ function $\gamma$ such that*

$$\forall \mathbf{x} \in Inv(l) \bullet \dot{V}_l(\mathbf{x}) \leq -\gamma\big(||\mathbf{x}_{\downarrow Var_l}||\big)$$

*for each $\dot{V}_l(\mathbf{x}) \in \{ \langle \nabla V_l(\mathbf{x}) \mid f(\mathbf{x}) \rangle \mid f(\mathbf{x}) \in Flow(l) \}$,*

*(C3) for each $(l_1, G, U, l_2) \in Trans$,*

$$\forall \mathbf{x} \in G \bullet V_{l_2}(U(\mathbf{x})) \leq V_{l_1}(\mathbf{x}),$$

*then $\mathcal{H}$ is globally asymptotically stable with respect to $Var^S$. Each $V_l$ is called a* local Lyapunov function (LLF) *of location $l$, and the function $V(l, \mathbf{x}) = V_l(\mathbf{x})$ is called* global Lyapunov function (GLF)*.*

*In this thesis, we denote by* location constraints *the Constraint C1 and the Constraint C2 and by* transition constraints *the Constraint C3.*

Figure 2.10 visualizes the Constraint C3. This constraint ensures that while switching from one location to another the Lyapunov function's value does not increase.

A localized version of the theorem — proving local asymptotic stability — can be obtained by restricting the constraints to a certain neighborhood of the origin $\mathcal{B}_\epsilon$ for some $\varepsilon$.

Global exponential stability (see Definition 2.27) can also be proven by Lyapunov theory by enforcing a particular shape for the $\mathcal{K}^\infty$ functions: $\alpha(||\mathbf{x}||) = a||\mathbf{x}||^d$, $\beta(||\mathbf{x}||) = b||\mathbf{x}||^d$, $\gamma(||\mathbf{x}||) = g||\mathbf{x}||^d$ where $a, b, c, d$ being positive constants. Global exponential stability is in particular interesting since this property allows us to estimate (1) the state of the system and (2) the convergence time. By setting $\mathsf{r} = g/b$ we can conclude from the Lyapunov theorem that

$$\dot{V}(\mathbf{x}) \leq -\gamma(||\mathbf{x}||) = -g||\mathbf{x}||^d \leq -\frac{g}{b}V(\mathbf{x}) = -\mathsf{r} \cdot V(\mathbf{x}),$$

This is a differential inclusion in a single variable $V(\mathbf{x})$ representing the Lyapunov function's value. This differential inclusion has the solution

$$V(\mathbf{x}(t)) \leq \exp(-\mathsf{r} \cdot t) \cdot V(\mathbf{x}(0)).$$

Hence, we can conclude global exponential stability.

We can also compute an upper bound on the convergence time as follows. Given a set of initial states *Init* and a target set $T$, we can compute two values $c_{Init}, c_T$ such that

$$\begin{aligned} \mathbf{x} \in Init &\Rightarrow V(\mathbf{x}) \leq c_{Init} \\ V(\mathbf{x}) \leq c_T &\Rightarrow \mathbf{x} \in T. \end{aligned}$$

Note that these values correspond to two level sets; one set that contains *Init* and one set that is contained in $T$. This allows us to compute the maximum time spent outside of the target set using the above solution of the differential inclusion as

$$\begin{aligned} & c_T \leq \exp(-\mathsf{r} \cdot t) \cdot c_{Init} \\ \Leftrightarrow \quad & t \leq \frac{1}{\mathsf{r}} \ln\left(\frac{c_{Init}}{c_T}\right). \end{aligned}$$

Thus, after time $t$ any trajectory starting in *Init* has entered $T$.

We will make related investigations in Section 4.3 where we obtain so-called safe (level) sets. A safe (level) set is a set for which it is guaranteed that a trajectory — once entered — will not leave the set and moreover will not reach an unsafe set.

### 2.4.2 S-Procedure

The $\mathcal{S}$-Procedure is a powerful technique to transform a conditioned constraint into an unconditioned constraint. Recall the location and transition constraints in Theorem 2.31. The constraints have the form $\forall \mathbf{x} \in R : f(\mathbf{x}) \geq 0$, we call $R \subseteq \mathbb{R}^n$ a *region* which may restrict the values for which $f : \mathbb{R}^n \mapsto \mathbb{R}$ has to be non-negative. If the region is the full state space, i. e., $R = \mathcal{X}$, then we call the constraint *unconditioned* and *conditioned* otherwise. As we consider polynomial hybrid systems and, thus, guards and invariants are given as boolean combinations over non-negativity constraints on polynomials, we can assume, wlog, that each region $R$ is given as a conjunction of equalities and inequalities.[3]

**Theorem 2.32 ($\mathcal{S}$-Procedure [Boy+94, p.23])**
*Let $f_i : \mathbb{R}^n \mapsto \mathbb{R}$ for $0 \leq i \leq m$. If there exists non-negative multipliers $\lambda_i \geq 0$, such that*

$$\forall \mathbf{x} \in \mathbb{R}^n \bullet \left( \sum_{i=1}^{m} \lambda_i \cdot f_i(\mathbf{x}) \right) \leq f_0(\mathbf{x})$$

*then*

$$\forall \mathbf{x} \in \left\{ \mathbf{y} \;\middle|\; \bigwedge_{i=1}^{m} 0 \leq f_i(\mathbf{y}) \right\} \bullet 0 \leq f_0(\mathbf{x}).$$

Note that if two functions $f_i(\cdot) \geq 0$ and $f_j(\cdot) \geq 0$ are used to encode equality, e. g., $f_i = -f_j$, then it is better to include only one function $f_i$ and remove the positivity requirement on $\lambda_i$. This reduces the number of newly introduced *free parameters* or *decision variables.*

In general, the $\mathcal{S}$-Procedure is a relaxation, i. e., it is not an equivalent rewriting. A notable exception is when $m = 1$ and $f_i$ are quadratic functions, then the two statements are equivalent.

**Example 2.33**
*Suppose we are interested in solutions to the constraint*

$$\forall x \in \left\{ y \;\middle|\; 0 \leq 100 - y^2 \right\} \bullet 0 \leq \rho - x^2$$

*with $\rho$ being the unknown free parameter. The problem obviously has no solution without taking the condition $\forall x \in \left\{ y \;\middle|\; 0 \leq 100 - y^2 \right\}$ into account. The $\mathcal{S}$-Procedure allows us to search for a solution to*

$$\forall x \bullet \lambda \cdot (100 - x^2) \leq \rho - x^2$$
$$\Leftrightarrow \forall x \bullet 0 \leq (\rho - 100\lambda) + (\lambda - 1)x^2$$

*with $0 \leq \lambda$, instead. And it follows that any solution to*

$$1 \leq \lambda \wedge 100\lambda \leq \rho$$

*yields a solution for the initial problem.* ◁

[3] Recall, that every formula can be rewritten in disjunctive normal form (DNF). If the region is not given as a conjunction of equalities and inequalities, we can rewrite it as a DNF and handle each disjunctive term separately.

On the other hand, we might not find a solution if the condition is not carefully selected:

**Example 2.34**
*Suppose we want to show non-negativity of a function over a certain "set" and have the constraint*

$$\forall x \in \{\, y \mid 10 = y \,\} \bullet 0 \leq 100 - x^2$$

*with no unknown free parameter. The function obviously is non-negative for "all" $x$ which can be easily proven by substitution. However, the $\mathcal{S}$-Procedure gives us*

$$\begin{aligned} &\forall x \bullet \lambda \cdot (10 - x) \leq 100 - x^2 \\ \Leftrightarrow\ &\forall x \bullet 0 \leq (100 - 10\lambda) - 1x^2 + \lambda x \end{aligned}$$

*with no constraint on $\lambda$, which is unfortunately* `false`*. Choosing the condition to be $100 = y^2$, on the other hand, yields*

$$\begin{aligned} &\forall x \bullet \lambda \cdot (100 - x^2) \leq 100 - x^2 \\ \Leftrightarrow\ &\forall x \bullet 0 \leq (100 - 100\lambda) + (\lambda - 1)x^2 \end{aligned}$$

*which is* `true` *for $\lambda = 1$.* ◁

This example shows that in order to successfully apply the $\mathcal{S}$-Procedure, one has to choose the right combination of conditions. Since non-negativity is invariant under multiplication, we can always add all combinations of conditions. While true in theory, in practice this introduces several additional unknown free parameters which on one hand increase the problem size and on the other hand might not help to find a solution in case they have to be set to zero anyhow. We will address this problem in Section 3.2.2. In that section, we will propose a simple heuristic to detect free parameters that have to be zero.

### 2.4.3 From Non-Negativity to Sum-of-Squares

An efficient way of proving non-negativity of a polynomial is to show that it can be rewritten as a sum-of-squares (SOS), i. e., $p(\mathbf{x}) = \sum_{i=1}^{n} q_i^2(\mathbf{x})$. A SOS is clearly non-negative since quadratic terms are non-negative and the sum of non-negative terms is also non-negative.

**Example 2.35 (Taken from [PW98])**
*Let $p(x, y, z) = x^6 + 4x^3y^2z + y^6 + 2y^4z^2 + y^2z^4 + 4z^6$. The task is to represented $f$ by a sum of squares, i. e., find terms $(q_i)$ such that*

$$p(\mathbf{x}) = \sum_{i=1}^{n} q_i^2(\mathbf{x}). \tag{2.1}$$

*One solution is* $(x^3+2y^2z)^2+(y^3-yz^2)^2+(2z^3)^2$ *which is a sum-of-squares and therefore non-negative. And indeed,*

$$\begin{aligned}&(x^3+2y^2z)^2+(y^3-yz^2)^2+(2z^3)^2\\ =&(x^6+4x^3y^2z+4y^4z^2)+(y^6-2y^4z^2+y^2z^4)+(4z^6)\\ =&x^6+4x^3y^2z+y6+2y^4z^2+y^2z^4+4z^6\\ =&p(\mathbf{x}).\end{aligned}$$

◁

However, it is important to note that the converse is not true. For example a well-known polynomial that is indeed positive semidefinite but has no sums-of-squares-decomposition is

$$f(x,y,z)=x^4y^2+x^2y^4+z^6-3x^2y^2z^2$$

also known as the Motzkin form [PL03, p.15]. The following important result was originally shown by Choi et al. in [MD 95] and later reformulated by Powers and Wörmann in [PW98].

**Lemma 2.36 (Sum-of-squares (SOS) [PW98])**
*Suppose* $p(\mathbf{x})\in\mathbb{R}[\mathbf{x}]$ *be a polynomial of degree* $2d$ *with* $\mathbf{z}^T=\left[\mathbf{x}^{\beta_1}\ldots\mathbf{x}^{\beta_u}\right]$ *being a vector of monomials. Then,* $p(\mathbf{x})$ *is sum-of-squares (SOS) if and only if there exists a symmetric positive semidefinite (PSD) matrix* $Q\in\mathbb{R}^{u^2}$ *such that*

$$p(\mathbf{x})=\mathbf{z}^TQ\mathbf{z}.$$

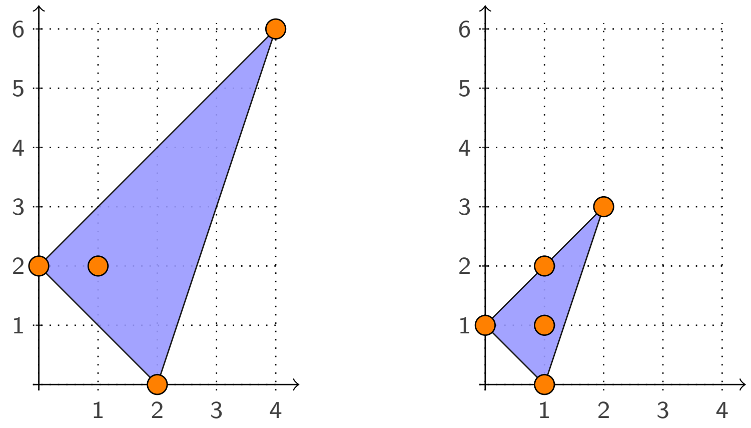
*Let* $Q$ *be such a matrix of rank* $t$*. We can construct polynomials* $q_1(\mathbf{x}),\ldots,q_t(\mathbf{x})\in\mathbb{R}[\mathbf{x}]$ *such that*

$$p(\mathbf{x})=\sum_{i=1}^{t}q_i^2(\mathbf{x})=\mathbf{z}^TAA^T\mathbf{z}=\mathbf{z}^TQ\mathbf{z}$$

*where* $A\in\mathbb{R}^{tu}$ *and* $q_i(\mathbf{x})=\sum_{j=1}^{u}A_{(\beta_j,i)}\mathbf{x}^{\beta_j}$.

Powers and Wörmann also showed that the size of $\mathbf{z}$ in Lemma 2.36 is bounded. Denote by $\Lambda(n,d)=\left\{\mathbf{e}=\left[e_1\ldots e_n\right]\,\middle|\,\sum_{i=1}^{n}e_i\leq d\right\}$ the finite set of all vectors of $n$ exponents up to a degree of $d$. If $p\in\mathbb{R}[x_1,\ldots,x_n]$ is a SOS polynomial, i. e., there exists $q_1,\ldots,q_t\in\mathbb{R}[\mathbf{x}]$ such that $p(\mathbf{x})=\sum_{i=1}^{m}q_i^2$, then $p(\mathbf{x})$ is of degree $2d$ and each $q_i(\mathbf{x})=\sum_{j=1}^{n}a_ix_j^{e_{i,j}}$ — or rather the corresponding vector of exponents $\left[e_{i,1}\ldots e_{i,n}\right]$ — is an element of $\Lambda(n,d)$. As $\Lambda(n,d)$ contains all combinations of exponents, its size is $\dim\Lambda(n,d)=\binom{n+d}{n}$.

**Note 2.37**
*The matrix* $Q=AA^T$ *is called a Gram matrix for* $f(\mathbf{x})$. ◁



(a) The Newton polytope of $p(\mathbf{x}) = 4\,x_1^4 x_2^6 - x_1 x_2^2 + x_1^2 + x_2^2$.

(b) Monomials used for the SOS decomposition.

Figure 2.11: Visualization of the Newton polytope [Pap+13].

**Example 2.38 (Continuation of Example 2.35)**
*In Example 2.35, the set of all possible vectors of exponents contains all combinations of three non-negative integers with a sum less or equal to three:*

$$1, x, y, z, x^2, y^2, z^2, xy, xz, yz, x^3, y^3, z^3, x^2y, x^2z, xy^2, y^2z, xz^2, yz^2, xyz$$

*Further investigation of the coefficients allows to reduce this vector to the five terms* $(q_i) = (x^3, y^3, y^2z, yz^2, z^3)$. ◁

One technique to algorithmically reduce the size of the monomial vector is the newton polytope.

**Definition 2.39 (Newton Polytope [Stu98])**
*Let* $p(\mathbf{x}) \in \mathbb{R}[\mathbf{x}]$, $\mathbf{x} = \begin{bmatrix} x_1 \dots x_n \end{bmatrix}$, *be a polynomial in* $n$ *variables. We can write* $p$ *as a weighted sum of monomials*

$$p(\mathbf{x}) = \sum_{i}^{m} a_i x_1^{e_{1,i}} \cdots x_n^{e_{n,i}}$$

*Let* $E$ *be the set of exponent vectors* $E = \{\mathbf{e}_1 \dots, \mathbf{e}_m\}$ *with* $\mathbf{e}_i = \begin{bmatrix} e_{1,i} \dots e_{n,i} \end{bmatrix}$. *The* Newton polytope *is a convex lattice polytope made of the convex hull of the exponent vectors:*

$$\mathrm{Newton}(p) = \mathrm{conv}(\{\, \mathbf{e}_i \in E \mid a_i \neq 0 \,\}),$$

*where any zero vector is omitted.* ♢

**Example 2.40**
*Figure 2.11 visualizes the Newton polytope for the planar polynomial* $p(\mathbf{x}) = 4\,x_1^4 x_2^6 - x_1 x_2^2 + x_1^2 + x_2^2$ *(taken from [Pap+13]). In Figure 2.11a, the points are the vectors of exponents of* $p$*, i. e.,*

$$(4,6),(1,2),(2,0),(0,2)$$

*where* $(1,2)$ *lies in the interior and the polytope is their convex hull.* ◁

Software that makes use of the Newton polytope for the SOS decomposition are SOSTools [Pap+13], YALMIP [Löf04], SOSOpt [Sei13], and Stabhyli. Early algorithmic approaches to SOS decomposition have been presented by Powers in [PW98]. Recently, Dai and Xia have proposed an optimization in [DX15] that reduces the size of the problem by first decomposing a polynomial into small, so-called, *split polynomials* for which then finding a SOS decomposition is more tractable. This is done because the size of the problem — as mentioned above — is $\mathcal{O}\left(\binom{n+d}{n}\right)$ where $2d$ is the degree of the polynomial and $n$ is the number of variables and, hence, the growth rate is high.

**Note 2.41**
*In [Rez78], Reznick has shown that the vector of monomials* $\mathbf{z}$ *needs to contain only monomials, whose squares have a degree in the newton polytope of* $p(\mathbf{x})$*. For Example 2.40, this is visualized in Figure 2.11b and the highlighted points correspond to exponents of the monomials* $x_1, x_2, x_1x_2, x_1x_2^2, x_1^2x_2^3$. ◁

### 2.4.4 Linear Matrix Inequalities

With the help of the $\mathcal{S}$-Procedure and the SOS decomposition, we can relax the constraints obtained from the Lyapunov theorem into a so-called linear matrix inequality.

**Definition 2.42 (Linear matrix inequality (LMI) [Boy+94, p.7])**
*Let* $F_i \in \mathbb{R}^{n\times n}$*,* $0 \le i \le m$*, be symmetric matrices and let* $\lambda_i \in \mathbb{R}$*,* $1 \le i \le m$*, be free parameters. A linear matrix inequality (LMI) has the form*

$$0 \preceq F_0 + \sum_{i=1}^{m} \lambda_i F_i .$$

*A valuation of the free parameters* $\lambda_i$ *such that the condition is fulfilled, is called a* solution to the LMI. ◇

Using a block diagonal matrix, a set of inequalities can be combined into a single LMI:

**Proposition 2.43**
*Simultaneously finding a solution to* $0 \preceq F_0 + \sum_{i=1}^{m} \lambda_{F,i} F_i$ *and* $0 \preceq G_0 + \sum_{i=1}^{l} \lambda_{G,i} G_i$

*can be achieved by finding a solution for*

$$0 \preceq \begin{bmatrix} F_0 & 0 \\ 0 & G_0 \end{bmatrix} + \sum_{i=1}^{m} \lambda_{F,i} \begin{bmatrix} F_i & 0 \\ 0 & 0 \end{bmatrix} + \sum_{i=1}^{l} \lambda_{G,i} \begin{bmatrix} 0 & 0 \\ 0 & G_i \end{bmatrix}.$$

Now, by solving a single LMI, we can simultaneously search for the sum-of-squares decomposition, valuations of the $\mathcal{S}$-Procedure parameters, the $\mathcal{K}^\infty$-functions, and the Lyapunov functions.

### 2.4.5 Computing Lyapunov Functions

To computationally obtain Lyapunov functions, each function is instantiated by a template involving free parameters. Using these Lyapunov function templates, a constraint system corresponding to Theorem 2.31 is generated and relaxed by the above described

1. $\mathcal{S}$-Procedure to restrict the constraints to certain regions and
2. sum-of-squares decomposition which allows us to rewrite the polynomials as linear matrix inequality [PP03].

These LMIs — in turn — can be solved by Semidefinite Programming (SDP) [BV04].

**Definition 2.44 (Semidefinite Programming [BV04, pp.168])**
*Let* $C, X, F_i \in \mathbb{R}^{n\times n}$ *be symmetric matrices and* $b_i \in \mathbb{R}^n$, $1 \leq i \leq m$, *be column vectors. A semidefinite program (SDP) has the from*

$$\begin{aligned} &\textit{minimize } \langle C, X\rangle \\ &\qquad \textit{s.t. } \langle A_i, X\rangle = b_i \textit{ for all } 1 \leq i \leq m \\ &\qquad \textit{and } X \succeq 0 \end{aligned}$$

*where* $\langle A, B\rangle = \operatorname{tr}(A^T B) = \sum_{i,j}(A_{i,j} \cdot B_{i,j})$ *for* $A, B \in \mathbb{R}^{n\times n}$. *A solution to the SDP problem is the matrix* $X$. ♢

A list of available SDP solvers at time of writing can be found in Table 2.12. The solvers that are supported by Stabhyli are CSDP [Bor99] and SDPA [Fuj+07]. Most of the solvers in Table 2.12 use some kind of interior point methods and numerically approximate a solution.

**Example 2.45 (SDP [BPT12, Beispiel 2.11])**

| Name | Language | Website |
|---|---|---|
| SDPA | C++ | http://sdpa.sourceforge.net/ |
| CSDP | C | https://projects.coin-or.org/Csdp/ |
| SDPT3 | Matlab | http://www.math.nus.edu.sg/~mattohkc/sdpt3.html |
| SeDuMi | Matlab | http://sedumi.ie.lehigh.edu/ |
| DSDP | C, Matlab | http://www.mcs.anl.gov/hs/software/DSDP/ |
| PENSDP | C, Fortran, Matlab | http://www.penopt.com/pensdp.html |
| SDPLR | C, Matlab | http://sburer.github.io/ |
| ConicBundle | C/C++ | https://www-user.tu-chemnitz.de/~helmberg/ConicBundle |
| CVXOPT⋆ | python | http://cvxopt.org/ |
| SCS | C | http://web.stanford.edu/~boyd/papers/scs.html |
| SuanShu | Java | http://numericalmethod.com/suanshu/ |
| SDPB | C++ | https://github.com/davidsd/sdpb |

Table 2.12: Listing of available SDP software.

*Consider the following SDP*

$$\begin{aligned} \textit{minimize}\ & 2x_{11} + 2x_{12} \\ \textit{s.t.}\ & x_{11} + x_{22} = 1 \\ \textit{and}\ & \begin{bmatrix} x_{11} & x_{12} \\ x_{12} & x_{22} \end{bmatrix} \succeq 0 \end{aligned}$$

*with the solver's inputs*

$$m := 1 \qquad C := \begin{bmatrix} 2 & 1 \\ 1 & 0 \end{bmatrix} \qquad A_1 := \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \qquad b_1 := 1.$$

*Every solution to* $x_{11}(1-x_{11}) \geq x_{12}^2$ *yields a valid solution while the optimal solution is*

$$X_{optimal} = \begin{bmatrix} \frac{2-\sqrt{2}}{4} & -\frac{1}{2\sqrt{2}} \\ -\frac{1}{2\sqrt{2}} & \frac{2+\sqrt{2}}{4} \end{bmatrix}$$

*which is irrational, and thus, cannot be returned by a numerical solver. However,* CSDP *returns*

$$X = \begin{bmatrix} \frac{5276295079350937}{36028797018963968} & \frac{-6369051706984595}{18014398509481984} \\ \frac{-6369051706984595}{18014398509481984} & \frac{30752501939613031}{36028797018963968} \end{bmatrix}$$

*which is close and each entry has an error of* $2.4 \cdot 10^{-9}$ *or less.* ◁

While numerical solvers are very fast, they sometimes suffer from numerical inaccuracies. Therefore, one has to double-check solutions returned by the solver. To double-check, we have multiple possibilities from which we make use of two: (1) computing minors and (2) computing eigenvalues.

**Definition 2.46 (Minors [Fis97, pp.191])**
*Let* $M : \mathbb{R}^{n \times n}$ *be a quadratic matrix. Let* $I, J \subseteq \{1, \ldots, n\}$ *be non-empty subsets of*

*M's indices. A* minor *is the determinate of a submatrix of $M$ obtained by deleting the $i$-th rows and the $j$-th column for all $(i, j) \in I \times J$. If $I = J$ one obtains a* principal minor*, that is the determinate of a submatrix that is obtained by deleting the $i$-th row and the $i$-th column for all $i \in I$. If $I = J = \{1, \ldots, k\}$ for some $k < n$ one obtains a* leading principal minor*.* ♢

**Theorem 2.47 (Checking (Semi-)definiteness [Bha07])**
*Let $M : \mathbb{R}^{n \times n}$ be a real symmetric matrix. The properties*

- *$M \succ 0$ (resp. $M \succeq 0$).*
- *All eigenvalues of $M$ are positive (resp. non-negative).*
- *All leading principal minors (resp. principal minors) of $M$ are positive (resp. non-negative).*

*are equivalent.*

As one can see in Definition 2.46, checking definiteness can be done efficiently as for a $n \times n$ square matrix, one has to check only $n$ leading principle minors, in contrast, checking semidefiniteness requires us to investigate $2^n$ principle minors. The means that in practice the semidefiniteness check based on the principal minors is costly due to their sheer number. However, we can still check the leading principal minors:

- if they are all positive, then $M$ is clearly positive semidefinite,
- if one is negative, then $M$ is clearly not positive semidefinite.

While this allows us to identify validness of results of the solver, it cannot help us to obtain valid solutions. Therefore, strengthening of the constraints by adding additional “gaps” might be used to make the constraints more robust against numerical issues. The drawback is that this sometimes results in the feasible set becoming empty.

Now, we have all ingredients for a first Lyapunov function-based stability verification algorithm:

(1) Choose templates for the Lyapunov functions.

(2) Generate the constraints as in Theorem 2.31.

(3) Apply the $\mathcal{S}$-Procedure to the constraints and relax the outcome to an LMI using the SOS decomposition.

(4) Hand the LMI over to an SDP solver.

- if not solvable, return `unknown`.

(5) Double check result of the SDP solver.

- if valid, return `stable`.
- otherwise, return `unknown`.

**Example 2.48 (Continuation of Example 2.28)**
*Constructing the constraints for our (running) example as imposed by the Lyapunov theorem, one obtains*

$$
\begin{aligned}
&\alpha, \beta, \gamma \geq 0\\
v \in [-15, 15] \wedge x \in [-500.500] &\Rightarrow \alpha(v^2 + x^2) \leq V_{\mathtt{Normal}}(x, v)\\
v \in [-15, 15] \wedge x \in [-500.500] &\Rightarrow V_{\mathtt{Normal}}(x, v) \leq \beta \cdot (v^2 + x^2)\\
v \in [-20, -5] \wedge x = 0 &\Rightarrow \alpha(v^2) \leq V_{\mathtt{Accelerate}}(x, v)\\
v \in [-20, -5] \wedge x = 0 &\Rightarrow V_{\mathtt{Accelerate}}(x, v) \leq \beta \cdot v^2\\
v \in [5, 20] \wedge x = 0 &\Rightarrow \alpha(v^2) \leq V_{\mathtt{Brake}}(x, v)\\
v \in [5, 20] \wedge x = 0 &\Rightarrow V_{\mathtt{Brake}}(x, v) \leq \beta \cdot v^2\\
\begin{matrix} v \in [-15, 15] \\ \wedge x \in [-500.500] \end{matrix} &\Rightarrow \left\langle \nabla V_{\mathtt{Normal}}(x, v) \,\middle|\, \begin{bmatrix} -0.001x - 0.052v \\ v \end{bmatrix} \right\rangle \leq -\gamma \cdot (v^2 + x^2)\\
v \in [-20, -5] \wedge x = 0 &\Rightarrow \left\langle \nabla V_{\mathtt{Accelerate}}(x, v) \,\middle|\, \begin{bmatrix} 1.5 \\ 0 \end{bmatrix} \right\rangle \leq -\gamma \cdot v^2\\
v \in [5, 20] \wedge x = 0 &\Rightarrow \left\langle \nabla V_{\mathtt{Brake}}(x, v) \,\middle|\, \begin{bmatrix} 2.5 \\ 0 \end{bmatrix} \right\rangle \leq -\gamma \cdot v^2\\
v \in [13, 15] \wedge x \in [-500, 500] &\Rightarrow V_{\mathtt{Brake}}(x, v) \leq V_{\mathtt{Normal}}(x, v)\\
v \in [-15, -14] \wedge x \in [-500, 500] &\Rightarrow V_{\mathtt{Accelerate}}(x, v) \leq V_{\mathtt{Normal}}(x, v)\\
v \in [-6, -5] \wedge x = 0 &\Rightarrow V_{\mathtt{Normal}}(0, v) \leq V_{\mathtt{Accelerate}}(x, v)\\
v \in [5, 11] \wedge x = 0 &\Rightarrow V_{\mathtt{Normal}}(0, v) \leq V_{\mathtt{Brake}}(x, v).
\end{aligned}
$$

*A reasonable choice for the Lyapunov function templates is*

$$
\begin{aligned}
V_{\mathtt{Normal}}(x, y) &= V_5 x^2 + V_4 v^2 + V_3 vx + V_2 x + V_1 v + V_0\\
V_{\mathtt{Accelerate}}(x, y) &= V_{11} x^2 + V_{10} v^2 + V_9 vx + V_8 x + V_7 v + V_6\\
V_{\mathtt{Brake}}(x, y) &= V_{17} x^2 + V_{16} v^2 + V_{15} vx + V_{14} x + V_{13} v + V_{12}.
\end{aligned}
$$

*Thus, the resulting Lyapunov constraint system has 21 free parameters before applying the $\mathcal{S}$-Procedure and afterwards at least 48 more because the $\mathcal{S}$-Procedure adds one free parameter for each condition. For example, the constraint*

$$v \in [-15, 15] \wedge x \in [-500.500] \Rightarrow \alpha(v^2 + x^2) \leq V_{\mathtt{Normal}}(x, v)$$

*becomes*

$$\lambda_1(v+15)+\lambda_2(15-v)+\lambda_3(x+500)+\lambda_4(500-x)+\alpha(v^2+x^2)\le V_{\mathtt{Normal}}(x,v)$$
$$\lambda_1,\lambda_2,\lambda_3,\lambda_4\ge 0.$$

*And as we do not necessarily know which $\mathcal{S}$-Procedure-terms are needed, we would also include the combinations*

$$\begin{aligned}\lambda_5(v+15)(15-v),\\ \lambda_6(v+15)(x+500),\\ \lambda_7(v+15)(500-x),\\ \lambda_8(15-v)(x+500),\\ \lambda_9(15-v)(500-x),\\ \lambda_{10}(x+500)(500-x)\end{aligned}$$

*resulting in ten free parameters due to $\mathcal{S}$-Procedure-terms for one constraint. For the whole constraint system, we have then $5\cdot 10+8\cdot 6=98$ free parameters due to $\mathcal{S}$-Procedure-terms, 18 free parameters due to Lyapunov function templates, and three parameters due to $\mathcal{K}^\infty$ functions. Fortunately, in this example all constraints are quadratic. Thus, the sums-of-squares-decomposition is trivial and does not introduce extra degrees of freedom which would result in additional free parameters.*

*The solution obtained by* Stabhyli *is .0000000000000000028079*

$$\begin{aligned}V_{\mathtt{Normal}}(v,x)&=+0.110238vx+0.003975x^2+1.084671v^2\\&\quad-0.000000x-0.000000v\\V_{\mathtt{Accelerate}}(v,x)&=-0.103610v^2-4.483182v+16.122491\\V_{\mathtt{Brake}}(v,x)&=-0.101502v^2+4.080301v+18.222108\end{aligned}$$

*and depicted in Figure 2.13. In the figure it can be seen that for transitions the values of the Lyapunov function of the source mode are higher than the values of the target mode. Also the value of each Lyapunov function decrease towards the origin.* ◁

## 2.5 Decompositional Stability Verification

In this section, we briefly introduce the decompositional construction of Lyapunov functions for self-containment and point the interested reader to [OT09] for more details.

The decomposition technique introduces a so-called *constraint graph.* A constraint graph $C=C(\mathcal{H})$ is a directed graph where additionally vertices are labeled with location constraints and transition constraints for self-loops, i. e., $l_1=l_2$ while edges are labeled with transition constraints for non-self-loops, i. e., $l_1\neq l_2$. The predicate $\operatorname{constr}(C)$ is used to denote the conjunct of the annotated constraints. Obviously, any solution to the

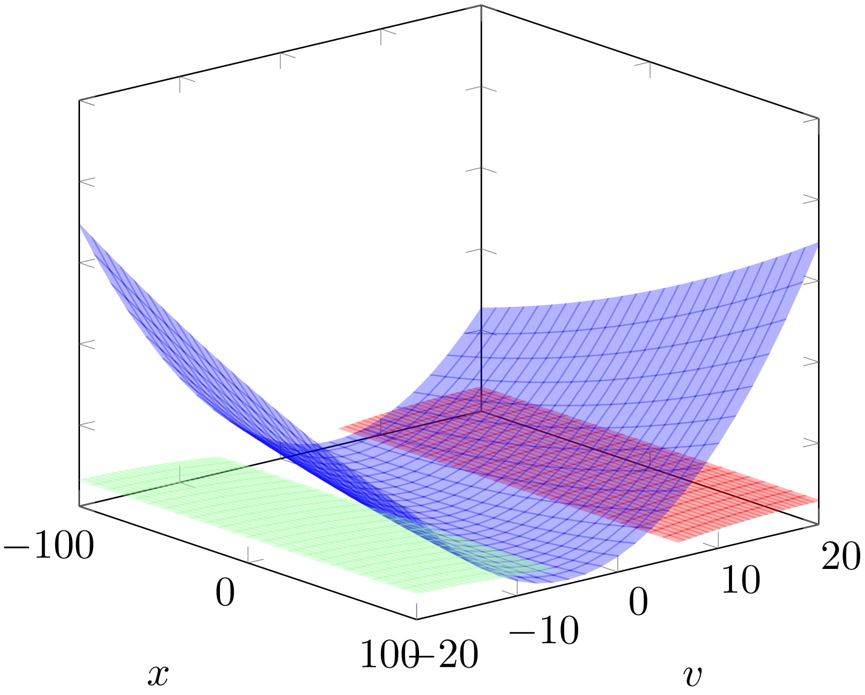


Figure 2.13: A Lyapunov function for the hybrid system from Example 2.28. The Lyapunov function for the mode `Normal` is plotted in blue, for the mode `Accelerate` is plotted in green, and for the mode `Brake` is plotted in red.

constraint graph ($\mathrm{constr}(C)$) is a solution to Theorem 2.31 for $\mathcal{H}$. The graph structure is exploited in two ways:

1) The constraint graph is partitioned into finitely many strongly connected components (SCCs) and Lyapunov functions can be computed for each SCC in isolation.

2) Each SCCs is further decomposed into (overlapping) cycles. In contrast to SCCs, Lyapunov functions cannot be computed in isolation, because the Lyapunov functions of the locations in the individual cycles are mutually dependent due to the transition constraints. To overcome this issue, we do not only compute a single Lyapunov function but a finite set of Lyapunov functions for every location $S_l = \{V_{l,1}, \dots, V_{l,n}\}$ and any conic combination $\mathrm{cone}^+(S)$ — clearly the tip of the cone has to be excluded — yields a valid Lyapunov function for location $l$.

### 2.5.1 Graph

Let us start with some basic graph notations.

**Definition 2.49 (Graph)**
*A* graph *is a tuple $\mathscr{G} = (\mathscr{V}, \mathscr{E})$ where $\mathscr{V}$ is a finite set of* vertices *and $\mathscr{E} \subseteq \mathscr{V} \times \mathscr{V}$ is a finite set of edges. For a* undirected graph*, it holds that $(v_1, v_2), (v_2, v_1) \in \mathscr{E}$, if $(v_1, v_2) \in \mathscr{E}$ then $(v_2, v_1) \in \mathscr{E}$. This is not necessarily true for a* directed graph *and we can distinguish two edges $(v_1, v_2), (v_2, v_1) \in \mathscr{E}$.* ♢

For an edge $(v_1, v_2)$, we call the first element $v_1$, the *source* and the second element $v_2$, the *target*.

Often, we focus our analysis only on a certain part of the graph, a so-called subgraph.

**Definition 2.50 (Subgraph)**
*A* subgraph *$\mathscr{G}' = (\mathscr{V}', \mathscr{E}')$ of a graph $\mathscr{G} = (\mathscr{V}, \mathscr{E})$ is a graph where $\mathscr{V}' \subseteq \mathscr{V}$ and $\mathscr{E}' \subseteq \mathscr{E} \subseteq \mathscr{V} \times \mathscr{V}$.* ♢

In order to reason about the graph structure of a hybrid automaton, we define the underlying graph as follows:

**Definition 2.51 (Underlying Graph)**
*An* underlying graph *$\mathscr{G}(\mathcal{H}) = (\mathscr{V}, \mathscr{E})$ of a hybrid automaton $\mathcal{H} = (Var, Loc, Trans, Flow, Inv, Inits)$ is a directed graph where $\mathscr{V} = Loc$ is a finite set of vertices and $\mathscr{E} = \{ (l_1, l_2) \in \mathscr{V} \times \mathscr{V} \mid (l_1, G, U, l_2) \in Trans \}$ is a finite set of edges.* ♢

**Definition 2.52 (Adjacent, Incident)**
*Let $\mathscr{G} = (\mathscr{V}, \mathscr{E})$ be a graph. Two vertices $v_1, v_2 \in \mathscr{V}$ are called* adjacent *if and only if $(v_1, v_2) \in \mathscr{E}$. Two edges $(v_1, v_2) = e_1 \in \mathscr{E}$ and $(v_3, v_4) = e_2 \in \mathscr{E}$ are called* incident *if and only if either $v_2 = v_3$ or $v_1 = v_4$.* ♢

The decomposition is done on two levels: cycles and strongly connected components (SCCs). They are formally defined as follows:

**Definition 2.53 (Path, Cycle, Clique)**
*Let $\mathscr{G} = (\mathscr{V}, \mathscr{E})$ be a graph. A sequence of vertices $v_1, \ldots, v_n \in \mathscr{V}$ is called a* path *from $v_1$ to $v_n$ if and only if the vertices are adjacent and the edges are incident, i. e., $(v_i, v_{i+1}) = e_i \in \mathscr{E}$, $1 \leq i < n$. If no vertex is repeated, i. e., $v_i = v_j$ if and only if $i = j$, then the path is a* simple path*. A path is called a* cycle *if and only if $v_1 = v_n$. Again, if only $v_1$ occurs twice, then the cycle is a* simple cycle*. A subgraph $(\mathscr{V}, \mathscr{E}) = \mathscr{G}' \subseteq \mathscr{G}$ is called a* clique *if and only if for every pair $v_i, v_j \in \mathscr{V}'$, $v_i$ and $v_j$ are adjacent.* ♢

In a directed graph, it is sometimes helpful to emphasize that we not taking edges in reverse direction. For this a path $v_1, \ldots, v_n$ is called a *forward path* if and only if we can traverse the graph along the path in a forward direction, i. e., $(v_i, v_{i+1}) \in \mathscr{E}$ for all $1 \leq i < n$. In contrast, we call the path a *backward path* if and only if we can traverse the graph along the path in a backward direction, i. e., $(v_{i+1}, v_i) \in \mathscr{E}$ for all $1 \leq i < n$.

**Definition 2.54 (Strongly Connected Component)**
*A strongly connected component (SCC) of a directed graph $\mathscr{G} = (\mathscr{V}, \mathscr{E})$ is a maximal subgraph $\mathscr{G}' = (\mathscr{V}', \mathscr{E}')$ such that for each pair of vertices $v_1 \neq v_2 \in \mathscr{V}'$, there exists a forward path from $v_1$ to $v_2$. Here,* maximality *means that no vertex may be added without violating the existence of a forward path. An edge $(v_1, v_2)$ connecting two SCCs is called a* bridge*.* $\diamondsuit$

To improve the performance of the decomposition, we will identify and investigate subgraphs with high connectedness (or density) in Section 4.2. In that context, a search for cliques is one possible way of finding subgraphs with a high density.

**Definition 2.55 (Graph Density)**
*Let $\mathscr{G} = (\mathscr{V}, \mathscr{E})$ be a graph. If $\mathscr{G}$ is undirected then the* graph density *$D$ is defined as $D = \frac{2|\mathscr{E}|}{|\mathscr{V}|\,(|\mathscr{V}|-1)}$ and if $\mathscr{G}$ is directed then the* graph density *$D$ is defined as $D = \frac{|\mathscr{E}|}{|\mathscr{V}|\,(|\mathscr{V}|-1)}$.* $\diamondsuit$

### 2.5.2 Decomposition into Strongly Connected Components

The main insight behind the decomposition into SCCs is that if a hybrid system is indeed stable, then any trajectory entering an SCC of the corresponding hybrid automaton $\mathcal{H}$ may either converge to the equilibrium point within the SCC or leave the SCC. In any case, once entered, an SCC might not be entered again. This allows us to compute local Lyapunov functions for each SCC separately, because for every pair $(l_1, l_2)$ of locations of different SCCs it holds that either

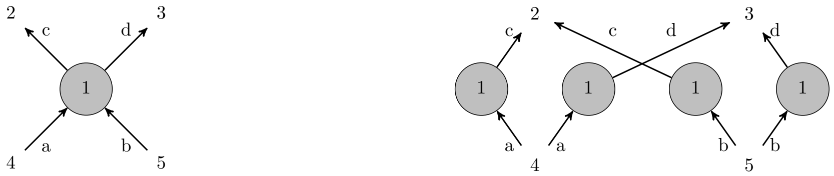
- there is no forward path between $l_1$ and $l_2$ or
- there is a forward path from $l_1$ to $l_2$ but no forward path from $l_2$ to $l_1$.

Thus, the transition constraints — as imposed by the Lyapunov theorem — enforce a decreasing order of the Lyapunov functions of the two locations at most in one direction. This is summarized in the following theorem.

**Theorem 2.56 (Decomposition into SCCs [Oeh11, Theorem 4.1])**
*Let $\mathcal{H}$ be a hybrid automaton with $\mathcal{H} = (Var, Loc, Trans, Flow, Inv, Inits)$. If all sub-automata pertaining to the strongly connected components of $\mathscr{G}(\mathcal{H})$ are globally attractive, then so is $\mathcal{H}$. If all SCCs are Lyapunov stable, and if all transitions $(l_1, G, U, l_2) \in Trans$ corresponding to bridges of $\mathscr{G}(\mathcal{H})$ are* sub-linear*, that is,*

$$\exists c > 0 \bullet \forall \mathbf{x} \in G \bullet ||U(\mathbf{x})|| \leq c||\mathbf{x}||,$$

*then $\mathcal{H}$ is Lyapunov stable. Therefore, $\mathcal{H}$ is GAS.*



(a) Before splitting

(b) After splitting

Figure 2.14: Visualization of the mode-splitting step

### 2.5.3 Decomposition into Overlapping Cycles

The decomposition into SCCs is the simpler part of the two levels of decomposition. For the decomposition into overlapping cycles, we need to compute sets of Lyapunov functions, which have to be further checked to be compatible in the sense of the transition constraints.

To guarantee compatibility, the cycles have to be treated in a certain order. Let us first introduce the concepts of outer cycles and border vertices. A cycle is called *outer cycle* if there is at most one vertex which connects the cycle with remainder of the graph. A vertex which is shared by at least two cycles is called a *border vertex*.

Now, compatibility can be guaranteed if the cycles are examined successively in the following way:

(1) **Preparation:** Generate the constraint graph $C = C(\mathcal{H})$ of the hybrid automaton $\mathcal{H}$.

(2) **Selection:** Search for an outer cycle $C' \subseteq C(\mathcal{H})$ in the constraint graph $C(\mathcal{H})$.

(3) **Reduction:** If such outer cycle exists, then

1. construct the constraint systems $\operatorname{constr}(C')$ of the subgraph $C'$, which is like certifying stability of a corresponding hybrid automaton $\mathcal{H}'$ containing only those locations and transitions which are in the underlying $\mathscr{G}'$ of $\mathcal{H}'$.
2. compute finitely many candidate local Lyapunov functions $(V_{l,i})$ which are given by solutions of the constraint systems $\operatorname{constr}(C')$. If the constraint systems are solved using optimization techniques, then different solutions can be obtained by using different objective functions. If no solution can be found, then return `failed`.
3. replace the annotations in the constraint graph: For each border vertex the location constraints are replaced by the conic combination of the candidate local Lyapunov functions

$$V_l = \operatorname{cone}^+(V_{l,i}) = \sum_i \lambda_i \cdot V_{l,i} \text{ with } \lambda_i \geq 0, \sum_i \lambda_i > 0,$$

where $(\lambda_i)$ are fresh parameters that are existentially quantified and shared by all such border vertex. All other vertices and edges forming the cycle can be removed.

4. if this was the last cycle, then return `success`, otherwise continue with Step 2.

(4) **Splitting:** If not outer cycle exists, then

1. select an arbitrary vertex and replace the vertex with one copy of that vertex for each pair of incoming and outgoing edges, such that afterwards, every copy is connected to exactly one incoming and one outgoing vertex. This is visualized in Figure 2.14. In Figure 2.14a, vertex `1` is connected to four other vertices by two incoming and two outgoing edges. In Figure 2.14b, vertex `1` is replaced by four copies, where each one is connected to exactly one incoming and one outgoing edge.

2. continue with Step 2.

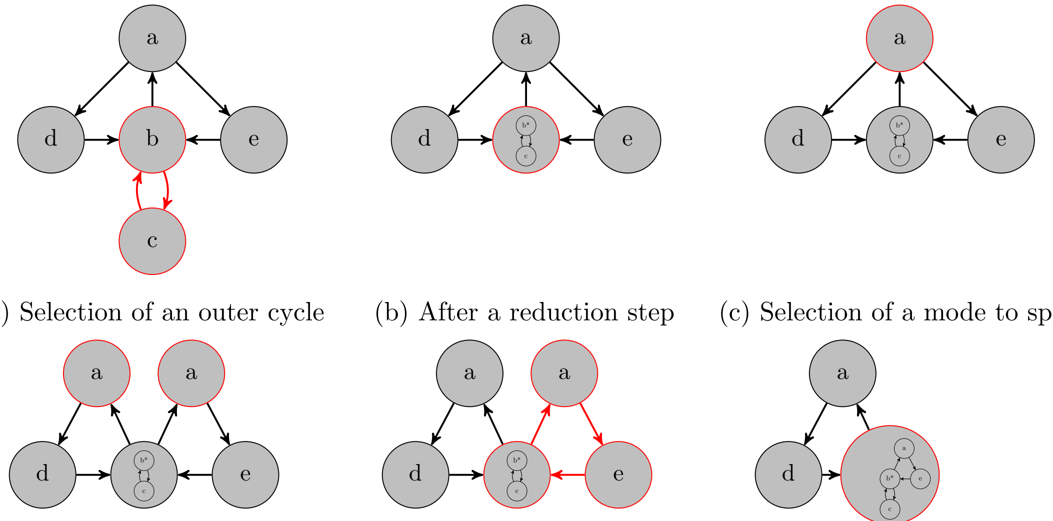
**The Reduction Step** Conical combinations of the candidate LLFs are valid local Lyapunov functions and satisfy the constraints of the Lyapunov theorem because the set of positive functions is a convex set and the constraints describe a convex satisfaction problem. This fact allows us to replace constraints corresponding to a — possibly large — cycle by — possibly small — conic combinations of candidate local Lyapunov functions.

Step 3 of the above algorithm is called the *reduction step*. The reduction step collapses all vertices that lie only on that outer cycle and replaces references to LLFs in the constraints of adjacent edges by conical combinations of the candidate LLFs. This allows us to prove stability of each cycle separately while, cycle-by-cycle, ensuring compatibility of the feasible sets of the (overlapping) cycles.

**The Location-Splitting Step** In the above algorithm, if the graph does not contain an outer cycle, then Step 4, called the *location-splitting* step, is performed. In the location-splitting step, a single vertex is replaced by multiple copies, one copy per pair of incoming and outgoing edges.

Depending on the order in which vertices are chosen for location-splitting, one can make a cycle connected to the rest of the graph by exactly one vertex and then perform a reduction step. Clearly, the order of location-splitting and reduction steps does not only affect the termination of the procedure, but also the size of the graph and, therefore, the number of cycles that have to be reduced. With a good order of reduction and location-splitting steps, one ends up with a single cycle for which the following holds: The successful computation of candidate LLFs implies the existence of a piecewise Lyapunov function for the whole SCC. In Section 3.2 on page 52, we introduce three heuristics for the location-splitting that have been implemented, namely, (1) selection by product, (2) prioritization of zippers, and (3) selection by pairwise degree.

**An Abstract Example** Figure 2.15 visualizes an abstract decomposition. Each vertex represents the constraints of a location of a hybrid automaton. In Figure 2.15a, an outer cycle can be found and a reduction step is performed. The (red) cycle (consisting of vertex `b` and vertex `c`) is selected and collapsed into a single vertex. This is done by replacing the border vertex with a finite set of solutions to the corresponding optimization



(a) Selection of an outer cycle (b) After a reduction step (c) Selection of a mode to split

(d) After a mode-splitting step (e) Selection of an outer cycle (f) After a reduction step

Figure 2.15: A sketch of the decomposition procedure.

problems — visualized by collapsing the cycle into a single vertex in Figure 2.15b. Thus, the feasible set of the cycle's constraints is replaced by a set of candidate LLFs. This set serves as a "summary of the solutions to the sub-proof" in subsequent proofs. Using conic combinations of these candidates in subsequent proofs, ensures the existence of Lyapunov functions for all locations in the cycle.

In Figure 2.15c, there are no outer cycles, thus, a location-splitting step is performed: the vertex a is selected, copied twice, and each path is routed through one copy. The result is shown in Figure 2.15d. Since the result contains outer cycles, we can select an outer cycle as in Figure 2.15e and perform another reduction step resulting a single cycle being left. Figure 2.15f shows the result. Now, if a solution to the constraint system of the last cycle can be found, then the original hybrid automaton is stable, too. On the other hand, if any reduction fails, then we cannot conclude that the system is not stable. Nevertheless, the designer of the system can make use of the knowledge and redesign the part of the system that was hard to prove stability of.

As stated before, the order and choice of reduction and splitting steps matters, Oehlerking suggested in [Oeh11] to use backtracking in case a certain reduction step was not successful.

**Benefits and Issues of the Decompositional Proofs** The general benefits of the decomposition are that

- a large problem is split into several smaller sub-problems,
- solutions to sub-problems can be reused and
- solving small problems is general less attractive to numerical issues.

| System | Theorem | Technique |
|---|---|---|
| system of linear ODEs | Theorem 2.29 | LMI |
| system of polynomial ODEs | Theorem 2.29 | SOS, LMI |
| switched system of linear ODEs | Theorem 2.29[3] | LMI |
| switched system of polynomial ODEs | Theorem 2.29[3] | SOS, LMI |
| switched system of linear ODEs | Theorem 2.31 | LMI |
| switched system of polynomial ODEs | Theorem 2.31 | SOS, LMI |
| linear hybrid automata | Theorem 2.31 | $\mathcal{S}$-Procedure, LMI |
| polynomial hybrid automata | Theorem 2.31 | $\mathcal{S}$-Procedure, SOS, LMI |

Table 2.16: Lyapunov theorem based proof schemata applicable to different classes of dynamical system.

However, the chance of success is also decreased due to the fact that the feasible sets are under-approximated by finitely many candidate solutions. Additionally, gaps — as mentioned in Section 2.4.5 on page 39 — further limit the use of the decomposition as each reduction now "doubly" shrinks the feasible set: via gaps and — as we have seen — via computing finitely many candidates LLFs.

## 2.6 Summary

In this chapter, we have defined basic notations and summarized Lyapunov-related ways to prove stability of different kinds of dynamical systems. Table 2.16 summarizes which theorem and techniques can be applied to prove GAS or GES for different classes of dynamical systems.

In Section 2.4.5, we have presented a technique based on semidefinite programming to automatically compute Lyapunov functions as a solution to linear matrix inequalities. Further, we summarized contemporary tools for solving and addressed issues and countermeasures. However, in the next chapter we will investigate the problem of numerical inaccuracies in more detail and presented further techniques for detected and partially overcome these issues.

Finally, we have presented the decompositional construction of Lyapunov functions as proposed by Oehlerking and Theel [OT09; Oeh11]. This technique allows us — depending on the hybrid automaton — to construct Lyapunov functions more effectively and more efficiently. However, we have identified that this way of constructing Lyapunov functions is not suitable in general. We will address some issues in the following chapters:

- numerical inaccuracies and algorithmic treatment, which will be addressed in the following Chapter 3,
- high connectivity of the underlying graph leads to an explosion of the automata's description, which will be addressed in Chapter 4.

[3] The theorem has to be extended to include mode constraints for each differential equation $f_i$ using a *common Lyapunov function*, i. e., enforce the same Lyapunov function for each $f_i$.

CHAPTER 3

# Automatic Stability Verification

This chapter presents one of the key contributions of this thesis: an automatized approach to stability verification using Lyapunov theory.

First, we give a short overview of existing techniques in Section 3.1. Then, we present STABHYLI which is the tool developed during this work. The theoretical background has been presented in Chapter 2. Section 3.2 gives details on the implementation that has been partially presented in [MT13a]. STABHYLI automatically proves stability of non-linear or rather polynomial hybrid systems. Stability certificates are obtained by Lyapunov theory combined with decomposition and composition techniques. The certificate involves finding Lyapunov function using one of the numerical solvers CSDP [Bor99] or SDPA [Fuj+07] as a backend. Numerical optimization is usually very fast but, it unfortunately, suffers from numerical issues which we have already mentioned in Section 2.4.5. STABHYLI— among others — integrates some smaller pre- and post-processing steps that counteract these issues.

We identified one key issue that leads to numerical issues which is: *implicit equalities* in the constraint system as obtained from the Lyapunov theorem Theorem 2.31. This issue often prevents us from automatically finding Lyapunov functions. It has been investigated in [MT13b]. We present a heuristic to detect and handle implicit equalities in Section 3.3, which has partially been implemented in STABHYLI and combined with a backtracking procedure by Zschoche in [Zsc15]. Using the presented heuristic, the number of false-negatives is reduced. Here, a *false-negative* means that the solver reports infeasibility of the numerical problem even if there is a solution.

Also *false-positives* are possible, i. e., the solver reports success even though the returned set of values does not satisfy the constraint system. Such a false-positive can easily be provoked by scaling the flows, invariants, guards, or updates beyond machine precision. We have addressed this issue in [MT14] and proposed a technique based one satisfiability modulo theory (SMT) to (a) rigorously validate the results of a numerical solver and (b) use a possible counterexample to guide the numerical solver towards a valid solution. The former will be presented in Section 3.4 and the later will be presented in Section 3.5.

## Affirmation

Most of the content of this chapter has already been published in [MT13a; MT13b; MT14] of which the main author and main contributor is also the author of this thesis.

## 3.1 State-of-the-Art

In contrast to safety properties, stability has not yet received that much attention with respect to automatic proving and therefore, only a few tools are available. Most of the tools have been developed within the transregional research project AVACS (Automatic Verification and Analysis of Comlex System) which can be said to be the first reserach project successfully developing integrated tools for the automated verification of stability properties of hybrid systems. Indeed, only the following automated tools — each one specialized for specific system classes — are known to the author.

- Podelski and Wagner presented a tool in [PW07b] which computes a sequence of snapshots and then tries to relate the snapshots in a decreasing sequence. If successful, then this certifies region stability, i. e., stability with respect to a region instead of a single equilibrium point. To compute the snapshots, their tool relies on sets of states which usually are overapproximations of the actual reachable states. In cases where the overapproximations are to coarse, no decreasing relation between the snapshots can be found.

- Oehlerking et al. implemented a powerful state space partitioning scheme to find Lyapunov functions for linear hybrid systems [OBT07].

- The RSolver by Ratschan and She computes Lyapunov-like functions for continuous system [RS10].

- Duggirala and Mitra proposed a tool that combines Lyapunov functions with searching for a well-foundedness relation for symmetric linear hybrid systems [DM12].

- Averist is a tool, that is based one a technique presented by Prabhakar and García Soto in [PS13] and allows proving stability of hybrid systems with (piecewise) constant derivatives. This technique has been extended to polyhedral switched systems in [PS14] and, recently, to switched linear hybrid systems [PS15; PS16] by constructing an enclosing polyhedral system and applying a specialized counterexample guided abstraction refinement (CEGAR) technique. It has also been shown that the technique is complete for locally asymptotically stable linear hybrid systems which are “uniformly converging in time” in [PS16]. While asymptotic convergence requires that for every trajectory and any region around the origin exists a time (that may differ for every trajectory and region) such that the region is reached, . . . *uniformly converge* is stronger as it requires the existance of a singe time (that is the same for every trajectory) after which every trajectory has reached half of it’s way to the origin.

- Matlab toolboxes (YALMIP [Löf04], SOSTools [Pap+13]) that require a by-hand generation of the constraint systems for the search of Lyapunov functions are available. These toolboxes do not automatically prove stability but assist in handling backend solvers, provide a uniform notation and a convenient integration into Matlab which internally handles the conversion of data structures.

> A toolbox integrating SDP solvers is VSDP [JCK07]. It computes rigorous error bounds of the true value of the objective function and verified certificates of infeasibility. It makes use of interval arithmetic and correctly handles the rounding in floating-point arithmetic. Of course, if VSDP reports infeasibility, we can rely on this. On the other side, if VSDP reports success, then we have an interval enclosures of the true value of the objective function and of the optimal feasible solution. The enclosure of the optimal feasible solution is guaranteed to contain the optimal feasible solution. However, not every enclosed candidate solution is guaranteed to be feasible. This means that each candidate needs to be checked.

All these tools, including Stabhyli, are incomplete if their system model is sufficiently expressive. This is due to the fact that not all stable hybrid systems can be proven stable because stability is in general undecidable. For example decidability has been shown for the very restrictive case of *planar rectangular switched hybrid systems* and undecidable for systems with five or more dimensions [PV13]. *Switched hybrid systems* are update-free hybrid systems where flows, invariants, and guards are specified by convex polyhedral sets. Nevertheless, sufficient conditions such as with Lyapunov theory can be checked instead.

## 3.2 Stabhyli: A Tool for Automatic Stability Verification

Stabhyli is a tool for the automatic verification of stability of non-linear hybrid systems. It can be used to obtain common Lyapunov functions, piecewise Lyapunov functions and employing the decompositional proof schemes presented in [OT09; Dam+10]. As real world systems tend to become very large and complicated, automated computer-aided verification and assistance in the design is very important. Stabhyli addresses both, verification and design.

To actually obtain the Lyapunov functions, and thereby deriving a proof, one out of four currently implemented proof schemes can be used. We call a *proof scheme* a method that takes a hybrid system as an input and returns either that the system is stable or that the method was not successful. In the following, the proof schemes are explained in more detail. All proof schemes have in common that they generate constraint systems that will be handed over to an external solver.

**(Proof Scheme 1) Common Lyapunov Function** This proof scheme tries to find a single *common Lyapunov function* $V$ such that $\forall l \in Loc \bullet V_l = V$ and roughly corresponds to Theorem 2.29 lifted to hybrid systems. This proof scheme has the advantage that the constraint system contains much fewer free parameters because 1. only a single Lyapunov function has to be found and 2. some transition constraints (constraint type C3 of Theorem 2.31) can be left out, e. g., for every transition where the update function is the identity function. As a drawback this approach is very restrictive since all locations have to share the same local Lyapunov function.

**(Proof Scheme 2) Piecewise Lyapunov Function** This proof scheme tries to find a *piecewise Lyapunov function* exactly as described in Theorem 2.31. The scheme is not that restrictive as it allows the use of a local Lyapunov function (LLF) for each location. These LLFs together form a piecewise Lyapunov function or discontinuous Lyapunov function which is valid for all runs of the hybrid automaton (HA). Here, all transition constraints (constraint type C3 of Theorem 2.31) are required to ensure that the global Lyapunov function does not increase when switching between locations.

**(Proof Scheme 3) Piecewise Lyapunov Function via Decomposition** The third proof scheme tries to find a piecewise Lyapunov function using decomposition as proposed by Oehlerking and Theel [OT09]. A short summary of this technique has already been given in Section 2.5. From the perspective of a Lyapunov function the decomposition into cycles can be seen as making the different sequences of locations, that a trajectory may visit, more explicit. This step, called *location-splitting* (or *mode-splitting*), is visualized in Figure 2.14 on page 45. Location-splitting allows us to have multiple nodes for the same location depending on the path in the graph. For every node we allow a different LLF. And, hence, a single location might have different LLFs and the actual one depends on the history and future of the location sequence. For example, a trajectory might enter a location $l_b$ either coming from $l_a$ or $l_{a'}$ with $a \neq a'$ and visiting $l_c$ next. Generating the constraint system exactly as described in Theorem 2.31 leads to three transition constraints involving location $l_b$. This might be unnecessarily restrictive since $l_b$ might operate differently depending on the preceding location. In this case, the decompositional proof scheme will split the paths by creating a copy of $l_b$ for each path. And, thereby, allowing every instance to have its own — possibly different — LLF.

As mentioned in Section 2.5, carefully selecting locations for splitting leads to outer cycles connected to the reset of the graph by a single border vertex. For each such outer cycle Stabhyli treads the cycle as a subgraph and generates a *sub-constraint system*. For the subgraph's constraint system the piecewise Lyapunov function scheme is used. Then, Stabhyli tries to solve the sub-constraint system in different optimization directions. If no solution could be found, then Stabhyli reports an error. Otherwise, the set of all found solutions is used as a basis for constructing a LLFs for the border vertex in subsequent constraint systems.

**Note 3.1**
*Termination of this procedure depends on the selection of vertices for splitting. Furthermore, the order of the splitting steps affects the total number of steps required to complete the proof and even the chance of success.* ◁

For controlling the order, we have implemented three heuristics:

**Selection by product.** Sort the vertices by their product of incoming and outgoing edges. Select a vertex with the smallest product greater than 1. Thus, this heuristic always chooses a vertex that will be split into the least possible number of vertices.

**Prioritization of zippers.** We call a vertex a *zipper* if and only if it either has exactly one incoming edge and multiple outgoing edges or vice versa. The heuristic performs a *selection by product* on zippers. If no zipper exists, then a *selection by product* on all vertices is performed. This heuristic tries to quickly obtain outer cycles.

**Selection by pairwise degree.** Sort the vertices according to the vector order of incoming and outgoing edges and select the first vertex having a vector greater than $\begin{bmatrix}1 & 1\end{bmatrix}^T$. This heuristic is a trade-off between the others.

Another benefit of the decompositional proof scheme is that, if the hybrid automaton cannot be proven stable, then the particular part of the automaton, that let the proof fail, can easily be identified. This is due to the fact that the proof scheme successively generates sub-proofs to derive a complete proof. If a sub-constraint system could not be solved, then this can be used to identify the parts of the automaton that potentially require a redesign. By using this knowledge, a guided analysis, refinement, or redesign of this part can be performed by the user. This can be seen as a redesign process for hybrid systems.

**(Proof Scheme 4) Piecewise Lyapunov Function via Composition** The last proof scheme is an incremental technique as proposed by Damm et al. in [Dam+10]. Stabhyli allows specifying *modules* or *components* with entry (the control is transferred to the component) and exit ports (the control is transferred from the component to another one) and automatically generates *interfaces* for these components. The interfaces keep annotations that allow to reuse the component in a compositional manner without knowing the implementation (i. e., the hybrid automaton) of the underlying module. That way, a structured design in a bottom-up fashion of stable hybrid systems is possible while the composition uses the annotations in the interfaces to simultaneously assure that a proof of stability exists.

In Chapter 5, we will present a so-called sequential composition operator. The Piecewise Lyapunov Function via Composition is the supplementing tool support for kind of composition.

Figure 3.1 gives an overview of the steps performed by Stabhyli.

(1) The input is a hybrid automaton, which is read from a hybrid automaton language (HAL) file. The description is parsed into an internal graph data structure.

(2) The given hybrid system is shifted such that the equilibrium point is at the origin of the state space, i. e., $\mathbf{0}$. Details also including further preprocessing steps are given in Section 3.2.1.

(3) Depending on the selected proof scheme, a (sub-)constraint system is generated.

(4) A backend solver is run and the result of the solver is then double-checked whether there are violated constraints. This is done because the solver might only be able to satisfy the constraints up to a certain accuracy.

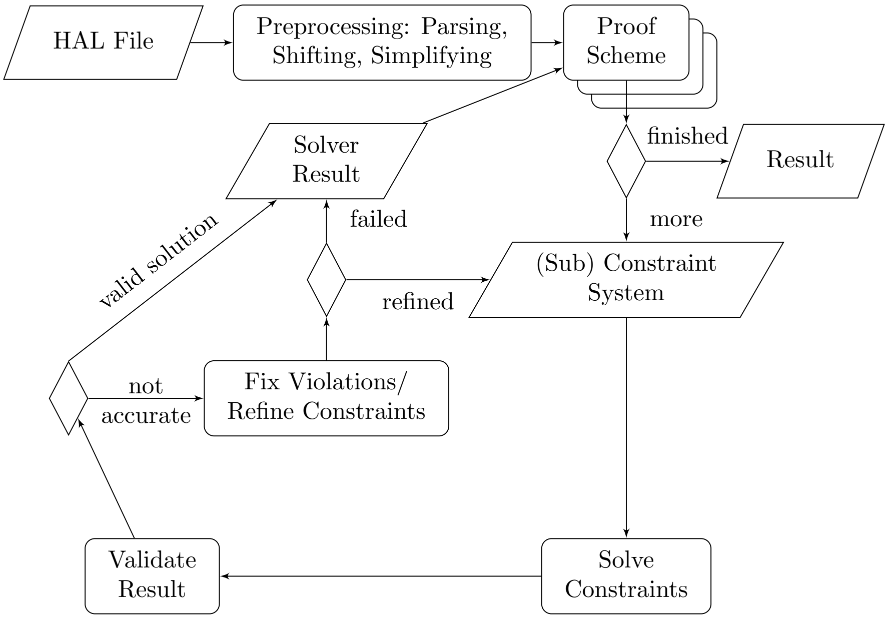


Figure 3.1: Overview of STABHYLI's internal steps.

(a) If a violated constraint is found, then we refine the constraint system and rerun the solver in Step 4.

(b) If no refinement can be found, then we stop and report `failed`.

(c) If the solution is valid and the current constant system is a sub-constraint system, then we continue with Step 3.

(d) If the solution is valid and this has been the last constraint system to be solved, then we stop and report `success`.

Next, we describe the preprocessing. The steps for solving constraints are described in more detail in Section 3.2.2.

### 3.2.1 Preprocessing

The preprocessing consists of

(1) parsing the HAL file,

(2) shifting the described hybrid automaton such that the equilibrium point is at point $\mathbf{0}$, and

(3) rewriting the description such that all terms used to define flows, invariants, guards and updates are in a canonical form.

Actually, the last two steps are not separated and performed together.

**Reading a hybrid automaton language file** The hybrid automaton language is a language similar to HLang [Frä+07] but specialized to describe hybrid automata while HLang targets a more general description of hybrid systems via predicates. It consists of five declaration sections.

1 A **variable section** declaring the set of available variables.

2 An **interface section** and a **submodule section** which are only allowed when using Proof Scheme 4.

3 A **location section** declaring the locations (or modes) of the hybrid automaton. Each location needs to have a unique name and optionally has an associated flow and an invariant.

4 A **transition part** declaring the transitions (or jumps) of the hybrid automaton where a transition has a source location, an optional guard, an update function which defaults to the identity function, and a target location.

The *variable declaration* section allows us to define constants, parameters, and variables of the hybrid system. Figure 3.2 show a syntax diagram which corresponding to the extended Backus–Naur form (BNF) for this section. *Constants* allow giving names to certain values or ranges. This increases readability of the description of the hybrid automata because values can be called by their meaning and not by their value. *Parameters* allow describing parameterized hybrid systems but this feature is not yet supported by Stabhyli. *Variables* which are neither constants nor parameters can be declared as *convergent*. For convergent variables an equilibrium point different from 0 can be specified. Additionally, variables may be bounded by globally specifying the domain as an interval.

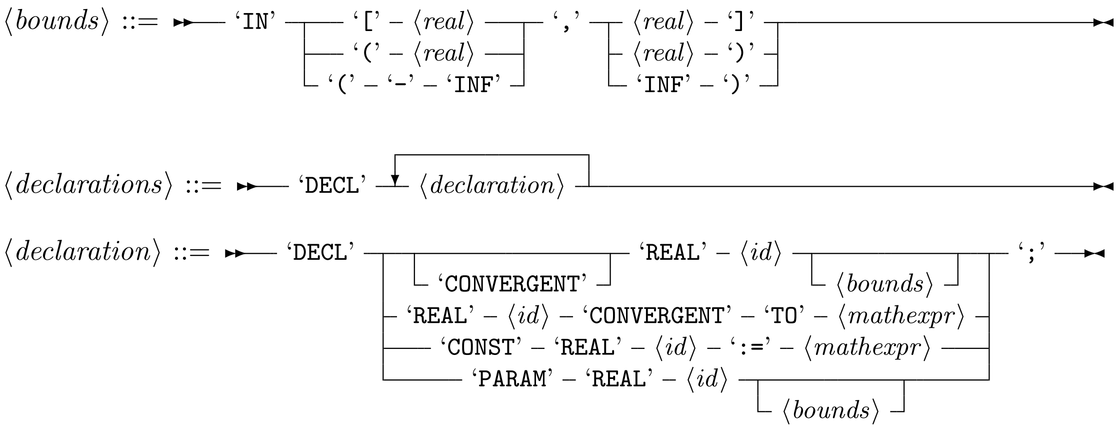


Figure 3.2: Syntax diagram of the variable declaration section.

The interface and submodule declaration sections are only allowed when using Proof Scheme 4. The declaration of a submodules allows us to define transitions to other hybrid

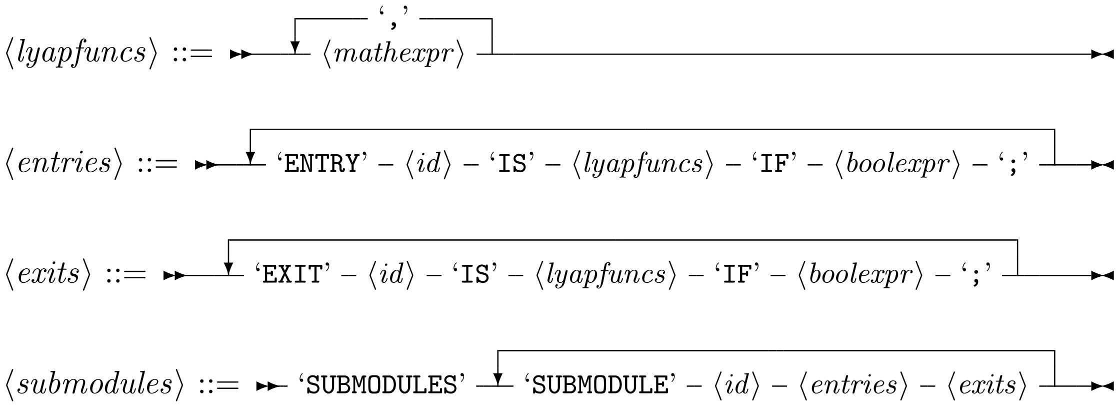

⟨lyapfuncs⟩ ::=
‘,’
⟨mathexpr⟩
⟨entries⟩ ::=
‘ENTRY’ – ⟨id⟩ – ‘IS’ – ⟨lyapfuncs⟩ – ‘IF’ – ⟨boolexpr⟩ – ‘;’
⟨exits⟩ ::=
‘EXIT’ – ⟨id⟩ – ‘IS’ – ⟨lyapfuncs⟩ – ‘IF’ – ⟨boolexpr⟩ – ‘;’
⟨submodules⟩ ::=
‘SUBMODULES’
‘SUBMODULE’ – ⟨id⟩ – ⟨entries⟩ – ⟨exits⟩

Figure 3.3: Syntax diagram of the submodule declaration section.

.

automata for which only an abstract interface is available. A *submodule declaration* (syntax given in Figure 3.3) describes the entry and exit ports together with a set of Lyapunov functions. The names of the entry and exit ports can be used as the source and target of a transition. This allows to transfer the control to and receive control from a component for which only the interface is available. The annotated Lyapunov functions are used by STABHYLI to prove stability of the composition. To create an

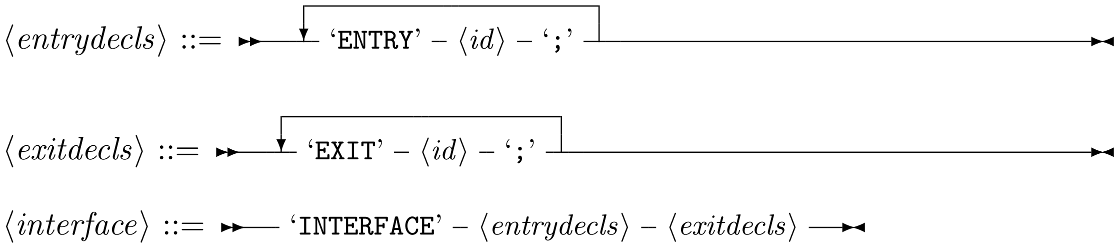


Figure 3.4: Syntax diagram of the interface declaration section.

.

abstract interface, the *interface declaration section* is used. A syntax diagram in given in Figure 3.4. This section serves as a template which is used by STABHYLI to generate a submodule description. This description can be used in subsequent compositions. The interface template consists of names for entry and exit ports. These names can be used as the source or the target of a transition. This allows defining how control may be transferred to this module and how this module transfers control to outer modules. Given such a template, STABHYLI computes a set of Lyapunov functions.

The *location declaration* is the main difference to HLANG. Instead of having a separated declaration section for invariants — which in case of HLANG are not even location specific — and for flows, hybrid automaton language (HAL) has only one section, called `MODES`, which is more natural for automata and combines both. While this decision makes it less flexible, it also makes the locations of the automaton explicit and is thus better suited for graph-based analysis without the need to extrapolate a graph structure from the predicates. The syntax can be seen in Figure 3.5. Each mode optionally

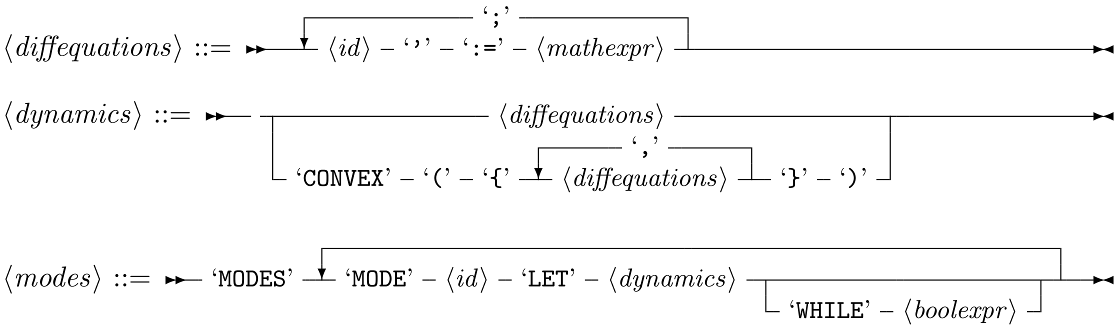


Figure 3.5: Syntax diagram of the mode declaration section.

has a flow and an invariant. Flows are either (1) differential equations where the time derivative is described by a polynomial function or (2) differential inclusions where the derivative lies inside a convex set of polynomial functions. Invariants are arbitrary Boolean combinations of relations of polynomials with the common logical operators `not`, `and`, `or`, `impl`, `equiv`, `nand`, and `nor`, comparison operators `<`, `<=`, `==`, `!=`, `>=`, and `>`, and the arithmetic operators `+`, `-` `*`, `/`, and `^`.

The *transition declaration* allows us to describe transitions of the hybrid automaton. This section is called `JUMPS` and its syntax is given in Figure 3.6. Transitions consist

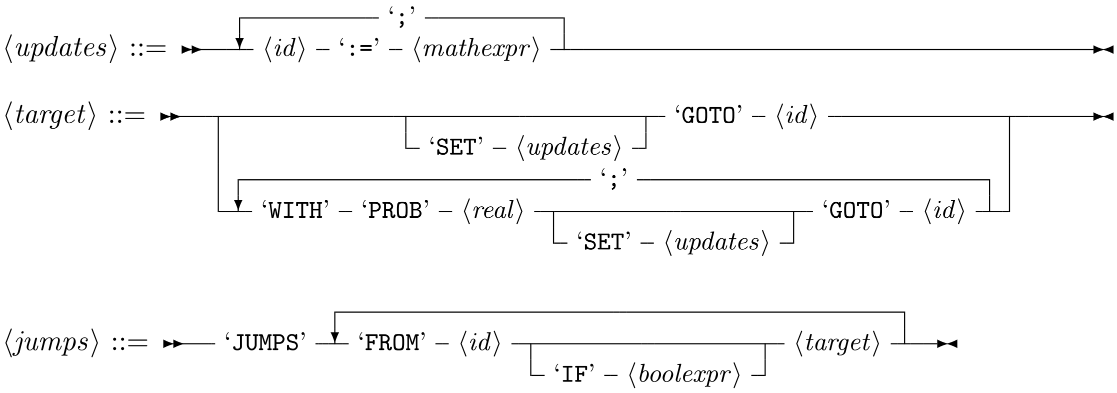


Figure 3.6: Syntax diagram of the transition declaration section.

of a source (`FROM`) location, a guard (`IF`) which is a Boolean combination of relations of polynomials just like invariants, and at least one pair of an optional update (`SET`) and target location (`GOTO`). If multiple locations are specified, then each location has to have an associated probability (`WITH PROP`) and the sum has to be 1. This allows us to specify probabilistic hybrid automata, albeit not yet exploited by STABHYLI. In fact STABHYLI treats each probabilistic choice as several transitions with a non-deterministic choice.

Preliminary support for specifying initial states as well as target states — to be used by reachability tools — has also been started but the format is not yet fixed.

### Variable Shifting

HAL allows the user to define variables that are required to converge to a fixed value different from the origin 0. This means that STABHYLI must prove convergence with respect to this specified equilibrium point. Although this eases modeling, it requires a tool to shift the system. A successful proof then guarantees that the difference between the value of the variable and the equilibrium point converges to zero which in turn guarantees the desired property. Such shifting can be done by replacing each occurrence of a variable $x$ by a new variable $x' := x - x_e$ for an equilibrium point $x_e$.

### Forms of Flows, Guards, Invariants, and Resets

HAL allows us to define flows and resets in terms of arbitrary mathematical expressions (`mathexpr`) of the following form

$$\texttt{expr} ::= \texttt{expr} \star \texttt{expr} \mid (\texttt{expr}) \mid -\texttt{expr} \mid \texttt{number} \mid \texttt{variable}$$

with operators $\star \in \{\texttt{-}, \texttt{+}, \texttt{*}, \texttt{/}, \texttt{\^{}}\}$ having their usual meaning. STABHYLI evaluates such expressions using symbolic substitution and rational arithmetic. and checks whether these expressions are polynomials since it currently handles only polynomials. For doing so, it tries to rewrite every such expression as a sum of monomials, i. e. $\sum_{j=0}^{n} c_j \prod_{v \in Var} v^{i_{v,j}}$, and checks whether all exponents $i_{v,j}$ evaluate to non-negative integers. During this step, any defined constant occurring in an expression is also be replaced by its definition.

For invariants and guards, HAL allows defining arbitrary logical formulae (`boolexpr`) of the form

$$\texttt{formula} ::= \texttt{formula} \bigtriangleup \texttt{formula} \mid (\texttt{formula}) \mid \neg\texttt{formula} \mid \texttt{formula} \sim \texttt{formula}$$

with logical operators $\bigtriangleup \in \{\texttt{and}, \texttt{or}, \texttt{impl}, \texttt{equiv}, \texttt{nand}, \texttt{nor}\}$ as well as comparators $\sim \in \{\texttt{<}, \texttt{<=}, \texttt{=}, \texttt{!=}, \texttt{>=}, \texttt{>}\}$ also having their usual meaning. Such logical formulae will be evaluated by STABHYLI and rewritten in DNF.

### Variables that are Required for Convergence

HAL allows to globally mark variables as *required to converge*, that is, $v \in Var'$ according to Definition 2.26 and, hence, STABHYLI will try to find a Lyapunov function which witnesses convergence of this set of marked variables. Since convergence of these variable might depend on the convergence of other variables for certain location, we have to identify these extra variables. In order to determine this set of variables that are required to converge locally to a location $l \in Loc$, we give the following recursive definition: a variable $v \in \mathit{Var}$ is *required to converge* if and only if the variable is marked as required to converge or it occurs on the right-hand side of a differential equation or inclusion of a variable that is required to converge. Thus, the *set of variables $\mathit{Var}_l$ that are required to converge locally to a location $l$* is:

$$\mathit{Var}_l := \mathit{Var}' \cup \left\{ v \;\middle|\; \exists v' \in \mathit{Var}_l \bullet v \in \mathit{RHS}\Big(\mathit{Flow}(l)_{\downarrow v'}\Big) \right\},$$

where $\mathit{Flow}(l)_{\downarrow v'}$ is the projection of $\mathit{Flow}(l)$ onto variable $v'$ which corresponds to the time derivative of $v'$. The set $\mathit{RHS}(f(\cdot))$ is the set of all variables occurring in $f(\cdot)$ with a non-zero coefficient. It is defined as:

$$\mathit{RHS}(f(\cdot)) := \mathit{Var} \setminus \left\{ v \in \mathit{Var} \;\middle|\; f(\mathbf{x}) = f(\mathbf{x}_{|\mathit{Var}\setminus\{v\}}) \right\}$$

Here, $\mathbf{x}_{|\mathit{Var}\setminus\{v\}}$ is the *orthogonal projection* on $\mathit{Var} \setminus \{v\}$ which sets all entries of $\mathbf{x}$ not in $\mathit{Var} \setminus \{v\}$ to zero.

Note, that the recursive definition of $\mathit{Var}_l$ is transitive and, thus, we can generate the set by the transitive closure, which can be compute in $\mathcal{O}\Big(|\mathit{Var}|^3\Big)$ time using the Floyd-Warshall algorithm [War62].

**Example 3.2**
*For the set of variables $\mathit{Var} = \{x, y, z\}$ with $\mathit{Var}' = \{y\}$ and a flow*

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} \in \operatorname{conv}\left(\left\{ \begin{bmatrix} -x \\ -x - 2.1y \\ 0 \end{bmatrix}, \begin{bmatrix} -1.5x \\ -1.1x - 2.1y \\ 0 \end{bmatrix} \right\}\right)$$

*for a location $l$, the set of variables, that are required to converge, is $\mathit{Var}_l = \{x, y\}$ since*

$$\begin{aligned} &\mathit{Var}' \cup \mathit{RHS}(\operatorname{conv}(\{-x - 2.1y, -1.1x - 2.1y\})) \cup \mathit{RHS}(\operatorname{conv}(\{0, 0\})) \\ =&\{y\} \cup \{x, y\} \cup \{\}. \end{aligned}$$

◁

**Remark 3.3**
*This is only a heuristic; for a monomial $x \cdot y$ that occurs on the right-hand side, it would be sufficient that either $x$ or $y$ converges to 0 but our heuristic demands both.* ◁

Another approach which is computationally too expensive would be to try every combination.

### 3.2.2 Solving Constraints

Stabhyli automatically generates the constraint system for a given hybrid automaton by applying Theorem 2.31. If the generated constraint system is feasible, then we obtain Lyapunov functions which — in turn — imply stability of the hybrid system. If the hybrid automaton is linear, then one can directly construct a system of LMIs. This approach and related techniques are discussed by Pettersson in his Dissertation [Pet99]. If the hybrid automaton is polynomial, then it is first required to transform the constraint system using the so-called sum-of-squares (SOS) decomposition (see Lemma 2.36) into an LMI [PP03]. In both cases, the $\mathcal{S}$-Procedure (see Theorem 2.32) is used to restrict the constraints to regions [BV04]. Finally, this yields an LMI problem, which can be solved by a semidefinite program (SDP) (see Definition 2.44).

These optimization problems can than be solved efficiently by tools like CSDP [Bor99], which is the solver that Stabhyli uses. Details have been given in Section 2.4.5.

A typical situation where numerical problems arise is that a constraint requires free parameters to have specific values (such as zero). This happens if for a location $l \in Loc$, the invariant contains the origin, i. e., $\mathbf{0} \in Inv(l)$ (in that case, it is required that $V(\mathbf{0}) = 0$). Although this specific case, where no additive constant for the Lyapunov function is allowed, is so common that Stabhyli handles it a priori. However, for others cases, it is more challenging to detect that a free parameter is required to have a specific value. A very common case are implicit equality constraints which we focus on in Section 3.3.

#### Obtaining Lyapunov Function Templates

Most common approaches use parameterized function templates for the search of Lyapunov functions see for example [**SankaranarayananCA2013**; BV04]. Providing such templates for the Lyapunov functions, allows us to search for a solution to the constraint system as defined by Theorem 2.31 since otherwise a search in the space of all (positive semi-definite) functions is needed.

An efficent approach is to use polynomials involving *free (or unknown) parameters* as the templates, and a solver's task is to find a good valuation of these parameters. As described above, having an additive term (i. e., the monomial of degree 0) in the Lyapunov function is only valid for locations whose invariant does not contain the origin, because otherwise the value of the Lyapunov function must be zero at the origin. Thus, Stabhyli will not add an additive free parameter if and only if the invariant contains the origin. Furthermore, Stabhyli allows the Lyapunov function template to involve only combinations of variables $v$ that have an associated flow function, i. e., $Flow(l)_{\downarrow v} \neq 0$. Apart from these exceptions, Stabhyli generates all combinations of variables that are contained in the Newton polytope as described in Definition 2.39 and whose degrees are bounded by a user-specified range.

### Refining Constraints

As mentioned above, Stabhyli uses a numerical solver as a backend. To further counteract numerical problems, we have implemented a refinement loop. This loop exploits the fact that due to technical reasons, numerical solvers — if not able to solve a constraint system — report valuations of free parameters that violate the least number of constraints. Our refinement loop therefore double-checks the result of the solver and reruns the solver using a refined problem. To obtain this refinement, Stabhyli replaces each occurrence of a parameter by

- zero if the parameter is supposed to be non-negative and the valuation is negative, or by
- a small positive number if the parameter is supposed to be positive and its valuation is non-positive.

In order not to narrow down the solution space too fast, Stabhyli eliminates those parameters first which do not occur in the objective and which have the largest distance to zero.

Stabhyli runs the check even if the solver reports success since it might happen that the solver returns solutions that satisfy the constraints only up to some threshold. Accordingly, we use the fact that a matrix is positive semidefinite if and only if it has only non-negative eigenvalues. Through this refinement, Stabhyli incrementally tries to build a valid solution even if the solver was not able to find one in the first place.

### Detection of Unsupportive Parameters

To detect unneeded parameters, Stabhyli analyzes the linear matrix inequality presentation of the constraint system. Given an LMI of the form

$$0 \preceq F_0 + \sum_{i=1}^{m} \rho_i F_i$$

where $F_i \in \mathbb{R}^{n \times n}$. We analyze the diagonal entries

$$\mathsf{diag}_j = F_{0,(j,j)} + \sum_{i=1}^{m} \rho_i F_{i,(j,j)}$$

for $1 \leq j \leq n$ in isolation. The diagonal entries are constraints linear in the parameters $\rho_1, \ldots, \rho_m$. For some of these parameters — like those belonging to the $\mathcal{K}^\infty$ functions $\alpha, \beta, \gamma$ as well as those belonging to $\mathcal{S}$-Procedure terms — we also know that they have to be non-negative. If we find any $\mathsf{diag}_j$ where $F_{i,(j,j)} \leq 0$ for $0 \leq i \leq m$, then we set all parameters $\rho_i$ to 0 where $F_{i,(j,j)} < 0$. We call these parameters *unsupportive*. This simplification exploits the fact that all principal minors including $\mathsf{diag}_j$ have to be non-negative (cf. Definition 2.46 and Theorem 2.47 ).

### Dimension Reduction

Due to the elimination of unsupportive parameters, it might happen that some rows and columns consists of zeros only. If there exists a $k = 1, \ldots, n$ such that for all $0 \leq i \leq m$ and all $1 \leq j \leq n$

$$0 = F_{i,(j,k)} \wedge 0 = F_{i,(k,j)}$$

holds, then we remove the $k$-th column and the $k$-th row of all matrices $F_i$.

### Strengthen Constraints

As mentioned in Section 2.4.5 we can use gaps to make constraints more robust against numerical issues. For STABHYLI we generate gaps as follows. Let $\epsilon$ be a — user-chosen — small positive value and let $\mathit{Var}_f \subseteq \mathit{Var}$ be the set of variables occurring in a constraint of the form

$$\forall \mathbf{x} \in P \subseteq \mathbb{R}^{|\mathit{Var}|} \bullet 0 \preceq f(\rho, \mathbf{x})$$

as imposed by Theorem 2.31. A possible gap function $g_\epsilon(\mathbf{x})$ has to take into account that this is a non-negativity constraint, i. e., $g_\epsilon(\mathbf{x}) \geq 0, \forall \mathbf{x} \in P$, and if $0 \in P$, then the gap should be zero. Therefore, the *gap function* $g_\epsilon(\mathbf{x})$ which we use for the constraint is

$$g_\epsilon(\mathbf{x}) = \epsilon \cdot \sum_{v \in \mathit{Var}_f} x_v^2$$

where $x_v$ is the entry in $\mathbf{x}$ corresponding to the variable $v$.

Clearly, finding a solution for

$$\forall \mathbf{x} \in P \subseteq \mathbb{R}^{|\mathit{Var}|} \bullet g_\epsilon(\mathbf{x}) \preceq f(\rho, \mathbf{x})$$

which we call the *strengthened constraint* — yields a valid solution for

$$\forall \mathbf{x} \in P \subseteq \mathbb{R}^{|\mathit{Var}|} \bullet 0 \preceq f(\rho, \mathbf{x}).$$

**Note 3.4**
*Let $\rho_{\mathsf{NUM}}$ be a candidate solution obtained by a numerical solver and let $P_{\mathsf{ERR}} \subseteq P$ be the set of all points where the strengthened constraint is violated, i. e., $g_\epsilon(\mathbf{x}_{\mathsf{ERR}}) \not\preceq f(\rho_{\mathsf{NUM}}, \mathbf{x}_{\mathsf{ERR}})$ if and only if $\mathbf{x}_{\mathsf{ERR}} \in P_{\mathsf{ERR}}$. In case $0 < f(\rho_{\mathsf{NUM}}, \mathbf{x}_{\mathsf{ERR}})$ for all $\mathbf{x}_{\mathsf{ERR}} \in P_{\mathsf{ERR}} \setminus \{\mathbf{0}\}$ and $0 = f(\rho_{\mathsf{NUM}}, \mathbf{0})$ whenever $\mathbf{0} \in P_{\mathsf{ERR}}$, then the candidate solution $\rho_{\mathsf{NUM}}$ is still valid for the original constraint.* ◁

## 3.3 Equality Detection and Handling

In the last section, we have introduced the general architecture of STABHYLI and have shown some basic concepts to counteract numerical issues. In this section, we present a more advanced heuristic to reduce numerical issues that prevent us from automatically computing Lyapunov functions. This simple but yet powerful heuristic detects implicitly specified equality constraints in a system of constraints that is generated for the search of Lyapunov functions.

The search for Lyapunov functions involves generating and solving sets of constraints as imposed by the Lyapunov theorem (see Theorem 2.31). As already mentioned, the methods we focus on require us to first generate a suitable set of Lyapunov function templates, i. e., the sequence $(V_l)$ as well as $\mathcal{K}^\infty$ functions. These templates involve *free parameters* $\rho = \begin{bmatrix}\rho_1 & \dots & \rho_n\end{bmatrix}$ for which we want to find a valuation such that the constraint system is satisfied. Given a hybrid system and specific Lyapunov function templates, the constraint system has the form

$$\bigwedge_i \forall \mathbf{x} \in P_i \bullet 0 \preceq f_i(\rho, \mathbf{x})$$

where $P_i \subseteq \mathcal{X}$ is a region corresponding to a guard of a transition or an invariant of a location and $0 \preceq f_i(\rho, \mathbf{x})$ is the non-negative condition of the individual location and transition constraints.

**Definition 3.5 (Parameterized Lyapunov Constraint System)**
*Let $\mathcal{H} = (Var, Loc, Trans, Flow, Inv, Inits)$ be a hybrid automaton with the continuous state space $\mathcal{X} = \mathbb{R}^{|Var|}$. Let $(V_l)_{l \in Loc}$ be a set of Lyapunov function templates with free parameters $\rho = \begin{bmatrix}\rho_1 & \dots & \rho_n\end{bmatrix}$. A* parameterized Lyapunov constraint system PLCS *for the hybrid automaton $\mathcal{H}$ and Lyapunov function templates $(V_l)$ is the set*

$$\mathsf{PLCS} := \{(P_i, f_i(\rho, \mathbf{x}))\}$$

*where each element $(P_i, f_i(\rho, \mathbf{x}))$ corresponds to either a location constraint or a transition constraint of the form*

$$\forall \mathbf{x} \in P_i \bullet 0 \preceq f_i(\rho, \mathbf{x})$$

*as imposed by Theorem 2.31. A* solution to an PLCS *is a valuation for $\rho$ such that all constraints corresponding to elements in the* PLCS *are satisfied. Hence, a solution yields a set of local Lyapunov function for the hybrid automaton $\mathcal{H}$.* ◇

The set of local Lyapunov function can be obtained by substituting the free parameters in the templates $(V_l)$ with the corresponding elements of valuations for $\rho$.

As mentioned in Section 2.4.2, we distinguish *conditioned* and *unconditioned* constraints.

**Definition 3.6 (Conditioned and Unconditioned Constraint)**
*A constraint of the form*

$$\forall \mathbf{x} \in P \bullet 0 \preceq f(\rho, \mathbf{x})$$

*where $f(\rho, \mathbf{x})$ is a parameterized polynomial, $P \subseteq \mathcal{X}$ is a subset of the continuous state space, and $P$ is called the condition. The constraint is either called*

- unconditioned *if and only if $P = \mathbb{R}^n$ or*
- conditioned *if and only if $P \subsetneq \mathbb{R}^n$.*

$\diamondsuit$

The $\mathcal{S}$-Procedure allows us to convert conditioned constraints into unconditioned constraints at the price of extra parameters. The conversion might hide simple relations between the functions $(f_i)$ as the following motivating example shows

**Example 3.7**
*Suppose the following excerpt of constraints was generated by naïvely applying the Lyapunov theorem:*

$$\forall \mathbf{x} \in P_1 : 0 \preceq f_1(\rho, \mathbf{x})$$
$$\forall \mathbf{x} \in P_2 : 0 \preceq f_2(\rho, \mathbf{x})$$

*where $P_1 = P_2 \subseteq \mathbb{R}^n$ and $f_1 = -f_2$. Since these conditioned constraints cannot directly be given to an SDP solver, we have to apply the $\mathcal{S}$-Procedure (see Theorem 2.32) which then constructs the following representation:*

$$0 \preceq f_1(\rho, \mathbf{x}) - p_1(\rho_1, \mathbf{x})$$
$$0 \preceq f_2(\rho, \mathbf{x}) - p_2(\rho_2, \mathbf{x})$$

*where $p_1$ (resp. $p_2$) are parameterized polynomials encoding the conditions $\mathbf{x} \in P_1$ (resp. $\mathbf{x} \in P_2$). Since $P_1$ equals $P_2$ also $p_1$ equals $p_2$, except that do to the $\mathcal{S}$-Procedure, $p_1$ and $p_2$ have disjoint sets of parameters, i. e., $\rho_1$ and $\rho_2$. Having the sets disjoint is in most cases good, because it allows more freedom in case $f_1 \neq -f_2$. But in the special case of $f_1 = -f_2$, it imposes unnecessary difficulties since a solver is required to assign some free parameters exact valuations, which is — unfortunately — bad.*

*Consider the case $p_1(\cdot, \mathbf{x}) = p_2(\cdot, \mathbf{x})$ and $f_1 = -f_2$:*

$$\begin{aligned}
&0 \preceq f_1(\rho, \mathbf{x}) - p_1(\rho_1, \mathbf{x}) \wedge 0 \preceq f_2(\rho, \mathbf{x}) - p_2(\rho_2, \mathbf{x}) \\
\Leftrightarrow &0 \preceq f_1(\rho, \mathbf{x}) - p_1(\rho_1, \mathbf{x}) \wedge 0 \preceq -f_1(\rho, \mathbf{x}) - p_1(\rho_2, \mathbf{x}) \\
\Leftrightarrow &p_1(\rho_1, \mathbf{x}) \preceq f_1(\rho, \mathbf{x}) \preceq -p_1(\rho_2, \mathbf{x}) \\
\Rightarrow &p_1(\rho_1, \mathbf{x}) \preceq -p_1(\rho_2, \mathbf{x}).
\end{aligned}$$

*Due to the definition of the $\mathcal{S}$-Procedure, we know that $p_1(\rho_1, \mathbf{x})$ is a polynomial which is linear in $\rho_1$ and we can write*

$$\begin{aligned} & p_1(\rho_1, \mathbf{x}) \preceq -p_1(\rho_2, \mathbf{x}) \\ \Leftrightarrow & 0 \preceq -(p_1(\rho_1, \mathbf{x}) + p_1(\rho_2, \mathbf{x})) \\ \Leftrightarrow & 0 \preceq -(p_1(\rho_1 + \rho_2, \mathbf{x})). \end{aligned}$$

*Further, since $p_1(\rho_1 + \rho_2, \mathbf{x})$ was obtained from the region $P_1$ by the $\mathcal{S}$-Procedure it is, therefore, of the form $p_1(\rho_1 + \rho_2, \mathbf{x}) = \sum_i (\rho_{1,i} + \rho_{2,i}) \cdot p_{1,i}(\mathbf{x})$ with $\rho_{1,i} + \rho_{2,i} \geq 0$. Moreover, we know that $\mathbf{x} \in P_1$ if and only if for all $i$ holds $0 \leq p_{1,i}(\mathbf{x})$.*

*In the following, we show that this implies that $\rho_1 + \rho_2 = \mathbf{0}$. Let us assume the contrary, i. e., at least one element $\rho_{1,j} + \rho_{2,j} > 0$, and show that this is a contradiction.*

*Let us exclude the trivial case — where all $p_{1,i}(\mathbf{x}) = 0$ for all $\mathbf{x} \in P_1$ — and choose an arbitrary point $\mathbf{x}^\star \in P_1 \setminus \{\mathbf{0}\}$ for which exists $p_{1,j}(\mathbf{x}^\star) > 0$. Now, in order to satisfy $0 \preceq -p_1(\rho_1 + \rho_2, \mathbf{x})$ it has to hold that*

$$0 < \underbrace{(\rho_{1,j} + \rho_{2,j})}_{>0} \cdot \underbrace{p_{1,j}(\mathbf{x}^\star)}_{>0} \leq -\sum_{i \neq j} \underbrace{(\rho_{1,i} + \rho_{2,i})}_{\geq 0} \cdot \underbrace{p_{1,i}(\mathbf{x}^\star)}_{\geq 0} \leq 0 \qquad \lightning$$

*which is a contradiction and, hence, $\rho_{1,j} + \rho_{2,j} > 0$ cannot be true.*

*We conclude that $\rho_1 + \rho_2 = \mathbf{0}$ must be true.* ◁

In practice, the problem is even worse, since numerical solvers sometimes suffer from numerical inaccuracies. As mentioned in the previous section, additional gap functions are added to the inequalities to render the constraints more robust against such numerical issues.

Consider the following example which often arises in case we have to express equalities.

**Example 3.8**

*Assume we want to express the equality constraint $\forall \mathbf{x} \in P : f(\mathbf{x}) = 0$, which we have to encode with two inequalities. Naivly, we would write*

$$\begin{aligned} \forall \mathbf{x} \in P_1 : 0 \preceq f_1(\rho, \mathbf{x}) \\ \forall \mathbf{x} \in P_2 : 0 \preceq f_2(\rho, \mathbf{x}) \end{aligned}$$

*with $f_1 = -f_2 = f$ and $P_1 = P_2 = P$.*

*Further, assume that in order to reduce numerical issues, we have introduced gap functions. Let $g_i$ with $\forall \mathbf{x} \in P_i : 0 \preceq g_i(\mathbf{x})$ and $0 < g_i(\mathbf{x})$ for some $\mathbf{x}^\star \in P_i \setminus \{\mathbf{0}\}$ where $i \in \{1, 2\}$ be such gap functions. Instead of solving the above constraints, we try to solve the following strengthened constraints:*

$$\begin{aligned} \forall \mathbf{x} \in P_1 : g_1(\mathbf{x}) \preceq f_1(\rho, \mathbf{x}) \\ \forall \mathbf{x} \in P_2 : g_2(\mathbf{x}) \preceq f_2(\rho, \mathbf{x}). \end{aligned}$$

*In this case where $P_2 = P_1$ and $f_2 = -f_1$, the strengthened constraints do not yield a solution since*

$$\begin{aligned} & 0 \leq g_1(\mathbf{x}^\star) \leq f_1(\rho, \mathbf{x}^\star) \wedge 0 \leq g_2(\mathbf{x}^\star) \leq -f_1(\rho, \mathbf{x}^\star) \\ \Leftrightarrow\ & 0 \leq g_1(\mathbf{x}^\star) \leq f_1(\rho, \mathbf{x}^\star) \wedge f_1(\rho, \mathbf{x}^\star) \leq -g_2(\mathbf{x}^\star) \leq 0 \\ \Leftrightarrow\ & 0 \leq \underbrace{g_1(\mathbf{x}^\star)}_{>0} \leq f_1(\rho, \mathbf{x}^\star) \leq \underbrace{-g_2(\mathbf{x}^\star)}_{<0} \leq 0 \end{aligned}$$

*is obviously a contradiction unless $P_1 \subseteq \{0\}$. But the original conditioned constraint $\forall \mathbf{x} \in P : f(\mathbf{x}) = 0$ may be easy to solve.* ◁

Our heuristic tries to detect such situations. The heuristic analyzes the conditioned constraints before they are converted into unconditioned constraints by the $\mathcal{S}$-Procedure. After using the heuristic to detects implicit equalities, we aim to eliminate them using a substitution of free parameters in the other constraints.

**Definition 3.9 (Implicit Equality Constraint over a Region)**
*Let $\mathcal{H}$ be a hybrid automaton and* PLCS *be the parameterized Lyapunov constraints system of $\mathcal{H}$. $(P, f_i(\rho, \mathbf{x}))$ is called an* implicit equality constraint over $P$ *if*

- *$(P_i, f_i(\rho, \mathbf{x})) \in$* PLCS *is an element of the parameterized Lyapunov constraint system,*
- *$P \neq \emptyset$,*
- *every solution $\rho$ of the* PLCS *implies $0 = f_i(\rho, \mathbf{x})$ for all $\mathbf{x} \in P$.*

◇

Our technique can be seen as the exploitation of linear dependence in the parameterized Lyapunov constraint system before this information is not preserved during the $\mathcal{S}$-Procedure. Similar techniques to detect implicit equalities are used, for example, in satisfiability modulo theories solving for linear arithmetic [Li+09]. However, we have to deal with universally quantified polynomial constraints for which we show how to handle them in the special settings, where the constraints arise from proving stability using Lyapunov functions.

To algebraically handle polynomial optimization problems, we introduce parameterized versions of monomials and polynomials.

**Definition 3.10 (Parameterized Polynomial)**
*A polynomial $f(\rho, \mathbf{x})$ is called a* parameterized polynomial *if it has the form $f(x) = \sum_j c_j \rho_j \prod_{v \in Var} v^{e_{v,j}}$ where $c_j$ is a* coefficient*, $\rho_j$ is a* parameter*, and $\rho_j \prod_{v \in Var} v^{e_{v,j}}$ is called a* parameterized monomial. ◇

In practice, the parameter in a parameterized monomial is rather optional, but to increase readability and shorten the formulas, we assume every summand in a parameterized polynomial to have a parameter where a dummy parameter might also represent a constant 1.

### 3.3.1 Simplifying Constraint Systems

Next, we give the problem statement and then describe the heuristic used to simplify the constraint systems.

For our heuristic we restrict the search for implicit equalities over a region $P$ to the case where $P$ is the condition of the constraint, i. e., $P = P_i$ in Definition 3.9. This restriction has practical reasons and reduces the number of constraints that have to be considered simultaneously.

**Definition 3.11 (Matching Constraint)**
*Let $(P, f(\rho, \mathbf{x})) \in \mathsf{PLCS}$ be a constraint of a parameterized Lyapunov constraint system. A constraint $(P, g(\rho, \mathbf{x}))$ is called a* matching constraint *if and only if $0 \preceq f(\rho, \mathbf{x}) \wedge 0 \preceq g(\rho, \mathbf{x})$ for all $\mathbf{x} \in P$ implies that $0 = f(\rho, \mathbf{x}) = g(\rho, \mathbf{x})$ for all $\mathbf{x} \in P$.* ♢

Note that $(P, g(\rho, \mathbf{x}))$ in Definition 3.11 might not be an element of the parameterized Lyapunov constraint system. Therefore,, usually, we have to construct such a matching constraint by combining other constraints.

**Problem 3.12**
*Let $\mathsf{PLCS}$ be a parameterized Lyapunov constraint system with elements $(P_i, f_i(\rho, \mathbf{x}))$. We want to find a constraint $(P_k, f_k(\rho, \mathbf{x}))$ such that there are non-negative scalars $d_1, \dots, d_n \geq 0$ where*

- *$P_i \neq P_k$ implies $d_i = 0$ and*
- *$\sum_i d_i \cdot f_i(\rho, \mathbf{x}) = -f_k(\rho, \mathbf{x})$ for all $\mathbf{x} \in P_k$.*

♣

**Theorem 3.13**
*Let $\mathsf{PLCS}$ be a parameterized Lyapunov constraint system with elements $(P_i, f_i(\rho, \mathbf{x}))$. Let $d_1, \dots, d_n \geq 0$ be non-negative scalars. If there is a $k$ such that $P_i \neq P_k$ implies $d_i = 0$ and $\sum_i d_i \cdot f_i(\rho, \mathbf{x}) = -f_k(\rho, \mathbf{x})$ for all $\mathbf{x} \in P_k$, then $(P_k, -f_k(\rho, \mathbf{x}))$ is a matching constraint and $(P_k, f_k(\rho, \mathbf{x}))$ is an implicit equality over $P_k$.*

**Proof.**
*Let $I$ be the set of indices for which $d_i > 0$. Let $\rho$ be an arbitrary solution of the $\mathsf{PLCS}$. This means that all constraints $(P_i, f_i(\rho, \mathbf{x})) \in \mathsf{PLCS}$ are satisfied.This especially includes the constraint $(P_k, f_k(\rho, \mathbf{x}))$ and all $(P_i, f_i(\rho, \mathbf{x}))$ for $i \in I$. Since $d_i > 0$ implies $P_i = P_k$, we also know that $(P_k, f_i(\rho, \mathbf{x}))$ for $i \in I$ are satis-*

> *fied. Since positive semidefinite functions are closed conic combination, we conclude that* $(P_k, \sum_i d_i \cdot f_i(\rho, \mathbf{x})) = (P_k, -f_k(\rho, \mathbf{x}))$ *is also satisfied. Finally, if both* $(P_k, -f_k(\rho, \mathbf{x}))$ *and* $(P_k, f_k(\rho, \mathbf{x}))$ *are satisfied, then they can be satisfied only with equality, i. e.,* $f_k(\rho, \mathbf{x}) = 0$ *for all* $\mathbf{x} \in P_k$*.* □

Roughly speaking, we are searching for constraints for which one can obtain a matching constraint via a conic combination of the other constraints. In theory, the factors $d_i$ do not need to be restricted to scalar factors. Instead, they can be arbitrary positive semidefinite monomials. This is due to the well-known fact that the result of a product of positive semidefinite functions is again positive semidefinite. Without this restriction, we could find the implicit equality in the following example:

> **Example 3.14**
> *Given the following system of conditioned constraints:*
>
> $$\forall x \in P : \rho_1 x^2 \preceq \rho_2 x^2 \tag{3.1}$$
> $$\forall x \in P : \rho_2 x^4 \preceq \rho_1 x^4 \tag{3.2}$$
>
> *by multiplying the inequality in Constraint 3.1 by* $x^2$*, we obtain*
>
> $$\forall x \in P : \rho_1 x^4 \preceq \rho_2 x^4 \wedge \rho_2 x^4 \preceq \rho_1 x^4$$
>
> *from which we conclude that*
>
> $$\forall x \in P : \rho_1 x^4 = \rho_2 x^4.$$
>
> ◁

However, we restrict the factors to scalars since implicit equality constraints are mainly caused by the transition constraints (constraint type C3 of Theorem 2.31). Therefore, a situation such as in Example 3.14 does not need to be handled because

- only the transition constraints are responsible for relating the individual Lyapunov function templates and
- a free parameter $\rho_i$ usually occurs only in a single parameterized monomial of the Lyapunov function template.

Furthermore, due to this restriction, a simple linear program (LP) can be used to search for implicit equality constraints. In the following, we show how to obtain such a linear program.

We start with a parameterized Lyapunov constraint system PLCS obtained by generating all constraints required by Theorem 2.31. Then, we group the constraints of the PLCS by their condition, thereby obtaining a finite number of groups of constraints.

Each such group has the following form:

$$\begin{aligned}&\forall \mathbf{x} \in P \bullet\\ &\qquad 0 \preceq f_1(\rho, \mathbf{x}) = \sum_j c_{(j,1)} \rho_{(j,1)} \prod_{v \in \mathit{Var}} v^{e_{(v,j,1)}}\\ &\wedge \quad \ldots\\ &\wedge\, 0 \preceq f_n(\rho, \mathbf{x}) = \sum_j c_{(j,n)} \rho_{(j,n)} \prod_{v \in \mathit{Var}} v^{e_{(v,j,n)}}.\end{aligned}$$

Here, $c_{j,k}$ is the $j$-th coefficient of the $k$-th constraint belonging to the $j$-th parameterized monomial of the $k$-th constraint and, similar, $\rho_{j,k}$ is the $j$-th parameter of the $k$-th constraint.

In each group, we now search for constraints $(P_k, f_k(\rho, \mathbf{x})) \in \mathsf{PLCS}$ for which we can syntactically construct a matching constraint as a conic combination of the other constraints:

$$\exists d_1, \ldots, d_n \geq 0 \bullet \sum_i d_i f_i(\rho, \mathbf{x}) = -f_k(\rho, \mathbf{x})$$

or equivalently:

$$\exists d_1, \ldots, d_n \geq 0 \bullet \sum_{i \neq k} d_i f_i(\rho, \mathbf{x}) + f_k(\rho, \mathbf{x}) = 0$$

By syntactically replacing all parameterized monomials $\rho_{(j,i)} \prod_{v \in \mathit{Var}} v^{e_{(v,j,i)}}$ by a new symbol $z_{(j,i)}$, we have:

$$\begin{aligned}\exists d_1, \ldots, d_n \geq 0 \bullet &\sum_i d_i \sum_j c_{(j,i)} z_{(j,i)}\\ &+ \sum_j c_{(j,k)} z_{(j,k)} = 0.\end{aligned}$$

Reordering the $m$ syntactical different monomials $z_1, \ldots, z_m$ leads to:

$$\begin{aligned}\exists d_1, \ldots, d_n \geq 0 \bullet &\sum_i d_i \sum_j^m c_{(j,i)} z_j\\ &+ \sum_j^m c_{(j,k)} z_j = 0.\end{aligned}$$

By grouping the monomials $z_j$, we obtain

$$\begin{aligned}&\exists d_1, \ldots, d_n \geq 0 \bullet\\ &\qquad \sum_j^m \left( \left( \sum_i d_i c_{(j,i)} \right) + c_{(j,k)} \right) z_j = 0.\end{aligned}$$

Now, observe that a solution for the linear feasibility problem

$$\begin{aligned}&\text{find}\, d_1, \ldots, d_n \text{ such that}\\ &\qquad \forall 1 \leq j \leq m \bullet \sum_i d_i c_{(j,i)} = -c_{(j,k)}\end{aligned}$$

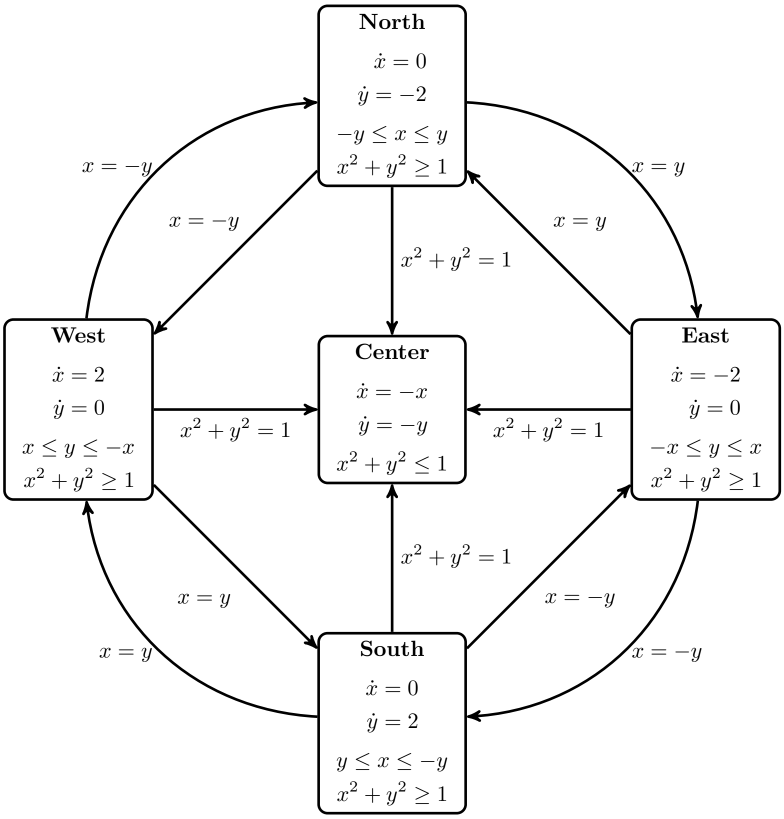


Figure 3.7: Hybrid system describing a robot automatically approaching a certain region `Center`.

with $d_i \geq 0$, yields a valid solution to the above constraint system and

$$\forall \mathbf{x} \in P_k \bullet 0 \preceq \sum_i d_i f_i(\rho, \mathbf{x})$$

is a constructible matching constraint for the $k$-th constraint. Note, that in order to ease finding a solution for the LP, the parameter $d_k$ should, a priori, be set to 0. Also note, that if it is known that the templates are not scaled, i. e. the constraints are not normalized, then the problem can even be reduced to a binary program where $d_i \in \{0, 1\}$.[1]

**Example 3.15**
*Figure 3.7 shows a hybrid automaton modeling a simple robot whose goal is to reach a certain target region* `Center` *independent of the starting point (somewhere in* `North`, `East`, `South`, *or* `West`*) with a maximal velocity of* $2km/h$. *Whenever it is*

[1] At first glance, this seems to be a bad idea in terms of complexity since binary or integer programs are NP-complete while linear programs can be solved in polynomial time. However, over the last decades impressive improvements have been made and SAT solving is very fast in practice. Thus, is might serve as an alternative.

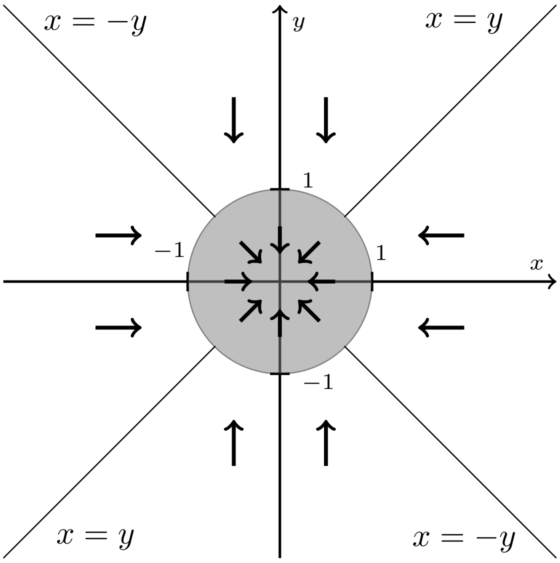


Figure 3.8: Vector field of the robot's movement defined in Figure 3.7.

*not yet close to the* `Center`*, it drives full throttle in the cardinal direction of the center (thus, its direction depends only on the quadrant the robot is in). Being close to (less than one kilometer) the target, the robot's strategy changes: now, it follows a radio signal directing it directly towards the target. The vector field is visualized in Figure 3.8.*

*Generating the parameterized Lyapunov constraint system for this hybrid automaton leads to three constraints per location and one constraint per transition — thus, 27 constraints in total. Here, we focus only on the constraints generated for the* *transitions which are:*

$$x = y \Rightarrow V_N \preceq V_E \tag{3.3}$$
$$x = y \Rightarrow V_E \preceq V_N \tag{3.4}$$
$$x = y \Rightarrow V_S \preceq V_W \tag{3.5}$$
$$x = y \Rightarrow V_W \preceq V_S \tag{3.6}$$
$$x = -y \Rightarrow V_N \preceq V_W \tag{3.7}$$
$$x = -y \Rightarrow V_W \preceq V_N \tag{3.8}$$
$$x = -y \Rightarrow V_E \preceq V_S \tag{3.9}$$
$$x = -y \Rightarrow V_S \preceq V_E \tag{3.10}$$
$$x^2 + y^2 = 1 \Rightarrow V_C \preceq V_N \tag{3.11}$$
$$x^2 + y^2 = 1 \Rightarrow V_C \preceq V_E \tag{3.12}$$
$$x^2 + y^2 = 1 \Rightarrow V_C \preceq V_S \tag{3.13}$$
$$x^2 + y^2 = 1 \Rightarrow V_C \preceq V_W \tag{3.14}$$

*where $V_C, V_N, V_E, V_S, V_W$ are the templates for the Lyapunov function for the individual locations* `Center`, `North`, `East`, `South`, *and* `West`, *respectively. By applying the procedure described above, we derive that Constraint 3.3 and Constraint 3.4, Constraint 3.5 and Constraint 3.6, Constraint 3.7 and Constraint 3.8, and Constraint 3.9 and Constraint 3.10 are pairwise matching constraints. In the textual presentation above, this can be easily seen, but matching constraints can be much harder to detect — even automatically — in practice, since the templates are not given in this explicit form but instead are given as parameterized polynomials. The templates might be $V_i = \rho_{(i,1)}x^2 + \rho_{(i,2)}xy + \rho_{(i,3)}y^2 + \rho_{(i,4)}x + \rho_{(i,5)}y + \rho_{(i,6)}$ for $i \in \{N, E, S, W\}$ and $V_C = \rho_{(C,1)}x^2 + \rho_{(C,2)}xy + \rho_{(C,3)}y^2 + \rho_{(C,4)}x + \rho_{(C,5)}y$, which are good candidates in this example. However, we conclude that*

$$x = y \Rightarrow V_N = V_E, \tag{3.15}$$
$$x = -y \Rightarrow V_N = V_W, \tag{3.16}$$
$$x = -y \Rightarrow V_E = V_S, \tag{3.17}$$
$$x = y \Rightarrow V_S = V_W. \tag{3.18}$$

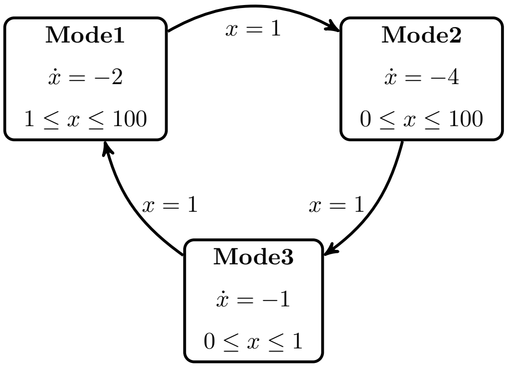


Figure 3.9: Hybrid system showing that implicit equality constraints might involve more than two transitions.

*Thus, in case of Equality 3.15 together with the condition, we have*

$$\begin{aligned} x = y \Rightarrow \quad & (\rho_{(N,1)} - \rho_{(E,1)})x^2 \\ & + (\rho_{(N,2)} - \rho_{(E,2)})xy \\ & + (\rho_{(N,3)} - \rho_{(E,3)})y^2 \\ & + (\rho_{(N,4)} - \rho_{(E,4)})x \\ & + (\rho_{(N,5)} - \rho_{(E,5)})y \\ & + (\rho_{(N,6)} - \rho_{(E,6)}) = 0. \end{aligned}$$

*By eliminating the condition, we get the unconditioned constraint*

$$\begin{aligned} & (\rho_{(N,1)} + \rho_{(N,2)} + \rho_{(N,3)} \\ & \quad - \rho_{(E,1)} + \rho_{(E,2)} + \rho_{(E,3)})y^2 \\ & + (\rho_{(N,4)} + \rho_{(N,5)} - \rho_{(E,4)} + \rho_{(E,5)})y \\ & + (\rho_{(N,6)} - \rho_{(E,6)}) = 0 \end{aligned}$$

*in which we can now isolate one of the parameterized monomials and use it to substitute it in the other constraints. Here, a promising candidate is to replace* $\rho_{(E,6)}$ *by* $(\rho_{(N,1)} + \rho_{(N,2)} + \rho_{(N,3)} - \rho_{(E,1)} + \rho_{(E,2)} + \rho_{(E,3)})y^2 + (\rho_{(N,4)} + \rho_{(N,5)} - \rho_{(E,4)} + \rho_{(E,5)})y + \rho_{(N,6)}$. *By doing so, we reduce the number of free parameters by one and the number of constraints by two. We can repeat the procedure with the other three implicit Equalities 3.16 to 3.18 and remove six more constraints and three more parameters.* ◁

**Example 3.16**
*The third example, which is given in Figure 3.9, does not describe a system for a*

> *particular purpose. But it shows that implicit equality constraints can involve more than two constraints. Here, the transitions require:*
>
> $$x = 1 \Rightarrow V_2 \preceq V_1 \tag{3.19}$$
> $$x = 1 \Rightarrow V_3 \preceq V_2 \tag{3.20}$$
> $$x = 1 \Rightarrow V_1 \preceq V_3 \tag{3.21}$$
>
> *where $V_1$, $V_2$, and $V_3$ are the Lyapunov function templates for* `Mode1`*,* `Mode2`*, and* `Mode3`*, respectively. By combination of the constraints, we can conclude:*
>
> $$x = 1 \Rightarrow V_1 = V_2 = V_3 \tag{3.22}$$
>
> *which allows us to remove three constraints and two free parameters from the constraint system.* ◁

For all three examples, Lyapunov functions can only be found if

(a) no gaps are used to strengthen the constraints which, then, risks that the values returned by a solver do not form a valid Lyapunov function or

(b) implicit equality constraints are detected and resolved which, thereby, discards the need for gaps on these constraints.

The current implementation in Stabhyli is still incomplete. For example Stabhyli does not check every possible isolateable term and uses a greedy approach instead. These limitations have been addressed in the thesis of Zschoche, where he developed a backtracking approach [Zsc15]. He has also further investigated the problem of implicit equalities. The contributions presented in [Zsc15] include

- generalization of the detection to implicit equalities without structural identical conditions, and
- identification of a strong relation between cycles in the hybrid automaton's underlying graph and implicit equalities.

On one hand, the latter allows us to check for implicit equalities local to a cycle of the underlying graph but, on the other hand, requires us to enumerate cycles. This enumeration is very costly because the number of cycles can be exponential in the number nodes. However, an integration into the decomposition method (see Section 2.5) seems to be interesting as the decomposition already enumerates cycles.

## 3.4 Validation of Candidate Solutions

In this section, we present experiments on combining satisfiability modulo theory (SMT) methods with state-of-the-art numerical solvers for the validation of candidate solutions during the search for Lyapunov functions. In the last sections, we have shown how to make the constraint systems — that arise during the search for Lyapunov functions — more robust with respect to numerical issues and how to detect implicit equalities. Both techniques increase the chance to obtain solutions but do not avoid getting invalid results. Therefore, one still needs to validate the result. Stabhyli uses the computation of eigenvalues and principal minors (cf. Theorem 2.47) to ratify the validity of a solution. However, the computation of eigenvalues is usually done via numerical algorithms which, again, may suffer from numerical issues.

Instead, we would like to identify and use a method which is precise. In [MT14], we propose the use of SMT solvers to validate candidate solutions. An SMT solver combines SAT solvers with theory solvers to check the satisfiability (SAT) over first order logic (FOL) formulae with respect to first-order theories. One instance of such a theory is non-linear real arithmetic (NRA). A formula is satisfiable if there exists a model — a valuation for each free variable — such that using the model, the formula evaluates to `true`. Since, general first order logic (FOL) is semi-decidable, i. e., there exists a procedure such that, given a formula, the procedure eventually decides whether the formula is valid. Due to Tarski [Tar51], we know that non-linear real arithmetic (NRA), on the other hand, is decidable. The complexity is, however, double exponential in the number of variables as proven for the upper bound by Collins [Col75] and for the lower bound by Davenport and Heintz [DH88] and Weispfenning [Wei88].

SMT solving has received quite some attention over the last years but, unfortunately, most available implementations are either restricted to linear arithmetic or do not support quantifiers. There is only a handful of SMT solvers which support NRA with quantifiers: Z3 [DB08], CVC3 [BT07], and CVC4 [Bar+11]. As stated above, SMT is already combining boolean satisfiability problem (SAT) solving with different theory solvers. Nevertheless, other combinations built on top of SMT solvers have been realized. One approach combines a linear SMT solver with interval constraint propagation to solve non-linear real arithmetic problems [Gao+10]. Another approach combines convex programming with SMT solving to solve non-linear convex constraints [Nuz+10]. Gao et al. [Gao+10] also use the result of a linear SMT solver to validate answers of the interval constraints propagation method. This is very similar to what we do, but they consider only quantifier-free formulas. In the search for a solution of the constraint systems, however, we have a single quantifier alternation that is an outer existential and an inner universal quantifier. Thus, we cannot apply their method due to the lack of the quantifiers.

Referring to the list of tools in Section 3.1, all Lyapunov-based tools somehow use numerical — approximative — methods. This means that the obtained candidate solutions should better be validated to ensure correctness.

### 3.4.1 Validating Solver Engine

In the following, we describe the validating solver engine. This engine is the integrated engine used by Stabhyli to compute Lyapunov function. It consists of

1. the implicit equality detection and elimination described in Section 3.3,

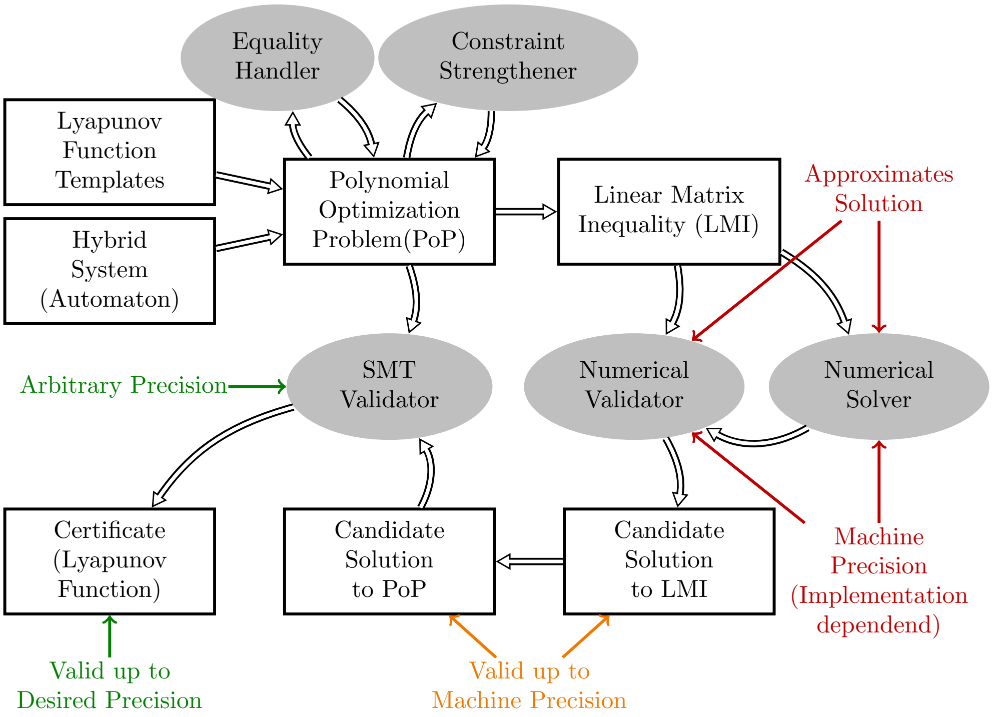


Figure 3.10: Overview of the validating solver engine.

2. the strengthening, the simplifications, and the conversion to linear matrix inequalities described in Section 3.2.2 and

3. the validation described in this section.

A hybrid automaton in this context might be a describtion of the full system or of a subsystem or even a system of systems. In case of the proof schemes Piecewise Lyapunov Function via Decomposition and Piecewise Lyapunov Function via Composition such subsystems are automatically constructed, e. g., the subsystems corresponding to cycles in the underlying graph. In case of the proof scheme common Lyapunov function scheme, the Lyapunov function templates might be identical for all locations.

Figure 3.10 gives an overview on the steps of the validating solver engine. In this sketch of the process, the big arrows describe the dependencies, boxes are input, output, and intermediate artefacts, and ovals are process steps that are addressed — in this thesis — in more detail. On the right it is indicated the numercial solver and validator do only approximate solutions, hence, it is indicated in the middle, that the candiate solutions are valid up to machine precision, and on the left it is indicated, that the SMT validated candidate solution are valid with arbitrary precision. The individual steps of the validating solver engine are as follows:

1. In the first step, a linear polynomial optimization problem (LPoP) is constructed from the Lyapunov function templates and the description of the hybrid automaton.

2. In the second step, an equality handler and a constraint strengthener post-process the linear polynomial optimization problem (LPoP).

3. In the third step, the LPoP is relaxed (via $\mathcal{S}$-Procedure and SOS decomposition) into a linear matrix inequality (LMI).

4. In the fourth step, an SDP solver is used to obtain a candidate solution for the LMI.

5. In the fifth step, the LMI candidate solution is numerically validated (e. g., by computing eigenvalues[2] and checking principal minors).
   If the candidate solution to the LMI did not pass this validation, then the process stops and reports no success.

6. In the sixth step, a candidate solution to the LPoP is derived from the candidate solution to the LMI.

7. In the last step, the candidate solution to the LPoP is further validated by an SMT solver. If the candidate solution to the LPoP did not pass this validation, then the process stops and reports no success. Otherwise, the candidate solution is successfully validated and a valid stability certificate will be reported..

The SDP solver and the numerical validator usually approximate a solution, i. e., they start with an initial valuation of the free parameters. Then, they optimize the valuation in the direction which minimizes a given objective function as well as the error.[3] If (a) a certain accuracy is obtained, (b) a maximum number of steps is reached, or (c) the progress becomes too small, then the algorithm stops. Clearly, we can always increase the maximum number of steps and the desired accuracy but the closer the current valuation gets to an optimal solution, the smaller the progress becomes. Therefore, it is not always possible to reach the optimal solution.

Recall, that the resulting LMIs is to be solved by a numerical solver and — unfortunately — this kind of solvers do only approximate solutions and additionally suffer from numerical issues. Prior to applying the $\mathcal{S}$-Procedure and SOS decomposition, the parameterized Lyapunov constraint system has the following form:

$$\bigwedge_i \forall \mathbf{x} \bullet \left( \bigwedge_j 0 \leq g_{(j,i)}(\mathbf{x}) \right) \Rightarrow 0 \preceq f_i(\rho, \mathbf{x})$$

where $\rho = \begin{bmatrix} \rho_1 & \dots & \rho_m \end{bmatrix}^T$ are the free parameters (stemming from the templates and $\mathcal{K}^\infty$ functions), $\mathbf{x} \in \mathbb{R}^n$ are the system variables (stemming from the hybrid system), each $g_{(j,i)}(\mathbf{x}) : \mathbb{R}^n \mapsto \mathbb{R}$ is a parameter-free polynomial describing a certain region (the

[2] Note, that computing eigenvalues has issues on its own since it is well-known to be numerically ill-conditioned [Wil94]. A problem is said to be ill-conditioned, if a small change in the input, may result in a huge change in the output.

[3] Roughly speaking, the error determines the quality of the solution: the closer to 0, the better (in the sense of feasibility) the solution is.

condition of the constraint), and $f_i(\rho, \mathbf{x}) : \mathbb{R}^m \times \mathbb{R}^n \mapsto \mathbb{R}$ are polynomials which are linear in the free parameters $\rho$. The goal is to find a valuation for $\rho$.

Example 3.17 shows a very simple instance of such a constraint system.

**Example 3.17**

$$\forall x \in \mathbb{R} \bullet \texttt{true} \Rightarrow 0 \preceq \rho \cdot x^2$$

◁

When trying to find a feasible solution for $\rho$, a numerical solver might return the candidate solution $\rho_{\mathsf{NUM}} = -1 \cdot 10^{-k}$, where $k$ is sufficiently large. On one hand, this is in most cases sufficient and allows us to conclude feasibility of the above problem. On the other hand, reusing this candidate solution might render any successive calculation invalid. However, Stabhyli composes Lyapunov functions as conic combinations of other Lyapunov functions. This means that in order to guarantee validity of the composed Lyapunov function, we need to assure validity of the combined Lyapunov functions.

We tried to directly solve constraint systems obtained from applying the Lyapunov Theorem, using the SMT solvers Z3 [DB08], CVC3 [BT07], and CVC4 [Bar+11]. Unfortunately, none of these solvers were able to solve relevant instances — the solvers always returned `unknown`. However, we were able to use SMT solvers to successfully (in-)validate candidate solutions. This use of SMT solvers for validating solutions is beneficial since it can be done with exact arithmetic, e. g., using rationals with full precision. This allows us to conclude that a candidate solution is actually valid and further computation that reuse the candidate solution is also trustworthy.

**Remark 3.18**
*If we want solve the constraint systems directly using an SMT solver, then we can only use solvers that support the logic `NRA` (non-linear real arithmetic) since the constraints involve polynomials. In contrast, if we want to validate candidate solutions, then we can use solvers that support `QF_NRA` (quantifier-free non-linear real arithmetic). Even though both logics are decidable in theory, the first one is computationally intractable in general.* ◁

To validate a candidate solution, we substitute the free parameters $\rho$ in the constraint system by the candidate solution $\rho_{\mathsf{NUM}}$. For Example 3.17, we obtain:

$$\forall x \in \mathbb{R} : \texttt{true} \Rightarrow 0 \preceq -1 \cdot 10^{-k} \cdot x^2. \tag{3.23}$$

Next, we simply iterate through all constraints and ask whether the negation is satisfiable. For Equation 3.23, we obtain:

$$\exists x \in \mathbb{R} : \texttt{true} \wedge -1 \cdot 10^{-k} \cdot x^2 < 0. \tag{3.24}$$

Then, an SMT solver reports one of the following:

**case** `unknown` We cannot assure validity of the candidate solution — this might happen due to insufficient memory or computation time.

**case** `unsat` We know that the candidate solution is valid.

**case** `sat` We have to reject the candidate solution because we found a witness of its invalidity.

For Equation 3.24, an SMT solver might return `sat` with the model $\mathbf{x}_{\mathsf{SMT}} = 1$ and, indeed, $\mathbf{x}_{\mathsf{SMT}}$ serves as a counterexample showing that $\rho_{\mathsf{NUM}}$ is an invalid solution.

**Remark 3.19**
*In our experiments, none of the solvers ever returned* `unknown` *while checking validity of candidate solutions.* ◁

We have automatized this simple validation scheme in Stabhyli using Z3 and allow the user to choose the threshold with which candidate solutions are rejected. This leads to the following rule on which Stabhyli decides to reject solutions.

**Definition 3.20 (Rejection of Candidate Solutions)**
*Given a parameterized Lyapunov constraint system* PLCS*, a candidate solution* $\rho_{\mathsf{NUM}} = \begin{bmatrix}\rho_{\mathsf{NUM},1} & \dots & \rho_{\mathsf{NUM},n}\end{bmatrix}$*, and user-chosen thresholds* $\theta_{SMT}$*,* $\theta_{EV}$*, and* $\theta_{PM}$*. If either*

- *for any* $(P, 0 \leq f(\rho, \mathbf{x})) \in \mathsf{PLCS}$*, it holds*
$$\exists \mathbf{x}_{\mathsf{SMT}} \in P \bullet f(\rho_{\mathsf{NUM}}, \mathbf{x}) < \theta_{SMT},$$
- *for any* $(P, 0 \leq f(\rho, \mathbf{x})) \in \mathsf{PLCS}$*, it holds that the corresponding LMI formulation for* $f(\rho_{\mathsf{NUM}}, \mathbf{x})$ *has eigenvalues or minors less than* $\theta_{EV}$*, or*
- *for any parameter, it holds that* $\rho_{\mathsf{NUM},i} < \theta_{PM}$ *is negative while it is supposed to be non-negative,*

*then the candidate solution* $\rho_{\mathsf{NUM}}$ *is* rejected. ◇

## 3.5 Guiding a Numerical Solver

In this section, we sketch a preliminary idea to guide the numerical solver away from bad candidate solutions and give a simple example which leads to numerical issues in the search for Lyapunov function. However, these issues can be detected and fixed due to the proposed method.

In the previous section, we have shown how to use SMT solvers to validate solutions. Due to the nature of the numerical solvers (i. e., that they approximate solutions), we

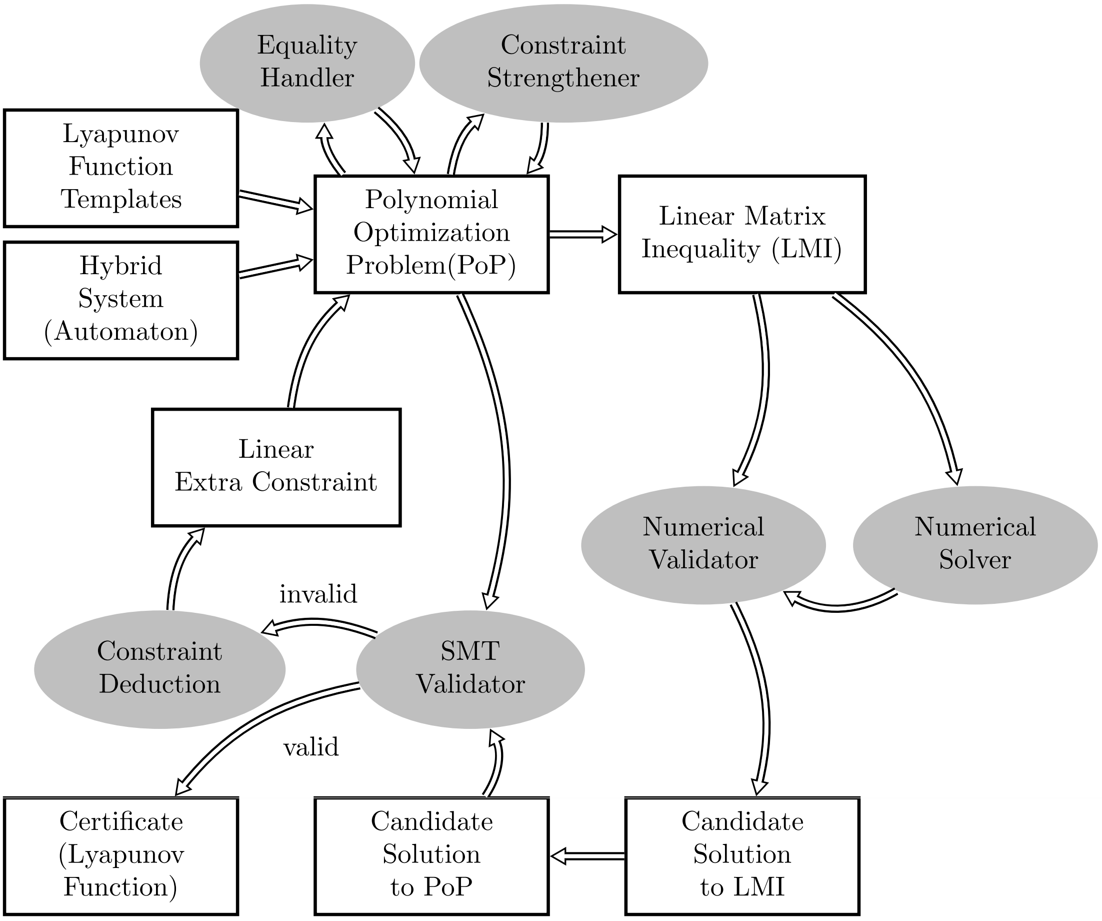
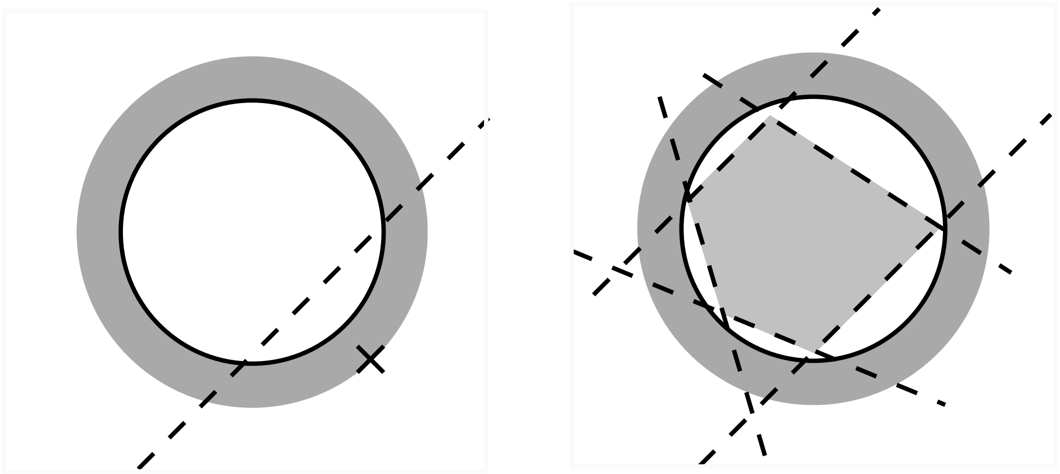

Equality Handler
Constraint Strengthener
Lyapunov Function Templates
Polynomial Optimization Problem(PoP)
Linear Matrix Inequality (LMI)
Hybrid System (Automaton)
Linear Extra Constraint
Numerical Validator
Numerical Solver
invalid
Constraint Deduction
SMT Validator
valid
Certificate (Lyapunov Function)
Candidate Solution to PoP
Candidate Solution to LMI

Figure 3.11: Overview of the guiding solver engine.

can expect many solutions returned by a numerical solver to be invalidated by the SMT solver. One of the reasons is the optimization goal as by default, the numerical solver tries to minimize $-\alpha$, $\beta$, and $-\gamma$ (see Theorem 2.31). In most cases, this leads to a situation where the optimal feasible solution lies on the border of the solution space. That means that the choice of optimization function increases the attractiveness of invalid candidate solutions.

To overcome this issue, we would like to gain knowledge from counterexamples, i. e., candidate solution which have been identified as invalid. As described in the previous section, in a situation where the SMT solver has detected an invalid solution, then, we obtain a witness $x_{\mathsf{SMT}}$ of the invalidity. Therefore, we propose to use this witness and compute the error or residual at the point $x_{\mathsf{SMT}}$ of the constraint which is not satisfied. Next, we extend the constraint system with an additional constraint which is supposed to prevent the invalid candidate solution.

Figure 3.11 gives an overview of the guiding (and validating) solver engine. It extends the proposed engine from Figure 3.10 by a constraint deduction which makes use of the



(a) Solution space with a first counterexample.

(b) Final solution space

Figure 3.12: Guiding the solver by shrinking the solution space of a constraint system.

generated counterexample from the SMT-based validator. The idea is as follows: If the SMT solver returns `sat` and a model $x_{\mathsf{SMT}}$, we construct an additional constraint that rules out the candidate solution $\rho_{\mathsf{NUM}}$. This is done iteratively: whenever we identify invalid solutions, then we construct an addition constraint until a valid solution is found. We call this iterative way of extending the constraint system, *guiding* because it does not guarantee that the numerical solver will return valid solutions. Instead, this approach does push the solver away from bad solutions. Moreover, we can also check for unsatisfiability of the added constraint — which are all linear constraints — to detect situation where the solution space becomes empty.

Figure 3.12 sketches the idea. The inner circle represents the solution space while the gray outer ring represents the invalid candidate solutions that might be returned by the solver due to numerical issues and approximation inaccuracies. In Figure 3.12a, the “X” marks a possible candidate solution which is then successfully identified as invalid. The dashed line represents a derived constraint that is added to rule out the invalid candidate solution — and hopefully other invalid candidate solutions as well. Figure 3.12b represents a final result in which the original solution space is narrowed down to the most inner gray polygon due to four more constraints that were added. Any solution that might be returned by the numerical solver that is within the gray polygon is valid. Even more, if again due to numerical issues, the solution lies slightly outside of the inner gray polygon but within the white circle, then this is not a problem since such a candidate would still satisfy the original constraint system.

**Remark 3.21**
*Note, that the goal is to find solutions to the original constraint system. This fact allows us to exclude the added constraint from validation as they do not need to be*

*satisfied since their sole purpose is to guide the numerical solver.* ◁

In the following, we explain how the additional constraints are constructed.

### 3.5.1 Guiding

By evaluating the all violated constraints $0 \leq f_i(\rho, \mathbf{x})$ using the counterexample $x_{\mathsf{SMT}}$ and the candidate solution $\rho_{\mathsf{NUM}}$, we obtain

$$\mathsf{res}_i = f_i(\rho_{\mathsf{NUM}}, \mathbf{x}_{\mathsf{SMT}}),$$

where $\mathsf{res}_i < 0$. Now, we can extend the constraint system from Example 3.17 by a constraint

$$|\mathsf{res}_i| \leq f_i(\rho, \mathbf{x}_{\mathsf{SMT}}),$$

which is linear in $\rho$ and does not contain quantifiers. In our running example, this leads to the constraint $|\mathsf{res}| \leq \rho \cdot x_{\mathsf{SMT}}^2$. The extended constraint system is as follows:

**Example 3.22**

$$\begin{aligned} \exists \rho \quad \textit{such that} \quad & \forall x \bullet \texttt{true} \Rightarrow 0 \preceq \rho \cdot x^2 \\ & \wedge |\mathit{res}| \leq \rho \cdot x_{\mathit{SMT}}^2 \end{aligned}$$

◁

This extended constraint system can be handed back to the numerical solver, asking for a new $\rho'_{\mathsf{NUM}}$. This will be again validated using the SMT solver. If we obtain a new counterexample $x'_{\mathsf{SMT}}$, then we extend the problem again until either

- a valid solution is found,
- a maximum number of iterations has been reached, or
- the set of constraints with which we extended the original problem, contains a contradiction.

The last alternative might happen if the real solution space is empty or has a very small interior. Note, that the additional constraints are all linear, unconditioned, and quantifier free. Thus, linear programming (or again an SMT solver) might be used to further check contradiction-freeness, i. e., satisfiability of the additional constraints.

Such contradictions can be introduced because the constraints that we add are more restrictive than the original constraints. The rationale behind this is the assumption that a numerical solver will achieve the same accuracy on the added constraint as on the original constraint. If that is the case, then the numerical solver might not return a solution to the extended constraint system. But the new candidate might nevertheless satisfy the original constraint system. Thus, we would have obtained a validatable solution.

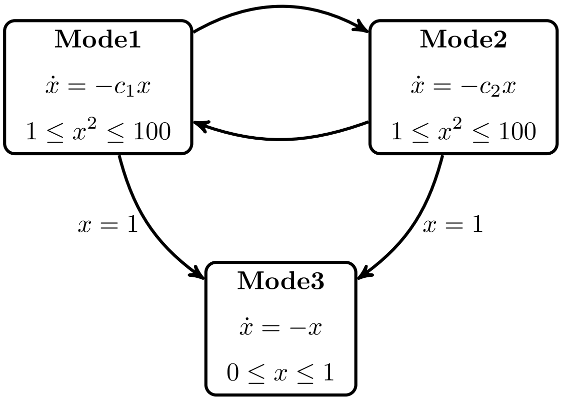


Figure 3.13: Badly scaled hybrid system.

### Accelerated Guiding

We can accelerate this method further by an alternative formulation of the added constraint. By *acceleration*, we mean that with each extra constraint, we narrow down the solution space more quickly by ruling out many points at once. This has the drawback that we might exclude good points, too. This can be achieve if we alternatively extend the constraint system as follows:

**Example 3.23**

$$\exists \rho \quad \textit{such that} \quad \forall x \bullet \mathtt{true} \Rightarrow 0 \preceq \rho \cdot x^2$$
$$\wedge\ \mathsf{acc} \cdot |\mathit{res}| \leq \rho \cdot x_{\mathit{SMT}}^2$$

◁

where $0 < \mathsf{acc}$ is an acceleration factor. Choosing $\mathsf{acc}$ such that $1 < \mathsf{acc}$ can lead to a validatable solution more quickly. The drawback is that with increasing $\mathsf{acc}$ the solution space is narrowed down faster. In the worst case, it might happen that the solution space becomes empty due to contradictions before a validatable solution has been found. On the other hand, if choosing $\mathsf{acc}$ such that $0 < \mathsf{acc} < 1$, then it might happen that there is nearly no progress concerning the quality of candidate solutions returned by the numerical solver.

**Remark 3.24**
*In our experiments, an acceleration factor of* $5 \leq \mathsf{acc} \leq 10$ *turned out to be a good choice.* ◁

### 3.5.2 Example

Consider the subsystem of the hybrid system given in Figure 3.13 consisting of the two locations **Mode1** and **Mode2**, only. The flow is described by the differential equation

$$\dot{x} = -c_i \cdot x$$

where $\mathsf{inv} = 1 \leq x^2 \wedge x^2 \leq 100$ is the invariant of both locations and arbitrary switching is allowed. Choosing the coefficients $c_i$ to be badly scaled, e.g. $c_1 = 10^{-10}$ and $c_2 = 10^{10}$ together with using $\rho_3 \cdot x^2 + \rho_2 \cdot x + \rho_1$ as the Lyapunov function templates for both locations leads to the constraint system:

$$\begin{aligned}
\exists \alpha, \beta, \gamma :& \\
&\mathsf{inv} \Rightarrow \alpha \cdot x^2 \preceq \rho_3 \cdot x^2 + \rho_2 \cdot x + \rho_1 \\
&\wedge\ \mathsf{inv} \Rightarrow \rho_3 \cdot x^2 + \rho_2 \cdot x + \rho_1 \preceq \beta \cdot x^2 \\
&\wedge\ \mathsf{inv} \Rightarrow \gamma \cdot x^2 \preceq 2c_1\rho_3 \cdot x^2 + c_1\rho_2 \cdot x \\
&\wedge\ \mathsf{inv} \Rightarrow \gamma \cdot x^2 \preceq 2c_2\rho_3 \cdot x^2 + c_2\rho_2 \cdot x \\
&\wedge\ 0 < \alpha \\
&\wedge\ 0 < \beta \\
&\wedge\ 0 < \gamma.
\end{aligned}$$

One might easily see that $\rho_1 = \rho_2 = 0$, $\rho_3 = \alpha = \beta = 1$, $\gamma = 2c_1$ is a valid solution. Nevertheless, both solver, SDPA and CSDP report “Lack of Progress” and thus, the quality of the solution is unknown. Indeed, both solution have the problem that $\gamma$ is slightly too large and CSDP additionally chooses $V_3$ and $\beta$ slightly too small.

However, the SMT-based validator finds counterexamples. And after deducing two more constraints in case of SDPA and four more constraints in case of CSDP, both solvers — even though they still report “Lack of Progress” — return a valid solution.

> **Remark 3.25**
> *The numerical validation is not helpful in this example, since the eigenvalues are very close to* 0*. For the initial constraint system, i. e., without any additional constraints, computing the eigenvalues correctly indicates that the candidate solutions are not valid. In case of the final constraint system — the one obtained through guidance — computing eigenvalues indicates that the solutions are invalid even though they are valid. Here, the SMT-based validator allows us to recover the solutions and let us conclude stability of the hybrid system.* ◁

## 3.6 Benchmarking

In this section, we will give some examples that were proven stable using Stabhyli. The first example is a simplified automatic cruise controller that might be used to adjust the velocity of a vehicle. The second example demonstrates that higher order Lyapunov functions allow stability proofs even if no quadratic Lyapunov function exists [PP03].

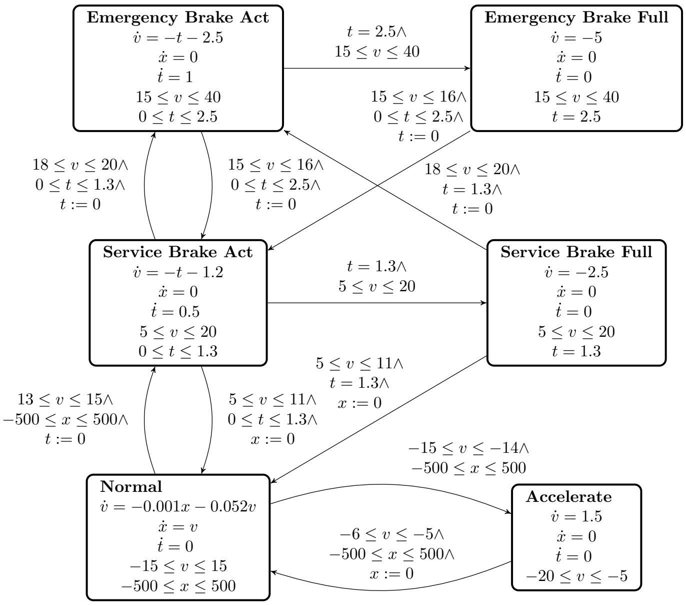


Figure 3.14: The automatic cruise controller [Oeh11].

**Example 1: The Automatic Cruise Controller** Figure 3.14 shows an automatic cruise controller (ACC) from [Oeh11]. This controller regulates the velocity (in $^{\mathrm{m}}/_{\mathrm{s}}$) of a vehicle. Its objective is to approach a desired velocity. In this example, the variable $v$ is the difference between this desired velocity and the current velocity. The automaton contains six locations; four of them model the brake system, one location models the acceleration, and one location implements a fine-tuning of the current velocity using a PI controller in non-critical situations. The brake system of the vehicle has two levels and, for convenience, slowly increases the deceleration towards a maximum.

In our experiments, STABHYLI was able to obtain a certificate of stability for this controller via decomposition with all three heuristics in twelve steps (sum of splitting and reduction steps).[4] While the sequences of steps performed by the "Prioritization of Zippers" and the "Selection by Product" heuristic are identical, the heuristic "Selection by Pairwise Degree" has chosen a different sequence of steps but obtained the same result.

Intentionally "sabotaging" the controller in a way that it cannot stabilize, causes STABHYLI to recognize and report the problem. E.g., changing the differential equation in location "Emergency Brake" to $\dot{v} = 5$, leads to an unstable cycle covering the locations "Emergency Brake", "Brake Activation", and "Emergency Brake Activation." In this case, STABHYLI terminates the search for a proof and blames the responsible cycle. This information allows guided redesign of this particular part of the automaton. The other proof schemes simply fail without any information about the cause.

**Example 2: An Artificial Example with a Sextic Lyapunov function** from [PP03] is

[4] With reduced accuracy of `1.00e-06`

$$\begin{aligned} &\mathit{Flow}(l_1)(x) && \mathit{Flow}(l_2)(x) \\ &= \begin{bmatrix} -5x_1 - 4x_2 \\ -x_1 - 2x_2 \end{bmatrix} && = \begin{bmatrix} -2x_1 - 4x_2 \\ 20x_1 - 2x_2 \end{bmatrix}, \end{aligned}$$

with $x = \begin{bmatrix} x_1 & x_2 \end{bmatrix}^T$ which has two locations $l_1, l_2$ and allows arbitrary switching.

For this system, Stabhyli cannot find a quadratic common Lyapunov function, which is clearly not a deficiency since no such function exists [PP03]. Note, that one cannot conclude that the system is not stable. In fact, it is indeed stable and can be proven stable by allowing Stabhyli to search for a sextic common Lyapunov function. The normalized result, Stabhyli returns, is

$$\begin{aligned} V(x) =& 19.576x_1^6 + 11.627x_1^5x_2 + 15.267x_1^4x_2^2 \\ &+ 3.0857x_1^3x_2^3 + 8.9471x_1^2x_2^4 - 1.3629x_1x_2^5 \\ &+ 1.0539x_2^6 \end{aligned}$$

which is nearly the same Lyapunov function as the one presented in [PP03]. By default, Stabhyli uses quadratic Lyapunov function templates. For this system, the default setting is not sufficient, but the tool user can allow Stabhyli to retry using higher degrees and thereby allowing more flexible templates. In our experiments, Stabhyli returned the above result within few seconds, while RSolver was not able to return any solution within nine hours. Relaxing the problem to region stability let RSolver report “Unknown.”

## 3.7 Summary

In this chapter, we have presented Stabhyli, a tool for automatically deriving proofs of stability of hybrid systems. Stabhyli uses state-of-the-art techniques like sum-of-squares to cast polynomial constraint systems to LMIs. In Section 3.2, we have presented the four different proof schemes that Stabhyli offers. These proof schemes are combined with powerful preprocessing and refinement steps. The decompositional proof scheme allows a designer to identify proof-critical parts of the hybrid system. The incremental proof scheme, in contrast, allows a bottom-up design of hybrid systems while Stabhyli ensures that a proof exists.

A simple but yet powerful heuristic that detects implicit equality constraints has been presented in Section 3.3. This heuristic searches for equality constraints using linear programming. In our case, the constraints that we have to consider, do have a special shape. Thanks to this shape, the heuristic is sufficient in many cases and especially in the settings of finding Lyapunov functions. After applying the heuristic, we can use the gained knowledge, i. e., the detected equality constraints, to eliminate free parameters in the parameterized Lyapunov constraint system. Eliminating free parameters does not only reduce the number of free parameters but also the number of constraints. We have presented three examples where the search for Lyapunov functions certifying stability of these hybrid systems would fail because of additional gaps that are required to make the constraint system robust against numerical issues. But by using our heuristic, it is easy to find Lyapunov functions for two of them.

In Section 3.4, we have presented an approach to validate candidate solutions as obtained by numerically solvers. We have applied this approach in the process of solving constraint systems that arise in during the search for Lyapunov functions. The validation works by using SMT solvers to check satisfiability of the negated constraints. Such a validation is needed if the Lyapunov functions are reused, e.g. as barrier certificates proving unreachability of certain bad states, as a basis for composition, or as an estimator on the rate of convergence.

While this validation is already very helpful, one would also like to be able to gain knowledge from invalid candidate solutions. Therefore, in Section 3.5, we have presented a preliminary idea usable to guide numerical solvers away from invalid candidate solutions. That way, it becomes more likely that valid candidate solutions will be found.

Finally, in Section 3.6, two examples that have been studied in the literature have been proven stable using Stabhyli. The first example is the automatic cruise controller from [Oeh11] for which a quadratic Lyapunov function can be found. The second example is from [PP03] for which a sextic common Lyapunov function can be found. Both can be proven stable fully automated using Stabhyli.

CHAPTER 4

# Transformations that Simplify Stability Verification

In this chapter, we propose two techniques to simplify the verification of stability properties using Lyapunov functions.

The outline of the chapter is as follows Section 4.1 gives a short overview of existing techniques for transformation, relaxation, or rewriting hybrid system for stability verification. In Section 4.2, we present a method for relaxing the graph structure of a hybrid automaton which — in the best case — reduces the effort for decomposition enormously, then step-by-step reconstructs the original graph structure, and — in the worst case — falls back to a decomposition of the original hybrid automaton. This technique has been presented in [MT15]. In Section 4.3 we present a second technique which is called *unrolling*. Unrolling automatizes parts of the techniques that we used to verify a steering controller in [MHR17]. It combines repeated but local reachability and stability analysis to simultaneously verify safety and stability properties for the overall hybrid system. The technique has been published in [MHT15; HM15; HMT15].

### Affirmation

Most of the content of this chapter has already been published in [MT15; MHT15] of which the main author and main contributor is also the author of this thesis. The main contribution in [MHT15] of Hagemann considers the conversion of safe sets — obtained from sublevel sets of Lyapunov function — to safe boxes. A preliminary idea addressing ellipsoidal sublevel sets only was proposed by the author of this thesis but never published. Hagemann has shown that with minor adjustments the idea can be generalized to arbitrary quadrics and proved the correctness of this technique. In the generalized case, we do not necessarily obtain boxes but $\mathcal{H}$-polyhedra. The results have been published in [HM15].

## 4.1 State-of-the-Art

Abstraction and refinement is widely used in software development. Such techniques are valuable since they allow handling complex systems. One key point is the ability to decompose a large system into subsystems, analyze those subsystems and deduce properties of the larger system. As cyber-physical systems tend to become more and more complex, such techniques become more important.

The topic of automatic transformation techniques for stability has not received much attention. This is different for reachability analyses where techniques like abstractions (e. g., combined with refinement in CEGAR) as in [Pra+13; Bog+14], bisimulations as in [HTP05], and relaxations as in [Jha+07; CTG06] are applied frequently and automated. Even worse, classical constructions based on bisimulations do not preserve stability [Cui07; PDV12; PLM13].

In a series of papers by Prabhakar et al. [PDV12; Pra12; PLM13] investigated and proposed a stronger bisimulation notion which is called *uniformly continuous bisimulation* and proved this notion to preserve stability. This notion additionally requires uniform continuity. A special case is the identity relation which trivially satisfies the requirement of uniform continuity [PS16]. It says that, given two hybrid automata $\mathcal{H}_1$ and $\mathcal{H}_2$, if $\mathcal{H}_1$ is asymptotically stable and exhibits at least all trajectories of $\mathcal{H}_2$, then $\mathcal{H}_2$ is also asymptotically stable.

Therefore, only two approaches in the area of transformation techniques for automatic stability verification are known:

1. The transformation used in the decompositional technique by Oehlerking and Theel [OT09] (see Section 2.5) and

2. The technique and the tool (Averist) proposed in [PS13; PS14; PS15; PS16] which is based on abstractions (see Section 3.1).

Unfortunately, Averist is currently restricted to switched linear hybrid systems. In theory, however, the technique implemented in Averist can be lifted to update-free non-linear hybrid systems, but for this class, no good strategy for state space partitioning is known. Nonetheless, the techniques presented in this chapter have the potential to be combined with the technique underlying Averist.

## 4.2 Relaxation: From Dense to Sparse Graph Structures

In this section, we present an improvement to the decomposition technique for proving stability of hybrid systems as proposed by Oehlerking and Theel in [OT09]. The improvement consists of relaxing the graph structure before applying the decomposition technique and was published in [MT15].

Our relaxation technique is developed for hybrid systems exhibiting dense graph structures. Applying the relaxation results in hybrid systems that are well suited for decomposition. This is because the connectivity of the hybrid system's graph is reduced significantly by rewiring the transitions of the graph. On one hand, the relaxation renders the decomposition technique more efficient and, on the other hand, the relaxation allows us to decompose a wider range of graph structures. This increases the likeliness of successfully identifying Lyapunov functions.

For hybrid systems with a complex discrete behavior, the decomposition technique decomposes the monolithic problem of proving stability into multiple subproblems. But if a hybrid system exhibiting a complex control structure — in the sense of a dense graph structure — is decomposed, then the blow-up can be enormous. The result is a high number of subproblems that must be solved – which is not bad per se. But since the decompositional technique involves underapproximating the feasible sets of each subproblem — when applied too often — results in the feasible set becoming empty. The relaxation technique presented here reduces the number of steps required by the decomposition and, therefore, the number of underapproximations. This has two benefits: (1) the runtime is reduced as well as (2) the effect of underapproximations is minimized.

The remainder of this section is organized as follows. First, we describe the relaxation technique in Section 4.2.1, prove termination, and show that stability is preserved when using Lyapunov functions for the proof. Second, in Section 4.2.2, we demonstrate our relaxation by proving stability of three examples:

1. the automatic cruise controller which is the motivating example for the decompositional technique,
2. an abstract example which shows what happens if decomposition is applied to complete graph structures, and
3. a spidercam which is an example that exhibits a dense graph structure for which proving stability using decomposition is not possible.

### 4.2.1 Relaxation of the Graph Structure

We show how the decomposition can be improved by our graph structure-based relaxation. Consider the underlying graph $\mathscr{G}(\mathcal{H}) = (\mathscr{V}, \mathscr{E})$ of a hybrid automaton $\mathcal{H} = (\mathit{Var}, \mathit{Loc}, \mathit{Trans}, \mathit{Flow}, \mathit{Inv})$, with the set of vertices $\mathscr{V} = \mathit{Loc}$ and the set of edges $\mathscr{E} = \{\, (l_1, l_2) \in \mathscr{V} \times \mathscr{V} \mid (l_1, G, U, l_2) \in \mathit{Trans} \,\}$. Note, that the underlying graph has at most a single edge between any two vertices while the hybrid automaton might have multiple transitions between two locations. The *density* of the graph $\mathscr{G}$ is the ratio of the number of edges in the graph and the maximal possible number of edges in a graph of the same size, i. e., $\frac{|\mathscr{E}|}{|\mathscr{V}|(|\mathscr{V}|-1)}$.[1]

The idea is to identify a set of locations of a hybrid automaton whose graph structure is dense. This can, for example, be done by a clique-finding or dense-subgraph-finding algorithm. A *clique* is a complete subgraph, i. e., having a density of 1 (see Definition 2.53 on 43).[2] Our relaxation then rewires the transitions such that the resulting automaton immediately exhibits a structure well suited for decomposition. In the context of decomposition, we call a graph structure *well-suited* if it contains mainly outer cycles.

The reason, that our relaxation technique combines so well with the decomposition technique, is as follows: if a hybrid system exhibits a dense graph structure, then the decomposition results in a huge blow-up. This blow-up is a result of the location-splitting

[1] We are referring to the definition of density for directed graphs.

[2] Finding the maximum clique is NP-hard. However, a *maximum clique* is not required, any *maximal clique* with more than two vertices is sufficient. Any clique for which no clique has more vertices is called a *maximum clique* while a clique which cannot be extended by including one more vertex is called a *maximal clique*. Even better, as we are interested in dense structures only, we can use quasi-cliques. A *quasi-clique* is a subgraph whose density is not less than a certain threshold. Thus, any greedy algorithm can be used.

step. The splitting step separates vertices shared between cycles, i. e., if there is more than one vertex shared between two or more cycles, then multiple copies of the vertex are created. Thus, the higher the density of the graph structure is, the higher the blow-up gets. Further, if many cycles share many vertices — as in dense graphs, — then whole cycles may get copied and each copy requires (1) solving an optimization problem and (2) underapproximating the feasible set of the problem. In contrast, our relaxation overapproximates the discrete behavior by putting each vertex in its own cycle connected only to the new dummy vertex. This reduces the number of optimization problems to be solved and the number of feasible sets to be underapproximated.

In the following, we define the relaxation operator. Then, we give an algorithm which applies the relaxation integrated with decomposition. Finally, we prove termination of the algorithm and that stability of the original hybrid automaton is implied by stability of the relaxed automaton.

**Definition 4.1 (Graph Structure Relaxation)**
*Given a hybrid automaton $\mathcal{H} = (\mathit{Var}, \mathit{Loc}, \mathit{Trans}, \mathit{Flow}, \mathit{Inv})$ and a subset of locations $\mathit{Loc}_d \subseteq \mathit{Loc}$. The* graph structure-relaxed hybrid automaton *$\mathit{Rlx}_{\mathsf{GS}}(\mathcal{H}, \mathit{Loc}_d) = (\mathit{Var}^\sharp, \mathit{Loc}^\sharp, \mathit{Trans}^\sharp, \mathit{Flow}^\sharp, \mathit{Inv}^\sharp)$ with respect to the sub-component $\mathit{Loc}_d$ is defined as follows*

$$\begin{aligned}
\mathit{Var}^\sharp &= \mathit{Var},\\
\mathit{Loc}^\sharp &= \mathit{Loc} \uplus \{l_c\},\\
\mathit{Trans}^\sharp &= \{\, (l_1, G, U, l_2) \in \mathit{Trans} \mid \{l_1, l_2\} \cap \mathit{Loc}_d = \emptyset \,\}\\
&\quad \bigcup \left\{ \begin{matrix} (l_1, G, \mathrm{id}, l_c), \\ (l_c, G, U, l_2) \end{matrix} \;\middle|\; \begin{matrix} (l_1, G, U, l_2) \in \mathit{Trans}, \\ \wedge\ \{l_1, l_2\} \cap \mathit{Loc}_d \neq \emptyset \end{matrix} \right\},\\
\mathit{Flow}^\sharp(l) &= \begin{cases} \mathtt{zero} & \text{if } l = l_c \\ \mathit{Flow}(l) & \text{otherwise,} \end{cases}\\
\mathit{Inv}^\sharp(l) &= \begin{cases} \emptyset & \text{if } l = l_c \\ \mathit{Inv}(l) & \text{otherwise,} \end{cases}
\end{aligned}$$

*where $l_c$ is a new location, called a* dummy or central location *whose flow is given by* $\mathtt{zero}$ *and the invariant is the empty set.* $\mathtt{zero} : \mathcal{X} \to \mathcal{P}(\mathcal{X})$ *is a function assigning* $\mathbf{0}$ *to each $\mathbf{x} \in \mathcal{X}$, i. e., $\dot{\mathbf{x}} \in \mathtt{zero}(\mathbf{x}) = \{\mathbf{0}\}$ for all $\mathbf{x} \in \mathcal{X}$.* ♢

In $\mathit{Trans}^\sharp$ in Definition 4.1, we replace each transition $(l_1, G, U, l_2) \in \mathit{Trans}$ connected to at least one location in $\mathit{Loc}_d$ with two transitions: one connecting the old source location $l_1$ with the new central location $l_c$ and the other one connecting $l_c$ with the old target location $l_2$. Note, that multiple applications of the relaxation operation are possible. We call each such application a *transition-splitting step*. By $ST$, we denote the set of all pairs of transitions, called *split transition*, that have been created during the transition splitting step.

Intuitively, the introduced location $l_c$ is a dummy location whose invariant always

**Algorithm 1:** The relaxation function `relax`.

```
input  : a hybrid automaton $\mathcal{H}$, a dense sub-component $Loc_d$ of $\mathcal{H}$
output : the relaxed version of $\mathcal{H}$, a set of split transitions $ST$, the central
         location $l_c$
$l_c \leftarrow$ newLoc();
$\mathcal{H}.Loc \leftarrow \mathcal{H}.Loc \cup \{l_c\}$;
$\mathcal{H}.Flow(l_c) \leftarrow$ zero;
$\mathcal{H}.Inv(l_c) \leftarrow \emptyset$;
$Trans \leftarrow \mathcal{H}.Trans$;
foreach $\tau = (l_1, G, U, l_2) \in Trans$ do
    if $\{l_1, l_2\} \cap Loc_d \neq \emptyset$ then
        // split the transitions into two parts
        $\tau_1 \leftarrow (l_1, G, \mathrm{id}, l_c)$;
        $\tau_2 \leftarrow (l_c, G, U, l_2)$;
        // replace the transition by the two parts
        $\mathcal{H}.Trans \leftarrow (\mathcal{H}.Trans \setminus \{\tau\}) \cup \{\tau_1, \tau_2\}$;
        // keep account of split transitions
        $ST \leftarrow ST \cup \{(\tau_1, \tau_2)\}$
```

evaluates to false and the flow function does not change the valuations of the continuous variables. Indeed, no run will dwell in the central location and a run taking an incoming transition of $l_c$ must, immediately, take an outgoing transition of $l_c$. The sole reason to add the central location is changing the structure of underlying graph of the hybrid system: the new structure contains mainly cycles that are connected via $l_c$.

Next, we show how to integrate decomposition and relaxation. Pseudo-code of the relaxation function and a reconstruction function — which step-by-step reverts the relaxation — can be found in Algorithm 1 and Algorithm 2, respectively. Algorithm 3 gives pseudo-code of the main algorithm. The main algorithm works as follows:

(1) The function `relax` relaxes the graph structure of the hybrid automaton $\mathcal{H}$ and generates the set of split transitions $ST$.

(2) If the set $ST$ is empty, then call `applyDecomposition` with the original automaton and return the result — this function applies the original decomposition technique as described in Section 2.5.

(3) Otherwise, apply `applyDecomposition` on the current relaxed form of the automaton. If the result is `stable`, then return the result. Otherwise, if the original decompositional technique has failed, then it returns a *failed subgraph* that is a subgraph for which it was unable to find Lyapunov functions.

(4) Choose a split transition from the set $ST$ which also belongs to the failed subgraph. It is then used to reconstruct a transition from the original hybrid automaton. Then, execution is continued with step 2.

**Algorithm 2:** The reconstruction function `reconstruct`.

```
input  : a relaxed hybrid automaton $\mathcal{H}$, a set of split transitions $ST$, a pair of
         split transitions $(\tau_1, \tau_2)$, where $\tau_1 = (l_1, G, \mathrm{id}, l_c)$, $\tau_2 = (l_c, G, U, l_2)$ and
         $l_c$ is the central location
output : a relaxed hybrid automaton $\mathcal{H}$ with one split transition being
         reconstructed, the set of split transitions $ST$
// reconstruct the original transition
$\tau \leftarrow (l_1, G, U, l_2)$;
// replace the split transition $(\tau_1, \tau_2)$ by $\tau$
$\mathcal{H}.Trans \leftarrow (\mathcal{H}.Trans \setminus \{\tau_1, \tau_2\}) \cup \{\tau\}$;
// update the set of split transitions
$ST \leftarrow ST \setminus \{(\tau_1, \tau_2)\}$;
// remove $l_c$ if and only if unconnected
if $ST = \emptyset$ then
    $\mathcal{H}.Loc \leftarrow \mathcal{H}.Loc \setminus \{l_c\}$;
```

(5) If no such split transition exists, then the algorithm fails and returns the failed subgraph since this failing subgraph will persist in the automaton. Further, reverting the relaxation cannot help because no split transition is contained in the failed subgraph.

Next, we prove termination and soundness of the algorithm. Here, soundness indicates that a Lyapunov function-based stability certificate for a relaxed automaton implies stability of the original, unmodified automaton. In particular, the local Lyapunov functions of the relaxed hybrid automaton are valid local Lyapunov functions for the original automaton.

### Termination of the Integrated Algorithm

**Theorem 4.2**
*The proposed algorithm presented in Algorithm 3 terminates.*

**Proof.**
*The function* `relax` *terminates since the copy of the set of transitions of $\mathcal{H}$ is finite and is not modified in the course of the algorithm. The while-loop terminates if either an* `applyDecomposition` *is successful, no pair for reconstruction can be identified, or the set $ST$ is empty. In the first two cases, the algorithm terminates directly. For the last case, we assume that no call to* `applyDecomposition` *is ever successful and a split transition is always found. Then, in each iteration of the loop, one edge gets removed from $ST$. The set $ST$ is finite because the relaxation function* `relax` *splits only finitely many edges. Thus, the set $ST$ eventually becomes empty.*

**Algorithm 3:** The integrated relaxation and decomposition algorithm.

```
input  : a hybrid automaton $\mathcal{H}$, a set of locations $Loc_d$ corresponding to a dense subgraph
output : returns stable if the $\mathcal{H}$ is stable and failed otherwise.
// relax the graph structure
$\mathcal{H}, ST, l_c \leftarrow$ relax($\mathcal{H}, Loc_d$);
while $ST \neq \emptyset$ do
    // apply decomposition
    result $\leftarrow$ applyDecomposition($\mathcal{H}$);
    if result is stable then
        return stable;
    // apply reconstruction
    if $\exists(\tau_1, \tau_2) \in ST : \{\tau_1, \tau_2\} \cap$ failedSubgraph(result) $\neq \emptyset$ then
        $\mathcal{H}, ST \leftarrow$ reconstruct($\mathcal{H}, ST, (\tau_1, \tau_2), l_c$);
    else
        return result;
// apply decomposition on the original automaton
result $\leftarrow$ applyDecomposition($\mathcal{H}$);
return result;
```

*Therefore, the loop terminates.* □

### Preservation of Stability

**Theorem 4.3**
*Let $\mathcal{H} = (Var, Loc, Trans, Flow, Inv)$ be a hybrid automaton and let $Loc_d \subseteq Loc$ be a subset of the locations. If a family of local Lyapunov functions $(V_l)$ proving $Rlx_{\mathsf{GS}}(\mathcal{H}, Loc_d)$ to be GAS exists, then there exists a family of local Lyapunov functions for $\mathcal{H}$ proving $\mathcal{H}$ to be GAS.*

**Proof.**
*Given a hybrid automaton $\mathcal{H} = (Var, Loc, Trans, Flow, Inv)$. Let the hybrid automaton $\mathcal{H}^\sharp = Rlx_{\mathsf{GS}}(\mathcal{H}, Loc_d) = (Var^\sharp, Loc^\sharp, Trans^\sharp, Flow^\sharp, Inv^\sharp)$, be a graph structure-relaxed version of $\mathcal{H}$ where $Loc_d \subseteq Loc$ is the sub-component of $\mathcal{H}$ that has been relaxed. Further, let $(V_l)_{l \in Loc^\sharp}$ be the family of local Lyapunov functions that prove GAS of $\mathcal{H}^\sharp$ and let $ST$ be the set of split transitions — some transition may have been reconstructed. Now, it must be shown that $(V_l)_{l \in Loc}$ are valid Lyapunov functions for $\mathcal{H}$.*
*The location constraints of Theorem 2.31 trivially hold, since $Rlx_{\mathsf{GS}}$ alters neither the flow functions nor the invariants, i. e., $\forall l \in Loc : Flow^\sharp(l) = Flow(l) \wedge Inv^\sharp(l) =$*

*$Inv(l)$. The transition constraint also holds for all transitions that are not altered by $Rlx_{\mathsf{GS}}$ or have been reconstructed, i. e., $Trans \cap Trans^{\sharp}$. Let $\tau \in Trans \setminus Trans^{\sharp}$ be an arbitrary transition. We show that the transition constraint holds. Due to the definition of $Rlx_{\mathsf{GS}}$, all transition in $Trans \setminus Trans^{\sharp}$ are split transitions and there is a corresponding pair in $ST$. Let $(\tau_1, \tau_2) \in ST$ be the pair corresponding to $\tau = (l_1, G, U, l_2)$. Since $(V_l)_{l \in Loc^{\sharp}}$ is a valid family of local Lyapunov function for $\mathcal{H}^{\sharp}$, the transition constraint holds for all transitions in $Trans^{\sharp}$. In particular, the transition constraint holds for $\tau_1 = (l_1, G, \mathrm{id}, l_c)$ and $\tau_2 = (l_c, G, U, l_2)$. Thus,*

$$\forall x \in G \bullet V_{l_c}(\mathrm{id}(x)) \leq V_{l_1}(x) \wedge \forall x \in G \bullet V_{l_2}(U(x)) \leq V_{l_c}(x).$$

*It follows, that*

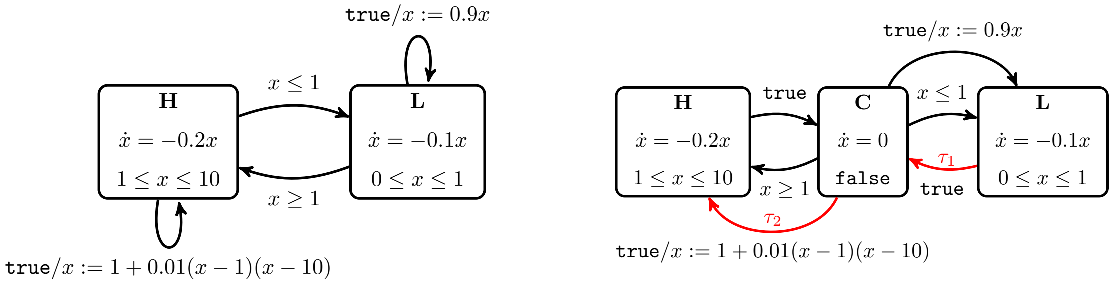
$$\forall x \in G \bullet V_{l_2}(U(x)) \leq V_{l_c}(x) \leq V_{l_1}(x).$$

*Therefore, the transition constraint holds for $\tau$.* □

While Theorem 4.3 shows that stability of the relaxed automaton yields stability of the original automaton, the contrary is not true. Figure 4.1 shows a hybrid system where the relaxation renders the system unstable. This example exploits that the relaxation may introduce spurious runs. This happens in case there are transitions with overlapping guard sets connected to the central location $l_c$. A trajectory of the relaxed automaton might then take the first part of a split transition to the central location $l_c$ and continues with the second part of a different split transition. A transition corresponding to this behavior might not exist in the unmodified hybrid automaton. While this does not render our approach being incorrect, it may lead to difficulties since these extra trajectories have to be GAS, too. In case of the system in Figure 4.1, new trajectories are introduced which allow a trajectory to jump back from the location `L` to `H` by taking the transitions $\tau_1, \tau_2$. This behavior corresponds to leaving `L` by the right self-loop and entering `H` by the left self-loop, which is obviously impossible. However, due to the update function, the value of $x$ might increase as $1 + 0.01(x - 1)(x - 10) > 1$ for $x < 1$.

In general, our relaxation introduces conservatism which is then reduced step-by-step during each reconstruction step. The degree of conservatism highly depends on the guards of the transitions since the central location relates all LLFs of locations in $Loc_d$. Therefore, when more guards are overlapping, more LLFs have to be compatible even when this was not needed in the original automaton.

One possibility to counter-act this issue is to introduce a new continuous variable in the relaxed automaton which is set to a unique value per split transition: the update function of the first part of a split transition sets the value used to guard the second part of the transition. Indeed, this trick discards any spurious trajectories for the price of an additional continuous variable. However, since the values of that variable are somewhat artificial, a Lyapunov function may not be able to make use of that variable. Thus, this trick will not ease satisfying the conditions of the Lyapunov theorem in general.



(a) The unmodified (stable) automaton.

(b) The relaxed (unstable) automaton.

Figure 4.1: A hybrid system which becomes unstable after relaxation.

### 4.2.2 Experiments

In this section, we present three examples where the graph structure-based relaxation improves the application of the decomposition technique. The first example is again the motivating example for the decomposition technique: an automatic cruise controller (ACC) [Oeh11]. The second example is the fully connected digraph $K_3$. $K_3$ (such as the $K_1, K_2, K_4, K_5$ do not represent a concrete hybrid automata but a potential graph structure of a hybrid automaton. The last example is a spidercam. Here, the graph is not as fully connected as the $K_3$ example, but its density is already too high to apply decomposition directly.

We have implemented the decomposition and relaxation in Python. Our implementation does not compute the Lyapunov function but merely performs the graph transformation. Lyapunov functions have been computed separately using STABHYLI. Table 4.2 gives the graph properties and a comparison of the number of reduction steps required by the decomposition with and without relaxation (in the best case).

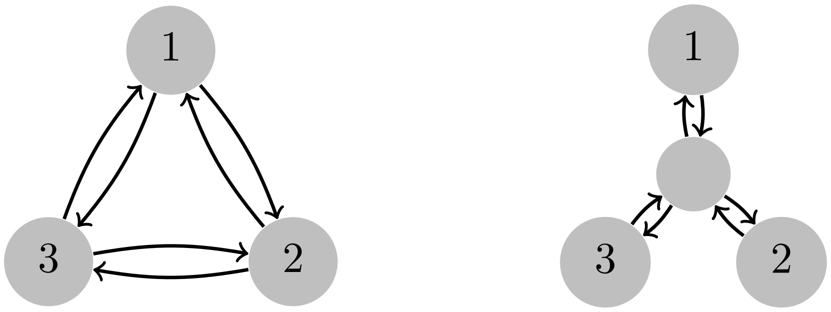
The given data was obtained without actually computing Lyapunov functions focusing on the graph related part of the decomposition.

| | | | Decomposition | | | With Relaxation | |
|---|---|---|---|---|---|---|---|
| Graph Structure | Nodes ($n$) | Edges | Reductions steps | Location-Splittings steps | Time | Reductions steps | Time |
| directed $K_1$ | 1 | 0 | 0 | 0 | 0.04s | 0 | 0.04s |
| directed $K_2$ | 2 | 2 | 1 | 0 | 0.04s | 2 | 0.04s |
| directed $K_3$ | 3 | 6 | 6 | 4 | 0.21s | 3 | 0.05s |
| directed $K_4$ | 4 | 12 | 47 | 25 | 1.15s | 4 | 0.05s |
| directed $K_5$ | 5 | 20 | 1852 | 352 | 13h22m | 5 | 0.05s |
| Spidercam | 9 | 32 | 753 | 287 | 1h46m | 9 | 0.06s |
| Cruise Controller | 6 | 11 | 7 | 6 | 0.060s | 6 | 0.06s |

Table 4.2: Comparison of the decomposition with and without relaxation.

**Example 1: The Automatic Cruise Controller (ACC)** has been presented in Example 1 and is globally asymptotically stable as it can be proven using the original decomposition technique (cf. [Oeh11; MT13a]). Indeed, the graph structure is sparse and thus, already well-suited for applying the decomposition technique directly. In fact, only one more cycle needs to be reduced compared to decomposition after relaxation (cf. Table 4.2). Even though the relaxation is not needed here, it also does not harm. Though, it may be used for sparse graphs structures, too.

**Example 3: The directed $\mathbf{K_3}$** is a fully connected digraph with $n$ nodes. The $K_3$ as well as a relaxed version of it is shown in Figure 4.3. In a fully connected digraph, there is a single edge from each node to each other node, resulting in a total number of $n(n-1)$ edges. The number of cycles, the decomposition technique has to reduce, grows very fast by $n$ which can be seen in Table 4.2. In comparison, the number of cycles in the relaxed version of the graph grows linearly with $n$, assuming that the edges can be concentrated.[3] Otherwise, after the relaxation, each original node has $n-1$ incoming and $n-1$ outgoing edges where each edge connects the node with the central node $l_c$. Each such combination forms a cycle between $l_c$ and an original location, giving a total of $n(n-1)(n-1)$ cycles in the worst case. This cubic growth is still much less than the number of reductions without relaxation.



(a) unmodified

(b) relaxed

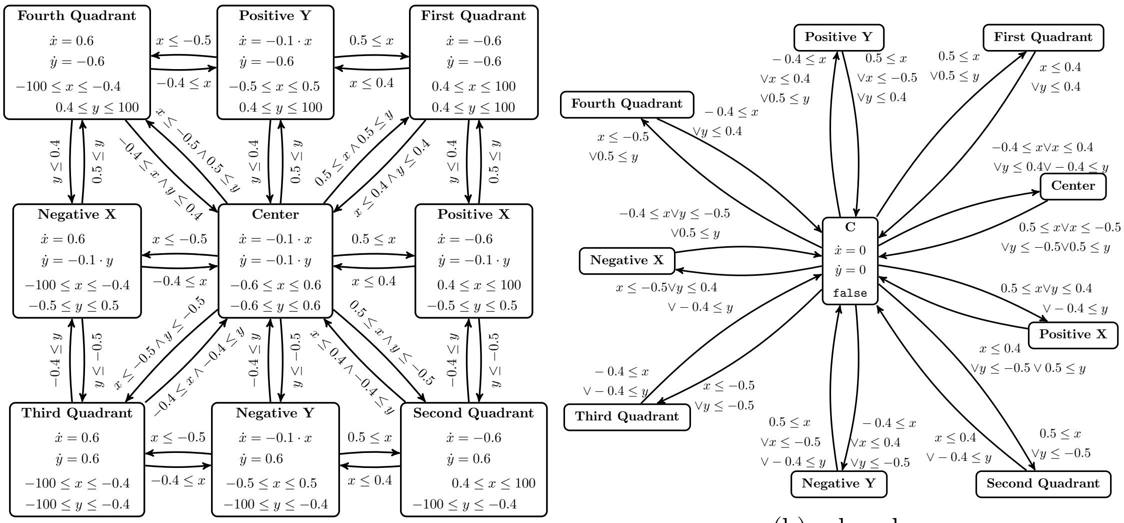
Figure 4.3: Relaxation of The $K_3$ example.

Such a graph might not be the result of a by-hand designed system but might be the outcome of a synthesis or an automatic translation. However, the fast growth of cycles also indicates the high number of reductions steps and therefore the high number of underapproximations of the solution space.

**Example 4: The Spidercam** is a movable robot equipped with a camera. It is used at sport events such as a football matches. The robot is connected to four cables. Each cable is attached to a motor that is placed high above the playing field in a corner of a stadium. By winding and unwinding the cables — and thereby controlling the length of the cables, — the spidercam is able to reach nearly any position in the three-dimensional space above the playing field. Figure 4.4 shows a simplified model of such a spidercam in the plane. The goal is to stabilize the camera at a certain position. The continuous

[3] By "concentrating edges," we mean that edges with the same source and target node are handled as a single edge for the cycle finding algorithm.

variables $x$ and $y$ denote the distance relative to the desired position on the axis induced by the cables.



(a) unmodified

(b) relaxed

Figure 4.4: Relaxation of the simple planar spidercam.

In the model, we assume a high-level control of the motor engines, i. e., the movement is on axis $\dot{x}$ and $\dot{y}$ instead of a low-level control of each individual motor. The model has nine locations: one location that controls the behavior while being close to the desired position, four locations corresponding to nearly straight movements along one of the axes and four locations cover the quadrants between the axes. The maximal velocity in the direction of each axis is limited from above by 0.6m. Thus, in the four locations corresponding to the quadrants, the movement in each direction is at full speed. In the four locations corresponding to the axes, the movement on the particular axis is at full speed while the movement orthogonal to the axis is proportional to the distance. In the last location, the speed in both directions is proportional to the distance.

The spidercam is globally asymptotically stable which can be proven fully automatically. However, it is not possible to obtain a piecewise Lyapunov function via decomposition without relaxation due to accumulating underapproximations of the partial solutions and the high number of cycles that have to be reduced.[4] This is because, each time a cycle is reduced, the feasible set of a subproblem is underapproximated by a finite set of solutions which finally results in the feasible set becoming empty and no LLFs can be found.

In contrast, relaxing the graph structure followed by applying the decomposition is immediately successful. In particular, no reconstruction step is required.

[4] Stabhyli currently does not contain strategies to handle the situation where no reduction is possible. The current implementation would then simply fail. Although, it is theoretically possible to perform some form of backtracking, it is hard to decide which underapproximation must be refined.

## 4.3 Unrolling: Hybrid Methods for Hybrid Systems

In this section, we describe an approach for simultaneously verifying safety and stability and focus on the author's contributions. The approach has been published in [MHT15; HM15; HMT15].

In industrial applications it is usually not sufficient to prove either safety or stability properties, instead, usually we need to prove both. The safety property guarantees that "something bad" never happens. The stability property guarantees that "something good" eventually happens. The analyses of both properties are usually performed in isolation. In this work, we consider analyzing both properties by a single automatic approach for hybrid systems. We basically merge analyses of both properties to exploit the knowledge gained from the analysis of each of them in the analysis of the other. We show how both analyses can be divided into multiple steps and interlocked such that both benefit from each other. In fact, we compute single-location Lyapunov functions,[5] unroll the automaton of the hybrid system via repeated reachability queries, and, finally, compute a global Lyapunov function. Each reachability query is simplified by exploiting the knowledge gained from the single-location Lyapunov functions. The final computation of the global Lyapunov function is simplified by a precise characterization of the reachable states and reuses the single-location Lyapunov functions.

We present an approach to verify safety and stability properties of hybrid systems at once. While the verification of safety and stability is usually done separately, we propose to integrate both analyses in a symbiotic fashion:

**From the safety perspective,** we use Lyapunov functions to detect regions that are guaranteed to be safe. Exploiting these regions helps us to shorten the reachability analysis. Indeed, if all trajectories eventually enter a safe region, then there is no need to compute infinite traces of the trajectories.

**From the stability perspective,** we obtain a more precise description of the reachable sets, and thereby, the trajectories of a system which are actually feasible. By discovering implicit knowledge and making it explicit, we lower the computational burden to obtain Lyapunov functions.

A rudimentary unrolling has been sketched in [Oeh11, page 126 f.]. Oehlerking suggested to use reachability analysis to show that certain guards are not reachable. Such a reachability analysis can be complicated as most tools will only be able to tell what is reachable within finite time and not the opposite. To achieve the opposite, invariants or fix point arguments are required. We solve this issue by the careful integration of safety and stability analysis and show the potential using a prototypical implementation combining the two tools STABHYLI and SOAPBOX.

[5] In [MHT15; HM15; HMT15], we called single-location Lyapunov functions "single-mode Lyapunov function" which is closer to the terminology used in control theory.

**Note 4.4**
*Since* STABHYLI *employs the sum-of-squares method, every computed Lyapunov function is representable as a sum-of-squares and therefore as quadratic matrices. However, in the following, we restrict ourselves to* quadratic *Lyapunov functions as they are sufficient for many applications.* ◁

### 4.3.1 Safety, Reachability, and Reach-Avoid Problems

Since we deal with both, safety and stability properties in this section, we recall some basic background on safety and reachability analysis and briefly present some properties of SOAPBOX [Hag14; Hag15] which is the tool we use to automatically compute reachable state sets and, thus, allows us to check for (non-)reachable states.

**Safety and reachability analysis** of hybrid systems has to deal with two problems: (1) how to tackle the dynamics of the system, and (2) how to represent the reachable states systematically. Both problems are related since the choice of the admissible dynamics has an impact on the required operations for post-image computation. For reachability analysis, we will consider systems whose dynamics are given by linear differential inclusions [Fre+11; Gir05; KV00]. Differential inclusions allow us to approximate systems with richer dynamics [ADG03; ADG07; DMT10].

For state representation we focus on convex approaches where reachable states are usually represented by unions of convex sets. Different representations, like polyhedra [CK03], template polyhedra [SDI08], zonotopes [Gir05], ellipsoids [KV00], and support functions [LG09], are commonly used.

In the following, we present two concepts which are related to our combined investigation of safety and stability.

**Reach-avoid problems** describe that one is interested in finding a control strategy or initial condition to reach a certain set of desired states while avoiding another set of undesired states. In [MT00], Mitchell and Tomlin propose an exact algorithm for this problem which — like our algorithm — makes use of sublevel sets. Abate et al. give a specification of the corresponding probabilistic problem in [Aba+08]. However, a reach-avoid problem is an *existential problem*, i. e., it asks for the existence of at least one trajectory, while we are interested in the case where *all* trajectories converge and avoid undesired states.

**Inevitability** of a hybrid system $\mathcal{H}$ with respect to a set $T$ denotes that every trajectory of the system reaches $T$ within finite time. This property has been studied for hybrid systems by Podelski et al. in [PW06; BMP10] and Duggirala and Mitra in [DM12]. The focus of this line of research is the relation to program termination and well-foundedness. However, we do not require a single $T$ to exist. Instead, every trajectory or bundle of trajectories might have its own set and additionally, they do not need to be given a priori but are identified implicitly.

### Checking Reachability via SoapBox

SoapBox is a tool for reachability analysis of hybrid systems. It is implemented in Matlab and recently parts of it have been reimplemented in C++.[6] SoapBox handles hybrid systems with continuous dynamics described by linear differential inclusions and arbitrary affine maps for discrete updates. The invariants, guards, and sets of reachable states are given as convex polyhedra. Internally, the reachability algorithm of SoapBox is based on symbolic orthogonal projections (sops) [Hag14; Hag15].

Although SoapBox is a fully functional model checker handling continuous and discrete updates of a hybrid system, for our approach, it suffices to present the location-specific continuous post-image computation provided by the method

$$\texttt{Reach}(\mathit{Init}, \mathit{Flow}(l), \mathit{Inv}(l), \mathit{Safe}, T).$$

As input it expects an $\mathcal{H}$-polyhedron representing the initial states *Init*, a differential inclusion $\mathit{Flow}(l)$ describing the differential inclusion of the form $\dot{\mathbf{x}} = A\mathbf{x} + \mathbf{E}$, where $\mathbf{E}$ is an $\mathcal{H}$-polytope modeling the bounded input, an $\mathcal{H}$-polyhedron representing the invariant $\mathit{Inv}(l)$, an $\mathcal{H}$-polyhedron representing the target set[7] $T$, and an $\mathcal{H}$-polyhedron *Safe*. The method computes a tight overapproximation of all reachable states on trajectories solving

$$\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + \mathbf{E},\ \mathbf{x}(0) \in \mathit{Init},\ \forall t \geq 0 \bullet \mathbf{x}(t) \in \mathit{Inv}(m),$$

until either all states leave the invariant or all states have entered the set *Safe*. Then, it returns a polyhedral set representing a tight polyhedral overapproximation of the intersection of the reachable states with the target set $T$.[8]

## 4.3.2 Safety and Stability at Once

We present our algorithm that combines safety and stability analysis. We show how the combination of both analyses mutually simplifies the verification task.

While GAS is defined over trajectories, Theorem 2.31 argues over hybrid states and searches for LLFs which have to be compatible with respect to the transition constraints. We observed that, often, we are able to find the LLFs but fail to establish the necessary compatibility with respect to the transition constraint. Among other reasons, this may be due to cyclic dependencies imposed by the transitions of the hybrid automaton as investigated in Section 3.3.

> **Note 4.5**
> *For each Lyapunov function $V$ (local or global), it holds that for any trajectory $\mathbf{x}(\cdot)$ the value of $V$ does not increase, i. e., $\forall t_0, t \geq 0 \bullet V(\mathbf{x}(t_0 + t)) \leq V(\mathbf{x}(t_0))$ (or*

[6] The experiments in the section have been performed using the Matlab version of SoapBox.

[7] Usually, `Reach()` handles finite unions of polyhedral target sets at once. For the sake of simplicity, our presentation is restricted to a single target set.

[8] As before, `Reach()` returns unions of polyhedra, each representing a single traversal of a target set.

> $\forall t_0, t \geq 0 \bullet V(l(t_0+t), \mathbf{x}(t_0+t)) \leq V(l(t_0), \mathbf{x}(t_0))$ *for the global Lyapunov function (GLF)).* $\triangleleft$

Our approach is to unroll the hybrid automaton to an equivalent hybrid automaton for which we can verify GAS via Theorem 2.31. Here, "equivalence" denotes that both automata exhibit an identical continuous behavior, i. e., for each trajectory of either automaton, there is a trajectory of the other automaton with the same continuous evolution. Hence, the original hybrid automaton is as well GAS.

We define the following notion of the *depth of a reachable state.* The set $\mathcal{R}_0 = \mathit{Inits}$ is the set of all reachable states of depth 0. Given the set $\mathcal{R}_n$ of all reachable states at depth $n$, we recursively define $\mathcal{R}_{n+1}$ as the set of all tuples $\left(l_{n+1}, \mathbf{x}_{n+1}\right)$ which are reachable from any tuple $(l_n, \mathbf{x}_n) \in \mathcal{R}_n$ by a combination of a continuous evolution and a subsequent discrete transition, i. e., there exists a flow $f \in \mathit{Flow}(l_n)$, an instant of time $t_s \geq 0$, and a transition $(l_n, G, U, l_{n+1}) \in \mathit{Trans}$ such that for $\mathbf{y}(t_s) = \int_0^{t_s} f(\mathbf{y})\ dt$ it holds that $\mathbf{y}(t) \in \mathit{Inv}(l_n)$ for all $t \in [0, t_s]$, $\mathbf{y}(0) = \mathbf{x}_n$, $\mathbf{y}(t_s) \in G$, and $U(\mathbf{y}(t_s)) = \mathbf{x}_{n+1}$. Clearly, a hybrid state $(l, \mathbf{x})$ is reachable at depth $n$ if and only if $(l, \mathbf{x}) \in \mathcal{R}_n$.

Our idea is as follows: First, we compute for each location a Lyapunov function, called a single-location Lyapunov functions (SLLFs). Then, starting with $\mathcal{R}_0$, we successively compute the sets $\mathcal{R}_n$. For each location this reduces to the reachability problem of computing the *reach set*, that is the set obtained by computing all reachable states in the transition guards and subsequent application of the respective discrete updates. During each reach set computation, we additionally assess safety, i. e., we decide whether *Unsafe* can be reached or not. Moreover, the single-location Lyapunov functions (SLLFs) enable us to compute safe sets which substantially simplify reachability analysis: An appropriate safe set has no intersection with neither the unsafe set nor any guard of an outgoing transition and it is invariant under the flow. Thus, we may stop the reachability analysis as soon as a safe set is entered.

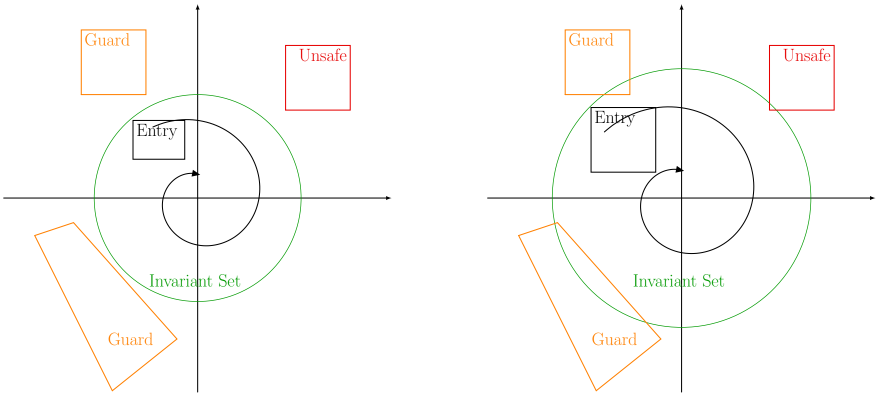
Successful termination of this approach does not only help to ensure safety, but also yields an unrolled (or unfolded) version of the original hybrid automaton. The unrolled automaton is free of cyclic dependencies. Additionally, guards are restricted to their intersection with the reachable states. Both facts enormously simplify the task of establishing compatibility of the SLLFs with the transition constraints, which finally yields a global Lyapunov function.

### Single-location Lyapunov Function Computation

As mentioned above, in the first step, we compute Lyapunov functions for each location in isolation. We call a Lyapunov function *single-location Lyapunov function (SLLF)*, if it satisfies the location constraints, but not necessarily the transition constraints of Theorem 2.31.

Stabhyli can compute Lyapunov functions automatically. To obtain *single-location Lyapunov functions*, we run Stabhyli with a single-location automaton $\mathcal{H}_l$ for every $l \in \mathit{Loc}$, where the *single-location hybrid automaton* is defined as

$$\mathcal{H}_l = (\mathit{Var}, \{l\}, \emptyset, \{l \mapsto \mathit{Flow}(l)\}, \{l \mapsto \mathit{Inv}(l)\}, \{(\mathit{Inv}(l), l)\}).$$



(a) Trajectories will not reach Unsafe.

(b) Trajectories might reach Unsafe.

Figure 4.5: A sketch of invariant sets.

Computing the SLLFs certifies that each single-location hybrid automaton is GAS. Note, that this is not sufficient to conclude that the full hybrid automaton is GAS. However, we exploit SLLFs during the following reachability analysis. In a final step, we combine the SLLFs with the results of the reachability analyses to verify GAS of the full hybrid automaton (see Global Lyapunov Function Computation on page 108).

### Safe Set Computation

As in the previous section, we restrict our attention to a single location $l$ of a hybrid automaton $\mathcal{H}$. First, we introduce the notion of a sublevel set of a Lyapunov function.

**Definition 4.6 (Sublevel Set of a Lyapunov Function)**
*Let* $V_l(\cdot)$ *be a Lyapunov function of location* $l$*. For any* $s \geq 0$*, we call* $L^{-}_{V_l,s} = \{\, \mathbf{x} \mid V_l(\mathbf{x}) \leq s \,\}$ *a* sublevel set of a Lyapunov function *of location* $l$*.* ♢

A Lyapunov function assigns a value to any state of the state space, and its values along any trajectory does not increase. Hence, all trajectories starting in a sublevel set will not leave the sublevel set. In general, we call any subset of states which cannot be left by the continuous trajectories an *invariant set* (for the formal definition see Definition 4.7). This property allows a strong prediction on the future of the trajectory. However, if the intersection of the invariant set (cf. the green circles in Figure 4.5) with *Unsafe* (cf. the red boxes in Figure 4.5) or some guards (cf. the orange boxes in Figure 4.5) is not empty, then this prediction does not suffice to establish safety. In this case, we can neither rule out that a trajectory enters *Unsafe* in this location nor that a trajectory leaves the location via a discrete transition and enters *Unsafe* via another location.

**Definition 4.7 (Safe Set, Avoid Set, and Invariant Set)**
*Let $A$ and $\mathcal{X}'$ be two subsets of $\mathcal{X}$ such that $\mathcal{X}' \subseteq Inv(l)$. If for all trajectories $\mathbf{x}(\cdot)$ of a hybrid automaton $\mathcal{H}$ with $\mathbf{x}(0) \in \mathcal{X}'$ and for all $t \geq 0$ it holds*

$$\forall t' \ (0 \leq t' \leq t \rightarrow \mathbf{x}(t') \in Inv(l)) \quad \Rightarrow \quad \forall t' \ (0 \leq t' \leq t \rightarrow \mathbf{x}(t') \in \mathcal{X}') \tag{4.1}$$
$$\forall t' \ (0 \leq t' \leq t \rightarrow \mathbf{x}(t') \in Inv(l)) \quad \Rightarrow \quad \forall t' \ (0 \leq t' \leq t \rightarrow \mathbf{x}(t') \notin A), \tag{4.2}$$

*then $A$ is called an* avoid set *and $\mathcal{X}'$ is called a* safe set *for $A$ of a location $l$. A safe set for $A$ is denoted by $Safe_A$.*

*The set $\mathcal{X}' \subseteq Inv(l)$ is an* invariant set *if condition (4.1) holds.* ♢

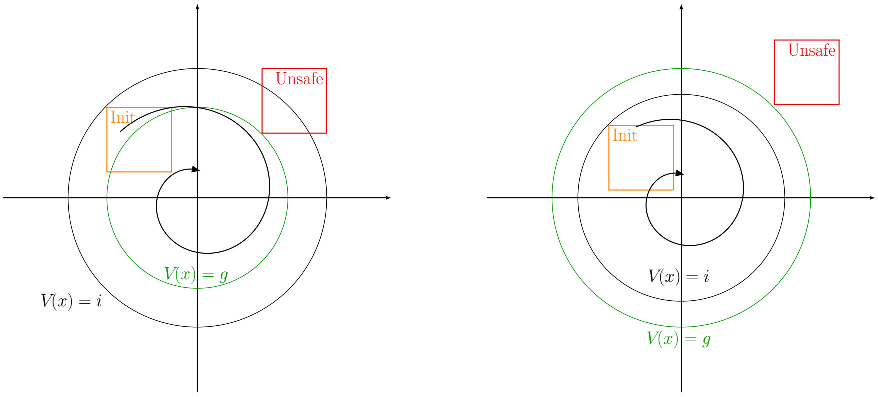
**Proposition 4.8**
*Any sublevel set of a Lyapunov function of a location is also an invariant set of the location.*

**Proof.**
*Let $L^-_{V_l,s}$ be a sublevel set of some location $l$, $\mathbf{x}(\cdot)$ a trajectory with $\mathbf{x}(0) \in L^-_{V_l,s}$, and $t_f \geq 0$. Assume, $\mathbf{x}(t')$ with $0 \leq t' \leq t_f$ is a trajectory segment that does not leave the invariant. A Lyapunov function assigns a value to any state of the state space, and the values along any trajectory do not increase. Hence, $\mathbf{x}(t')$ will not leave the sublevel set.* □

**Proposition 4.9**
*Let $\mathcal{X}'$ be an invariant set of a location. If $A \cap \mathcal{X}' = \emptyset$ holds for some set $A$, then $\mathcal{X}'$ is a safe set for the avoid set $A$.*

**Proof.**
*$\mathcal{X}'$ is an invariant set. Hence, each point on any trajectory which emanates from $\mathcal{X}'$ is either within $\mathcal{X}'$ or it violates the invariant. Since $\mathcal{X}'$ and $A$ have no points in common, none of the points in $A$ can be reached by a trajectory emanating from $\mathcal{X}'$ without violating the invariant.* □

**Corollary 4.10**
*Any sublevel set of a location which has an empty intersection with some set $A$ is a safe set for the avoid set $A$.*



(a) safe after entering $L_{V,g}$

(b) initially safe

Figure 4.6: A sketch of safe sets.

**Proof.**
*Follows immediately from Proposition 4.9 and Proposition 4.8.* □

Now, the alleviating argument for the reachability analysis is that a certain set, like *Unsafe* or a guard, cannot be reached as soon as a safe set for the respective set is entered.

Following Corollary 4.10, we aim at maximizing the extent of the sublevel set $L_{V,s} = \{\, \mathbf{x} \mid V(\mathbf{x}) \leq s \,\}$ for a given avoid set $A$. Let $g$ be a strict lower bound for the Lyapunov function over $A$, i. e., $g < \inf_{\mathbf{x} \in A} V(\mathbf{x})$. Since all states with a lower value are guaranteed not to be in $A$, the set $Safe_A = L_{V,g}$ is a safe set for $A$. Furthermore, if we find an upper bound $i \geq \sup_{\mathbf{x} \in Init} V(\mathbf{x})$ with $g \geq i$, then $A$ is unreachable from all initial states, and we can omit the reach set computation entirely. Both cases are visualized in Figure 4.6.

This yields the following basic algorithm:

(1) Determine the infimum of the Lyapunov function $b = \inf\{V(\mathbf{x}) \,|\, \mathbf{x} \in\ A\}$ over the avoid set $A$.

(2) Determine the supremum of the Lyapunov function $i = \sup\{V(\mathbf{x}) \,|\, \mathbf{x} \in\ Init\}$ over the initial set *Init*.

(3) If $i < b$, then the *Init* is safe (*initially safe*). Otherwise, subtract a safety margin $\epsilon$ from the infimum, $g = b - \epsilon$, to build the safe set $Safe_A = L_{V,g}$ (*safe after entering* $L_{V,g}$).

Although finding the infimum (resp. supremum) of a Lyapunov function over a compact set coincides with the search for the minimum (resp. maximum) and can be done via numerical optimization, our implementation performs a bi-sectioning combined on top of Z3. Further, our implementation returns a lower (resp. upper) bound instead of the infimum (resp. supremum) which is sufficient in our case. The procedure is as follows:

(1) Guess an initial bounding interval $[a, b]$ with $a = 0$ for the infimum (resp. supremum) of the Lyapunov function

**Algorithm 4:** The prepare function().

```
input  : a hybrid automaton $\mathcal{H}$, a set $Unsafe$
output : a set of Lyapunov functions $LFs$, and a prepared version of $\mathcal{H}$
$Inits', Trans', LFs \leftarrow \emptyset$;
// compute Lyapunov functions for each location in isolation
foreach $l \in \mathcal{H}.Loc$ do $LFs(l) \leftarrow$ computeLF($Flow(l), Inv(l)$);
$Trans' \leftarrow \mathcal{H}.Trans$;                              // copy transitions
foreach $(Init, l) \in \mathcal{H}.Inits$ do               // separate each initial location
    $l' \leftarrow$ copyLoc($\mathcal{H}, Unsafe, LFs, l$);                    // copy the location
    $Inits' \leftarrow Inits' \cup \{(Init, l')\}$ ;                  // add the new initial set
    // copy each outgoing transition to the new location
    foreach $(l, G, U, l_2) \in Trans'$ do $\mathcal{H}.Trans \leftarrow \mathcal{H}.Trans \cup \{(l', G, U, l_2)\}$;
$\mathcal{H}.Inits \leftarrow Inits'$ ;                            // replace initial states
```

(2) Refine the initial interval:
Ask Z3 to find an $\mathbf{x} \in A$ (resp. $\mathbf{x} \in Init$) such that $V(\mathbf{x}) \leq b$ (resp. $V(\mathbf{x}) \geq b$). While such an $\mathbf{x}$ cannot (resp. can) be found increase $b$.

(3) If a refined interval $[a, b]$ is found, then we enter the bi-sectioning:
Ask Z3 to find an $\mathbf{x} \in A$ (resp. $\mathbf{x} \in Init$) such that $V(\mathbf{x}) \leq \frac{b-a}{2}$ (resp. $V(\mathbf{x}) \geq \frac{b-a}{2}$). While $b - a \geq \epsilon$ continue with $[a, \frac{b-a}{2}]$ (resp. $[\frac{b-a}{2}, b]$) if such an $\mathbf{x}$ can be found and continue with $[\frac{b-a}{2}, b]$ (resp. $[a, \frac{b-a}{2}]$) otherwise.

(4) A safe approximation of the infimum (resp. supremum) of the Lyapunov function is given by $a$ (resp. $b$).

### SafeBox Conversion

In order to use safe sets for trajectory truncation in a polyhedral-based tool like SoapBox, we generate polyhedral underapproximations of safe sets. In this section, we shortly describe the idea of our method. Details of this method can be found in [HM15; HMT15]. Stabhyli generates quadratic Lyapunov functions. Hence, a sublevel set in our context is a quadric set $\left\{ \mathbf{x} \mid \mathbf{x}^T V \mathbf{x} \leq c^2 \right\}$ with $c > 0$ and a symmetric matrix $V$. Projectively principal axis transformation yields an invertible matrix $L$ and a diagonal matrix $E$, whose coefficients are equal to $-1$, 1, or 0, and sorted in descending order such that $V = LEL^T$. Using homogeneous coordinates this can also be expressed as

$$\tilde{V} = \begin{pmatrix} V & \mathbf{0} \\ \mathbf{0}^T & -c^2 \end{pmatrix} = \tilde{L}\tilde{E}\tilde{L}^T \text{ with } \tilde{E} = \begin{pmatrix} E & \mathbf{0} \\ \mathbf{0}^T & -1 \end{pmatrix} \text{ and } \tilde{L} = \begin{pmatrix} L & \mathbf{0} \\ \mathbf{0}^T & 1 \end{pmatrix}.$$

By Sylvester's law of inertia, the numbers of negative, positive, and zero coefficients in $E$ and $\tilde{E}$, respectively, are uniquely determined. Furthermore, let $E'$ be the matrix obtained from $E$ by replacing all occurrences of $-1$ by 0. This yields the implication $\mathbf{y}^T E' \mathbf{y} \leq 1 \Rightarrow \mathbf{y}^T E \mathbf{y} \leq 1$. Hence, the cylinder $\left\{ \mathbf{y} \mid \mathbf{y}^T E' \mathbf{y} \leq 1 \right\}$ over a lower-dimensional unit sphere is the largest inscribed convex and cylindrical set of $\left\{ \mathbf{y} \mid \mathbf{y}^T E \mathbf{y} \leq 1 \right\}$. Now, given template $\mathcal{H}$-polyhedra with circumspheres of arbitrary dimension, it is easy to generate an inscribed $\mathcal{H}$-polyhedron of the spherical cylinder. For our experiments, we used hypercubes and cross-polytopes. It remains to compute the image of the resulting polyhedra under the inverse transformation to $\binom{\mathbf{y}}{\mu} = \tilde{L}^T \binom{\mathbf{x}}{\lambda}$, which is computationally easy but involves non-trivial insights in projective geometry.[9]

### Unrolling Algorithm

Algorithm 4 and Algorithm 5 show the unrolling algorithm. Algorithm 4 is a *preparation function* that creates copies of the locations as well as all outgoing edges for each initial state set. The function `copyLoc` creates a fresh copy of a location with the same flow, invariant, SLLF, and unsafe set as the original location. Algorithm 5 is the main *unrolling algorithm*. It is executed after the preparation function. It maintains a job queue which is initialized with the initial state sets. Until the job queue is empty, it selects a job, computes safe sets and reach sets with respect to *Unsafe* and the guards of the outgoing transition. An intersection of the reach set with *Unsafe* shows that the hybrid system is unsafe. Intersections with guards are used to tighten the guards of transitions and are enqueued for further exploration by replacing the guard of the transition with the actual reach set. This unrolls the hybrid automaton in a breadth-first manner. If the job queue is empty, then the unrolling is followed by a post-processing. The post-processing removes all nodes that are not connected to a location of the initial set. The result is a forest of hybrid systems describing the trajectories abstractly. This unrolled hybrid automaton can be proven stable very efficiently according to Corollary 4.12. In fact, since the unrolled hybrid automaton is acyclic, the computed SLLFs may be reused.

### Global Lyapunov Function Computation

Now that we have established safety, we verify GAS of the unrolled hybrid automaton reusing the SLLFs. Since both automata are equivalent in the sense of feasible trajectories, this implies GAS of the original hybrid automaton.

For completeness, we introduce notations as well as a theorem from [Oeh11] which we will adapt to our needs in Corollary 4.12.

A decomposition of the underlying graph into SCCs allows us to apply the following theorem:

**Theorem 4.11 (Decomposition into strongly connected components [Oeh11, Theorem 4.1, Remark 4.3])**
*Let $\mathcal{H}$ be a hybrid automaton with $\mathcal{H} = (\mathit{Var}, \mathit{Loc}, \mathit{Trans}, \mathit{Flow}, \mathit{Inv}, \mathit{Inits})$. If all sub-automata pertaining to the SCCs of $\mathscr{G}(\mathcal{H})$ are globally attractive, then so is $\mathcal{H}$. If all SCCs are Lyapunov stable and if for all transitions $(l_1, G, U, l_2) \in \mathit{Trans}$*

[9] Actually, we use a projective generalization of polyhedra similar to the notion of *projective polyhedra* as it has been introduced in [Gal09].

**Algorithm 5:** The unrolling algorithm.

```
input : A hybrid automaton $\mathcal{H}$, a set $\mathit{Unsafe}$, and a set of Lyapunov functions $\mathit{LFs}$.
output: An unrolled version of $\mathcal{H}$.
Jobs $\leftarrow \mathcal{H}.\mathit{Inits}$;                       // start from each initial state set
while Jobs $\neq \emptyset$ do
    $(\mathit{Init}, l) \leftarrow$ pop(Jobs);                          // select a job
    // compute safe sets with respect to unsafe and convert them to
       safe boxes
    $\mathit{Safe}_{\mathit{Unsafe}} \leftarrow$ convertToBoxes(safeSets($\mathit{LFs}(l), \mathit{Init}, \mathit{Unsafe}(l)$));
    // compute reach set with respect to unsafe
    $R \leftarrow$ Reach($\mathit{Init}, \mathit{Flow}(l), \mathit{Inv}(l), \mathit{Safe}_{\mathit{Unsafe}}, \mathit{Unsafe}(l)$);
    if $R \neq \emptyset$ then markUnsafe($l$,$R$);                  // model is unsafe
    $\mathit{Trans}' \leftarrow \mathcal{H}.\mathit{Trans}$;                           // copy transitions
    // check reachability of each outgoing transition
    foreach $(l, G, U, l_2) \in \mathit{Trans}'$ do
        $\mathcal{H}.\mathit{Trans} \leftarrow \mathcal{H}.\mathit{Trans} \setminus \{(l, G, U, l_2)\}$;       // remove old transition
        // compute safe sets with respect to guard and convert them to
           safe boxes
        $\mathit{Safe}_G \leftarrow$ convertToBoxes(safeSets($\mathit{LFs}(l), \mathit{Init}, G$));
        // compute reach set with respect to guard
        $R \leftarrow$ Reach($\mathit{Init}, \mathit{Flow}(l), \mathit{Inv}(l), \mathit{Safe}_G, G$);
        if $R = \emptyset$ then continue;                       // guard unreachable
        $l' \leftarrow$ copyLoc($\mathcal{H}$, Unsafe, $\mathit{LFs}, l_2$);              // copy the location
        $\mathcal{H}.\mathit{Trans} \leftarrow \mathcal{H}.\mathit{Trans} \cup \{(l, R, U, l')\}$;      // add refined incoming
          transition
        // copy each outgoing transitions to the new location
        foreach $(l_2, G_2, U_2, l_3) \in \mathit{Trans}'$ do
          $\mathcal{H}.\mathit{Trans} \leftarrow \mathcal{H}.\mathit{Trans} \cup \{(l', G_2, U_2, l_3)\}$;
        Jobs $\leftarrow$ Jobs $\cup\{$(apply($R, U$), $l'$)$\}$;         // append updated postset
```

*corresponding to bridges of $\mathscr{G}(\mathcal{H})$, it holds that*

$$\exists c > 0 \bullet \forall \mathbf{x} \in G \bullet V_{C_2}(l_2, U(\mathbf{x})) \leq c \cdot V_{C_1}(l_1, \mathbf{x})$$

*where $C_i$ is the SCC containing $l_i$ for $i \in 1, 2$, then $\mathcal{H}$ is Lyapunov stable. Therefore, $\mathcal{H}$ is GAS.*

Therefore, if a hybrid automaton consists of SCCs which contain at most one location — as it is the case for an unrolled hybrid automaton, — then we can compute Lyapunov functions for each location in isolation, and we can check satisfiability of the transition constraints afterwards.

This leads to the following corollary which is basically a reformulation of Theorem 4.11

in the context of unrolled hybrid automata.

**Corollary 4.12 (GAS of an unrolled Hybrid Automaton)**
*Let $\mathcal{H}$ be an unrolled hybrid automaton with $\mathcal{H} = (\mathit{Var}, \mathit{Loc}, \mathit{Trans}, \mathit{Flow}, \mathit{Inv}, \mathit{Inits})$. If all locations are globally attractive, then so is $\mathcal{H}$. If all locations are Lyapunov stable and for all transitions $(l_1, G, U, l_2) \in \mathit{Trans}$ it holds that*

$$\exists c > 0 \bullet \forall \mathbf{x} \in G \bullet V_{l_2}(U(\mathbf{x})) \leq c \cdot V_{l_1}(\mathbf{x}),$$

*then $\mathcal{H}$ is Lyapunov stable. Consequently, $\mathcal{H}$ is GAS.*

**Proof.**
*Follows from Theorem 4.11, if every location of $\mathcal{H}$ belongs to exactly one SCC. This is the case because $\mathcal{H}$ is unrolled and, in turn, acyclic.* □

Since we already have SLLFs, it remains to show that for each transition the factor $c$ as used in Corollary 4.12 actually exists. If this is successful, then we can conclude that the hybrid automaton is GAS.

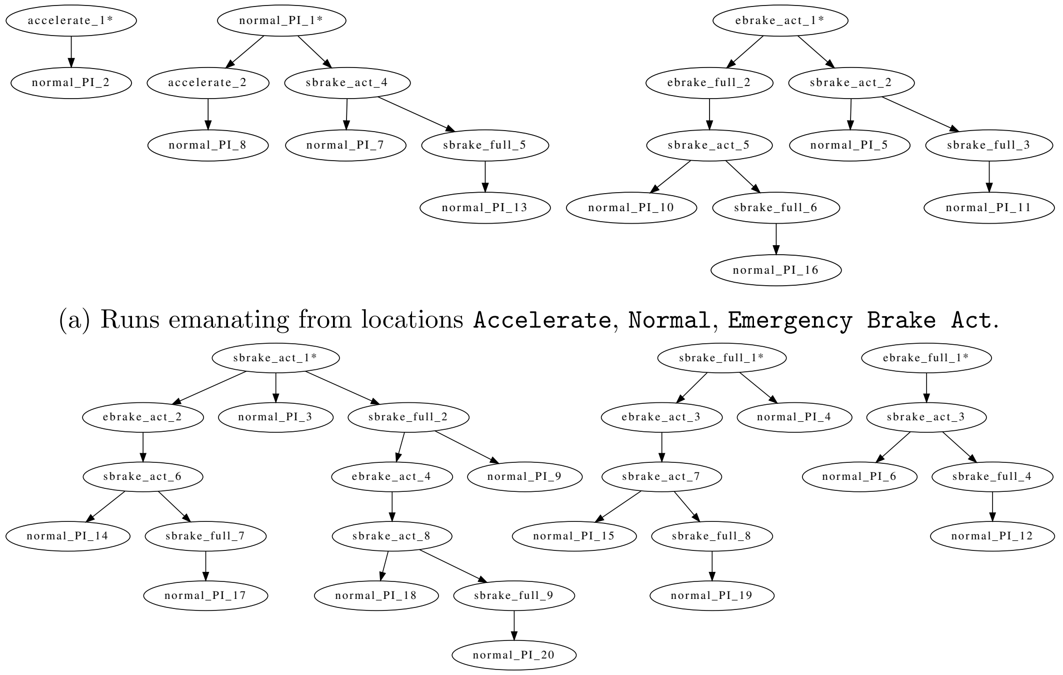
### 4.3.3 Experiments

In this section, we present our benchmark set and compare the time needed for verification of their respective properties. The benchmark set consists of four examples

1. the automatic cruise controller (see Example 1),
2. the spidercam (see Example 4 and Figure 4.4a), and
3. a velocity controller for which we verify stability and safety and which is part of the case study presented in Chapter 5,
4. an artificial example for which it is impossible to prove stability without further reachability analysis.

Examples 1 and 2 can trivial be extended to also include unsafe sets. Doing so requires us to further verify safety. Due to our combined approach this verification can be done with no additional costs because we already have the complete reachability information at hand.

**Example Automatic Cruise Controller (ACC)** The unrolled version of the ACC (see Figure 3.14 on page 85) is given in Figure 4.7. It has six initial locations and at most five location switches occur until every trajectory enters and stays in the location `Normal PI`.



(a) Runs emanating from locations `Accelerate`, `Normal`, `Emergency Brake Act`.

(b) Runs emanating from locations `Service Brake Act`, `Service Brake Full`, `Emergency Brake Full`.

Figure 4.7: The unrolled automatic cruise controller.

**Example Spidercam** Every trajectory of the spidercam needs at most five transitions until it enters and stays in the location `center`. But due to the number of initial locations, the unrolled version has 193 locations and 184 transitions. A small snippet which contains only those runs emanating from the location `Positive X` is shown in Figure 4.8.

**Example 5: A Velocity Controller** (VC) — visualized in Figure 4.9 — is part of the Advanced Driver Assistance System (ADAS) presented and analyzed in [Dam+14; DMR14; MHR17] and briefly introduced in Section 5.4. The ADAS consists of two concurrent controllers (as well as helper components) that cooperatively achieve the following objectives:

**(Obj1)** maintain a centrifugal force comfortable for a driver,

**(Obj2)** bring and then keep the car on the center of its lane,

**(Obj3)** control the speed whereby also considering driver requests for a certain speed value.

The VC contributes to Objective Obj1 and Objective Obj3.
The model has three modes: one location with a constant acceleration, one location with a constant deceleration and one location doing the fine-tuning via a PI controller. The

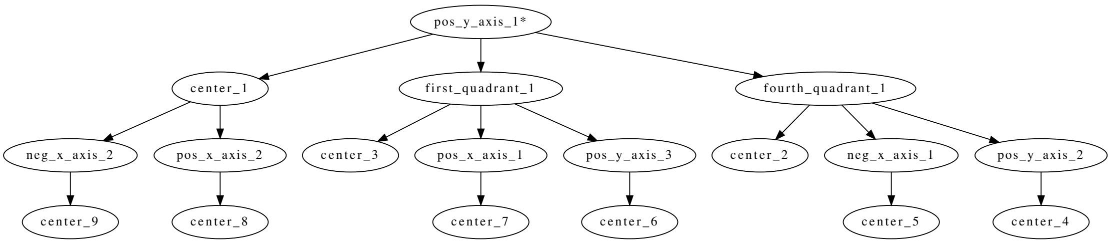


Figure 4.8: The unrolled spidercam (only runs emanating from the location `Positive X` are shown).

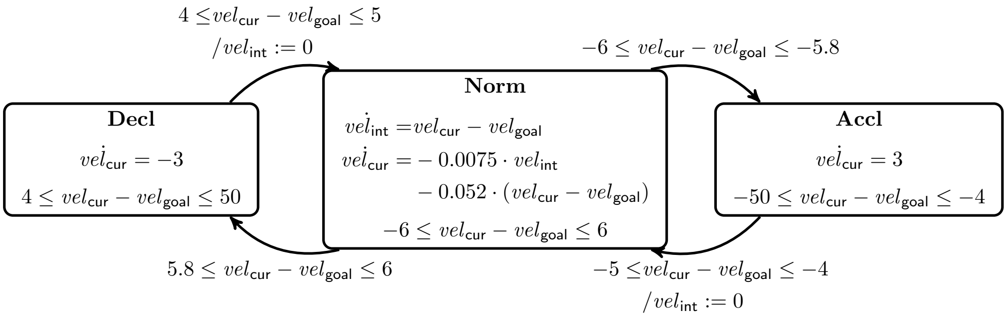

$4 \le vel_{\text{cur}} - vel_{\text{goal}} \le 5$
$/vel_{\text{int}} := 0$
$-6 \le vel_{\text{cur}} - vel_{\text{goal}} \le -5.8$
Norm
$\dot{vel}_{\text{int}} = vel_{\text{cur}} - vel_{\text{goal}}$
$\dot{vel}_{\text{cur}} = -0.0075 \cdot vel_{\text{int}} - 0.052 \cdot (vel_{\text{cur}} - vel_{\text{goal}})$
$-6 \le vel_{\text{cur}} - vel_{\text{goal}} \le 6$
Decl
$\dot{vel}_{\text{cur}} = -3$
$4 \le vel_{\text{cur}} - vel_{\text{goal}} \le 50$
Accl
$\dot{vel}_{\text{cur}} = 3$
$-50 \le vel_{\text{cur}} - vel_{\text{goal}} \le -4$
$5.8 \le vel_{\text{cur}} - vel_{\text{goal}} \le 6$
$-5 \le vel_{\text{cur}} - vel_{\text{goal}} \le -4$
$/vel_{\text{int}} := 0$

Figure 4.9: The velocity controller [Dam+14].

VC's task is to drive the current velocity of the vehicle $vel_{\text{cur}}$ to a desired velocity $vel_{\text{goal}}$. This desired velocity is given by an external input that might be updated discretely. The verification task is to show that

- $vel_{\text{cur}}$ converges to $vel_{\text{goal}}$,
- if $vel_{\text{cur}} - vel_{\text{goal}} \le -3$ initially holds, then $vel_{\text{cur}} - vel_{\text{goal}} \le 3$ always holds,
- if $vel_{\text{cur}} - vel_{\text{goal}} \in [-3, -2]$ initially holds, then $vel_{\text{cur}} - vel_{\text{goal}} \le 2$ always holds,
- if $vel_{\text{cur}} - vel_{\text{goal}} \in [-2, 0]$ initially holds, then $vel_{\text{cur}} - vel_{\text{goal}} \le 1$ always holds.

The later three are safety properties and restrict the peak-overshoot. For the verification, all properties are considered under the assumption that $vel_{\text{goal}}$ remains constant once set.

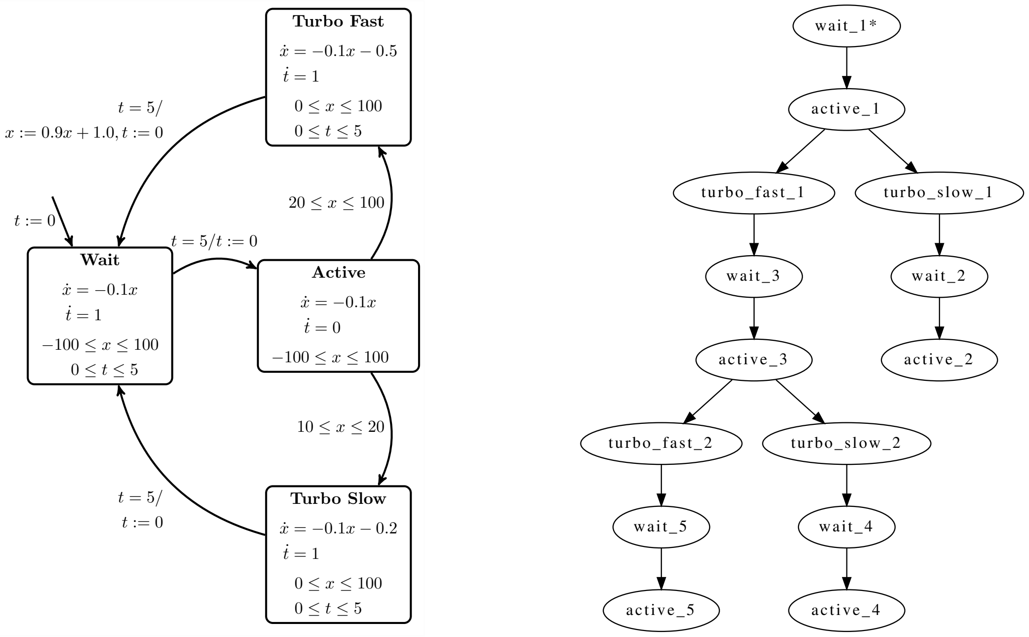
**Example 6: An Artificial Example** is a model (see Figure 4.10a) which cannot be proven stable without further reachability information. The reason is that the guard of the transition from location `Turbo Fast` to `Wait` does not restrict values of $x$. Thus, naïvely generating constraints due to Theorem 2.31 leads to the following snippet of constraints

$$\begin{aligned} \forall x, t \bullet x \in [-100, 100] \wedge t \in [0, 5] \Rightarrow \alpha(||x||) \preceq V_{\texttt{Wait}}(x, t) \\ \forall t \bullet t \in [0, 5] \Rightarrow V_{\texttt{Turbo Fast}}(0, t) = 0 \\ \forall x \bullet V_{\texttt{Wait}}(0.9x + 0.1, 0) \leq V_{\texttt{Turbo Fast}}(x, 5). \end{aligned}$$

Obviously, no such $V_{\texttt{Wait}}, V_{\texttt{Turbo Wait}}$ exist. On the other hand, due to the unrolling, we can conclude that the transition may only be taken with $x \in [10.1, 38.9]$ which allows us to replace the last constraint by

$$\forall x \bullet x \in [10.1, 38.9] \Rightarrow V_{\texttt{Wait}}(0.9x + 0.1, 0) \leq V_{\texttt{Turbo Fast}}(x, 5),$$

and, indeed, for $x \geq 10$, the Lyapunov function value may not increase.[10] The unrolled automaton is sketched in Figure 4.10b.



(a) unmodified

(b) unrolled

Figure 4.10: Unrolling of an artificial example.

## Results

Table 4.11 shows, in order, the depth of the unrolled hybrid automaton and the time

[10] Note, that the update increases $x$ in case the transition is taken with $x < 10$. This renders the system non-LS (cf. Definition 2.26). However, reachability information reveals that no such trajectory exists.

| | Depth | Stabhyli | SafeSet | BoxConvert | SoapBox | Total Time |
|---|---|---|---|---|---|---|
| VC | 3 | 0.39 | 0.16 | 2.40 | 38.60 | 41.55 |
| ACC | 6 | 0.82 | 1.00 | 57.20 | 291.60 | 350.62 |
| Spidercam | 5 | 2.14 | 580.30 | 11.00 | 3814.70 | 4408.14 |
| Example 6 | 8 | 1.65 | 5.67 | 0.06 | 58.04 | 65.42 |

(a) including Matlab startup

| | Depth | Stabhyli | SafeSet | BoxConvert | SoapBox | Total Time |
|---|---|---|---|---|---|---|
| VC | 3 | 0.39 | 0.16 | 2.40 | 14.60 | 17.55 |
| ACC | 6 | 0.82 | 1.00 | 57.20 | 39.60 | 98.62 |
| Spidercam | 5 | 2.14 | 580.30 | 11.00 | 1098.70 | 1692.14 |
| Example 6 | 8 | 1.65 | 5.67 | 0.06 | 10.04 | 17.42 |

(b) excluding Matlab startup

Table 4.11: Detailed computation times.

needed to compute the SLLFs, the safe sets, the inscribed polygon (hypercube and cross-polytope), the reachability information, and, in the last column, the total runtime.[11] Since SoapBox is written in Matlab and our current prototype tool runs SoapBox for each reach set computation, we did two comparisons: (a) with and (b) without the time for the Matlab (re-)initialization (once for each computation).

| | Stabhyli (common) | Stabhyli (piecewise) | Stabhyli (decomposition) | Unrolling |
|---|---|---|---|---|
| VC | X | 0.21s | 22.29s | 17.55 |
| ACC | X | 1.70s | 112.99s | 98.62 |
| Spidercam | 1.37s | 6.97s | X | 1692.14 |
| Example 6 | X | X | X | 17.42 |

Table 4.12: Comparison of computation times.

In Table 4.12, we compare the runtime of the proposed approach with the runtime of Stabhyli searching for a common Lyapunov function, Stabhyli searching for a piecewise Lyapunov function, and Stabhyli using the decompositional approach. We can conclude that although the proposed unrolling technique is not the fastest, its runtime is comparable to the runtime of the decompositional approach and may handle examples that other approaches cannot handle. Furthermore, if a benchmark exposes a safety property, too (like the VC), then additional time is required to verify the safety properties which nullifies the advantage of the shorter runtime.

## 4.4 Summary

In this chapter, we have presented two techniques to simplify the search for Lyapunov functions. The first technique relaxes the graph structure of the hybrid automaton: if the graph of the hybrid automaton is dense, then the relaxed version contains significantly fewer cycles. The idea is to re-route every transition through a new dummy location.

[11] An Intel© Core™ i7-3770T CPU with 2.50GHz and 8GB of RAM (in single-core mode) was used to run the benchmarks.

Thus, if in the original automaton a single transition is taken, then the relaxed automaton has to take the cascade of two transitions to achieve the same result. This is an advantage for the decompositional proof scheme because it implicitly computes cycle covers for every strongly connected components of the underlying graph. Therefore, fewer cycles lead to a reduced effort. Even more, the cycle covers have to satisfy that each pair of cycles shares at most one node. Our relaxation immediately produces such a structure. However, the main advantage is technical: a cycle encountered by the decompositional proof scheme results in a stability problem whose solution set is underapproximated by a finite number of solutions. This underapproximated solution set is reused in further computations. Therefore, the more subproblems have to be solved and underapproximated, the more likely we end up with an empty solution set due to underapproximation. That means that due to our relaxation technique the decompositional proof technique is available for hybrid automata whose graph structure is very dense. This is desired as the decomposition is particularly well-suited to prove stability of large-scale hybrid systems because it allows:

1. to decompose a monolithic proof into several smaller subproofs,
2. to reuse subproofs after modifying the hybrid system, and
3. to identify critical parts of the hybrid automaton.

All these benefits are not given when the hybrid system exhibits a very dense graph structure of the automaton because that would lead to an enormous number of computational steps required in the decomposition. The proposed relaxation overcomes these matters in the best case. Nevertheless, our relaxation is not for free and may introduce runs which are not stable and may render the search for Lyapunov function unsuccessful. Thus, if the relaxation is too loose, then our technique falls back to step-by-step reconstructing the original automaton. Each step increases the effort needed for the decomposition until a proof succeeds or ultimately — in the worst case — the original automaton gets decomposed.

Furthermore, the procedure can be automated which is very much desired as our focus is the automation of Lyapunov function-based stability proofs. In Section 4.2.2, by successfully employing the proposed technique in some examples, we showed usefulness of our approach.

The second technique is called unrolling or unfolding and simultaneously allows us to verify both, safety and stability properties of hybrid systems. In contrast to the simple approach of verifying the properties separately, we merge the verification procedures such that it makes either verification task simpler. Our approach allows us to exploit the knowledge that is gained during the verification of one problem in the verification of the other one. From the safety perspective, we use the fact that sublevel sets — which are obtained from Lyapunov functions — reveal subsets of the state space which are known to be safe. We stop the reachability analysis as soon as the safe set is entered. From the stability perspective, we use an unrolled version of the hybrid system's automaton — which is obtained by repeated reach set computations — to have a more precise characterization of the feasible trajectories. Obtaining Lyapunov functions for this unrolled hybrid automaton reduces the computational effort. It allows us to find Lyapunov functions for systems where the representation of the system contains implicit information that is needed to successfully prove stability.

For some hybrid automata this approach might not be applicable. This is the case when the sequence of locations in a run is infinite and thus a run keeps switching locations. In this case, we believe that the presented techniques still can fruitfully be applied to parts of the automaton. Reachability analysis profits from safe sets even when the stability of the overall automaton cannot be established. Additionally, knowledge gained by partial reachability computations on sub-components of the automaton helps us to relax the transition constraints for the computation of a global Lyapunov function. The usefulness of our approach has been shown by some promising experiments (see Section 4.3.3).

# CHAPTER 5

# A Framework for Designing Safe and Stable Hybrid Systems

In this chapter, we focus on the structured design of stable and safe hybrid systems. The key idea is to facilitate the reuse of components and employ composition to create large-scale hybrid systems out of smaller sub-components. Indeed, we would gain the following benefits:

- **reduced verification effort:** smaller models tend to be more tractable with respect to verification; minor changes in a sub-component do not necessarily trigger a complete rerun of the verification task,
- **improved manual inspection:** smaller models are easier to comprehend by humans,
- **reduced development costs:** creating new models does not require an engineer to start from scratch but rely on mature and well-known components; the influence of changes in sub-components are easier to locate and contract breaking components can be replaced.

In these settings, we assume a clear separation of the plant $\mathsf{P}$, a controller $\mathsf{C}$, and their common environment $\mathsf{env}$ as visualized in Figure 5.1. The controller might read the plant's state via *sensors* $\mathsf{Sens}$ and manipulate the state via *actuators* $\mathsf{Act}$ while an external (not modeled) environment may additionally manipulate the plant's state via *disturbances* $\mathsf{Disturb}$. Given a plant our goal is to systematically construct a controller that achieves certain objectives on the plant. Such objectives include stability properties, e. g., driving the plant state to a certain equilibrium, and safety properties, e. g., the plant state will always avoid a certain set of unsafe states. The design flow is then as follows: first, we design small controllers that either achieve some objectives on their own or contribute to cooperatively achieving an objective. Second, we compose the smaller controllers to larger ones and deduce properties of the composite based on the sub-component's properties.

To be able to deduce properties of a composed component from properties of its sub-components, contracts and interfaces are used. This is called *contract-based design* [Ben+08; SDP12]. *Interfaces* are used to describe properties of a component's assumption on its *deployment context* as well as the guarantees it will achieve in that context, which we also call its *service*. Using an interface instead of the implementation allows us

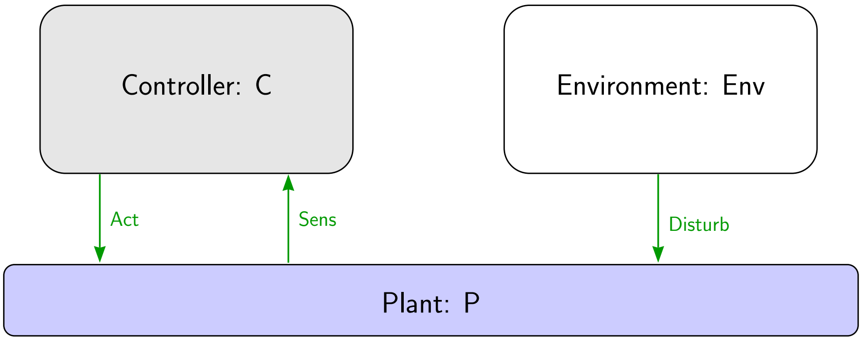


Figure 5.1: General setting: a clear separation between the controller, the plant, and their common environment

to (1) once check satisfaction of the interface in isolation and (2) later – during the composition – rely on the properties annotated in the interface to check compliance with the contract imposed by the composition operation. This last step is called *virtual integration*.

In this chapter, we focus on a parallel composition operator which complements the sequential composition operator introduced by Damm et al. in [Dam+10]. The sequential composition operator allows passing the burden of controlling a plant to another component while the parallel composition operator allows to have multiple controllers run simultaneously and controlling distinct parts of a common plant.

Neither interfaces nor implementations are unique. On one hand, there can be multiple interfaces to an implementation, for example, covering different operation modes, hiding or exposing certain implementation details, or describing certain levels of degradation either of the component itself or its environment. On the other hand, different implementations might satisfy the same interface, for example, we might have two controllers with different deployment contexts if one controller's context comprises the other controller's context while achieving the same guarantees it also satisfies the interface of the other controller but not necessarily vice versa.

The problem addressed in this chapter can be summarized as: carefully design a framework (1) making realistic assumptions on the deployment context, (2) allowing local verification of safety and stability properties of sub-controllers, (3) defining composition operators that preserve the locally established properties of the sub-controllers. Table 5.2 summarizes the contributions and the main responsible authors.

The remainder of the chapter is organized as follows: In Section 5.1, we formally introduce an extended version of hybrid automata, namely hybrid input/output automata, environmental predicates that capture the behavior of a controller's expected environment, and adapt the definition of runs to these new settings. Section 5.2 presents a brief introduction on the sequential composition operator as proposed in [Dam+10]. Section 5.3 is the main part of this chapter and contains the stability-related theory to the framework as published in [DMR16]. The stability-related part is solely contributed by this thesis's

| **Contribution** | **main contributor** |
|---|---|
| Framework [DMR16] | |
| • design of composition operator | shared |
| • correctness of composition | Astrid Rakow |
| • design of event mechanism | shared |
| • correctness of composition with events | Astrid Rakow |
| Framework-tailored stability notions | |
| • definition | author |
| • correctness | author |
| • compositionality | author |
| Case Study [MHR17; DMR14; Dam+14] | |
| • controller design and specification | author |
| • manual unrolling technique | author |
| • over-approximation techniques | author |
| • hybridization technique | shared |
| • local stability proofs | author |
| • local safety proofs | author |
| • reachability tool extension | Willem Hagemann |

Table 5.2: Contributions addressed in this chapter

author, while the remainder of the framework is mostly joint work. In this section, we first extend global asymptotic stability and the Lyapunov theorem to hybrid I/O automata and then show how to apply these in the context of parallel composition. Indeed, we introduce two new stability notions:

- global asymptotic stability under assumptions (GAS-Asm) and

- conditional global asymptotic stability (conGAS).

The later says that stability holds only while a certain (external) condition is satisfied by the environment and the former is a simple instantiation where the condition is globally assumed to hold. These two notions allow us to encode stability objectives that are always guaranteed and stability objectives that are only guaranteed in some situations such as after negotiating a contract with a cooperating controller. Finally, in Section 5.4, we briefly sketch a case study — published in [Dam+14; MHR17; DMR14] — which has been used to develop the presented framework.

## 5.1 Preliminaries

Let us first extend the hybrid automaton definition to include inputs and outputs. These are continuous variables whose evolution is not described by the automaton but are externally driven and continuous variables whose evolution is determined by the automaton and may be read by another entity. Thus, the set of variables of a hybrid I/O automaton consists of three disjoint sets of variables: input variables, local variables, and output variables. Hybrid I/O automata have been introduced by Lynch et al. in [LSV03] and in contrast to our definition, they also have a finite alphabet of input, local, and output actions. We have not included them in favor of communication channels that we will introduce later. The main difference is that we do not need to require some form of receptiveness [AL91] for the actions.

**Definition 5.1 (Hybrid I/O Automaton)**
*A hybrid I/O automaton (HIOA) is a tuple*

$$\mathcal{A} = (\mathit{Loc}, \mathit{Var}, \mathit{Trans}, \mathit{Flow}, \mathit{Inv}, \mathit{Inits})$$

- *$\mathit{Loc}$ is a finite set of* locations,
- *$\mathit{Var} = \mathit{Var}^L \,\dot\cup\, \mathit{Var}^I \,\dot\cup\, \mathit{Var}^O$ is a finite set of* variables *where $\mathit{Var}^L$, $\mathit{Var}^I$, $\mathit{Var}^O$ are disjoint sets of* local, input, and output variables *over $\mathbb{R}$ with $\mathit{Var}^C := \mathit{Var}^L \,\dot\cup\, \mathit{Var}^O$ denoting the* controlled variables *and $\mathcal{X} = \mathbb{R}^{|\mathit{Var}|}$, $\mathcal{X}^I = \mathbb{R}^{|\mathit{Var}^I|}$, and $\mathcal{X}^C = \mathbb{R}^{|\mathit{Var}^C|}$, being the* global, input, and controlled state space*.*
- *$\mathit{Trans}$ is a finite set of* transitions *$(l_1, G, U, l_2)$ where*
  - *$l_1, l_2 \in \mathit{Loc}$ are the* source and target location *of the transition, respectively,*
  - *$G \subseteq \mathcal{X}$ is a guard which restricts the valuations of the variables for which this transition can be taken,*
  - *$U : \mathcal{X} \to \mathcal{X}^C$ is the update* function *which might update some valuations of the variables,*
- *$\mathit{Flow} : \mathit{Loc} \to [\mathcal{X} \to \mathcal{P}(\mathcal{X}^C)]$ is the* flow function *that assigns a flow to every location. A flow $f : \mathcal{X} \to \mathcal{P}(\mathcal{X}^C)$ in turn assigns a closed subset of $\mathcal{X}^C$ to each $\mathbf{x} \in \mathcal{X}$, which can be seen as the right-hand side of a differential inclusion $\dot{\mathbf{x}}_{\downarrow \mathit{Var}^C} \in f(\mathbf{x})$,*
- *$\mathit{Inv} : \mathit{Loc} \to \mathcal{P}(\mathcal{X})$ is the invariant* function *that assigns a closed subset of the global continuous state space to each location $l \in \mathit{Loc}$, and therefore restricts valuations of the variables for which this location can be active.*
- *$\mathit{Inits} \subseteq \mathit{Loc} \times \mathcal{X}$ is a closed set of* initial (hybrid) states *with elements $(l, \mathit{Init})$ where $l$ is the* initial discrete state *and $\mathit{Init}$ is the* initial continuous state*.*

*Let $\mathit{Var}^H \subseteq \mathit{Var}^O$ be a subset of the output variables. $\mathcal{A} \setminus \mathit{Var}^H$ denotes the hybrid automaton $(\mathit{Loc}, \mathit{Var}', \mathit{Trans}, \mathit{Flow}, \mathit{Flow})$ with $\mathit{Var}' = \mathit{Var}^{L'} \,\dot\cup\, \mathit{Var}^I \,\dot\cup\, \mathit{Var}^{O'}$ where $\mathit{Var}^{L'} := \mathit{Var}^L \cup \mathit{Var}^H$ and $\mathit{Var}^{O'} := \mathit{Var}^O \setminus \mathit{Var}^H$ where the variables $\mathit{Var}^H$ are hidden.* ♢

The discrete updates will often be implicitly defined by a sequence of assignments of the form $v := e$, with $v \in \mathit{Var}^C$ and $e$ an expression over $\mathit{Var}$. We identify such a sequence of assignments with the predicate relating the pre- and post-states of its sequential execution. If a set of assignments is empty, it will be left out, meaning that all variables in $\mathit{Var}$ remain unchanged upon switching. For a predicate $\phi$, $\phi[\mathtt{Assgn}]$ is the result of the substitution induced by $\mathtt{Assgn}$ on $\phi$.

As a shorthand notation, let $\phi$ be a predicate over the set of variables $\mathit{Var} \cup \mathit{Var}^\star$ where $\mathit{Var}^\star$ are decorated conterparts of the variables $v \in \mathit{Var}$ and let $\mathbf{x}^\star \in \mathbb{R}^{|\mathit{Var}^\star|}$ be a valuation, we write $\mathbf{x}, \mathbf{x}^\star \models \phi$ to denote $\mathbf{x} \models \phi[\mathit{Var}^\star/\mathbf{x}^\star]$ as a shorthand notation for decorations $\star \in \{', \dot{}\}$ representing discrete updates and derivatives, respectively.

To capture an environment, we define an environment predicate:

**Definition 5.2 (Environment Predicate [DMR16])**
*Let $\mathit{Var}^E$ be the finite set $\mathit{Var}^L \uplus \mathit{Var}^I$ and $\mathcal{X}^E = \mathbb{R}^{|\mathit{Var}^E|}$ be the corresponding* environmental state space. *$\varphi^{env}$ is an* environment predicate *on $\mathit{Var}^E$ if it is of the form*

$$\varphi^{env} := \varphi^{env}_{inv} \wedge (\neg \mathsf{flow} \Rightarrow \varphi^{env}_{dscr}) \wedge (\mathsf{flow} \Rightarrow \varphi^{env}_{flow})$$

*where*

- *$\varphi^{env}_{inv}$ is a first order predicate over $v \in \mathit{Var}^E$ denoting invariant restrictions on the valuations of the variables as the subset $\mathit{Inv}^E := \left\{ \mathbf{x} \in \mathcal{X}^E \;\middle|\; \mathbf{x} \models \varphi^{env}_{inv} \right\} \subseteq \mathcal{X}^E$,*
- *$\mathsf{flow}$ is an atomic proposition which is true during a flow and false otherwise*
- *$\varphi^{env}_{dscr}$ is a first order predicate over $v \in \mathit{Var}^I \cup \mathit{Var}^{I'}$ denoting restrictions on the discrete updates performed by the environment as $U^E(\mathbf{x}^I) := \left\{ \mathbf{x}^{I'} \in \mathcal{X}^I \;\middle|\; \mathbf{x}^{I'}, \mathbf{x}^I \models \varphi^{env}_{dscr} \right\}$ where set $\mathit{Var}^{I'}$ holds decorated variants of the variables in $\mathit{Var}^I$, representing the value after the discrete update,*
- *$\varphi^{env}_{flow}$ is a first order predicate over $v \in \mathit{Var}^I \cup \mathit{Var}^{I^\bullet}$ denoting restrictions on the continuous flow. The predicate allows the environment the exhibit any flow in $\mathit{Flow}^E(\mathbf{x}^I) := \left\{ \dot{\mathbf{x}}^I \in \mathcal{X}^I \;\middle|\; \dot{\mathbf{x}}^I, \mathbf{x}^I \models \varphi^{env}_{flow} \right\}$, where set $\mathit{Var}^{I^\bullet}$ holds decorated variants of the variables in $\mathit{Var}^I$, representing differential equations or inclusions.*

♢

An environmental predicate allows us to specify the behavior on inputs and local variables in terms of invariants over the continuous flow, the discrete updates as well as the effects on the local state. At first, it might seem counterintuitive that an environment predicate, refers to inputs and also to local variables because this means that if we want to check that an environment adheres to an environment predicate then we have to also consider the composition with the automaton whose environment is specified. However,

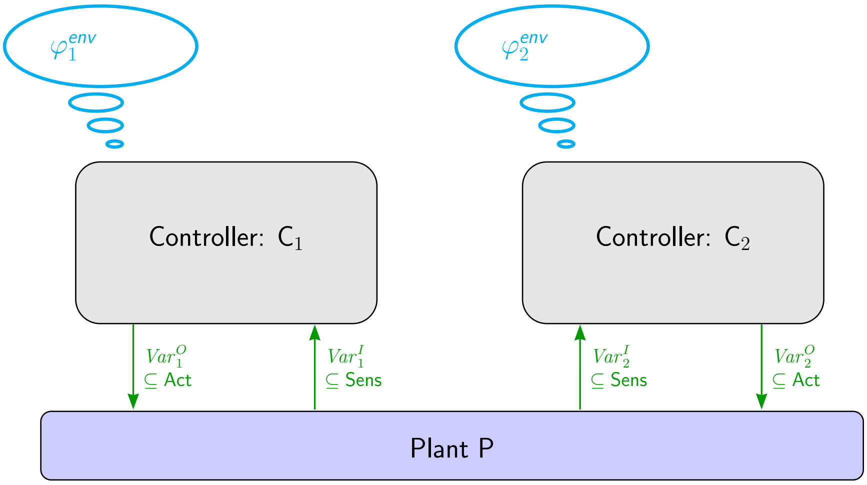

$\varphi_1^{env}$
$\varphi_2^{env}$
Controller: $C_1$
Controller: $C_2$
$Var_1^O$ $\subseteq$ Act
$Var_1^I$ $\subseteq$ Sens
$Var_2^I$ $\subseteq$ Sens
$Var_2^O$ $\subseteq$ Act
Plant P

Figure 5.3: The deployment context of a controller $\mathsf{C}_i$ is specified using an environment predicate $\varphi_i^{env}$.

in Section 5.3, we will restrict the local variables the predicate might constrain to the plant's variables. This makes sense, since in our setting, where we consider a controller for a given plant, we have to be able to restrict the behavior of controller's (unknown) environment, i. e., everything that might happen at the controllers inputs and everything that might happen at the plant's open inputs (its actuators). That way, an engineer is able to specify the deployment context of a controller with respect to the controller's sensor readings and with respect to actuators that are not under its control. By imposing syntactical restrictions on the assumptions of a controller, we are able to check whether an environment satisfies the environment predicate by considering the plant and the environment without the controller. This is needed during the parallel composition of two controllers as a means to show that each controller yields a valid environment for the other one. Indeed, we have to check whether each controller together with the plant satisfies the environment predicate of the other controller. Consider the visualization in Figure 5.3, let $\varphi_1^{env}$ be the environment predicate for the controller $\mathsf{C}_1$. To ensure composability, we have to check whether $\mathsf{C}_2$ and the common plant $\mathsf{P}$ satisfy $\varphi_1^{env}$ and vice versa.

As shorthand notations we refer to $\mathbf{x}^I, \mathbf{x}^L, \mathbf{x}^O, \mathbf{x}^C$ as the projection of a continuous state $\mathbf{x}$ of a HIOA $\mathcal{A}$ onto the set of input variables $Var^I$, local variables $Var^L$, output variables $Var^O$, or controlled variables $Var^C$. Likewise, for a vector-valued function over time $\mathbf{x}(t)$, we use $\mathbf{x}^I(t), \mathbf{x}^L(t), \mathbf{x}^O(t), \mathbf{x}^C(t)$ to refer to the projections onto the input, local, output, and controlled variables, respectively.

Next, we define satisfaction of an environmental predicate for a hybrid I/O automaton. It basically says that every invariant, every flow, and every update is covered by the environmental predicate. It is a very strong property in the sense that it is does not take into account the reachable states of an automaton but this conservatism allows us to check satisfaction without computing the trajectories of the system.

**Definition 5.3 (Satisfaction of an Environmental Predicate)** *Let $\mathcal{A}$ be a hybrid I/O automaton $\mathcal{A} = (Loc, Loc, Trans, Flow, Inv, Inits)$ and let $\varphi^{env} = \varphi^{env}_{inv} \wedge (\neg \mathsf{flow} \Rightarrow \varphi^{env}_{dscr}) \wedge (\mathsf{flow} \Rightarrow \varphi^{env}_{flow})$ be an environmental predicate on $Var^L \uplus Var^I = Var^E \subseteq Var$. We say $\mathcal{A}$* satisfies the environmental predicate *$\varphi^{env}$ if and only if*

**(Initial Satisfaction)** *for all initial states $(l, Init) \in Inits_i$ holds*

$$\forall \mathbf{x} \in Init \bullet \mathbf{x} \models \varphi^{env}_{inv},$$

**(Invariant Satisfaction)** *for all locations $l \in Loc$ holds*

$$\forall \mathbf{x} \in Inv(l) \bullet \mathbf{x} \models \varphi^{env}_{inv},$$

**(Flow Satisfaction)** *for all locations $l \in Loc$ holds*

$$\forall \mathbf{x} \in Inv(l) \bullet \forall \dot{\mathbf{x}} \in Flow(l)(\mathbf{x}) \bullet \dot{\mathbf{x}}, \mathbf{x} \models \varphi^{env}_{flow},$$

**(Update Satisfaction)** *for all transitions $(l, G, U, l') \in Trans$ holds*

$$\forall \mathbf{x} \in G \bullet (\mathbf{x} \models \varphi^{env}_{inv}) \Rightarrow (\mathbf{x}, U(\mathbf{x}) \models \varphi^{env}_{dscr}).$$

*Satisfaction of an environmental predicate is denoted by $\mathcal{A} \models \varphi^{env}$.* ♢

The above definition is very conservative in the sense that satisfaction of the environmental predicate depends on the static information only and not on runtime information. This has the advantage that checking satisfaction is much simpler because we do not need to generate all (possibly infinitely many) (possibly infinite) solutions but instead, we have to inspect only the invariants, flow functions, and transitions. This also follows roughly the spirit of the Lyapunov theorem, which allows checking satisfaction in almost the same manner, i.e., we investigate the possible states of the system instead of all possible trajectories. An alternative definition is given in [DMR16], which is not that conservative and allows to prove satisfaction based on runtime information: As $\varphi^{env}$ is a simple time-invariant safety property, we might use any form of model checking or reachability analysis to obtain a certificate stating that a hybrid automaton $\mathcal{A}$ is a valid instantiation of the environment specified by $\varphi^{env}$. One way of proving this is the method using level sets in combination with reachability analysis as shown in Section 4.3 and [HMT15].

In the following, we will define satisfaction of an environment predicate for a (environmental) run, such that satisfaction of an environment predicate for an automaton implies that all runs of a hybrid I/O automata satisfy the environment predicate.

**Definition 5.4 (Runs of an Environmental Predicate)**
*Let $Var = Var^L \uplus Var^I \uplus Var^O$ be a set of variables. An* environment run *on $Var^I$ is a – possibly infinite – sequence $\left(\mathbf{x}_i^I\right)$ for which holds*

- $\mathbf{x}_i^I : [t_i, t_{i+1}] \mapsto \mathcal{X}^I$ *(or $\mathbf{x}_i^I : [t_i, t_{i+1}) \mapsto \mathcal{X}^I$, in case $t_{i+1} = \infty$), is a function which is continuous on $[t_i, t_{i+1}]$ (or $[t_i, t_{i+1})$ in case $t_{i+1} = \infty$),*
- *$(t_i)$ is a non-decreasing sequence of switching times where either both $(t_i)$ and $\left(\mathbf{x}_i^I\right)$ are infinite or $(t_i)$ is one element longer than $\left(\mathbf{x}_i^I\right)$.*

*We say a run – not necessarily an environment run – $(\mathbf{x}_i)$* satisfies an environmental predicate $\varphi^{env} = \varphi^{env}_{inv} \wedge (\neg \mathsf{flow} \Rightarrow \varphi^{env}_{dscr}) \wedge (\mathsf{flow} \Rightarrow \varphi^{env}_{flow})$ *on $Var^L \uplus Var^I = Var^E \subseteq Var$ denoted by $\forall t \bullet \mathbf{x}(t) \models \varphi^{env}$ if and only if*

- *for all $t_i \leq t < t_{i+1}$ holds $\mathbf{x}_i(t) \models \varphi^{env}_{inv}$,*
- *for almost all $t_i \leq t < t_{i+1}$ holds $\dot{\mathbf{x}}_i^I(t), \mathbf{x}_i^I(t) \models \varphi^{env}_{flow}$,*
- *for all $t_i$, $i > 0$ but the last element of $(t_i)$ holds $\mathbf{x}_i^I(t_i) \models \varphi^{env}_{dscr}[\mathbf{x}^{I\prime}/\mathbf{x}^I_{i+1}(t_i)]$.*

♢

**Note 5.5**
*Note, that an environment predicate does not restrict the flow if and only if $\varphi^{env}_{flow}$ is* `true` *and similarly does not restrict the discrete updates if and only if $\varphi^{env}_{dscr}$ is* `true`. ◁

**Proposition 5.6**
*Note, that if an automaton satisfies an environmental predicate, then each trajectory of the automaton will satisfy the predicate.*

If the discrete updates allowed by the environmental predicate are unique, i. e., for any $\mathbf{x}^I \in \mathcal{X}^E$ there is exactly one $\mathbf{x}^{I\prime}$ such that $\mathbf{x}^{I\prime}, \mathbf{x}^I \models \varphi^{env}_{dscr}$, then we can construct a hybrid automaton that is as permissive as the environmental predicate.

**Definition 5.7 (Induced Environmental Hybrid Automaton)**
*Let $\mathcal{A} = (Loc, Var, Trans, Flow, Inv, Inits)$ be a hybrid I/O automaton with $Var = Var^L \uplus Var^I \uplus Var^O$ and let $\varphi^{env} = \varphi^{env}_{inv} \wedge (\neg \mathsf{flow} \Rightarrow \varphi^{env}_{dscr}) \wedge (\mathsf{flow} \Rightarrow \varphi^{env}_{flow})$ be an environmental predicate on $Var^E = Var^L \uplus Var^I$. The* Induced Environmental Hybrid I/O Automaton *is a single-location hybrid I/O automaton*

$$\mathcal{A}_{\mathsf{env}} = (Loc_{\mathsf{env}}, Var_{\mathsf{env}}, Trans_{\mathsf{env}}, Flow_{\mathsf{env}}, Inv_{\mathsf{env}}, Inits_{\mathsf{env}})$$

*where*

- $Loc_{\mathsf{env}} := \{l_{\mathsf{env}}\}$ *is a singleton set of locations,*
- $Var_{\mathsf{env}} = Var^I$ *is the set of variables,*
- $\mathcal{X}_{\mathsf{env}} := \mathbb{R}^{|Var_{\mathsf{env}}|}$ *is the continuous state space,*
- $Trans_{\mathsf{env}}$ *is a finite set of transitions such that for every* $\mathbf{x}', \mathbf{x} \in \mathcal{X}_{\mathsf{env}}$ *holds if* $\mathbf{x}', \mathbf{x} \models \varphi^{env}_{dscr}$ *then there exists a transition* $(l_{\mathsf{env}}, G_{\mathsf{env}}, U_{\mathsf{env}}, l_{\mathsf{env}}) \in Trans_{\mathsf{env}}$ *where* $\mathbf{x} \in G_{\mathsf{env}} \subseteq \mathcal{X}_{\mathsf{env}}$ *and* $U_{\mathsf{env}}(\mathbf{x}) := \mathbf{x}' \in \mathcal{X}_{\mathsf{env}}$
- $Inv_{\mathsf{env}}$ *is an invariant function such that*
$$Inv_{\mathsf{env}}(l_{\mathsf{env}}) = \left\{ \mathbf{x}_{\downarrow Var_{\mathsf{env}}} \;\middle|\; \mathbf{x} \in \mathbb{R}^{|Var^E|} \wedge \mathbf{x} \models \varphi^{env}_{inv} \right\},$$
- $Flow_{\mathsf{env}}$ *is a flow function such that for all* $\mathbf{x} \in Inv_{\mathsf{env}}(l_{\mathsf{env}})$ *holds*
$$Flow_{\mathsf{env}}(l_{\mathsf{env}})(\mathbf{x}) = \{ \dot{\mathbf{x}} \in \mathcal{X}_{\mathsf{env}} \mid \dot{\mathbf{x}}, \mathbf{x} \models \varphi^{env}_{flow} \},$$
- $Inits_{\mathsf{env}} := \{l_{\mathsf{env}}\} \times Inv_{\mathsf{env}}(l_{\mathsf{env}})$ *is the set of initial states.*

$\diamondsuit$

It should be clear that any trajectory allowed by the environmental predicate can be generated by the environmental hybrid automaton.

**Corollary 5.8**
*The induced environmental hybrid automaton is at least as permissive as the environmental predicate. That is, each input trajectory that satisfies the environmental predicate* $\varphi^{env}$ *is a trajectory of* $\mathcal{A}_E$.

We define runs and trajectories of a HIOA under an environmental predicate as follows:

**Definition 5.9 (Runs of a Hybrid I/O Automaton under an Environmental Predicate)**
*Let* $\mathcal{A} = (Loc, Var, Trans, Flow, Inv, Inits)$ *be a hybrid I/O automaton, let* $\varphi^{env} = \varphi^{env}_{inv} \wedge (\neg \mathsf{flow} \Rightarrow \varphi^{env}_{dscr}) \wedge (\mathsf{flow} \Rightarrow \varphi^{env}_{flow})$ *be an environmental predicate on* $Var^E$ *and let* $(t_i)$ *be a – possibly infinite – sequence of* switching times *for which holds*

- $t_i \in Time_\infty$ *with* $Time_\infty = \mathbb{R}_{\geq 0} \cup \{\infty\}$,
- $t_0 = 0$,
- $\forall i > 0 \bullet t_{i-1} \leq t_i$,
- *at most the last element of the sequence might be* $\infty$.

*A* run *of $\mathcal{A}$ is a – possibly infinite – sequence of tuples $(\pi_i) = (l_i, \mathbf{x}_i)$ for which holds*

- *$l_i \in Loc$ is a location of $\mathcal{A}$,*
- *$\mathbf{x}_i : [t_i, t_{i+1}] \mapsto \mathcal{X}$ (or $\mathbf{x}_i : [t_i, t_{i+1}) \mapsto \mathcal{X}$, in case $t_{i+1} = \infty$), is a function which is continuous on $[t_i, t_{i+1}]$ (or $[t_i, t_{i+1})$ in case $t_{i+1} = \infty$),*
- *$(t_i)$ is the sequence of switching times where either both $(t_i)$ and $(\pi_i)$ are infinite or $(t_i)$ is one element longer than $(\pi_i)$,*

*that also satisfies*

1. *$\pi_0 = (l_0, \mathbf{x}_0(0)) \in Inits$ and $\mathbf{x}_0^I(0) \models \varphi_{inv}^{env}$,*
2. *for all $t_i \leq t < t_{i+1}$ holds $\mathbf{x}_i(t) \in Inv(l_i)$ and $\mathbf{x}_i(t) \models \varphi_{inv}^{env}$,*
3. *for almost all $t_i \leq t < t_{i+1}$ holds $\dot{\mathbf{x}}_i^C(t) \in Flow(l_i)(\mathbf{x}_i(t))$ and $\dot{\mathbf{x}}_i^I(t), \mathbf{x}_i^I(t) \models \varphi_{flow}^{env}$,*
4. *for all $t_i$ of $(t_i)$, but the first and the last element, there either exists a transition $(l_i, G, U, l_{i+1}) \in Trans$ with*
   *a) $\lim_{t \to t_{i+1}} \mathbf{x}_i(t) \in G$ and*
   *b) the automaton performed an update*

   $$\mathbf{x}_{i+1}^C(t_i) = U(\mathbf{x}_i(t_i)) \wedge \mathbf{x}_{i+1}^I(t_i) = \mathbf{x}_i^I(t_i)$$

   *or (no transition and) the environment performed an update in which case*

   $$\mathbf{x}_{i+1}^C(t_i) = \mathbf{x}_i^C(t_i) \wedge \mathbf{x}_i^I(t_i) \models \varphi_{dscr}^{env}[\mathbf{x}^{I\prime}/\mathbf{x}_{i+1}^I(t_i)]$$

5. *if $(t_i)$ is infinite, then $\lim_{i \to \infty} t_i = \infty$.*

*We call the run* infinite *if $(t_i)$ is infinite and diverges to infinity or $(t_i)$ is finite and its last element is $\infty$. We call the run* finite *otherwise. The sequence $(l_i)$ is called the* location sequence*.* ♢

### 5.1.1 Plants and Controllers

As already mentioned, in our settings we assume that there is a distinct common plant which is controlled by a multitude of controllers. We define the plant and controllers as follows.

**Definition 5.10 (Plant [DMR16])**
*A* plant *is a lock-free HIOA $P$ of the form*

$$P = (Loc_P, Var_P, Trans_P, Flow_P, Inv_P, Inits_P)$$

*where* $Trans_{\mathsf{P}}$ *updates only* $Var^{L}_{\mathsf{P}}$.

*For* $Var^{O}_{\mathsf{P}}$, *we also write* $\mathsf{Sens}$, *which are the* sensors *of the plant. The set* $Var^{I}_{\mathsf{P}}$ *consists of disjoint sets,* $\mathsf{Act} \uplus \mathsf{Disturb}$, actuators *and* disturbances. ◊

**Definition 5.11 (Controller [DMR16])**
*A* controller *is a HIOA* $\mathsf{C}$ *of the form*

$$\mathsf{C} = (Loc_{\mathsf{C}}, Var_{\mathsf{C}}, Trans_{\mathsf{C}}, Flow_{\mathsf{C}}, Inv_{\mathsf{C}}, Inits_{\mathsf{C}}).$$

*We call* $Var^{I}_{\mathsf{C}} \cap \mathsf{Sens}$, *the* sensors read by a controller *and* $Var^{O}_{\mathsf{C}} \cap \mathsf{Act}$, *the* actuators driven by a controller. ◊

In our sense, *lock-free* means that an automaton might not stall time for a future time horizon of $\Delta^{time}_{\mathsf{lat}} > 0$ from any reachable state because we are aiming at a practical framework where we are interested only in processes where time always evolves.[1]

Definition 5.26 in Section 5.3.2 intentionally defines control components in a way that syntactical restrictions prevent direct communication between controllers. This means that two controllers cannot communicate via shared variables and, instead, they have to use communication channels, that may only transmit limited information potential even with a certain delay. This is motivated by (1) the assumption that signals may have a latency, (2) the fact that communication channels are clocked in practice, and (3) the observation that sane controllers should not update their discrete-valued outputs arbitrarily fast. Likewise, inertia of physical models usually prevent instantaneous responses as well as dramatic environmental changes. Therefore, we assume the existence of

- a minimal latency $\Delta^{time}_{\mathsf{lat}}$ between discrete actions like sending or receiving events and
- a minimal persistence time $\Delta^{\mathsf{sep}}$ for which a signal remains unchanged.

To not prevent communication at all or forcing engineers to encode their communication protocol into continuous variables and have the plant acting as some form of relay, we equip them with predefined but abstract communication channels that already capture physical effects. Our communication channels are binary but can be easily extended to any finite domain.

**Definition 5.12 (Communication Channel)**
*A* communication channel for $Var^{Chnl}$ *is a HIOA* $\mathsf{Com}$ *of the form*

$$\mathsf{Com} = (Loc_{\mathsf{Com}}, Var_{\mathsf{Com}}, Trans_{\mathsf{Com}}, Flow_{\mathsf{Com}}, Inv_{\mathsf{Com}}, Inits_{\mathsf{Com}})$$

*where*

- $Var^{Chnl}$ *is a finite set of boolean communication variables.*

[1] More details can be found in [DMR16].

- $\mathit{Var}_{\mathsf{Com}} = \mathit{Var}^{L}_{\mathsf{Com}} \uplus \mathit{Var}^{I}_{\mathsf{Com}} \uplus \mathit{Var}^{O}_{\mathsf{Com}}$,
- $\mathit{Var}^{I}_{\mathsf{Com}} = \left\{ v^{I}_{\mathsf{Com}} \mid v \in \mathit{Var}^{\mathit{Chnl}} \right\}$ *is a finite set of* input communication variables*,*
- $\mathit{Var}^{O}_{\mathsf{Com}} = \left\{ v^{O}_{\mathsf{Com}} \mid v \in \mathit{Var}^{\mathit{Chnl}} \right\}$ *is a finite set of* output communication variables*,*
- *$\mathsf{Com}$ is lock-free,*
- *output communication variables are boolean and only changed via transitions, i. e., for every output communication variable $v^{O}_{\mathsf{Com}} \in \mathit{Var}^{O}_{\mathsf{Com}}$ holds*

  $$\mathsf{Com} \models (\mathit{flow} \Rightarrow \dot{v}^{O}_{\mathsf{Com}} = 0) \wedge v \in \{0, 1\},$$

- *the initial hybrid states $\mathit{Inits}_{\mathsf{Com}}$ encode that initially no output communication variable $v^{O}_{\mathsf{Com}} \in \mathit{Var}^{O}_{\mathsf{Com}}$ is set, i. e., $\forall v^{O}_{\mathsf{Com}} \in \mathit{Var}^{O}_{\mathsf{Com}} \bullet v^{O}_{\mathsf{Com}} = 0$ and do not restrict the valuations of the input communication variables $v^{I}_{\mathsf{Com}} \in \mathit{Var}^{I}_{\mathsf{Com}}$,*
- *an output communication variables is set only if its corresponding input communication variable has been set before, i. e., for every trajectory $\mathbf{x}(\cdot)$ holds*

  $$\begin{aligned}\forall v \in \mathit{Var}^{\mathit{Chnl}} \bullet \forall t_{\mathsf{rcv}} \bullet \exists \delta_{lat} \in [\Delta^{Min}_{lat}, \Delta^{Max}_{lat}] \bullet \\ \mathsf{Com} \models \mathrm{rise}(v^{O}_{\mathsf{Com}}, t_{\mathsf{rcv}}) \Rightarrow \mathrm{rise}(v^{I}_{\mathsf{Com}}, t_{\mathsf{rcv}} - \delta_{lat})\end{aligned}$$

  *where* $\mathrm{rise}(v, t)$ *is a predicate indicating that there is a rising edge on the variable $v$ which is essentially a change of the valuation from* 0 *to* 1 *due to a discrete transition firing,*

- *if an output communication variable is changed, then it remains unchanged for at least $\Delta^{\mathsf{pers}} > 0$ time, i. e., for every trajectory $\mathbf{x}(\cdot)$ of $\mathsf{Com}$ and every output communication variable $v^{O}_{\mathsf{Com}} \in \mathit{Var}^{O}_{\mathsf{Com}}$ holds*

  $$\forall t_{\mathsf{snd}} \bullet \mathrm{change}(v, t_{\mathsf{snd}}) \Rightarrow \forall t \in [0, \Delta^{\mathsf{pers}}] \bullet \mathbf{x}(t_{\mathsf{snd}} + t)_{\downarrow v^{O}_{\mathsf{Com}}} = \mathbf{x}(t_{\mathsf{snd}})_{\downarrow v^{O}_{\mathsf{Com}}}$$

  *where* $\mathrm{change}(v, t)$ *is a predicate indicating that there is a rising edge or a falling edge on the variable $v$ which is essentially a change of the valuation from* 0 *to* 1 *(or vice versa due) to a discrete transition firing.*

$\diamondsuit$

We do not enforce a specific implementation as long as it satisfies the above constraints. An exemplary implementation of a hybrid automaton satisfying the above definition is given in [DMR16].

**Note 5.13**
*The definition of a communication channel allows binary communication for simplicity. However, using multiple communication channels on can communicate different messages, one could even encode numbers.* ◁

To reason about the parallel composition of controllers, plant, and communication channels modeled as hybrid I/O automata, we define parallel composability and parallel composition for HIOAs.

**Definition 5.14 (Parallel Composability of Hybrid I/O Automata)**
*Let $\mathcal{A}_1, \mathcal{A}_2$ be two hybrid I/O automata with*

$$\mathcal{A}_i = (Loc_i, Var_i, Trans_i, Flow_i, Inv_i, Inits_i).$$

*$\mathcal{A}_1$ and $\mathcal{A}_2$ are* parallel composable *if and only if*

- $Var_1^L \cap Var_2^L = \emptyset$ *and*
- $Var_1^O \cap Var_2^O = \emptyset$.

♢

**Definition 5.15 (Parallel Composition of Hybrid I/O Automata [DMR16])**
*Let $\mathcal{A}_1, \mathcal{A}_2$ be two parallel composable hybrid I/O automata with*

$$\mathcal{A}_i = (Loc_i, Var_i, Trans_i, Flow_i, Inv_i, Inits_i).$$

*The* parallel composition $\mathcal{A}_{1\|2} = \mathcal{A}_1 \parallel \mathcal{A}_2$ *is the hybrid I/O automaton*

$$\mathcal{A}_{1\|2} := \left(Loc_{1\|2}, Var_{1\|2}, Trans_{1\|2}, Flow_{1\|2}, Inv_{1\|2}, Inits_{1\|2}\right)$$

*where*

- *the locations are the Cartesian product of the locations of the sub-HIOAs* $Loc_{1\|2} := Loc_1 \times Loc_2$
- *the variables are the union of variables* $Var_{1\|2} := Var_1 \cup Var_2 = Var_{1\|2}^I \uplus Var_{1\|2}^L \uplus Var_{1\|2}^O$ *with*
  - *output variables being the disjoint union of the individual output variables, i. e.,* $Var_{1\|2}^O := Var_1^O \uplus Var_2^O$,
  - *input variables being the union of the individual input variables without those which are driven by output variables of the other HIOA, i. e.,* $Var_{1\|2}^I := \left(Var_1^I \cup Var_2^I\right) \setminus Var^O$,

  - *local variables being the union of local variable plus the input variables of a HIOA that are driven by the output variable of the other HIOA, i. e.,* $Var^L_{1\|2} := \left(Var^L_1 \uplus Var^L_2\right) \cup \left(\left(Var^I_1 \cup Var^I_2\right) \cap Var^O\right)$,

- *the state space is the union of the state spaces* $\mathcal{X}_{1\|2} := \mathbb{R}^{|Var|} = \mathbb{R}^{|Var_1 \cup Var_2|}$ *and the controlled state space is* $\mathcal{X}^C_{1\|2} := \mathbb{R}^{|Var^C|} = \mathbb{R}^{|Var^C_1 \uplus Var^C_2|}$,

- *the transitions are the interleaved transitions*

$$Trans_{1\|2} := \left\{ \left((l_1, l_2), G, U, (l'_1, l'_2)\right) \;\middle|\; \begin{array}{c} \exists i \in \{1,2\} \bullet (l_i, G_i, U_i, l'_i) \in Trans_i, \\ G = \left\{ \mathbf{x} \in \mathcal{X}_{1\|2} \;\middle|\; \mathbf{x}_{\downarrow Var_i} \in G_i \right\} \\ U(\mathbf{x}) = \left(U_i(\mathbf{x}_{\downarrow Var_i}), \mathbf{x}_{\downarrow Var^C \setminus Var^C_i}\right) \\ \forall j \bullet i \neq j \to l_j = l'_j, l_j \in Loc_j \end{array} \right\}$$

- *the continuous flow for each location* $l = (l_1, l_2) \in Loc_{1\|2}$ *is given by*

$$Flow_{1\|2}(l)(\mathbf{x}) := \left\{ \mathbf{y} \in \mathcal{X}^C_{1\|2} \;\middle|\; \bigwedge_i \mathbf{y}_{\downarrow Var^C_i} \in Flow_i(l_i)(\mathbf{x}_{\downarrow Var_i}) \right\}$$

- *the invariant for each location* $l = (l_1, l_2) \in Loc_{1\|2}$ *is given by*

$$Inv_{1\|2}(l) := \left\{ \mathbf{x} \in \mathcal{X}_{1\|2} \;\middle|\; \bigwedge_i \mathbf{x}_{\downarrow Var_i} \in Inv_i(l_i) \right\}$$

- *the set of initial states is given by*

$$Inits_{1\|2} := \left\{ ((l_1, l_2), \mathbf{x}) \in Loc \times \mathcal{X} \;\middle|\; \bigwedge_i (l_i, \mathbf{x}_{\downarrow Var_i}) \in Inits_i \right\}$$

*The state of the parallel composed system is the tuple* $\pi = (l, \mathbf{x})$ *where* $l \in Loc_{1\|2}$ *and we might think of the vector* $\mathbf{x} \in \mathcal{X}_{1\|2}$ *as*

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_{\downarrow Var^I_1 \setminus Var^O_2} \\ \mathbf{x}_{\downarrow Var^I_2 \setminus Var^O_1} \\ \mathbf{x}_{\downarrow Var^I_1 \cap Var^O_2} \\ \mathbf{x}_{\downarrow Var^I_2 \cap Var^O_1} \\ \mathbf{x}_{\downarrow Var^L_1} \\ \mathbf{x}_{\downarrow Var^L_2} \\ \mathbf{x}_{\downarrow Var^O_1} \\ \mathbf{x}_{\downarrow Var^O_2} \end{bmatrix} = \begin{bmatrix} \Bigg\} \mathbf{x}_{\downarrow Var^I_{1\|2}} \\ \left.\begin{matrix} \\ \\ \\ \\ \end{matrix}\right\} \mathbf{x}_{\downarrow Var^L_{1\|2}} \\ \Bigg\} \mathbf{x}_{\downarrow Var^O_{1\|2}} \end{bmatrix}.$$

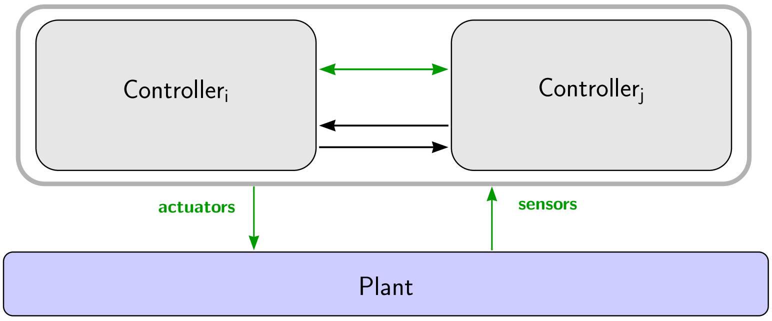


Figure 5.4: Sequential composition: controllers may pass around the responsibility of controlling the plant.

*The parallel composition is only* well-defined *if each local variable and each output variable is controlled by exactly one HIOA.* ◇

Now, we can successively construct a controller for a plant by composing controllers and channels until the plant has no more open actuators. This is called a closed loop and we can apply the usual techniques to prove stability or safety.

**Definition 5.16 (Closed Loop [DMR16])**
*We refer to the parallel composition $\mathcal{A} = C \parallel P$ of a controller $C$ and a plant $P$ as a* closed loop *if and only if $\mathcal{A}$ has no input variables, i. e., $Var^I = \emptyset$.* ◇

Since our goal is the analysis of *open loop* systems and the compositional derivation of properties such as stability, we will adapt GAS to our settings in the next sections.

## 5.2 State-of-the-Art: Sequential Composition

In [Dam+10] Damm et al. proposed a library-based design methodology for constructing hybrid controllers. Interfaces and interface satisfaction were defined to support cataloging components for later reuse. The importance of contract-based compositional design — as, e. g., imposed by interface satisfaction — has long been recognized as an instrument to reduce the number of late integration errors [SDP12] and to boost re-use of components [Dam+05].

The sequential composition framework defines *transition composition*, a way of sequential orchestration of controllers, that preserves safety and stability properties. Verification conditions were identified to alleviate the application of automatic verification tools. The idea is that each controller is controlling the plant in a different operation range like close to, far from, or very far from the equilibrium point. Each controller might then implement a specialized strategy to achieve the common safety and stability objectives on the plant. Here, interfaces are used to annotate (1) the plant for which the controller is designed, (2) the operation range of the plant which is covered by the controller, (3) the safety and stability objectives which are addressed by the controller, and (4) the conditions in which control may be transferred to (via an *entry port*) and from (via an *exit port*) this controller. Figure 5.4 sketches the system architecture.

Exploiting only knowledge annotated at the interfaces of the individual controllers, the transition composition ensures that

(Req 1) the union individual controller's operation range covers the overall operation range of the plant and

(Req 2) switching from one controller to another does not prevent achieving the common objectives.

Satisfaction of global safety objectives can be checked compositionally — at controller level — by locally inspecting all possible trajectories emanating from the controller's entry regions (as specified by the entry ports) and by additionally ensureing that — during transition composition — components are only entered via their entry ports. In contrast, to allow compositional derivation of stability properties, the framework uses extra interface annotations for the entry and exit ports. The basic idea is to annotate Lyapunov functions at the interface and check their compatibility during the transition composition. Since these Lyapunov functions might refer to local variables, that are hidden at the component's interface, so called *Lyapunov function projections (LFP)* are introduced. These Lyapunov function projections are projections of Lyapunov functions onto the set of visible (input and output) variables. Let $\mathsf{C}$ be a controller, let $\mathsf{P}$ be the plant associated with the controller, let $V$ be a global Lyapunov function for $\mathsf{C} \parallel \mathsf{P}$, and let *entry* (resp. *exit*) be an entry (resp. exit) port. Further let $\varphi^{entry}$ describe the set of legal entry states and $\varphi^{exit}$ be the set of reachable exit states for the particular port. We denote by $l_{entry}, l_{exit}$ locations corresponding to the entry port *entry* and exit port *exit*. Consequently, $Vl_{entry}, \cdot$ and $Vl_{exit}, \cdot$ are the Lyapunov functions associated with the locations corresponding to the entry and exit ports, respectively. A Lyapunov function projection $V_{entry}$ for port *entry* has to satisfy

$$\forall \mathbf{x} \in Inv(l_{entry}) \bullet (\mathbf{x}_{\downarrow Var^I \cup Var^O} \models \varphi^{entry}) \Rightarrow V(l_{entry}, \mathbf{x}) \leq V_{entry}\left(\mathbf{x}_{\downarrow Var^I \cup Var^O}\right),$$

which roughly asks the Lyapunov function projections to overapproximate the local Lyapunov function of the location associated with the entry port. The constraint for exit ports *exit* can be obtained by replacing *entry* with *exit* and inversing $\leq$, that is, the location's Lyapunov function's values have to be greater than the LFP's values, or formally

$$\forall \mathbf{x} \in Inv(l_{exit}) \bullet (\mathbf{x}_{\downarrow Var^I \cup Var^O} \models \varphi^{exit}) \Rightarrow V_{exit}\left(\mathbf{x}_{\downarrow Var^I \cup Var^O}\right) \leq V(l_{exit}, \mathbf{x}),$$

which roughly asks the Lyapunov function projections to underapproximate the local Lyapunov function of the location associated with the exit port. If both constraints are satisfied — which is required by the definition of interface satisfaction — and there is a trajectory emanating from $\varphi^{entry}$ reaching $\varphi^{exit}$, then it holds, that

$$\forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^{\left|Var^I \cup Var^O\right|} \bullet (\mathbf{x} \models \varphi^{entry}) \wedge (\mathbf{y} \models \varphi^{exit}) \Rightarrow V_{exit}(\mathbf{y}) \leq V_{entry}(\mathbf{x})$$

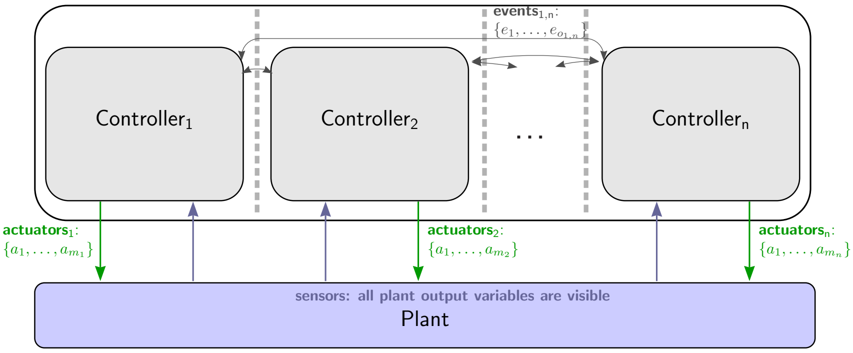

events$_{1,n}$:
$\{e_1, \ldots, e_{o_{1,n}}\}$
Controller$_1$
Controller$_2$
…
Controller$_n$
actuators$_1$:
$\{a_1, \ldots, a_{m_1}\}$
actuators$_2$:
$\{a_1, \ldots, a_{m_2}\}$
actuators$_n$:
$\{a_1, \ldots, a_{m_n}\}$
sensors: all plant output variables are visible
Plant

Figure 5.5: Application class: the controllers are loosely coupled, directly communicate via events (black arrows), may share the sensors (blue arrows), and have separate sets of actuators (green arrows).

because the value of the Lyapunov function $V$ decreases along every trajectory, i. e., $V(l_{exit}, \mathbf{x}) \leq V(l_{entry}, \mathbf{x})$. During the transition composition exit ports are connected to entry ports. That way the control is passed from one controller via its exits port to the next controller via its entry port. Considering the trajectories, this can be viewed as gluing snippets together. In order to conclude stability of the composed system the transition operator has to ensure that there is a global Lyapunov for the composed system such that this Lyapunov function's values always decreases. This boils down to a simple check: An entry port of one controller is allowed to be connected to the exit port of another controller if

$$\forall \mathbf{x} \in \mathbb{R}^{|Var^I \cup Var^O|} \bullet (\mathbf{x} \models \varphi^{exit}) \Rightarrow V_{entry}(\mathbf{x}) \leq V_{exit}(\mathbf{x}).$$

Together, these relations between exit and entry ports or rather their LFP's values allow us to conclude that for any trajectory of the sequentially composed controller and its plant, there is a Lyapunov function whose value decreases.

The described technique allows us to construct large controllers out of small subcontrollers. However, transition composition requires a controller to be capable of controlling all aspects of the plant. For plants whose control objectives are not only complex but also plenty, we need another composition operator called *parallel composition* which we will introduce in the following sections.

## 5.3 Extension to Parallel Composition

Targeting loosely coupled systems, we propose a parallel composition operator in this section. Even though we do not have a formal definition of the term "loosely coupled", we think of it as:

> "A controller C on its plant P is *loosely coupled* if and only if the control objectives can be achieved with few or no knowledge of other involved controllers and the environment."

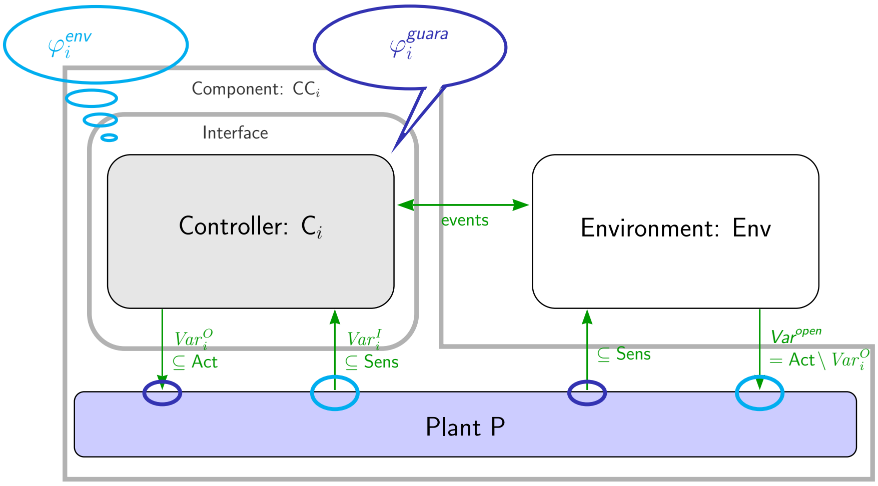

$\varphi_i^{env}$
$\varphi_i^{guara}$
Component: $\mathsf{CC}_i$
Interface
Controller: $\mathsf{C}_i$
events
Environment: Env
$Var_i^O$ $\subseteq$ Act
$Var_i^I$ $\subseteq$ Sens
$\subseteq$ Sens
$Var^{open}$ $=$ Act $\setminus Var_i^O$
Plant P

Figure 5.6: Analyzed in isolation: assumptions and guarantees define deployment context and the component's service.

Figure 5.5 sketches the application class. We make the following assumptions on the application domain which mainly restrict communication and visibility of a controller's environment:

(Asm 1) every controller can read all sensors of the plant but neither the local variables of the plant nor the plants inputs,

(Asm 2) every controller has its private set of actuators and no other controller might drive these actuators or read their values,

(Asm 3) every controller has its private set of local variables and no other controller might drive or read their values,

(Asm 4) controllers might communicate via private channels in a unicast fashion and the communication is neither synchronous nor reliable (packet may be late or get lost),

(Asm 5) controllers might state assumptions in form of predicates on the environment (deployment context) which describe under which environmental conditions they are able to achieve their objectives (service), and

(Asm 6) for simplicity, we assume clocks of the individual controller to be synchronized.

The above assumptions allow us to employ an assume-guarantee principle as visualized in Figure 5.6. The full details — including interface definitions and how to handle assumptions and guarantees — can be found in [DMR16]. This section is focused on the contribution of this thesis's author which is deriving stability proofs (1) for subcontrollers in isolation and (2) for composed controllers.

We consider establishing two kinds of stability notions

**event-unbounded stability** a controller will always achieve a certain stability objective as long as the controllers assumptions are satisfied.

**event-bounded stability** the controller will achieve a certain stability objective during periods of time where the environment satisfies an extra condition, that is a promise communicated via events.

Using these two notions, we can describe the service of a controller on the plant in its assumed environment either independent of other controllers or dependent on other controllers. If the service depends on other controllers, then event-based communication is assumed to negotiate cooperation, e. g., a suitable equilibrium point.

> **Example 5.17**
> *Consider a comfort system, a safety-critical system, and an energy generator where the comfort system and the safety-critical system draw energy and the energy supplied by the generator is predictable but changing. Assuming that the energy is always enough for the safety-critical system, we might assign priorities to the two energy consuming systems in order to have the safety-critical system always running. Traditionally, we cannot guarantee any service of the comfort system without extra information. However, we can guarantee — by event-bounded stability — that whenever the energy generator supplies enough energy (communicated via events) for a suitable period of time, then the comfort system achieves its service.*
>
> *Interestingly, this local property of the comfort system can be composed with an energy generation profile of the generator, to obtain knowledge on the combined system.* ◁

Note, that the first notion, event-unbounded stability, can be seen as a special case of the second where the period covers the full time domain. To capture these two notions, we define two properties that a system might establish. For the first notion, we define global asymptotic stability under assumptions (GAS-Asm) and for the second notion, we define conditional global asymptotic stability (conGAS).

In this section, we show that given two components $\mathsf{CC}_1, \mathsf{CC}_2$ if both $\mathsf{CC}_i$ are GAS-Asm with respect to $Var_i^S$, then its composite $\mathsf{CC}_{1\|2}$ is GAS-Asm with respect to $Var_1^S \cup Var_2^S$. That is, parallel composition preserves this form of stability. Additionally, we show that one can construct composite Lyapunov functions for the composed component $\mathsf{CC}_{1\|2}$ from the Lyapunov functions of the sub-components $\mathsf{CC}_i$.

### 5.3.1 Global Asymptotic Stability in an Environment

To cope with inputs and outputs, we introduce GAS-Asm which is global asymptotic stability under assumptions. Roughly speaking, an invariant assumption — a coarse abstraction of the remainder of the system — is used to “close” the open loop system and then apply the usual techniques to analyze a closed loop systems with tools like Stabhyli (see [MT13a] or Chapter 3). As we will see, it is indeed one method to construct an hybrid automaton from a given environmental assumption and to obtain a closed loop system by the parallel composition of the hybrid I/O automata. This closed loop system can then be used to obtain Lyapunov functions and therefore prove GAS-Asm. Another method — which we will focus on — is to adapt the Lyapunov theorem to incorporate the assumption directly.

**Definition 5.18 (Global Asymptotic Stability under Assumptions.)**
*Let*

$$\mathcal{A} = (\mathit{Loc}, \mathit{Var}, \mathit{Trans}, \mathit{Flow}, \mathit{Inv}, \mathit{Inits})$$

*be a hybrid I/O automaton, let* $\varphi^{env} = \varphi^{env}_{inv} \wedge (\neg \mathsf{flow} \Rightarrow \varphi^{env}_{dscr}) \wedge (\mathsf{flow} \Rightarrow \varphi^{env}_{flow})$ *be an environment predicate on* $\mathit{Var}^E$*, and let* $\mathit{Var}^S \subseteq \mathit{Var}^C$ *be a subset of variables required to converge to the equilibrium point* $\mathbf{0} \in \mathbb{R}^{|\mathit{Var}^S|}$*. A continuous-time dynamic system* $\mathcal{A}$ *is called* Lyapunov stable with respect to $\mathit{Var}^S$ under assumptions $\varphi^{env}$ (LS-Asm)*, if for all functions* $\mathbf{x}(\cdot)$ *with* $\forall t \bullet \mathbf{x}(t) \models \varphi^{env}$*, holds*

$$\forall \epsilon > 0 \bullet \exists \delta > 0 \bullet \forall t \geq 0 \bullet ||\mathbf{x}(0)|| < \delta \Rightarrow \left|\left|\mathbf{x}^S(t)\right|\right| < \epsilon.$$

$\mathcal{A}$ *is called* globally attractive with respect to $\mathit{Var}^S$ under assumptions $\varphi^{env}$ (GA-Asm)*, if for all functions* $\mathbf{x}(\cdot)$ *with* $\forall t \bullet \mathbf{x}(t) \models \varphi^{env}$*,*

$$\lim_{t \to \infty} \mathbf{x}^S(t) = \mathbf{0}, \quad \textit{i.e.,}\ \forall \epsilon > 0 \bullet \exists t_e \geq 0 \bullet \forall t > t_e \bullet \left|\left|\mathbf{x}^S(t)\right|\right| < \epsilon.$$

$\mathbf{x}^S$ *refers to the projection of* $\mathbf{x}_{\downarrow \mathit{Var}^S}$*. A system is* globally asymptotically stable wrt. $\mathit{Var}^S$ under assumptions $\varphi^{env}$ (global asymptotic stability under assumptions) *if and only if it is, both, Lyapunov stable wrt.* $\mathit{Var}^S$ *under assumption* $\varphi^{env}$ *and globally attractive wrt.* $\mathit{Var}^S$ *under assumption* $\varphi^{env}$*.* ♢

GAS-Asm is a generalized version of GAS that allows a hybrid automaton to have inputs and outputs and requires the trajectories to converge in case the automaton's environment (the valuation of the input variables) behaves according to an environmental predicate $\varphi^{env}$. The special case of GAS for hybrid automata without inputs and outputs can be obtained by setting $\varphi^{env}$ to `true`.

Next, we present two methods to prove GAS-Asm. As mentioned above, one method is to construct an induced environmental hybrid automaton for the environmental predicate and prove GAS for the parallel composition of the automata. This yields the following lemma.

**Lemma 5.19**
*Let*

$$\mathcal{A} = (\mathit{Loc}, \mathit{Var}, \mathit{Trans}, \mathit{Flow}, \mathit{Inv}, \mathit{Inits})$$

*be a hybrid I/O automaton, let* $\varphi^{env} = \varphi^{env}_{inv} \wedge (\neg \mathsf{flow} \Rightarrow \varphi^{env}_{dscr}) \wedge (\mathsf{flow} \Rightarrow \varphi^{env}_{flow})$ *be an environmental predicate on* $\mathit{Var}^E$*, and let* $\mathit{Var}^S \subseteq \mathit{Var}^C$ *be a subset of variables required to converge to the equilibrium point* $\mathbf{0} \in \mathbb{R}^{|\mathit{Var}^S|}$*. Further, let*

*$\mathcal{A}_{\mathsf{env}} = \big(l_{\mathsf{env}}, \mathit{Var}^{I}, \emptyset, \mathit{Trans}_{\mathsf{env}}, \mathit{Flow}_{\mathsf{env}}, \mathit{Inv}_{\mathsf{env}}, \mathit{Inits}_{\mathsf{env}}\big)$ be the induced environmental hybrid automaton. If the closed loop $\mathcal{A} \parallel \mathcal{A}_{\mathsf{env}}$ is GAS, then $\mathcal{A}$ is GAS-Asm.*

**Proof.**
*The definition of GAS-Asm constrains only those trajectories for which $\varphi^{env}$ always holds to be GAS. Due to Corollary 5.8, all trajectories that always satisfy $\varphi^{env}$ are essentially the trajectories of $\mathcal{A}_{\mathsf{env}}$. Thus, if $\mathcal{A}_{\mathsf{env}} \parallel \mathcal{A}$ is GAS, then $\mathcal{A}$ is GAS-Asm.* □

The benefit of constructing an induced hybrid automaton is that if we obtain a closed loop, then we can use the usual Lyapunov theorem to prove GAS. Unfortunately, for example

- if the environmental predicate does not constrain some inputs, then the parallel composition does not yield a closed loop, in which case we cannot apply the above lemma and
- if the environmental predicate allows external discrete updates to infinitely many different values, then we would need infinitely many transitions in the automaton, in which case the automaton does not exists.

Although, it might be possible to work around the issues in some way, the second method — which we think is more practical — is to use a slightly modified version of the Lyapunov theorem which deals with assumption predicates directly. As GAS-Asm requires Lyapunov stability and global attractivity to hold only for trajectories that always satisfy the environmental predicate, we simply escalate this to the Lyapunov theorem by adding it as a premise to every constraint. This basically expresses that there is a function which serves as a Lyapunov function only for states (of trajectories) that satisfy the predicate. Note that, this adds the same form of pessimism like the original Lyapunov theorem as we do not check whether these states belong to actual runs of the automaton.

**Theorem 5.20 (Discontinuous Lyapunov Functions under Assumptions.)** *Let*

$$\mathcal{A} = (\mathit{Loc}, \mathit{Var}, \mathit{Trans}, \mathit{Flow}, \mathit{Inv}, \mathit{Inits})$$

*be a hybrid I/O automaton, let $\varphi^{env} = \varphi^{env}_{inv} \wedge (\neg \mathsf{flow} \Rightarrow \varphi^{env}_{dscr}) \wedge (\mathsf{flow} \Rightarrow \varphi^{env}_{flow})$ be an environmental predicate on $\mathit{Var}^E$, and let $\mathit{Var}^S \subseteq \mathit{Var}^C$ be a subset of variables required to converge to the equilibrium point $\mathbf{0} \in \mathbb{R}^{|\mathit{Var}^S|}$. If for each mode $l \in \mathit{Loc}$ there exists a set of variables $\mathit{Var}^{S_l}$ with $\mathit{Var}^S \subseteq \mathit{Var}^{S_l} \subseteq \mathit{Var}^C$ and there exists a continuously differentiable function $V_l : \mathcal{X} \to \mathbb{R}$ such that*

*(C1) for each $l \in \mathit{Loc}$, there exist two class-$\mathcal{K}^\infty$ functions $\alpha$ and $\beta$ such that*

$$\forall \mathbf{x} \in \mathit{Inv}(l) \bullet \mathbf{x} \models \varphi^{env}_{inv} \Rightarrow \alpha\big(||\mathbf{x}^{S_l}||\big) \leq V_l(\mathbf{x}) \leq \beta\big(||\mathbf{x}^{S_l}||\big),$$

*(C2) for each $l \in Loc$, there exists a class $\mathcal{K}^\infty$ function $\gamma$ such that*

$$\forall \mathbf{x} \in \mathit{Inv}(l) \bullet \mathbf{x} \models \varphi_{\mathit{inv}}^{\mathit{env}} \Rightarrow \dot{V}_l(\mathbf{x}) \leq -\gamma\left(\left|\left|\mathbf{x}^{S_l}\right|\right|\right)$$

$$\text{for each } \dot{V}_l(\mathbf{x}) \in \left\{ \left\langle \begin{bmatrix} \frac{\partial V_l(\mathbf{x})}{\partial \mathbf{x}^C} \\ \frac{\partial V_l(\mathbf{x})}{\partial \mathbf{y}^I} \end{bmatrix} \middle| \begin{bmatrix} f(\mathbf{x}) \\ \dot{\mathbf{y}}^I \end{bmatrix} \right\rangle \middle| \begin{array}{c} f(\mathbf{x}) \in \mathit{Flow}(l) \\ \wedge\, \dot{\mathbf{y}}^I, \mathbf{x}^I \models \varphi_{\mathit{flow}}^{\mathit{env}} \end{array} \right\},$$

*(C3) for each transition $(l, G, U, l') \in \mathit{Trans}$,*

$$\forall \mathbf{x} \in G, \mathbf{x}' \bullet (\mathbf{x} \models \varphi_{\mathit{inv}}^{\mathit{env}}) \wedge \mathbf{x}^{C'} = U(\mathbf{x}) \wedge \mathbf{x}^{I'} = \mathbf{x}^I \Rightarrow V_{l'}(\mathbf{x}') \leq V_l(\mathbf{x}),$$

*(C4) for each $l \in Loc$ for the external discrete updates*

$$\begin{aligned} \forall \mathbf{x}, \mathbf{x}' \in \mathcal{X} \bullet \quad & (\mathbf{x} \models \varphi_{\mathit{inv}}^{\mathit{env}}) \wedge \mathbf{x}^{C'} = \mathbf{x}^C \wedge \left(\mathbf{x}^{I'}, \mathbf{x}^I \models \varphi_{\mathit{dscr}}^{\mathit{env}}\right) \\ & \Rightarrow V_l(\mathbf{x}') \leq V_l(\mathbf{x}), \end{aligned}$$

*where $\mathbf{x}^S$ refers to the projection of $\mathbf{x}_{\downarrow \mathit{Var}^S}$ and $\mathbf{x}^{S_l}$ refers to the projection of $\mathbf{x}_{\downarrow \mathit{Var}^{S_l}}$, then $\mathcal{A}$ is global asymptotically stable with respect to $\mathit{Var}^S$ under assumptions $\varphi^{\mathit{env}}$ (GAS-Asm) and $V_l$ is called a* local Lyapunov function (LLF) *for location $l$, and the function $V(l, \mathbf{x}) = V_l(\mathbf{x})$ is called a* global Lyapunov function (GLF) *for $\mathcal{A}$.*

**Proof.**
*First note that for any trajectory $\mathbf{x}(\cdot)$ of $\mathcal{A}$ with $\forall t \bullet \mathbf{x}(t) \models \varphi^{\mathit{env}}$ it holds that*

$$0 \leq V(l(u), \mathbf{x}(u)) \leq V(l(0), \mathbf{x}(0)) + \int_0^u \dot{V}(l(t), \mathbf{x}(t))\, \mathrm{d}t \leq V(l(0), \mathbf{x}(0)),$$

*because the global Lyapunov function $V$ is non-negative, always decreasing for every (parallel) flow, non-increasing for each transition firing, and non-increasing for each external discrete update. Thus, the projection of the trajectory onto $\mathit{Var}^S$ is bounded by $\{\, \mathbf{y}^S \mid \exists l \in Loc \bullet V(l, \mathbf{y}) \leq V(l(0), \mathbf{x}(0)) \,\}$ and each sublevel set $L_c^-$ of $V$ is invariant with respect to $\mathit{Var}^S$.*

*Since the GAS-Asm constrains only trajectories $\mathbf{x}(\cdot)$ that satisfy $\varphi^{\mathit{env}}$, we fix an arbitrary trajectory which satisfies $\varphi^{\mathit{env}}$ and show that the trajectory is Lyapunov stable and globally attractive.*

***Lyapunov Stable:*** *We have to show that for each $\epsilon > 0$ there exists a $\delta > 0$ such that $\forall t \geq 0 \bullet ||\mathbf{x}(0)|| < \delta \Rightarrow \left|\left|\mathbf{x}^S(t)\right|\right| < \epsilon$. Given $\epsilon > 0$, select any $r \in ]0, \epsilon[$ and a corresponding $\epsilon$-ball*

$$\mathcal{B}_\epsilon = \left\{ \mathbf{y} \in \mathbb{R}^{\left|\mathit{Var}^S\right|} \,\middle|\, ||\mathbf{y}|| \leq r \right\},$$

*we can construct a level set* $L_c = \left\{ \mathbf{y}^S \mid \exists l \in Loc \bullet \mathbf{y} \in Inv(l) \land V(l,\mathbf{y}) = c \right\}$ *with*

$$0 < c < \inf\left( \bigcup_{l \in Loc} \left\{ V(l,\mathbf{y}) \mid \mathbf{y} \in Inv(l) \land \left|\left|\mathbf{y}^S\right|\right| = \epsilon \right\} \right)$$

*which lies in the interior of* $\mathcal{B}_\epsilon$ *and is bounded in all directions of* $Var^S$*. Now,* $0 < \delta = \inf\left(\left\{ \left|\left|\mathbf{y}^S\right|\right| \mid \mathbf{y}^S \in L_c \right\}\right)$ *corresponds to a* $\delta$*-ball inside* $L_c$*. Since* $L_c$ *is invariant, we know that any trajectory starting in the* $\delta$*-ball*

$$\mathcal{B}_\delta = \{ \mathbf{y} \mid \exists l \in Loc \bullet \mathbf{y} \in Inv(l) \land ||\mathbf{y}|| \leq \delta \},$$

*will not leave* $L_c$ *and hence will always stay in* $\mathcal{B}_\delta$*, i. e.,*

$$\mathbf{x}(0) \in \mathcal{B}_\delta \Rightarrow \mathbf{x}^S(0) \in L_c \Rightarrow \mathbf{x}^S(t) \in L_c \Rightarrow \mathbf{x}^S(t) \in \mathcal{B}_\epsilon.$$

*And therefore,*

$$\forall t \geq 0 \bullet ||\mathbf{x}(0)|| < \delta \Rightarrow \left|\left|\mathbf{x}^S(t)\right|\right| < \epsilon.$$

*The construction is visualized in Figure 5.7a.*

***Globally Attractive:*** *We have to show that for each* $\epsilon > 0$*, there exists a* $t_\epsilon \geq 0$ *such that* $\forall t > t_\epsilon \bullet \left|\left|\mathbf{x}^S(t)\right|\right| < \epsilon$*.*

*Again, given* $\epsilon > 0$*, we construct the* $\epsilon$*-ball* $\mathcal{B}_\epsilon$ *and the level set* $L_c$ *as above. Note, that if* $c_0 = 0$*, which corresponds to starting at the equilibrium point, it immediately holds that*

$$0 \leq V(l(u),\mathbf{x}(u)) \leq V(l(0),\mathbf{x}(0)) \leq c_0 = 0 \text{ for all } u \geq 0.$$

*Hence, we assume* $c_0 > 0$*. Further, we can construct a similar level set* $L_{c_0}$ *with* $c_0 = V(l(0),\mathbf{x}(0))$ *and the sets*

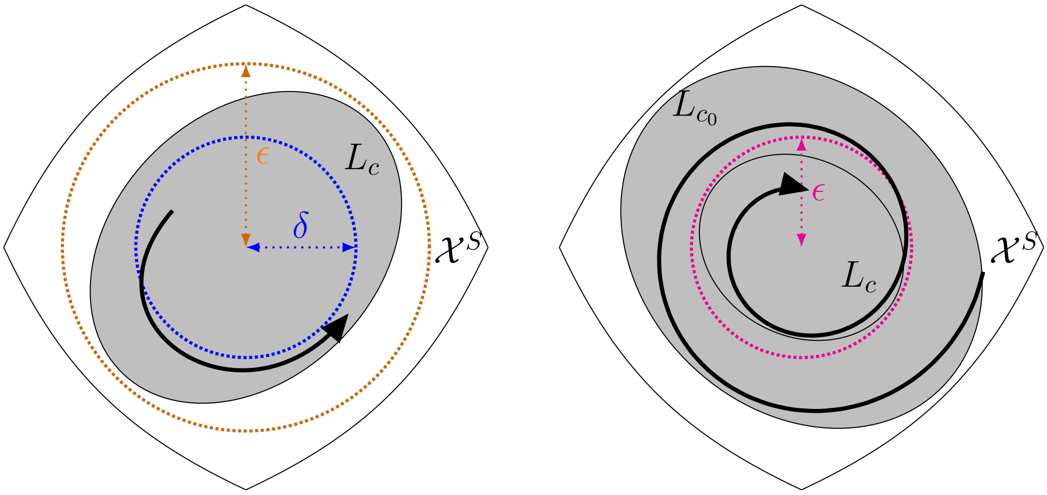
$$C_l = \{ \mathbf{y} \in Inv(l) \mid c \leq V(l,\mathbf{y}) \leq c_0 \}$$

*for every location* $l \in Loc$*, which are bounded sets "between" the initial level set of the trajectory and a level set in the* $\epsilon$*-ball. Since all* $V(l,\cdot)$*,* $l \in Loc$ *are continuous, we can compute*

$$-r_l^C = \sup_{\mathbf{y} \in C_l} \dot{V}(l,\mathbf{y}) < 0$$

*and take* $r^C = \min_{l \in Loc} r_l^C$ *which is the slowest decrease rate in* $\bigcup_l C_l$*. Thus,* $\dot{V}(\mathbf{x}(t)) \leq -r^C$ *for all* $t$ *and we have*

$$V(l(u),\mathbf{x}(u)) \leq V(l(0),\mathbf{x}(0)) + \int_0^u \dot{V}(l(t),\mathbf{x}(t))\,\mathrm{d}t \leq V(l(0),\mathbf{x}(0)) - r^C u,$$



(a) Construction for Lyapunov stability.

(b) Construction for global attractivity.

Figure 5.7: Visualization of the proof of Theorem 5.20

*which states that the Lyapunov function's value is bounded from above and we can compute the time required by the trajectory to reach a certain level set. Thus,*

$$t_\epsilon = (c_0 - c)/r^C$$

*yields an upper bound on the time to reach $L_c$ and therefore reaching $\mathcal{B}_\epsilon$. Since $L_c$ is invariant, it holds $||\mathbf{x}^S(t)|| < \epsilon$ for all $t \geq t_\epsilon$. The construction is visualized in Figure 5.7b.* □

Sometimes we want to express that a system converges to a certain equilibrium point only in periods of time where a certain conditions are satisfied instead of converging no matter what.

**Example 5.21**
*Given the system*

$$\dot{x} = x_{\texttt{ref}} - x$$
$$\dot{y} = -y$$

*with the external input $x_{\texttt{ref}}$. With global asymptotic stability under assumptions (GAS-Asm), we can describe the behavior of a system $\mathcal{A}$ like this:*

- *the variable $y$ **always** converges to $0$,*
- *if condition $\phi_1 \equiv x_{\texttt{ref}} = 5$, holds then the variable $x$ converges to $5$,*
- *if condition $\phi_2 \equiv x_{\texttt{ref}} = 23$, holds then the variable $x$ converges to $23$, and*

- *if condition* $\phi_3 \equiv x_{ref} = 42$*, holds then the variable* $x$ *converges to* $42$.

$\triangleleft$

Such sets of properties can be expressed by the following modified version of GAS, namely, conditional global asymptotic stability (conGAS). Intuitively, a HIOA $\mathcal{A}$ is $\varphi^{con}$-conGAS wrt. $\varphi^{env}$ if $\mathcal{A}$ is GAS as long as an condition predicate $\varphi^{con}$ is satisfied.

**Definition 5.22 (Conditional Global Asymptotic Stability.)**
*Let*

$$\mathcal{A} = (Loc, Var, Trans, Flow, Inv, Inits)$$

*be a hybrid I/O automaton, let* $\varphi^{con} = \varphi^{con}_{inv} \wedge (\neg \mathit{flow} \Rightarrow \varphi^{con}_{\mathit{flow}}) \wedge (\mathit{flow} \Rightarrow \varphi^{con}_{dscr})$ *be an environmental predicate on* $Var^E$*, and let* $Var^S \subseteq Var^C$ *be a subset of variables required to converge to the equilibrium point* $\mathbf{0} \in \mathbb{R}^{|Var^S|}$*. A continuous-time dynamic system* $\mathcal{A}$ *is called* $\varphi^{con}$-conditionally Lyapunov stable (LS) with respect to $Var^S$ *if for all functions* $\mathbf{x}(t)$ *holds*

$$\forall t_0 \geq 0, \forall \epsilon > 0, \forall t_f \geq t_0 \bullet \exists \delta > 0 \bullet$$
$$(\forall t \in [t_0, t_f] \bullet \mathbf{x}(t) \models \varphi^{con}) \Rightarrow \big(\forall t \in [t_0, t_f] \bullet ||\mathbf{x}(t_0)|| < \delta \Rightarrow \big|\big|\mathbf{x}^S(t)\big|\big| < \epsilon\big).$$

$\mathcal{A}$ *is called* $\varphi^{con}$-conditionally globally attractive (GA) with respect to $Var^S$ *if for all functions* $\mathbf{x}(t)$ *holds*

$$\forall t_0 \geq 0, \forall \epsilon > 0, \exists t_\epsilon \geq t_0 \bullet \forall t_f \geq t_\epsilon \bullet$$
$$\big((\forall t \in [t_0, t_f] \bullet \mathbf{x}(t) \models \varphi^{con}) \Rightarrow \big(\forall t \in [t_\epsilon, t_f] \bullet \big|\big|\mathbf{x}^S(t)\big|\big| < \epsilon\big)\big),$$

$\mathbf{x}^S$ *refers to the projection of* $\mathbf{x}_{\downarrow Var^S}$*. A system is* $\varphi^{con}$-conditionally globally asymptotically stable ($\varphi^{con}$-conGAS) with respect to $Var^S$ *if and only if it is, both,* $\varphi^{con}$*-conditionally Lyapunov stable with respect to* $Var^S$ *and* $\varphi^{con}$*-conditionally globally attractive with respect to* $Var^S$*.* $\diamondsuit$

Like for GAS-Asm, we give an adapted version of the Lyapunov theorem for conGAS which employs discontinuous conditional Lyapunov functions. This can be understood as: there exists functions that serves as Lyapunov function given that a certain condition $\varphi^{con}$ is satisfied.

**Theorem 5.23 (Discontinuous Conditional Lyapunov Functions.)**
*Let*

$$\mathcal{A} = (Loc, Var, Trans, Flow, Inv, Inits)$$

*be a hybrid I/O automaton, let* $\varphi^{con} = \varphi^{con}_{inv} \wedge (\neg \mathit{flow} \Rightarrow \varphi^{con}_{\mathit{flow}}) \wedge (\mathit{flow} \Rightarrow \varphi^{con}_{dscr})$ *be an environmental predicate on* $Var^E$*, and let* $Var^S \subseteq Var^C$ *be a subset of variables required to converge to the equilibrium point* $\mathbf{0} \in \mathbb{R}^{|Var^S|}$*. If for each mode* $l \in Loc$ *there exists a set of variables* $Var^{S_l}$ *with* $Var^S \subseteq Var^{S_l} \subseteq Var^C$ *and there exists a* *continuously differentiable function $V_l : \mathcal{X} \to \mathbb{R}$ such that*

*(C1) for each $l \in Loc$, there exist two class-$\mathcal{K}^\infty$ functions $\alpha$ and $\beta$ such that*

$$\forall \mathbf{x} \in Inv(l) \bullet \mathbf{x} \models \varphi^{con}_{inv} \Rightarrow \alpha\big(||\mathbf{x}^{S_l}||\big) \leq V_l(\mathbf{x}) \leq \beta\big(||\mathbf{x}^{S_l}||\big),$$

*(C2) for each $l \in Loc$, there exists a class $\mathcal{K}^\infty$ function $\gamma$ such that*

$$\forall \mathbf{x} \in Inv(l) \bullet \mathbf{x} \models \varphi^{con}_{inv} \Rightarrow \dot{V}_l\big(\mathbf{x}^C\big) \leq -\gamma\big(||\mathbf{x}^{S_l}||\big)$$

$$\text{for each } \dot{V}_l(\mathbf{x}) \in \left\{ \left\langle \begin{bmatrix} \frac{\partial V_l(\mathbf{x})}{\partial \mathbf{x}^C} \\ \frac{\partial V_l(\mathbf{x})}{\partial \mathbf{y}^I} \end{bmatrix} \middle| \begin{bmatrix} f(\mathbf{x}) \\ \dot{\mathbf{y}}^I \end{bmatrix} \right\rangle \middle| \begin{array}{c} f(\mathbf{x}) \in Flow(l) \\ \wedge\, \dot{\mathbf{y}}^I, \mathbf{x}^I \models \varphi^{env}_{flow} \end{array} \right\},$$

*(C3) for each transition $(l, G, U, l') \in Trans$,*

$$\forall \mathbf{x} \in G, \mathbf{x}' \bullet \Big(\mathbf{x} \models \varphi^{con}_{inv} \wedge \mathbf{x}^{C'} = U(\mathbf{x}) \wedge \mathbf{x}^{I'} = \mathbf{x}^I\Big) \Rightarrow V_{l'}\big(\mathbf{x}'\big) \leq V_l(\mathbf{x}),$$

*(C4) for each $l \in Loc$ for the external discrete updates*

$$\forall \mathbf{x}, \mathbf{x}' \in \mathcal{X} \bullet \Big(\mathbf{x} \models \varphi^{con}_{inv} \wedge \mathbf{x}^{C'} = \mathbf{x}^C \wedge \mathbf{x}^{I'}, \mathbf{x}^I \models \varphi^{con}_{dscr}\Big) \Rightarrow V_l\big(\mathbf{x}'\big) \leq V_l(\mathbf{x}),$$

*where $\mathbf{x}^S$ refers to the projection of $\mathbf{x}_{\downarrow Var^S}$ and $\mathbf{x}^{S_l}$ refers to the projection of $\mathbf{x}_{\downarrow Var^{S_l}}$, then $\mathcal{A}$ is $\varphi^{con}$-conditionally globally asymptotically stable (conGAS) with respect to $Var^S$ and $V_l$ is called a* local Lyapunov function (LLF) *of $l$, and the function $V(l, \mathbf{x}) = V_l(\mathbf{x})$ is called a* global Lyapunov function (GLF) *for $\mathcal{A}$.*

**Note 5.24**

*The difference between discontinuous Lyapunov function and conditional discontinuous Lyapunov functions is that for the former, the system has to converge all the time and in the latter it has to converge only while the predicate $\varphi^{con}$ evaluates to* `true`*.*

*Furthermore, the original Lyapunov theorem is a special case of our theorem. This can easily be seen by replacing $\varphi^{con}$ with* `true`*.* ◁

**Proof.**

*The proof is analogous to the proof of GAS-Asm but we assume that $\mathbf{x}(\cdot)$ is a trajectory with $\forall t \in [t_0, t_f] \bullet \mathbf{x}(t) \models \varphi^{con}$ where $t_0, t_f$ are two fixed but arbitrary timepoints with $t_0 \leq t_f$. First, we observe that satisfaction of $\varphi^{con}$ implies non-negativity of the Lyapunov function as well as decreasingness for every flow and (internal or external) discrete update. Then, the rest of proof as well as the constructions follows is similar to the proof of Theorem 5.20.* □

**Note 5.25**
*For* ***global attractivity****, it is noticeable that the $\epsilon$-ball $\mathcal{B}_\epsilon$ is guaranteed to be reached in case $t_f \geq t_\epsilon$.* ◁

### 5.3.2 Global Asymptotic Stability under Parallel Composition

In the following, we show how to make use of the new stability notions in the context of the composition of control components. The goal is to specify interfaces annotating GAS-Asm and conGAS properties. Further, these properties shall be preserved under composition. That is, if a sub-component of a composed component is GAS-Asm (resp. conGAS), then the composed component is also GAS-Asm (resp. conGAS) with respect to the same equilibrium. This essentially allows us to prove the properties in isolation and composed components simply inherit these properties just like safety properties. The idea is illustrated in Figure 5.8.

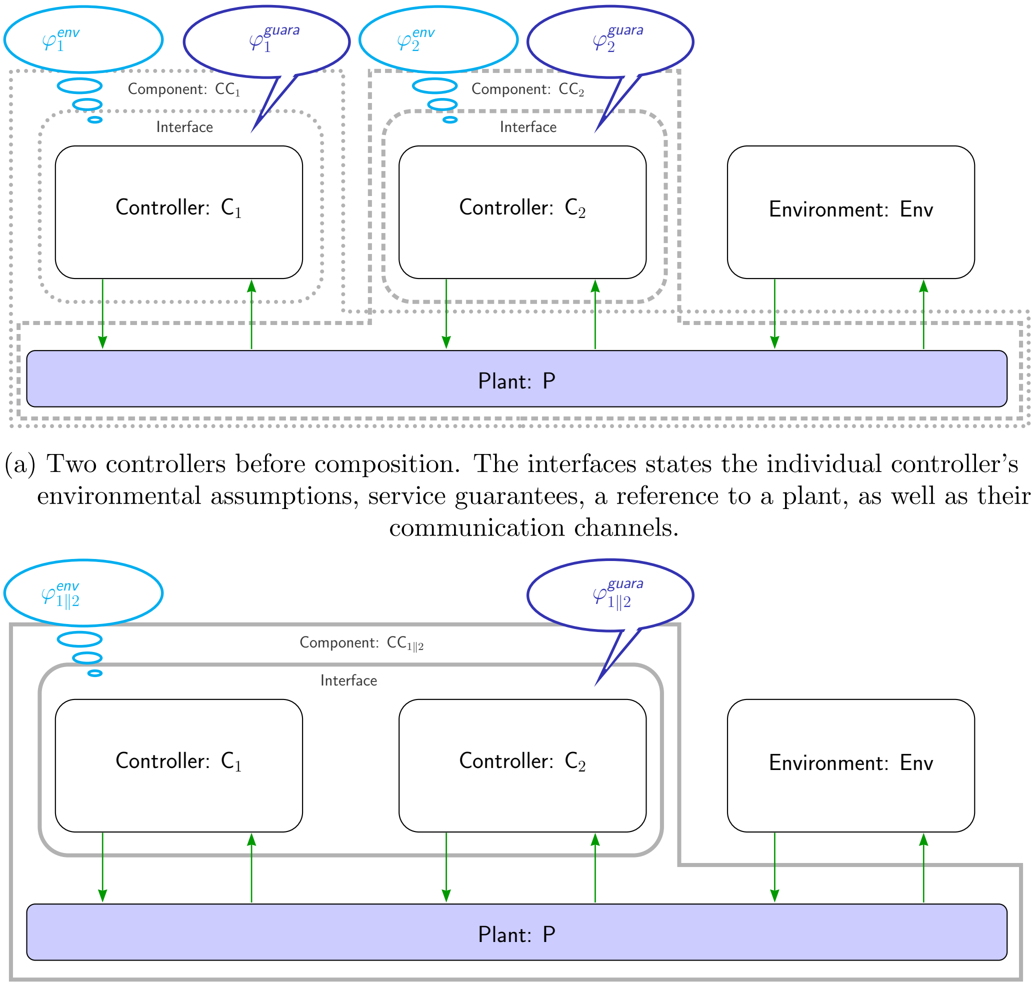
Note that directly applying the Lyapunov theorem Theorem 2.31 requires to construct the full hybrid automaton resulting of the parallel composition of the automata of all controllers, communication channels, the plant, and the environment. In most cases, this is not tractable due to a state space explosion. Instead, we propose the use of control components consisting of a controller, a reference to a plant, and an environment predicate as a means of verification in isolation. Consider the following formal definition of control components.

**Definition 5.26 (Control Component)**
*A control component* $\mathsf{CC} = (\mathsf{C}, \mathsf{P}, \varphi^{env})$ *is a tuple where*

- $\mathsf{C} = (Loc_\mathsf{C}, Var_\mathsf{C}, Trans_\mathsf{C}, Flow_\mathsf{C}, Inv_\mathsf{C}, Inits_\mathsf{C})$ *is a controller with* $Var_\mathsf{C} = Var^L_\mathsf{C} \uplus Var^I_\mathsf{C} \uplus Var^O_\mathsf{C}$,
- $Var^I_\mathsf{Com} \subseteq Var^I_\mathsf{C}$ *is a set of input communction variables,*
- $Var^O_\mathsf{Com} \subseteq Var^O_\mathsf{C}$ *is a set of output communction variables,*
- $\mathsf{P} = (Loc_\mathsf{P}, Var_\mathsf{P}, Trans_\mathsf{P}, Flow_\mathsf{P}, Inv_\mathsf{P}, Inits_\mathsf{P})$ *is a reference to a plant with* $Var_\mathsf{P} = Var^L_\mathsf{P} \uplus Var^I_\mathsf{P} \uplus Var^O_\mathsf{P}$, $Var^I_\mathsf{P} = \mathsf{Act} \uplus \mathsf{Disturb}$, $Var^I_\mathsf{P} = \mathsf{Sens}$,
- $\varphi^{env}$ *is an environmental predicate on* $Var^E_\mathsf{CC} = Var^I_\mathsf{C} \uplus (Var^I_\mathsf{P} \setminus Var^O_\mathsf{C})$.

*and where it holds that*

- *input variables of the controller* $Var^I_\mathsf{C}$ *are either input communication variables* $Var^I_\mathsf{Com}$ *or sensors of the plant* $\mathsf{Sens}$*, i. e.,* $\mathsf{Sens} \uplus Var^I_\mathsf{Com} \supseteq Var^I_\mathsf{C}$,
- *output variables of the controller* $Var^O_\mathsf{C}$ *are either output communication variables* $Var^O_\mathsf{Com}$ *or actuators of the plant* $\mathsf{Act}$*, i. e.,* $\mathsf{Act} \uplus Var^O_\mathsf{Com} \supseteq Var^O_\mathsf{C}$,



(a) Two controllers before composition. The interfaces states the individual controller's environmental assumptions, service guarantees, a reference to a plant, as well as their communication channels.

(b) Two controllers after composition. The common interfaces states their remaining environmental assumptions, their common service guarantees, a reference to the shared plant, as well as their remaining communication channels.

Figure 5.8: Composition of two controllers whose interfaces (assumptions and guarantees) have been proven in isolation. The composition operator allows to integrated the indidivual interfaces to form a common interface.

- *output communication variables are boolean and only changed via transitions, i. e., for every output communication variable $v \in Var^{O}_{Com}$ holds*

$$\mathit{C} \models ((\mathit{flow} \Rightarrow \dot{v} = 0) \wedge v \in \{0, 1\}),$$

- *the initial hybrid states $Inits_{C}$ encode that initially no output communication variable $v \in Var^{O}_{Com}$ is set, i. e., $\forall v \in Var^{O}_{Com} \bullet \forall (l, \mathbf{x}) \in Inits_{C} \bullet \mathbf{x}_{\downarrow v} = 0$ and do* *not restrict the valuations of the input communication variables* $v \in \mathit{Var}^I_{\mathsf{Com}}$,

- *if an output communication variable is changed, then it remains unchanged for at least* $\Delta^{\mathsf{pers}} > 0$ *time, i. e., for every trajectory* $\mathbf{x}(\cdot)$ *of* $C \parallel P$ *under* $\varphi^{env}$ *and every output communication variable* $v \in \mathit{Var}^O_{\mathsf{Com}}$ *holds*

$$\forall t_{\mathsf{snd}} \bullet \mathrm{change}(v, t_{\mathsf{snd}}) \Rightarrow \forall t \in [0, \Delta^{\mathsf{pers}}] \bullet \mathbf{x}(t_{\mathsf{snd}} + t)_{\downarrow v} = \mathbf{x}(t_{\mathsf{snd}})_{\downarrow v}.$$

*We denote by* $\mathit{Var}_{\mathsf{CC}} := \mathit{Var}_{\mathsf{C}} \cup \mathit{Var}_{\mathsf{P}}$ *the set of variables of the control component.* ◇

Now, here it can been seen why it is allowed that the environment predicates talk about local variables of a hybrid I/O automaton. This is because, we want components to make assumptions not only about the open actuators and disturbances of the plant $\mathit{Var}^I_{\mathsf{P}} \setminus \mathit{Var}^O_{\mathsf{C}} \subseteq \mathsf{Act} \cup \mathsf{Disturb}$ but also about the effects on the plant's (visible) state $\mathit{Var}^I_{\mathsf{C}} \cap \mathsf{Sens}$ as well as the allowed communication $\mathit{Var}^I_{\mathsf{Com}}$.

Runs of control components are defined as follows:

**Definition 5.27 (Runs of a Control Component)**
*Let* $CC = (C, P, \varphi^{env})$ *be a control component. A – possibly infinite – sequence* $(\pi_i)$ *is a called* run of CC *if and only if* $(\pi_i)$ *is a run of* $C \parallel P$ *under assumption* $\varphi^{env}$. ◇

We define GAS for control components to be GAS-Asm of the controller and the plant with the assumed environment.

**Definition 5.28 (Global Asymptotic Stability of a Control Component)**
*A control component* $CC = (C, P, \varphi^{env})$ *is* globally asymptotically stable with respect to $\mathit{Var}^S$ *if and only if* $C \parallel P$ *is GAS-Asm with respect to* $\mathit{Var}^S$ *under the environmental predicate* $\varphi^{env}$. ◇

Next, we introduce a composition operator and state constraints under which this operator composes components such that local properties of the sub-components are preserved and passed to the composed component. The composition operator is of such a form that

- composability can be decided and
- the resulting interfaces of the composed components can be obtained,

solely based on the information annotated at the sub-component's interface. Indeed, the operator allows the composed component to inherit guarantees from its sub-components. In that sense, safety and stability properties established by a sub-component are also be established by the composed component and $\varphi^{env}$, Com, $\parallel$-composability defines when such a property-preseving-composition is actually possible. The idea is that, if there is a common assumption on the environment such that one component is able to satisfy the other component's assumptions, then the properties (locally) established by each sub-component are established by the composition, too. This finally allows us to establish the properties or guarantees local to a component and annotate them on the component's interface. Using composition, we might then integrate these components without reverifying properties of the integrated component.

For simplicity, the composition operator used in this thesis employs a stronger — state-based — interpretation of the $\models$-operator as in [DMR16]. Here, $\mathsf{C} \models \phi$ states that $\phi$ holds for all valid states according to the invariant of the locations and for all valid states according to the transitions of of the controller $\mathsf{C}$. In contrast, in [DMR16], we require only the reachable state to satisfy $\phi$. Note, that using techniques such as unrolling (c.f. Section 4.3) allow to transform an automaton such that reachability information is made explicit. This different interpretation also fits better the state-based argument used in Lyapunov theory.

**Definition 5.29 ($\varphi^{env}$, Com, $\|$-Composability)**
*Given two control components $\mathsf{CC}_i = (\mathsf{C}_i, \mathsf{P}, \varphi_i^{env})$, $i \in \{1,2\}$, an environmental predicate $\varphi^{env}$, and an communication channel $\mathsf{Com}$ where*

- $\mathsf{C}_i = \big(Loc_{\mathsf{C}_i}, Var_{\mathsf{C}_i}, Trans_{\mathsf{C}_i}, Flow_{\mathsf{C}_i}, Inv_{\mathsf{C}_i}, Inits_{\mathsf{C}_i}\big)$ *are two hybrid I/O automata with* $Var_{\mathsf{C}_i} = Var^L_{\mathsf{C}_i} \uplus Var^I_{\mathsf{C}_i} \uplus Var^O_{\mathsf{C}_i}$,
- $\mathsf{P} = (Loc_{\mathsf{P}}, Var_{\mathsf{P}}, Trans_{\mathsf{P}}, Flow_{\mathsf{P}}, Inv_{\mathsf{P}}, Inits_{\mathsf{P}})$ *is a common plant,*
- $\varphi_i^{env} = \varphi_{inv,i}^{env} \wedge (\neg \mathsf{flow} \Rightarrow \varphi_{dscr,i}^{env}) \wedge (\mathsf{flow} \Rightarrow \varphi_{flow,i}^{env})$ *are two environmental predicates on* $Var_i^E := Var^I_{\mathsf{C}_i} \uplus (Var^I_{\mathsf{P}} \setminus Var^O_{\mathsf{C}_i})$,
- $\varphi^{env}$ *is a environmental predicate on* $Var^E$, *and*
- $\mathsf{Com} = (Loc_{\mathsf{Com}}, Var_{\mathsf{Com}}, Trans_{\mathsf{Com}}, Flow_{\mathsf{Com}}, Inv_{\mathsf{Com}}, Inits_{\mathsf{Com}})$ *is a communication channel.*

*We call $\mathsf{CC}_1$ and $\mathsf{CC}_2$* $\varphi^{env}, \mathsf{Com}, \|$-composable *if and only if*

- *the set of locations may not be empty* $Loc_i \neq \emptyset$ *for all* $i, j \in \{1, 2, \mathsf{P}, \mathsf{Com}\}$, *($Loc_{\mathsf{Com}}$ may contain a dummy location)*
- *the local variables are pairwise disjoint, i. e.,* $Var^L_{\mathsf{C}_i} \cap Var^L_{\mathsf{C}_j} = \emptyset$ *for all* $i, j \in \{1, 2, \mathsf{P}, \mathsf{Com}\}$, $i \neq j$,
- *an output variable is controlled by at most one subcontroller, i. e.,* $Var^O_{\mathsf{C}_1} \cap Var^O_{\mathsf{C}_2} = \emptyset$
- *the communication channel and the plant have no common variables, i. e.,* $Var_{\mathsf{Com}} \cap Var_{\mathsf{P}} = \emptyset$,

- *output communication variables of the channel are bound to input variables of a controller, i. e.,* $Var^{O}_{\mathsf{Com}} \subseteq Var^{I}_{\mathsf{C}_1} \cup Var^{I}_{\mathsf{C}_2}$*,*
- *input communication variables of the channel are bound to ouput variables of a controller,* $Var^{I}_{\mathsf{Com}} \subseteq Var^{O}_{\mathsf{C}_1} \cup Var^{O}_{\mathsf{C}_2}$*,*
- *the input variables of the composed controller are the input variables of the sub-controllers without those variables that are bound by the communication channel and are bound to sensors of the plant, i. e.* $Var^{I}_{1\|2} := (Var^{I}_{\mathsf{C}_1} \cup Var^{I}_{\mathsf{C}_2}) \setminus Var^{O}_{\mathsf{Com}} \subseteq \mathsf{Sens}$*,*
- *the output variables of the composed controller are the output variables of the sub-controllers without those variables that are bound by the communication channel and are bound to actuators of the plant, i. e.* $Var^{O}_{1\|2} := (Var^{O}_{\mathsf{C}_1} \cup Var^{O}_{\mathsf{C}_2}) \setminus Var^{I}_{\mathsf{Com}} \subseteq \mathsf{Act}$*,*
- *the controlled variables of the composed controller is the union the controlled variables of sub-controllers and the communication channel, i. e.,* $Var^{C}_{1\|2} := Var^{C}_{\mathsf{C}_1} \cup Var^{C}_{\mathsf{C}_2} \cup Var^{C}_{\mathsf{Com}}$*,*
- *for the environmental predicate* $\varphi^{env}$ *holds that*

  *(1) the predicate refers only to inputs of the composed controller or uncontrolled inputs of the plant, i. e.,* $Var^{E} \subseteq Var^{E}_{1\|2}$ *with* $Var^{E}_{1\|2} := Var^{I}_{1\|2} \mathbin{\dot\cup} (Var^{I}_{\mathsf{P}} \setminus Var^{O}_{1\|2})$*,*

  *(2) the i-th controller and the communication channel guarantee the j-th environment predicate*

  $$\mathsf{C}_i \parallel \mathsf{Com} \parallel \mathsf{P} \models \varphi^{env} \wedge \varphi^{env}_{i} \Rightarrow \varphi^{env}_{j}$$

  *and*

  *(3) the common environment predicate* $\varphi^{env}$ *covers the predicate* $\varphi^{env}_{j}$ *regarding the imposed restrictions on updates (i. e.,* $\varphi^{env}_{inv,j}$ *and* $\varphi^{env}_{dscr,j}$*) of external variables (i. e.,* $Var^{E}_{1\|2}$*)*

$$\begin{aligned} \forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}_{1\|2} \bullet \qquad & \left(\mathbf{x}_{\downarrow Var^{E}_{j} \setminus Var^{E}_{1\|2}}, \mathbf{x}'_{\downarrow Var^{E}_{j} \setminus Var^{E}_{1\|2}} \models \varphi^{env}_{inv,j} \wedge \varphi^{env}_{dscr,j}\right) \\ & \wedge \left(\mathbf{x}_{\downarrow Var^{E}_{1\|2}}, \mathbf{x}'_{\downarrow Var^{E}_{1\|2}} \models \varphi^{env}_{inv} \wedge \varphi^{env}_{dscr}\right) \\ & \Rightarrow \mathbf{x}_{\downarrow Var^{E}_{j}}, \mathbf{x}'_{\downarrow Var^{E}_{j}} \models \varphi^{env}_{inv,j} \wedge \varphi^{env}_{dscr,j} \end{aligned} \tag{5.1}$$

*The composed control component is denoted by* $\mathsf{CC}_{1\|2} = \mathsf{CC}_1 \parallel \mathsf{CC}_2 = \left(\mathsf{C}_{1\|2}, \mathsf{P}, \varphi^{env}_{1\|2}\right)$ *with* $\mathsf{C}_{1\|2} := \mathsf{C}_1 \parallel \mathsf{C}_2 \parallel \mathsf{Com}$, $\varphi^{env}_{1\|2} := \varphi^{env}$. ♢

The last requirement in $\varphi^{env}$, Com, $\|$-composability ensures that assumptions of a control component either have to be satisfied by the other control component or have to be passed to the composed control component's environment predicate (cf. Item (2)). However, the composed control component's environment predicate may not talk about actuators which are already bound to this control component. This is due to the fact that no other control component may be able to establish assumption on actuators which the component does not control (cf. Item (1)). Item (3) in the last requirement in $\varphi^{env}$, Com, $\|$-composability is a bit technical because the parallel composition of a controller, the communication channel, and the plant might not have any transitions and the $\models$-operator is evaluated based on the states imposed by the definition of locations and transitions and not on trajectories. Consider the following example showing that the slightly more technical formulation of the requirement is needed.

**Example 5.30**
*Consider two control components* $\mathsf{CC}_1$ *and* $\mathsf{CC}_2$ *each controlling a variable* $x$ *and* $y$ *of a common plant, respectively. Let the plant have a third variable* $z$ *which is not yet bound to a control component, i. e., the variable is controlled by an unspecified environment. Let* $\mathsf{CC}_1$ *have the assumption* $\varphi^{env}_{\mathsf{CC}_1} \equiv (\mathit{flow} \Rightarrow \dot{z} = 0) \wedge (\neg \mathit{flow} \Rightarrow z' \in \{1, 2\})$ *and* $\mathsf{CC}_2$ *have the assumption* $\varphi^{env}_{\mathsf{CC}_2} \equiv \mathtt{true}$*. Suppose that both,* $\mathsf{CC}_1$ *and* $\mathsf{CC}_2$ *have no transitions, if we would simply ask all transitions to satisfy discrete updates – as in the definition of satisfaction of an environment predicate (cf. Definition 5.3) – then composing* $\mathsf{CC}_1$ *with* $\mathsf{CC}_2$ *with an empty communication channel and the remaining assumption* $\varphi^{env}_{\mathsf{CC}_{1\|2}} \equiv (\mathit{flow} \Rightarrow \dot{z} = 0) \wedge (\neg \mathit{flow} \Rightarrow \mathtt{true})$ *for* $\mathsf{CC}_{1\|2} = \mathsf{CC}_1 \parallel \mathsf{CC}_2$ *would have been legal, even though, the composed environmental predicate does not satisfy the predicate of* $\mathsf{CC}_1$*. Note, that the assumption on the discrete updates on* $z$ *has disappeared with no control component being in charge for the assumption.* ◁

Therefore, we have to explicitly ask $\varphi^{env}$ to cover updates of variables that are not yet covered by either the communication channel and the other controller $Var^E_j \setminus Var^E_{1\|2} = Var^O_{\mathsf{Com}} \cup Var^O_{\mathsf{C}_j}$. Otherwise, it would be legal to compose sub-components without transitions while having guarantees on discrete updates that are not satisfied.

Note, that it is not needed to take care of updates of the plant as the plant may only perform updates on its local variables. In [DMR16], this extra requirement is not needed because in the paper satisfaction of an environmental predicate is a property on the runs of the automaton.

One important result from [DMR16] is that runs of the composed component can be projected down to any of its sub-components and one obtains a valid run of the sub-component. It is summarized in the following lemma and we refer to its proof to [DMR16].

**Lemma 5.31 (Global Runs Imply Local Runs [DMR16, Lemma 4 and Remark 1])**
*Given two $\varphi^{env}$, Com, $\|$-composable control components $CC_1$ and $CC_2$ with $CC_j = \left(C_j, P, \varphi_j^{env}\right)$, $j = 1, 2$. If $(\pi_i) = (l_i, \mathbf{x}_i)$ with $(l_i) = (l_{i,1}, l_{i,2}, l_{i,Com}, l_{i,P})$ is a run of $CC_1 \parallel CC_2$, then $(\pi_{j,i}) = \left((l_{i,j}, l_{i,P}), \mathbf{x}_{i\downarrow Var_{CC_j}}\right)$ is a run of $CC_j$, $j = 1, 2$.*

Using Lemma 5.31, we can proof that GAS or rather GAS-Asm is compatible with composition in our framework.

**Theorem 5.32**
*Given two $\varphi^{env}$, Com, $\|$-composable control components $CC_1$ and $CC_2$. If $CC_i = (C_i, P, \varphi_i^{env})$ is GAS wrt. $Var_i^S$, then $CC_1 \parallel CC_2$ is GAS wrt. $Var_1^S \cup Var_2^S$.*

**Proof.**
*Let $CC_1$ and $CC_2$ be two $\varphi^{env}$, Com, $\|$-composable control components with $CC_i = (C_i, P, \varphi_i^{env})$ being GAS with respect to $Var_i^S$. We show that $CC_1 \parallel CC_2$ is both, Lyapunov stable and globally attractive with respect to $Var^S := Var_1^S \cup Var_2^S$. Let us fix an arbitrary trajectory $\mathbf{x}(\cdot)$ of $CC_1 \parallel CC_2$. We know that $\forall t \bullet \mathbf{x}(t) \models \varphi^{env}$ holds. Further, by Lemma 5.31, we know that the projection $\mathbf{x}(\cdot)_{\downarrow Var_{CC_i}}$ belongs to a run of $CC_i$.*

***Lyapunov Stable:*** *Given $\epsilon > 0$ we have to construct a $\delta > 0$ for $CC_{1\|2}$ such that $\forall t \geq 0 \bullet ||\mathbf{x}(0)|| < \delta \Rightarrow \left|\left|\mathbf{x}(t)_{\downarrow Var^S}\right|\right| < \epsilon$. By choosing $r \in ]0, \epsilon[$ and $\epsilon_i \leq \frac{1}{\sqrt{2}} r$, we obtain the set*

$$C_{\epsilon_1,\epsilon_2} = \left\{ \mathbf{y} \in \mathbb{R}^{|Var^S|} \;\middle|\; \left|\left|\mathbf{y}_{\downarrow Var_1^S}\right|\right| \leq \epsilon_1 \wedge \left|\left|\mathbf{y}_{\downarrow Var_2^S}\right|\right| \leq \epsilon_2 \right\}$$

*which is in the interior of $\mathcal{B}_r = \left\{ \mathbf{y} \in \mathcal{X}_{1\|2} \;\middle|\; \left|\left|\mathbf{y}_{\downarrow Var^S}\right|\right| \leq r \right\}$ since $Var_1^S$ and $Var_2^S$ are disjoint and, thus, for all $\forall \mathbf{y} \in C_{\epsilon_1,\epsilon_2}$ holds*

$$\left|\left|\mathbf{y}_{\downarrow Var^S}\right|\right|^2 = \left|\left|\mathbf{y}_{\downarrow Var_1^S}\right|\right|^2 + \left|\left|\mathbf{y}_{\downarrow Var_2^S}\right|\right|^2 \leq \epsilon_1^2 + \epsilon_2^2 \leq 2(\frac{1}{\sqrt{2}} r)^2 \leq r^2 < \epsilon^2.$$

*Since $CC_i$ is GAS with respect to $Var_i^S$, there exists a $\delta_i > 0$ such that*

$$\forall t \geq 0 \bullet \left|\left|\mathbf{x}(0)_{\downarrow Var_{CC_i}}\right|\right| < \delta_i \Rightarrow \left|\left|\mathbf{x}(t)_{\downarrow Var_i^S}\right|\right| < \epsilon_i,$$

*hence, any trajectory starting in the set*

$$C_{\delta_1,\delta_2} = \left\{ \mathbf{y} \in \mathcal{X}_{1\|2} \;\middle|\; \left|\left|\mathbf{y}_{\downarrow Var_{CC_1}}\right|\right| \leq \epsilon_1 \wedge \left|\left|\mathbf{y}_{\downarrow Var_{CC_2}}\right|\right| \leq \epsilon_2 \right\}$$

*stays in $C_{\epsilon_1,\epsilon_2}$ forever. Now, with $\delta = \min(\delta_1, \delta_2)$, we obtain the largest $\delta$-ball inside $C_{\delta_1,\delta_2}$. And therefore,*

$$\begin{aligned} & \mathbf{x}(0) \in \mathcal{B}_\delta \\ \Rightarrow & \mathbf{x}(0) \in C_{\delta_1,\delta_2} \\ \Rightarrow & \mathbf{x}(t)_{\downarrow Var^S} \in C_{\epsilon_1,\epsilon_2} \\ \Rightarrow & \mathbf{x}(t)_{\downarrow Var^S} \in \mathcal{B}_r \\ \Rightarrow & \left|\left|\mathbf{x}(t)_{\downarrow Var^S}\right|\right| < \epsilon \end{aligned}$$

.

***Globally Attractive:*** *We have to show that for each $\epsilon > 0$ there exists a $t_\epsilon > 0$ for $\mathsf{CC}_{1\|2}$ such that $\forall t > t_\epsilon \bullet \left|\left|\mathbf{x}(t)_{\downarrow Var^S}\right|\right| < \epsilon$. Again, given $\epsilon$, we construct the $r$-ball $\mathcal{B}_r$, $\epsilon_i$, and $C_{\epsilon_1,\epsilon_2}$ as above. Since $\mathsf{CC}_i$ is GAS with respect to $Var_i^S$, there exists $t_{\epsilon_i} > 0$ such that $\forall t > t_{\epsilon_i} \bullet \left|\left|\mathbf{x}(t)_{\downarrow Var_i^S}\right|\right| < \epsilon_i$. Now, choosing $t_\epsilon = \max(t_{\epsilon_1}, t_{\epsilon_2})$, we know $\forall t > t_\epsilon \bullet \left|\left|\mathbf{x}(t)_{\downarrow Var_i^S}\right|\right| < \epsilon_i$. By construction of $C_{\epsilon_1,\epsilon_2}$ for all $t > t_\epsilon$ holds*

$$\left|\left|\mathbf{x}(t)_{\downarrow Var_1^S}\right|\right| < \epsilon_1 \wedge \left|\left|\mathbf{x}(t)_{\downarrow Var_2^S}\right|\right| < \epsilon_2 \Rightarrow \mathbf{x}(t)_{\downarrow Var^S} \in C_{\epsilon_1,\epsilon_2} \Rightarrow \mathbf{x}(t)_{\downarrow Var^S} \in \mathcal{B}_r.$$

*And therefore, $\left|\left|\mathbf{x}(t)_{\downarrow Var^S}\right|\right| \leq r < \epsilon$ for all $t \geq t_\epsilon$.* □

With GAS-Asm, we are able to describe in a compositional way that all individual sub-components of the system approach a local equilibrium regardless of the actual behavior of the environment as long as the environment predicate is not violated. We can decompose — although a task far from being trivial — a large system into smaller subsystems, prove GAS-Asm in isolation using a strong invariant on the environment, and compose the obtained knowledge to obtain GAS-Asm for the original system.

On the other hand, with conGAS, we are able to describe that the equilibrium, the system converges to, dynamically depends on the system's circumstances. With conGAS, we can, for instance, describe the convergence to two different setpoints, e. g., a velocity controller under the circumstance of being throttled to a low speed $vel_{\mathsf{lmax}}$, converges to some $vel_{\mathsf{goal}} \leq x_l$ and converges to some $vel'_{\mathsf{goal}} \leq vel_{\mathsf{hmax}}$ while the maximum velocity is as high as $vel_{\mathsf{hmax}}$.

**Definition 5.33 (Conditional Global Asymptotic Stability of a Control Component)**
*A control component $\mathsf{CC} = (\mathsf{C}, \mathsf{P}, \varphi^{env})$ is $\varphi^{con}$-*conditional globally asymptotically stable with respect to $Var^S$ ($\varphi^{con}$-conGAS) *if and only if $\varphi^{con}$ is an environmental predicate on $\mathsf{Sens}$ and $\mathsf{C} \parallel \mathsf{P}$ is $\varphi^{con}$-conditional globally asymptotically stable with respect to $Var^S$.* ♢

Note, that the environmental predicate $\varphi^{con}$ is not allowed to talk about the output variables of the controller. Therefore, the actual setpoint cannot be set from another controller because $Var^S \cup \mathsf{Sens} = \emptyset$. To overcome this, communication variables can be used through which a controller might update its local setpoint. In Section 5.4, we will make use of *events* which are a high-level abstraction of the communication channels. These events are specified in so called interfaces and allow us to encode control objectives like:

(Obj1) if an event $\mathcal{E}$ is active and promising that $vel_{\mathsf{max}} = 25\text{m/s}$ holds for some timespan, then $vel_{\mathsf{cur}}$ converges $vel_{\mathsf{goal}} = \mathcal{E}.vel_{\mathsf{max}} - 2$ during that timespan,

(Obj2) if an event $\mathcal{E}$ is active and promising that $vel_{\mathsf{max}} = 50\text{m/s}$ holds for some timespan, then $vel_{\mathsf{cur}}$ converges $vel_{\mathsf{goal}} = \mathcal{E}.vel_{\mathsf{max}} - 4$ during that timespan.

Next, we show that conGAS is compatible with composition.

**Theorem 5.34**
*Given two $\varphi^{env}$, $\mathsf{Com}$, $\|$-composable control components $\mathsf{CC}_1, \mathsf{CC}_2$ and an environmental predicate $\varphi^{con}$ on $\mathsf{Sens} \uplus (Var^I_{\mathsf{P}} \setminus Var^O_{\mathsf{C}})$. If one sub-component $\mathsf{CC}_i = (\mathsf{C}_i, \varphi^{env}_i, \mathsf{P})$ is $\varphi^{con}$-conGAS with respect to $Var^S_i$, then $\mathsf{CC}_1 \parallel \mathsf{CC}_2$ is $\varphi^{con}$-conGAS with respect to $Var^S_i$.*

**Proof.**
*By Lemma 5.31 follows that for every trajectory $\mathbf{x}(t)$ of $\mathsf{CC}_{1\|2} := \mathsf{CC}_1 \parallel \mathsf{CC}_2$ the projection $\mathbf{x}(t)_{\downarrow Var_i}$ is a trajectory corresponding to a run of $\mathsf{CC}_i$. For Lyapunov stability one has to note that the $\delta$ obtained from Lyapunov stability of $\mathsf{CC}_i$ also yields a full-dimensional $\delta$-ball for $\mathsf{CC}_{1\|2}$ whose projection onto $Var_i$ corresponds to the lower-dimensional $\delta$-ball for $\mathsf{CC}_i$. For global attractivity there is nothing to show since the upper bound $t_\epsilon$ on the time, obtained from global attractivity, needed by the sub-component $\mathsf{CC}_i$ to enter the $\epsilon$-ball, is the same as for the composed system.* □

So far, we have established that we can proof GAS of a larger control component by finding a suitable (parallel) decomposition of the controllers and the proving GAS for each sub-component in isolation. Although this is a non-trivial task, it allows us to do a structured design.

Next, we show that we cannot only inherit stability from the sub-components but we can also reuse their Lyapunov functions to create a so-called *composite Lyapunov function* [Kha00]. The idea is that the sum of the sub-component's Lyapunov function is a valid Lyapunov function for the composed component. The main difference is that in our special settings syntactical restrictions and invariance properties are exploited such that no further proof obligations arise during the composition of sub-components. Other methods employ vanishing terms or investigate the interconnecting term a posteriori. However, in our framework validity of the composite Lyapunov functions is assured by requiring that a component's environment satisfies the component's environmental predicate $\varphi^{env}$.

To ease readability, we introduce some shorthand notations that already incorporate the requirement that assumptions have to be satisfied. For a control component $\mathsf{CC} = (\mathsf{C}, \mathsf{P}, \varphi^{env})$ with a controller $\mathsf{C} = (\mathit{Loc}, \mathit{Var}, \mathit{Trans}, \mathit{Flow}, \mathit{Inv}, \mathit{Inits})$, a reference to a plant $\mathsf{P} = (\mathit{Loc}_\mathsf{P}, \mathit{Var}_\mathsf{P}, \mathit{Trans}_\mathsf{P}, \mathit{Flow}_\mathsf{P}, \mathit{Inv}_\mathsf{P}, \mathit{Inits}_\mathsf{P})$, and an environmental predicate $\varphi^{env}$, let $\mathsf{C} \parallel \mathsf{P} = \left(\mathit{Loc}_{\mathsf{C}\parallel\mathsf{P}}, \mathit{Var}_{\mathsf{C}\parallel\mathsf{P}}, \mathit{Trans}_{\mathsf{C}\parallel\mathsf{P}}, \mathit{Flow}_{\mathsf{C}\parallel\mathsf{P}}, \mathit{Inv}_{\mathsf{C}\parallel\mathsf{P}}, \mathit{Inits}_{\mathsf{C}\parallel\mathsf{P}}\right)$. We define the sets

$$\begin{aligned}
\mathit{Loc}_\mathsf{CC} &:= \mathit{Loc}_{\mathsf{C}\parallel\mathsf{P}}\\
\mathit{Var}_\mathsf{CC} &:= \mathit{Var}_{\mathsf{C}\parallel\mathsf{P}}\\
\mathit{Inv}_\mathsf{CC}(l) &:= \left\{ \mathbf{y} \in \mathit{Inv}_{\mathsf{C}\parallel\mathsf{P}}(l) \,\middle|\, \mathbf{y} \models \varphi^{env}_{inv} \right\}\\
\mathit{Flow}_\mathsf{CC}(l)(\mathbf{x}) &:= \left\{ \dot{\mathbf{y}} \,\middle|\, \dot{\mathbf{y}}^C \in \mathit{Flow}_{\mathsf{C}\parallel\mathsf{P}}(l)(\mathbf{x}) \wedge \dot{\mathbf{y}}^I, \mathbf{x} \models \varphi^{env}_{flow} \right\}\\
\mathit{Trans}_\mathsf{CC} &:= \left\{ (l, G_\mathsf{CC}, U, l') \,\middle|\, \begin{array}{l} (l, G, U, l') \in \mathit{Trans}_{\mathsf{C}\parallel\mathsf{P}} \\ \wedge G_\mathsf{CC} = \{ \mathbf{y} \in G \mid \mathbf{y} \models \varphi^{env}_{inv} \} \end{array} \right\}\\
\mathit{Inits}_\mathsf{CC} &:= \mathit{Inits}_{\mathsf{C}\parallel\mathsf{P}}.
\end{aligned}$$

The following is a technical lemma stating that the projection of a hybrid state of the composed system is a valid hybrid state of the subsystem, i. e., if $\mathbf{x}$ is in the invariant of the composed control component then $\mathbf{x}_{\downarrow \mathit{Var}_i}$ is in the invariant of the sub-component and likewise for the flow and satisfaction of the environmental predicate. For transitions, it holds that if $\tau$ is a transition of the composed control component, then there is exactly one sub-component with a corresponding transition while all other sub-components may not change their location.

**Lemma 5.35 (Invariant, Flow, Transition, and External Update Inclusion)**
*Given two $\varphi^{env}$, $\mathsf{Com}$, $\parallel$-composable control components $\mathsf{CC}_i = (\mathsf{C}_i, \mathsf{P}, \varphi^{env}_i)$ with controllers $\mathsf{C}_i = \left(\mathit{Loc}_{\mathsf{C}_i}, \mathit{Var}_{\mathsf{C}_i}, \mathit{Trans}_{\mathsf{C}_i}, \mathit{Flow}_{\mathsf{C}_i}, \mathit{Inv}_{\mathsf{C}_i}, \mathit{Inits}_{\mathsf{C}_i}\right)$ for $i = 1, 2$, a reference to a plant $\mathsf{P} = (\mathit{Loc}_\mathsf{P}, \mathit{Var}_\mathsf{P}, \mathit{Trans}_\mathsf{P}, \mathit{Flow}_\mathsf{P}, \mathit{Inv}_\mathsf{P}, \mathit{Inits}_\mathsf{P})$, the communication channel $\mathsf{Com} = (\mathit{Loc}_\mathsf{Com}, \mathit{Var}_\mathsf{Com}, \mathit{Trans}_\mathsf{Com}, \mathit{Flow}_\mathsf{Com}, \mathit{Inv}_\mathsf{Com}, \mathit{Inits}_\mathsf{Com})$, and an environmental predicate $\varphi^{env}_i$ on $\mathit{Var}^E_i$. For $\mathsf{CC}_{1\parallel 2} = \mathsf{CC}_1 \parallel \mathsf{CC}_2 = \left(\mathsf{C}_{1\parallel 2}, \mathsf{P}, \varphi^{env}\right)$ holds that*

**(Invariant Inclusion)** *For all locations $(l_1, l_2, l_\mathsf{Com}, l_\mathsf{P}) = l \in \mathit{Loc}_{\mathsf{CC}_{1\parallel 2}}$ holds*

$$\begin{aligned}
&\forall \mathbf{x} \in \mathcal{X}_{\mathsf{CC}_{1\parallel 2}} \bullet \mathbf{x} \in \mathit{Inv}_{\mathsf{CC}_{1\parallel 2}}(l)\\
&\qquad \Rightarrow \left(\forall i \bullet (l_i, l_\mathsf{P}) \in \mathit{Loc}_{\mathsf{CC}_i} \wedge \mathbf{x}_{\downarrow \mathit{Var}_{\mathsf{CC}_i}} \in \mathit{Inv}_{\mathsf{CC}_i}((l_i, l_\mathsf{P}))\right).
\end{aligned}$$

**(Flow Inclusion)** *For all locations* $(l_1, l_2, l_{\mathsf{Com}}, l_{\mathsf{P}}) = l \in Loc_{\mathsf{CC}_{1\|2}}$ *holds*

$$\begin{aligned}\forall \mathbf{x} \in Inv_{\mathsf{CC}_{1\|2}}(l) \bullet \mathbf{y} \in Flow_{\mathsf{CC}_{1\|2}}(l)(\mathbf{x})& \\ \Rightarrow \quad & (\forall i \bullet (l_i, l_{\mathsf{P}}) \in Loc_{\mathsf{CC}_i} \\ & \wedge \mathbf{y}_{\downarrow Var_{\mathsf{CC}_i}} \in Flow_{\mathsf{CC}_i}((l_i, l_{\mathsf{P}}))\Big(\mathbf{x}_{\downarrow Var_{\mathsf{CC}_i}}\Big)\Big).\end{aligned}$$

**(Transition Inclusion)** *For all transitions* $\Big(l, G_{1\|2}, U_{1\|2}, l'\Big) \in Trans_{\mathsf{CC}_{1\|2}}$*, i. e.,* $l = (l_1, l_2, l_{\mathsf{Com}}, l_{\mathsf{P}})$ *and* $l' = (l'_1, l'_2, l'_{\mathsf{Com}}, l'_{\mathsf{P}})$ *holds*

$$\exists\, i \in \{1, 2, \mathsf{Com}, \mathsf{P}\} \bullet \exists \big(l_i, G_i, U_i, l'_i\big) \in Trans_i \bullet \forall j \neq i \bullet l_j = l'_j$$

*such that* $\mathbf{x} \in G_{1\|2} \wedge \mathbf{x}' \in \mathcal{X}_{\mathsf{CC}_{1\|2}} \wedge \mathbf{x}^{C'} = U_{1\|2}(\mathbf{x})$ *implies*

$$\begin{aligned} & \mathbf{x}_{\downarrow Var_{\mathsf{CC}_i}} \in G_i \\ \wedge\ & \mathbf{x}'_{\downarrow Var^C_{\mathsf{CC}_i}} = U_i\Big(\mathbf{x}_{\downarrow Var_{\mathsf{CC}_i}}\Big) \\ \wedge\ & \mathbf{x}'_{\downarrow Var_{\mathsf{CC}_{1\|2}} \setminus Var^C_{\mathsf{CC}_i}} = \mathbf{x}_{\downarrow Var_{\mathsf{CC}_{1\|2}} \setminus Var^C_{\mathsf{CC}_i}}. \end{aligned} \tag{5.2}$$

**(External Update Inclusion)**
*For all external behavior* $\forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}_{\mathsf{CC}_{1\|2}}$ *holds*

$$\begin{aligned} & (\mathbf{x} \models \varphi^{env}_{inv}) \wedge \Big(\mathbf{x}^{I'}, \mathbf{x}^{I} \models \varphi^{env}_{dscr}\Big) \\ \Rightarrow \forall i \bullet & \Big(\mathbf{x}_{\downarrow Var_{\mathsf{CC}_i}} \models \varphi^{env}_{inv,i}\Big) \wedge \Big(\mathbf{x}^{I}{}_{\downarrow Var^I_{\mathsf{CC}_i}}, \mathbf{x}^{I'}{}_{\downarrow Var^I_{\mathsf{CC}_i}} \models \varphi^{env}_{dscr,i}\Big) \end{aligned}$$

**Proof.**
*We investigate each statement separately and only sketch the proofs. Each statement involves the argument that the influence of the communication channel is covered by the environmental predicate as defined in* $\varphi^{env}$*,* Com*,* $\|$*-composability.*

***(Invariant Inclusion):*** *Follows from the definition of the parallel composition of hybrid I/O automata by rearranging terms since the invariant of the composed component is the conjunction of the sub-controller's invariants.*

***(Flow Inclusion):*** *Follows from the definition of the parallel composition of hybrid I/O automata and (Invariant Inclusion) by rearranging terms since each variable is either controlled by the plant, the communication channel, or exactly one controller and the flow of the composed controller is the Cartesian product of the sub-controller's flows and the flow of the communication channel's internal variables.*

***(Transition Inclusion):*** *Follows from the definition of the parallel composition of hybrid I/O automata by rearranging terms since each controlled variable is either updated by the plant, the communication channel, or exactly one controller. Further the set of transitions is the interleaving of the sub-controller's, the plant's, and the communication channel's sets of transitions.*

***(External Update Inclusion):*** *Follows from the definition of $\varphi^{env}$, $\mathsf{Com}$, $\|$-composability of control components: Since $\mathsf{CC}_1$ and $\mathsf{CC}_2$ are $\varphi^{env}$, $\mathsf{Com}$, $\|$-composable, we know that*

$$\mathsf{C}_j \parallel \mathsf{Com} \parallel \mathsf{P} \models \varphi^{env}_{inv} \wedge (\neg \mathit{flow} \Rightarrow \varphi^{env}_{dscr}) \Rightarrow \varphi^{env}_{inv,i} \wedge (\neg \mathit{flow} \Rightarrow \varphi^{env}_{dscr,i}) \tag{5.3}$$

*and*

$$\begin{aligned} \forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}_{1\|2} \bullet \quad & \left( \mathbf{x}_{\downarrow Var^E_j \setminus Var^E_{1\|2}}, \mathbf{x}'_{\downarrow Var^E_j \setminus Var^E_{1\|2}} \models \varphi^{env}_j \wedge \varphi^{env}_{dscr,j} \right) \\ & \wedge \left( \mathbf{x}_{\downarrow Var^E_{1\|2}}, \mathbf{x}'_{\downarrow Var^E_{1\|2}} \models \varphi^{env} \wedge \varphi^{env}_{dscr} \right) \\ & \Rightarrow \mathbf{x}_{\downarrow Var^E_j}, \mathbf{x}'_{\downarrow Var^I_j} \models \varphi^{env}_j \wedge \varphi^{env}_{dscr,j} \end{aligned} \tag{5.4}$$

*for all $i, j \in \{1, 2\}$, $i \neq j$. Formula 5.3 means that — given the common assumptions are satisfied — updates, on variables the $j$-th controller or the communication channel may do during their transition firings, satisfy the $i$-th component's assumptions and Formula 5.4 means that the remaining updates of variables, as performed by the common environment during its transition fireings, also satisfy the $i$-th component's assumptions. Together with the premise of (External Update Inclusion), i. e.,*

$$\forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}_{\mathsf{CC}_{1\|2}} \bullet (\mathbf{x} \models \varphi^{env}_{inv}) \wedge (\mathbf{x}^{I\prime}, \mathbf{x}^{I} \models \varphi^{env}_{dscr}),$$

*we conclude that for all $\mathbf{x}, \mathbf{x}' \in \mathcal{X}_{\mathsf{CC}_{1\|2}}$*

$$\begin{aligned} & (\mathbf{x} \models \varphi^{env}_{inv}) \wedge (\mathbf{x}^{I\prime}, \mathbf{x}^{I} \models \varphi^{env}_{dscr}) \\ \Rightarrow\ & \forall i \bullet \left( \mathbf{x}_{\downarrow Var_{\mathsf{CC}_i}} \models \varphi^{env}_{inv,i} \right) \wedge \left( \mathbf{x}^{I}{}_{\downarrow Var^I_{\mathsf{CC}_i}}, \mathbf{x}^{I\prime}{}_{\downarrow Var^I_{\mathsf{CC}_i}} \models \varphi^{env}_{dscr,i} \right) \end{aligned}$$

*which completes the proof.* □

Next, we show compatibility of the Lyapunov functions with parallel composition which means that a Lyapunov function with respect to some set of variables for the sub-component is also a valid Lyapunov function for the composed system with respect to the same set of variables.

**Theorem 5.36 (Compatibility of Lyapunov Function with Parallel Composition)**
*Let $\mathsf{CC}_{1\|2} = \mathsf{CC}_1 \parallel \mathsf{CC}_2$ be the $\varphi^{env}_{1\|2}$, $\mathsf{Com}$, $\|$-composite control component of the*

*$\varphi^{env}$, $\mathsf{Com}$, $\|$-composable sub-components $\mathsf{CC}_1, \mathsf{CC}_2$ with $\mathsf{CC}_i = (\mathsf{C}_i, \mathsf{P}, \varphi_i^{env})$. If $V_{\mathsf{CC}_i} : Loc_{\mathsf{CC}_i} \times \mathcal{X}_{\mathsf{CC}_i} \to \mathbb{R}$ is a global Lyapunov function with respect to $Var^S \subseteq Var^C_{\mathsf{CC}_i} \subseteq Var_{\mathsf{CC}_i}$ for $\mathsf{CC}_i$, then $V_{\mathsf{CC}_{1\|2}} : Loc_{\mathsf{CC}_{1\|2}} \times \mathcal{X}_{\mathsf{CC}_{1\|2}} \to \mathbb{R}$ with*

$$V_{\mathsf{CC}_{1\|2}}((l_1, l_2, l_{\mathsf{Com}}, l_{\mathsf{P}}), \mathbf{x}) = V_{\mathsf{CC}_i}\Big((l_i, l_{\mathsf{P}}), \mathbf{x}_{\downarrow Var_{\mathsf{CC}_i}}\Big)$$

*is a global Lyapunov function with respect to $Var^S \subseteq Var^C_{\mathsf{CC}_{1\|2}} \subseteq Var_{\mathsf{CC}_{1\|2}}$ for $\mathsf{CC}_{1\|2}$.*

**Proof.**
*Let $\mathsf{CC}_{1\|2} = \mathsf{CC}_1 \parallel \mathsf{CC}_2$ be the $\varphi^{env}_{1\|2}$, $\|$-composite control component of the $\varphi^{env}_{1\|2}$, $\mathsf{Com}$, $\|$-composable sub-components $\mathsf{CC}_1, \mathsf{CC}_2$ with $\mathsf{CC}_i = (\mathsf{C}_i, \mathsf{P}, \varphi_i^{env})$ with $\varphi_\star^{env} = \varphi^{env}_{inv,\star} \wedge (\neg \mathit{flow} \Rightarrow \varphi^{env}_{dscr,\star}) \wedge (\mathit{flow} \Rightarrow \varphi^{env}_{flow,\star})$, $\star \in \{1, 2, 1 \parallel 2\}$. Let $i$ be arbitrary but fixed and $V_{\mathsf{CC}_i} : Loc_{\mathsf{CC}_i} \times \mathcal{X}_i \to \mathbb{R}$ be a global Lyapunov function with respect to $Var^S \subseteq Var^C_{\mathsf{CC}_i} \subseteq Var_{\mathsf{CC}_i}$ for $\mathsf{CC}_i$ satisfying the four constraints of Theorem 5.23.*

*We show that $V_{\mathsf{CC}_{1\|2}} : Loc_{\mathsf{CC}_{1\|2}} \times \mathcal{X}_{\mathsf{CC}_{1\|2}} \to \mathbb{R}$ with $V_{\mathsf{CC}_{1\|2}}((l_1, l_2, l_{\mathsf{Com}}, l_{\mathsf{P}}), \mathbf{x}) = V_{\mathsf{CC}_i}\Big((l_i, l_{\mathsf{P}}), \mathbf{x}_{\downarrow Var_{\mathsf{CC}_i}}\Big)$ is a global Lyapunov function with respect to $Var^S \subseteq Var^C_{\mathsf{CC}_i} \subseteq Var^C_{\mathsf{CC}_{1\|2}} \subseteq Var_{\mathsf{CC}_{1\|2}}$ for $\mathsf{CC}_{1\|2}$ by inspecting each of the four constraints of Theorem 5.23 individually.*

***Constraint C1:*** *Since Constraint C1 holds for $\mathsf{CC}_i$, we know that for every location $l_i \in Loc_{\mathsf{CC}_i}$ there exists a set of variables $Var^{S_l}$ with $Var^S \subseteq Var^{S_l} \subseteq Var^C_{\mathsf{CC}_i} \subseteq Var^C_{\mathsf{CC}_{1\|2}}$ such that*

$$\forall \mathbf{x} \in Inv_{\mathsf{CC}_i}(l_i) \bullet \alpha\big(\big|\big|\mathbf{x}^{S_l}\big|\big|\big) \leq V_{\mathsf{CC}_i}(l_i, \mathbf{x}) \leq \beta\big(\big|\big|\mathbf{x}^{S_l}\big|\big|\big).$$

*Together with Invariant Inclusion of Lemma 5.35 by hypothetical syllogism, we get that for every $l \in Loc_{1\|2}$ holds*

$$\forall \mathbf{x} \in Inv_{\mathsf{CC}_{1\|2}}(l) \bullet \alpha\Big(\Big|\Big|\mathbf{x}_{\downarrow Var^{S_l}}\Big|\Big|\Big) \leq V_{\mathsf{CC}_i}\Big(l_i, \mathbf{x}_{\downarrow Var_{\mathsf{CC}_i}}\Big) \leq \beta\Big(\Big|\Big|\mathbf{x}_{\downarrow Var^{S_l}}\Big|\Big|\Big)$$

*and by choice of $V_{\mathsf{CC}_{1\|2}}$, we conclude*

$$\forall \mathbf{x} \in Inv_{\mathsf{CC}_{1\|2}}(l) \bullet \alpha\Big(\Big|\Big|\mathbf{x}_{\downarrow Var^{S_l}}\Big|\Big|\Big) \leq V_{\mathsf{CC}_{1\|2}}(l, \mathbf{x}) \leq \beta\Big(\Big|\Big|\mathbf{x}_{\downarrow Var^{S_l}}\Big|\Big|\Big).$$

*And therefore, Constraint C1 holds for $\mathsf{CC}_{1\|2}$.*

***Constraint C2:*** *Since Constraint C2 holds for $\mathsf{CC}_i$, we know that for every location $l_i \in Loc_{\mathsf{CC}_i}$ there exists a set of variables $Var^{S_l}$ with $Var^S \subseteq Var^{S_l} \subseteq Var^C_{\mathsf{CC}_i} \subseteq$*

*$Var^{C}_{CC_{1\|2}}$ such that*

$$\forall \mathbf{x} \in Inv_{CC_i}(l_i) \bullet \forall \mathbf{y} \in Flow_{CC_i}(l_i)(\mathbf{x}) \bullet \left\langle \frac{\partial V_{CC_i}(l_i,\mathbf{x})}{\partial \mathbf{x}_{CC_i}} \middle| \mathbf{y} \right\rangle \leq -\gamma(||\mathbf{x}^{S_l}||).$$

*Together with Invariant Inclusion of Lemma 5.35 by hypothetical syllogism, we get that for every $l \in Loc_{1\|2}$ holds*

$$\forall \mathbf{x} \in Inv_{CC_{1\|2}}(l) \bullet \forall \mathbf{y} \in Flow_{CC_i}(l_i)\left(\mathbf{x}_{\downarrow Var_{CC_i}}\right)\bullet \left\langle \frac{\partial V_{CC_i}\left(l_i, \mathbf{x}_{\downarrow Var_{CC_i}}\right)}{\partial \mathbf{x}_{CC_i}} \middle| \mathbf{y} \right\rangle \leq -\gamma\left(\left|\left|\mathbf{x}_{\downarrow Var^{S_l}}\right|\right|\right).$$

*Again by hypothetical syllogism with Flow Inclusion of Lemma 5.35, we get that*

$$\forall \mathbf{x} \in Inv_{CC_{1\|2}}(l) \bullet \forall \mathbf{y} \in Flow_{CC_{1\|2}}(l)(\mathbf{x})\bullet \left\langle \frac{\partial V_{CC_i}\left(l_i, \mathbf{x}_{\downarrow Var_{CC_i}}\right)}{\partial \mathbf{x}_{CC_i}} \middle| \mathbf{y}_{\downarrow Var_{CC_i}} \right\rangle \leq -\gamma\left(\left|\left|\mathbf{x}_{\downarrow Var^{S_l}}\right|\right|\right).$$

*Note, that by construction of $V_{CC_{1\|2}}$ it holds, that if $V_{CC_i}$ is differentiable in all directions of $Var_{CC_i}$, then also $V_{CC_{1\|2}}$ is differentiable in all directions of $Var_{CC_i}$ and the derivate in each "new" direction $v \in Var_{CC_{1\|2}} \setminus Var_{CC_i}$ is $0$. Together with $Var_{CC_i} \subseteq Var_{CC_{1\|2}}$, we obtain*

$$\forall \mathbf{x} \in Inv_{CC_{1\|2}}(l) \bullet \forall \mathbf{y} \in Flow_{CC_{1\|2}}(l)(\mathbf{x})\bullet \left\langle \frac{\partial V_{CC_{1\|2}}(l,\mathbf{x})}{\partial \mathbf{x}_{CC_{1\|2}}} \middle| \mathbf{y} \right\rangle \leq -\gamma\left(\left|\left|\mathbf{x}_{\downarrow Var^{S_l}}\right|\right|\right).$$

*And therefore, Constraint C1 holds for $CC_{1\|2}$.*

***Constraint C3:*** *We have to show that for all transitions $(l, G, U, l') \in Trans_{CC_{1\|2}}$ with $l = (l_1, l_2, l_{Com}, l_P)$ and $l' = (l'_1, l'_2, l'_{Com}, l'_P)$ holds*

$$\mathbf{x} \in G \wedge \mathbf{x}'_{\downarrow Var^{C}_{CC_{1\|2}}} = U(\mathbf{x}) \wedge \mathbf{x}'_{\downarrow Var^{I}_{CC_{1\|2}}} = \mathbf{x}_{\downarrow Var^{I}_{CC_{1\|2}}}$$
$$\Rightarrow V_{CC_{1\|2}}(l,\mathbf{x}') \leq V_{CC_{1\|2}}(l',\mathbf{x}).$$

*By Transition Inclusion, we know that there are four cases (one for each involved sub-component):*

*(Case A) the transition was obtained from the controller $C_i$ of component $CC_i$ and only its location has changed $l_i \neq l'_i$,*

(Case B) the transition was obtained from the other controller $\mathsf{C}_j$ of the component $\mathsf{CC}_j$ for $i \neq j$ and only its location has changed $l_j \neq l'_j$,

(Case C) the transition was obtained from the common communication channel $\mathsf{Com}$ and only its location has changed $l_{\mathsf{Com}} \neq l'_{\mathsf{Com}}$, or

(Case D) the transition was obtained from the common plant $\mathsf{P}$ and only the plant's location has changed $l_{\mathsf{P}} \neq l'_{\mathsf{P}}$.

We show that in all cases the Lyapunov function's value does not increase.

**Case A and Case D :** Suppose that $(l, G, U, l') \in Trans_{\mathsf{CC}_{1\|2}}$ was obtained either from the controller $\mathsf{C}_i$ or from the plant $\mathsf{P}$, which together form component $\mathsf{CC}_i$. By Transition Inclusion, we know there exists a transition $(l_i, G_i, U_i, l'_i) \in Trans_{\mathsf{CC}_i}$ with $l_i, l'_i \in Loc_{\mathsf{C}_i} \times Loc_{\mathsf{P}}$ such that

$$\mathbf{x} \in G \wedge \mathbf{x}'_{\downarrow Var^{C}_{\mathsf{CC}_{1\|2}}} = U(\mathbf{x}) \Rightarrow \ \mathbf{x}_{\downarrow Var_{\mathsf{CC}_i}} \in G_i \wedge \mathbf{x}'_{\downarrow Var^{C}_{\mathsf{CC}_i}} = U_i\Big(\mathbf{x}_{\downarrow Var_{\mathsf{CC}_i}}\Big) \\ \wedge \mathbf{x}'_{\downarrow Var_{\mathsf{CC}_{1\|2}} \setminus Var^{C}_{\mathsf{CC}_i}} = \mathbf{x}_{\downarrow Var_{\mathsf{CC}_{1\|2}} \setminus Var^{C}_{\mathsf{CC}_i}}. \qquad (5.5)$$

Note, that the transition was obtained from the component $\mathsf{CC}_i$ and due to syntactical restrictions its update function $U_i$ may only change the valuation of variables in $Var^{C}_{\mathsf{CC}_i}$ but not of any variable in $Var_{\mathsf{CC}_{1\|2}} \setminus Var^{C}_{\mathsf{CC}_i} \supseteq Var^{I}_{\mathsf{CC}_i}$. By $\varphi^{env}$, $\mathsf{Com}$, $\|$-composability, we know that

$$\mathsf{C}_j \parallel \mathsf{Com} \parallel \mathsf{P} \models \varphi^{env}_{1\|2} \Rightarrow \varphi^{env}_{i}.$$

Since $\varphi^{env}_{1\|2}$ holds for all $\mathbf{x} \in G$ and by Update Satisfaction, we obtain that $\mathbf{x} \in G \Rightarrow \mathbf{x}_{\downarrow Var_{\mathsf{CC}_i}} \models \varphi^{env}_{inv,i}$. Together with Formula 5.5 this leads to

$$\mathbf{x} \in G \wedge \mathbf{x}'_{\downarrow Var^{C}_{\mathsf{CC}_{1\|2}}} = U(\mathbf{x}) \Rightarrow \ \mathbf{x}_{\downarrow Var_{\mathsf{CC}_i}} \in G_i \wedge \mathbf{x}_{\downarrow Var_{\mathsf{CC}_i}} \models \varphi^{env}_{inv,i} \\ \wedge \mathbf{x}'_{\downarrow Var^{C}_{\mathsf{CC}_i}} = U_i\Big(\mathbf{x}_{\downarrow Var_{\mathsf{CC}_i}}\Big) \wedge \mathbf{x}'_{\downarrow Var^{I}_{\mathsf{CC}_i}} = \mathbf{x}_{\downarrow Var^{I}_{\mathsf{CC}_i}}.$$

Since Constraint C3 holds for $\mathsf{CC}_i$, we know that for all transitions $(l_i, G_i, U_i, l'_i) \in Trans_{\mathsf{CC}_i}$, it holds that

$$\forall \mathbf{x} \in G_i, \mathbf{x}' \bullet \quad \mathbf{x}_{\downarrow Var_{\mathsf{CC}_i}} \models \varphi^{env}_{inv,i} \wedge \mathbf{x}'_{\downarrow Var^{C}_{\mathsf{CC}_i}} = U_i(\mathbf{x}) \wedge \mathbf{x}'_{\downarrow Var^{I}_{\mathsf{CC}_i}} = \mathbf{x}_{\downarrow Var^{I}_{\mathsf{CC}_i}} \\ \Rightarrow V_{\mathsf{CC}_i}(l_i, \mathbf{x}') \leq V_{\mathsf{CC}_i}(l'_i, \mathbf{x})$$

and conclude that for the transition $(l, G, U, l')$, it holds that

$$\mathbf{x} \in G \wedge \mathbf{x}'_{\downarrow Var^{C}_{\mathsf{CC}_{1\|2}}} = U(\mathbf{x}) \Rightarrow V_{\mathsf{CC}_i}(l_i, \mathbf{x}') \leq V_{\mathsf{CC}_i}(l'_i, \mathbf{x}).$$

*and by choice of $V_{\mathsf{CC}_{1\|2}}$, we have*

$$\mathbf{x} \in G \wedge \mathbf{x}'_{\downarrow Var^C_{\mathsf{CC}_{1\|2}}} = U(\mathbf{x}) \Rightarrow V_{\mathsf{CC}_{1\|2}}(l, \mathbf{x}') \leq V_{\mathsf{CC}_{1\|2}}(l', \mathbf{x}).$$

***Case B* and *Case C*:** *Suppose that $(l, G, U, l') \in Trans_{\mathsf{CC}_{1\|2}}$ was obtained from either the communication channel $\mathsf{Com}$ or the other controller $\mathsf{C}_j$. By Transition Inclusion, we know there exists a transition $(l_\star, G_\star, U_\star, l'_\star) \in Trans_\star$ with $\star \in \{\mathsf{C}_j, \mathsf{Com}\}$ such that*

$$\begin{aligned}\mathbf{x} \in G \wedge \mathbf{x}'_{\downarrow Var^C_{\mathsf{CC}_{1\|2}}} = U(\mathbf{x}) \Rightarrow\ & \mathbf{x}_{\downarrow Var_\star} \in G_\star \\ & \wedge\ \mathbf{x}'_{\downarrow Var^C_\star} = U_\star(\mathbf{x}_{\downarrow Var_\star}) \\ & \wedge\ \mathbf{x}'_{\downarrow Var_{\mathsf{CC}_{1\|2}} \setminus Var^C_\star} = \mathbf{x}_{\downarrow Var_{\mathsf{CC}_{1\|2}} \setminus Var^C_\star}.\end{aligned}$$

*Note, that the transition was obtained from $\star$ (the communication channel $\mathsf{Com}$ or the other controller $\mathsf{C}_j$) and due to syntactical restrictions its update function $U_\star$ may only change the valuation of variables in $Var^C_\star$ but not of any variable in $Var_{\mathsf{CC}_{1\|2}} \setminus Var^C_\star \supseteq Var^C_{\mathsf{CC}_i}$. By $\varphi^{env}$, $\mathsf{Com}$, $\|$-composability, we know that*

$$\mathsf{C}_j \parallel \mathsf{Com} \parallel \mathsf{P} \models \varphi^{env}_{1\|2} \Rightarrow \varphi^{env}_i.$$

*Since $\varphi^{env}_{1\|2}$ holds for all $\mathbf{x} \in G$ and by Update Satisfaction, we obtain that $\mathbf{x} \in G \Rightarrow \mathbf{x}_{\downarrow Var_{\mathsf{CC}_i}} \models \varphi^{env}_{inv,i}$ and $\mathbf{x} \in G \Rightarrow \mathbf{x}_{\downarrow Var^I_{\mathsf{CC}_i}}, \mathbf{x}'_{\downarrow Var^I_{\mathsf{CC}_i}} \models \varphi^{env}_{dscr,i}$. This leads to*

$$\begin{aligned}\mathbf{x} \in G \wedge \mathbf{x}'_{\downarrow Var^C_{\mathsf{CC}_{1\|2}}} = U(\mathbf{x}) \Rightarrow\ & \left(\mathbf{x}_{\downarrow Var_{\mathsf{CC}_i}} \models \varphi^{env}_{inv,i}\right) \\ & \wedge\ \left(\mathbf{x}_{\downarrow Var^I_{\mathsf{CC}_i}}, \mathbf{x}'_{\downarrow Var^I_{\mathsf{CC}_i}} \models \varphi^{env}_{dscr,i}\right) \\ & \wedge\ \mathbf{x}'_{\downarrow Var_{\mathsf{CC}_{1\|2}} \setminus Var^C_\star} = \mathbf{x}_{\downarrow Var_{\mathsf{CC}_{1\|2}} \setminus Var^C_\star}.\end{aligned}$$

*Together with the fact that the part of the location corresponding to component $\mathsf{CC}_i$ does not change, i. e., $l = (l_1, l_2, l_{\mathsf{Com}}, l_{\mathsf{P}})$ and $l' = (l'_1, l'_2, l'_{\mathsf{Com}}, l_{\mathsf{P}})$ with $l_i = l'_i$, we can make use of Constraint C4,i. e.,*

$$\begin{aligned}& \left(\mathbf{x}_{\downarrow Var_{\mathsf{CC}_i}} \models \varphi^{env}_{inv,i}\right) \\ & \wedge \left(\mathbf{x}'_{\downarrow Var^I_{\mathsf{CC}_i}}, \mathbf{x}_{\downarrow Var^I_{\mathsf{CC}_i}} \models \varphi^{env}_{dscr,i}\right) \wedge \mathbf{x}'_{\downarrow Var_{\mathsf{CC}_{1\|2}} \setminus Var^C_\star} = \mathbf{x}_{\downarrow Var_{\mathsf{CC}_{1\|2}} \setminus Var^C_\star} \\ \Rightarrow & V_{\mathsf{CC}_i}((l_i, l_{\mathsf{P}}), \mathbf{x}') \leq V_{\mathsf{CC}_i}((l_i, l_{\mathsf{P}}), \mathbf{x})\end{aligned}$$

*and conclude that for the transition $(l, G, U, l')$, it holds that*

$$\mathbf{x} \in G \wedge \mathbf{x}'_{\downarrow Var^C_{CC_{1\|2}}} = U(\mathbf{x}) \Rightarrow V_{CC_i}\Big((l_i, l_P), \mathbf{x}'_{\downarrow Var_{CC_i}}\Big) \leq V_{CC_i}\Big((l_i, l_P), \mathbf{x}_{\downarrow Var_{CC_i}}\Big).$$

*Which by choice of $V_{CC_{1\|2}}$ leads to*

$$\mathbf{x} \in G \wedge \mathbf{x}'_{\downarrow Var^C_{CC_{1\|2}}} = U(\mathbf{x}) \Rightarrow V_{CC_{1\|2}}(l, \mathbf{x}') \leq V_{CC_{1\|2}}(l', \mathbf{x}).$$

*We have inspected every case and conclude that Constraint C3, it holds for $CC_{1\|2}$.*

***Constraint C4:*** *We have to show that for every location $l \in Loc_{CC_{1\|2}}$ holds that for updates performed by the common environment on variables $Var^I_{CC_{1\|2}}$*

$$\begin{aligned}\forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}_{CC_{1\|2}} \bullet \qquad & \left(\mathbf{x} \models \varphi^{env}_{inv,1\|2}\right) \wedge \mathbf{x}'_{\downarrow Var^C_{CC_{1\|2}}} = \mathbf{x}_{\downarrow Var^C_{CC_{1\|2}}} \\ & \wedge \left(\mathbf{x}_{\downarrow Var^I_{CC_{1\|2}}}, \mathbf{x}'_{\downarrow Var^I_{CC_{1\|2}}} \models \varphi^{env}_{dscr,1\|2}\right) \\ & \Rightarrow V_{CC_{1\|2}}(l, \mathbf{x}') \leq V_{CC_{1\|2}}(l, \mathbf{x})\end{aligned}$$

*is satisfied.*

*The premise says*

$$\begin{aligned}\forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}_{CC_{1\|2}} \bullet \quad & \left(\mathbf{x} \models \varphi^{env}_{inv,1\|2}\right) \wedge \mathbf{x}'_{\downarrow Var^C_{CC_{1\|2}}} = \mathbf{x}_{\downarrow Var^C_{CC_{1\|2}}} \\ & \wedge \left(\mathbf{x}_{\downarrow Var^I_{CC_{1\|2}}}, \mathbf{x}'_{\downarrow Var^I_{CC_{1\|2}}} \models \varphi^{env}_{dscr,1\|2}\right)\end{aligned}$$

*by External Update Inclusion, i. e., $\forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}_{CC_{1\|2}}$*

$$\begin{aligned}& \left(\mathbf{x} \models \varphi^{env}_{inv,1\|2}\right) \wedge \left(\mathbf{x}_{\downarrow Var^I_{CC_{1\|2}}}, \mathbf{x}'_{\downarrow Var^I_{CC_{1\|2}}} \models \varphi^{env}_{dscr,1\|2}\right) \\ \Rightarrow\ & \forall i \bullet \left(\mathbf{x}_{\downarrow Var_{CC_i}} \models \varphi^{env}_{inv,i}\right) \wedge \left(\mathbf{x}^I{}_{\downarrow Var^I_{CC_i}}, \mathbf{x}^{I\prime}{}_{\downarrow Var^I_{CC_i}} \models \varphi^{env}_{dscr,i}\right)\end{aligned}$$

*we obtain*

$$\begin{aligned}\forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}_{CC_{1\|2}} \bullet \qquad & \left(\mathbf{x} \models \varphi^{env}_{inv,1\|2}\right) \wedge \mathbf{x}'_{\downarrow Var^C_{CC_{1\|2}}} = \mathbf{x}_{\downarrow Var^C_{CC_{1\|2}}} \\ & \wedge \left(\mathbf{x}_{\downarrow Var^I_{CC_{1\|2}}}, \mathbf{x}'_{\downarrow Var^I_{CC_{1\|2}}} \models \varphi^{env}_{dscr,1\|2}\right) \\ \Rightarrow \quad & \left(\mathbf{x}_{\downarrow Var_{CC_i}} \models \varphi^{env}_{inv,i}\right) \wedge \mathbf{x}'_{\downarrow Var^C_{CC_i}} = \mathbf{x}_{\downarrow Var^C_{CC_i}} \\ & \wedge \left(\mathbf{x}^I{}_{\downarrow Var^I_{CC_i}}, \mathbf{x}^{I\prime}{}_{\downarrow Var^I_{CC_i}} \models \varphi^{env}_{dscr,i}\right).\end{aligned}$$

*Since the location $l$ of $\mathsf{CC}_{1\|2}$ does not change during external updates and Constraint C4 holds for $\mathsf{CC}_i$, we know for the part of the location $l_i \in Loc_{\mathsf{CC}_i}$ corresponding to $\mathsf{CC}_i$ there exists a set of variables $Var^{S_l}$ with $Var^S \subseteq Var^{S_l} \subseteq Var^C_{\mathsf{CC}_i} \subseteq Var^C_{\mathsf{CC}_{1\|2}}$ such that*

$$\begin{aligned}\forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}_{\mathsf{CC}_i} \bullet \quad & \left(\mathbf{x}_{\downarrow Var_{\mathsf{CC}_i}} \models \varphi^{env}_{inv,i}\right) \wedge \mathbf{x}'_{\downarrow Var^C_{\mathsf{CC}_i}} = \mathbf{x}_{\downarrow Var^C_{\mathsf{CC}_i}} \\ & \wedge \left(\mathbf{x}_{\downarrow Var^I_{\mathsf{CC}_i}}, \mathbf{x}'_{\downarrow Var^I_{\mathsf{CC}_i}} \models \varphi^{env}_{dscr,i}\right) \\ & \Rightarrow V_{\mathsf{CC}_i}(l_i, \mathbf{x}') \leq V_{\mathsf{CC}_i}(l_i, \mathbf{x}).\end{aligned}$$

*Together with $Var^C_{\mathsf{CC}_i} \subseteq Var^C_{\mathsf{CC}_{1\|2}}$, this leads to*

$$\begin{aligned}\forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}_{\mathsf{CC}_{1\|2}} \bullet \quad & \left(\mathbf{x} \models \varphi^{env}_{inv,1\|2}\right) \wedge \mathbf{x}'_{\downarrow Var^C_{\mathsf{CC}_{1\|2}}} = \mathbf{x}_{\downarrow Var^C_{\mathsf{CC}_{1\|2}}} \\ & \wedge \left(\mathbf{x}_{\downarrow Var^I_{\mathsf{CC}_{1\|2}}}, \mathbf{x}'_{\downarrow Var^I_{\mathsf{CC}_{1\|2}}} \models \varphi^{env}_{dscr,1\|2}\right) \\ & \Rightarrow V_{\mathsf{CC}_i}\left(l_i, \mathbf{x}'_{\downarrow Var_{\mathsf{CC}_i}}\right) \leq V_{\mathsf{CC}_i}\left(l_i, \mathbf{x}_{\downarrow Var_{\mathsf{CC}_i}}\right)\end{aligned}$$

*and by choice of $V_{\mathsf{CC}_{1\|2}}$, we conclude*

$$\begin{aligned}\forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}_{\mathsf{CC}_{1\|2}} \bullet \quad & \left(\mathbf{x} \models \varphi^{env}_{inv,1\|2}\right) \wedge \mathbf{x}'_{\downarrow Var^C_{\mathsf{CC}_{1\|2}}} = \mathbf{x}_{\downarrow Var^C_{\mathsf{CC}_{1\|2}}} \\ & \wedge \left(\mathbf{x}_{\downarrow Var^I_{\mathsf{CC}_{1\|2}}}, \mathbf{x}'_{\downarrow Var^I_{\mathsf{CC}_{1\|2}}} \models \varphi^{env}_{dscr,1\|2}\right) \\ & \Rightarrow V_{\mathsf{CC}_{1\|2}}(l, \mathbf{x}') \leq V_{\mathsf{CC}_{1\|2}}(l, \mathbf{x}).\end{aligned}$$

*And therefore, Constraint C4 holds for $\mathsf{CC}_{1\|2}$.*

*This completes our proof as we have shown that all constraints are satisfied.* □

In the following, we will show how to construct a Lyapunov function with respect to the union of two sets of variables $Var^S = Var^S_i \cup Var^S_2$ based on Lyapunov functions for the individual sets $Var^S_1$ and $Var^S_2$. The ideas we are using are closely related to sum-type (or sum-separable) Lyapunov functions [Ito+11; Ito+13].

First, we introduce some technical straight forward properties of class-$\mathcal{K}, \mathcal{K}^\infty$ functions.

**Lemma 5.37 (Properties of class-$\mathcal{K}, \mathcal{K}^\infty$ functions [Kel14])**
*Given two class-$\mathcal{K}$ functions $\alpha_1, \alpha_2 \in \mathcal{K}$. It holds for each aggregation,*

$$\alpha(s) := \alpha_1(s) + \alpha_2(s), \tag{5.6}$$

$$\alpha(s) := \max(\alpha_1(s), \alpha_2(s)), \text{ or} \tag{5.7}$$

$$\alpha(s) := \min(\alpha_1(s), \alpha_2(s)) \tag{5.8}$$

*that $\alpha \in \mathcal{K}$. Further, if $\alpha_1, \alpha_2 \in \mathcal{K}^\infty$, then $\alpha \in \mathcal{K}^\infty$ in case of summation (i. e., Equation 5.6) or maximization (i. e., Equation 5.7).*

Now, we recall some basic properties of the Euclidean norm, inner products, and the gradient of a function.

**Lemma 5.38 (Bounds on the Euclidean Norm for Sub-vectors)**
*Given a set of variables $Var$. Let $Var_i \subseteq Var$ for $i = 1, 2$ be two subsets with $Var_i \cup Var_i = Var$. For all $\mathbf{x} \in \mathbb{R}^{|Var|}$, it holds that $||\mathbf{x}|| \leq ||\mathbf{x}_{\downarrow Var_1}|| + ||\mathbf{x}_{\downarrow Var_2}|| \leq 2||\mathbf{x}||$. Further, it holds that $||\mathbf{x}_{\downarrow Var_1}|| \leq ||\mathbf{x}||$.*

**Lemma 5.39 (Properties of the Inner Product [Fis97, p.262])**
*Let $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{R}^n$ be vectors and $a \in \mathbb{R}$ be a scalar. The inner product $\langle \cdot \mid \cdot \rangle$ satisfies*

$$\begin{aligned} \langle a\mathbf{x} \mid \mathbf{z} \rangle &= a\langle \mathbf{x} \mid \mathbf{z} \rangle \\ \langle \mathbf{x} + \mathbf{y} \mid \mathbf{z} \rangle &= \langle \mathbf{x} \mid \mathbf{z} \rangle + \langle \mathbf{y} \mid \mathbf{z} \rangle \end{aligned} \qquad \text{(Linearity)}$$

$$\langle \mathbf{x} \mid \mathbf{z} \rangle = \langle \mathbf{z} \mid \mathbf{x} \rangle \qquad \text{(Symmetry)}$$

$$\begin{aligned} \langle \mathbf{x} \mid \mathbf{x} \rangle &\geq 0 \\ \langle \mathbf{x} \mid \mathbf{x} \rangle &= 0 \textit{ iff } \mathbf{x} = \mathbf{0}. \end{aligned} \qquad \text{(Positive-Definiteness)}$$

**Lemma 5.40 (Properties of the Gradient Operator)**
*Let $f, g : \mathbb{R}^n \to \mathbb{R}$ be differentiable functions and $a \in \mathbb{R}$ be a scalar. The gradient $(\nabla f)_\mathbf{x} = \frac{\partial f}{\partial \mathbf{x}}$ satisfies*

$$\begin{aligned} a(\nabla f(\mathbf{x}))_\mathbf{x} &= (\nabla(a \cdot f(\mathbf{x})))_\mathbf{x} \\ (\nabla(f(\mathbf{x}) + g(x)))_\mathbf{x} &= (\nabla f(\mathbf{x}))_\mathbf{x} + (\nabla g(x))_\mathbf{x}. \end{aligned} \qquad \text{(Linearity)}$$

This finally allows us to compose Lyapunov functions as stated by the following theorem in case of global exponential stability. In this case, the class-$\mathcal{K}^\infty$ functions in Theorem 5.20 have the form $\alpha(s) = \alpha \cdot s$, $\beta(s) = \beta \cdot s$, and $\gamma(s) = \gamma \cdot s$. Therefore, linearity also holds for the functions $\alpha, \beta, \gamma$, i. e., $\alpha(s_1) + \alpha(s_2) = \alpha \cdot s_1 + \alpha \cdot s_2 = \alpha \cdot (s_1 + s_2) = \alpha(s_1 + s_2)$ and analogous for $\beta$ and $\gamma$.

**Theorem 5.41**
*Let*

$$\mathcal{A} = (Loc, Var, Trans, Flow, Inv, Inits)$$

*be a hybrid I/O automaton, and let $\varphi^{env}$ be an environmental predicate on $Var^E$. If $V_i : Loc \times \mathcal{X} \to \mathbb{R}$, for $i = 1, 2$ are two global GES-Lyapunov functions with respect to $Var_i^S \subseteq Var^C$, then $V\colon Loc \times \mathcal{X} \to \mathbb{R}$ with $V(l, \mathbf{x}) = V_1(l, \mathbf{x}) + V_2(l, \mathbf{x})$ is a global*

*Lyapunov function with respect to* $Var_1^S \cup Var_2^S \subseteq Var^C$.

**Proof.**
*Let* $\mathcal{A} = (Loc, Var, Trans, Flow, Inv, Inits)$ *be a hybrid I/O automaton, and let* $\varphi^{env}$ *be an environmental predicate on* $Var^E$. *Let* $V_i : Loc \times \mathcal{X} \to \mathbb{R}$, *for* $i = 1, 2$ *are two global Lyapunov functions with respect to* $Var_i^S \subseteq Var^C$. *We have to show that* $V : Loc \times \mathcal{X} \to \mathbb{R}$ *with* $V(l, \mathbf{x}) = V_1(l, \mathbf{x}) + V_2(l, \mathbf{x})$ *is a global Lyapunov function with respect to* $Var_1^S \cup Var_2^S \subseteq Var^C$.

*Since each* $V_i$ *is a global Lyapunov function with respect to* $Var_i^S$ *for* $i = 1, 2$, *we know that for each* $l \in Loc$ *there exist two sets of variables* $Var_1^{S_l}$, $Var_2^{S_l}$ *with* $Var_i^S \subseteq Var_i^{S_l} \subseteq Var^C$ *such that constraints C1 to C4 of Theorem 5.20 hold. Since* $Var^S = \cup_i Var_i^S \subseteq \cup_i Var_i^{S_l} \subseteq Var^C$, *we choose* $Var^{S_l} := Var_1^{S_l} \cup Var_2^{S_l}$ *and investigate each constraint individually.*

***Constraint C1:*** *For each location* $l \in Loc$, *we have to show existence of two class-*$\mathcal{K}^\infty$ *functions* $\alpha$ *and* $\beta$ *such that*

$$\forall \mathbf{x} \in Inv(l) \bullet \mathbf{x} \models \varphi^{env} \Rightarrow \alpha(||\mathbf{x}^{S_l}||) \leq V(l, \mathbf{x}) \leq \beta(||\mathbf{x}^{S_l}||).$$

*Wlog, we assume* $l$ *to be arbitrary but fixed.*

*Since* $V_i$ *is a global Lyapunov functions with respect to* $Var_i^S$ *for* $i = 1, 2$, *we know that Constraint C1 holds for* $V_i$: *for each* $l \in Loc$, *there exists class-*$\mathcal{K}^\infty$ *functions* $\alpha_i$ *and* $\beta_i$ *such that*

$$\forall \mathbf{x} \in Inv(l) \bullet \forall i \bullet \mathbf{x} \models \varphi^{env} \Rightarrow \alpha_i\left(\left|\left|\mathbf{x}_i^{S_l}\right|\right|\right) \leq V_i(l, \mathbf{x}) \leq \beta_i\left(\left|\left|\mathbf{x}_i^{S_l}\right|\right|\right).$$

*Since* $\leq$ *is compatible with addition, we can write*

$$\forall \mathbf{x} \in Inv(l) \bullet \mathbf{x} \models \varphi^{env} \Rightarrow \sum_i \alpha_i\left(\left|\left|\mathbf{x}_i^{S_l}\right|\right|\right) \leq \sum_i V_i(l, \mathbf{x}) \leq \sum_i \beta_i\left(\left|\left|\mathbf{x}_i^{S_l}\right|\right|\right)$$

*By choice of* $Var^{S_l}$, $V$, *and by Lemma 5.38, we know* $\beta_i\left(\left|\left|\mathbf{x}_i^{S_l}\right|\right|\right) \leq \beta_i(||\mathbf{x}^{S_l}||)$ *for* $i \in \{1, 2\}$, *and obtain*

$$\forall \mathbf{x} \in Inv(l) \bullet \mathbf{x} \models \varphi^{env} \Rightarrow \sum_i \alpha_i\left(\left|\left|\mathbf{x}_i^{S_l}\right|\right|\right) \leq V(l, \mathbf{x}) \leq \sum_i \beta_i(||\mathbf{x}^{S_l}||)$$

*We choose* $\alpha(s) := \alpha \cdot s$ *with* $\alpha = \min(\alpha_1, \alpha_2)$ *which is obviously a class-*$\mathcal{K}^\infty$ *function and* $\beta(s) := \sum_i \beta_i(s)$ *which is a class-*$\mathcal{K}^\infty$ *function according to Lemma 5.37. By choice of* $V$, *we obtain*

$$\forall \mathbf{x} \in Inv(l) \bullet \mathbf{x} \models \varphi^{env} \Rightarrow \sum_i \alpha\left(\left|\left|\mathbf{x}_i^{S_l}\right|\right|\right) \leq V(l, \mathbf{x}) \leq \beta(||\mathbf{x}^{S_l}||)$$

*Since $\alpha(s)$ is linear in $s$, it holds that $\sum_i \alpha\left(\left\|\mathbf{x}_i^{S_l}\right\|\right) = \alpha(\sum_i \left\|\mathbf{x}_i^{S_l}\right\|)$ and we can write*

$$\forall \mathbf{x} \in \mathit{Inv}(l) \bullet \mathbf{x} \models \varphi^{env} \Rightarrow \alpha(\sum_i \left\|\mathbf{x}_i^{S_l}\right\| \leq V(l,\mathbf{x}) \leq \beta(\left\|\mathbf{x}^{S_l}\right\|)$$

*By Lemma 5.38, it holds that $\left\|\mathbf{x}^{S_l}\right\| \leq \sum_i \left\|\mathbf{x}_i^{S_l}\right\|$ and we conclude*

$$\forall \mathbf{x} \in \mathit{Inv}(l) \bullet \mathbf{x} \models \varphi^{env} \Rightarrow \alpha(\left\|\mathbf{x}^{S_l}\right\| \leq V(l,\mathbf{x}) \leq \beta(\left\|\mathbf{x}^{S_l}\right\|).$$

***Constraint C2:*** *We have to show that for each location $l \in Loc$, there exists a class $\mathcal{K}^\infty$ function $\gamma$ such that*

$$\forall \mathbf{x} \in \mathit{Inv}(l) \bullet \mathbf{x} \models \varphi^{env}_{inv} \Rightarrow \dot{V}(l,\mathbf{x}) \leq -\gamma(\left\|\mathbf{x}^{S_l}\right\|)$$

*for each $\dot{V}(l,\mathbf{x}) \in \left\{ \left\langle \begin{bmatrix} \frac{\partial V(l,\mathbf{x})}{\partial \mathbf{x}^C} \\ \frac{\partial V(l,\mathbf{x})}{\partial \mathbf{y}^I} \end{bmatrix} \middle| \begin{bmatrix} f(\mathbf{x}) \\ \dot{\mathbf{y}}^I \end{bmatrix} \right\rangle \middle| \begin{bmatrix} f(\mathbf{x}) \in \mathit{Flow}(l) \\ \wedge \dot{\mathbf{y}}^I, \mathbf{x}^I \models \varphi^{env}_{flow} \end{bmatrix} \right\}$ or in a more compact form*

$$\forall \mathbf{x} \in \mathit{Inv}(l) \bullet \mathbf{x} \models \varphi^{env}_{inv} \Rightarrow \left\langle \frac{\partial V(l,\mathbf{x})}{\partial \mathbf{x}} \middle| g(\mathbf{x}) \right\rangle \leq -\gamma(\left\|\mathbf{x}^{S_l}\right\|)$$

*for each $g(x) \in \left\{ \begin{bmatrix} f(\mathbf{x}) \in \mathit{Flow}(l) \\ \wedge \dot{\mathbf{y}}^I, \mathbf{x}^I \models \varphi^{env}_{flow} \end{bmatrix} \right\}$. Wlog, we assume $l, g(x)$ to be arbitrary but fixed.*

*Since $V_i$ is a global Lyapunov functions with respect to $Var_i^S$ for $i = 1, 2$, we know that Constraint C2 holds for $V_i$: there exists a class-$\mathcal{K}^\infty$ function $\gamma_i$ such that*

$$\forall \mathbf{x} \in \mathit{Inv}(l) \bullet \forall i \bullet \mathbf{x} \models \varphi^{env}_{inv} \Rightarrow \left\langle \frac{\partial V_i(l,\mathbf{x})}{\partial \mathbf{x}} \middle| g(\mathbf{x}) \right\rangle \leq -\gamma_i\left(\left\|\mathbf{x}_i^{S_l}\right\|\right).$$

*Since $\leq$ is compatible with addition, we can write*

$$\forall \mathbf{x} \in \mathit{Inv}(l) \bullet \mathbf{x} \models \varphi^{env}_{inv} \Rightarrow \sum_i \left\langle \frac{\partial V_i(l,\mathbf{x})}{\partial \mathbf{x}} \middle| g(\mathbf{x}) \right\rangle \leq -\sum_i \gamma_i\left(\left\|\mathbf{x}_i^{S_l}\right\|\right).$$

*We choose $\gamma(s) := \gamma \cdot s$ with $\gamma = \min(\gamma_1, \gamma_2)$ which is a class-$\mathcal{K}^\infty$ function and we obtain*

$$\forall \mathbf{x} \in \mathit{Inv}(l) \bullet \mathbf{x} \models \varphi^{env}_{inv} \Rightarrow \sum_i \left\langle \frac{\partial V_i(l,\mathbf{x})}{\partial \mathbf{x}} \middle| g(\mathbf{x}) \right\rangle \leq -\sum_i \gamma(\left\|\mathbf{x}_i^{S_l}\right\|).$$

*Since $\gamma(s)$ is linear in $s$, it holds that $\sum_i \gamma\left(\left|\left|\mathbf{x}_i^{S_l}\right|\right|\right) = \gamma(\sum_i \left|\left|\mathbf{x}_i^{S_l}\right|\right|)$. Additionally, by linearity (Lemma 5.39 and Lemma 5.40), we can write*

$$\forall \mathbf{x} \in Inv(l) \bullet \mathbf{x} \models \varphi_{inv}^{env} \Rightarrow \left\langle \frac{\partial \sum_i V_i(l,\mathbf{x})}{\partial \mathbf{x}} \,\middle|\, g(\mathbf{x}) \right\rangle \leq -\gamma(\sum_i \left|\left|\mathbf{x}_i^{S_l}\right|\right|).$$

*By Lemma 5.38, it holds that $\left|\left|\mathbf{x}^{S_l}\right|\right| \leq \sum_i \left|\left|\mathbf{x}_i^{S_l}\right|\right|$ together with the choice of $V$, we conclude*

$$\forall \mathbf{x} \in Inv(l) \bullet \mathbf{x} \models \varphi_{inv}^{env} \Rightarrow \left\langle \frac{\partial V(l,\mathbf{x})}{\partial \mathbf{x}} \,\middle|\, g(\mathbf{x}) \right\rangle \leq -\gamma(\left|\left|\mathbf{x}^{S_l}\right|\right|).$$

***Constraint C3:*** *We have to show that for each transition $\tau = (l, G, U, l') \in Trans$ holds*

$$\forall \mathbf{x} \in G, \mathbf{x}' \in \mathcal{X} \bullet \left(\mathbf{x} \models \varphi_{inv}^{env} \wedge \mathbf{x}^{C'} = U(\mathbf{x}) \wedge \mathbf{x}^{I'} = \mathbf{x}^{I}\right) \Rightarrow V(l', \mathbf{x}') \leq V(l, \mathbf{x}).$$

*Wlog, we assume $\tau$ to be arbitrary but fixed.*

*Since $V_i$ is a global Lyapunov functions with respect to $Var_i^S$ for $i = 1, 2$, we know that Constraint C3 holds for $V_i$:*

$$\begin{aligned}\forall \mathbf{x} \in G, \mathbf{x}' \in \mathcal{X} \bullet \forall i \bullet \quad & \left(\mathbf{x} \models \varphi_{inv}^{env} \wedge \mathbf{x}^{C'} = U(\mathbf{x}) \wedge \mathbf{x}^{I'} = \mathbf{x}^{I}\right) \\ & \Rightarrow V_i(l', \mathbf{x}') \leq V_i(l, \mathbf{x}).\end{aligned}$$

*Since $\leq$ is compatible with addition, we can write*

$$\begin{aligned}\forall \mathbf{x} \in G, \mathbf{x}' \in \mathcal{X} \bullet \quad & \left(\mathbf{x} \models \varphi_{inv}^{env} \wedge \mathbf{x}^{C'} = U(\mathbf{x}) \wedge \mathbf{x}^{I'} = \mathbf{x}^{I}\right) \\ & \Rightarrow \sum_i V_i(l', \mathbf{x}') \leq \sum_i V_i(l, \mathbf{x}).\end{aligned}$$

*By choice of $V$, we conclude*

$$\begin{aligned}\forall \mathbf{x} \in G, \mathbf{x}' \in \mathcal{X} \bullet \quad & \left(\mathbf{x} \models \varphi_{inv}^{env} \wedge \mathbf{x}^{C'} = U(\mathbf{x}) \wedge \mathbf{x}^{I'} = \mathbf{x}^{I}\right) \\ & \Rightarrow V(l', \mathbf{x}') \leq V(l, \mathbf{x}).\end{aligned}$$

***Constraint C4:*** *We have to show that for each external discrete update and each location $l \in Loc$, it holds that*

$$\forall \mathbf{x}, \mathbf{x}' \in \mathcal{X} \bullet \left(\mathbf{x} \models \varphi_{inv}^{env} \wedge \mathbf{x}^{C'} = \mathbf{x}^{C} \wedge \mathbf{x}^{I'}, \mathbf{x}^{I} \models \varphi_{dscr}^{env}\right) \Rightarrow V(l, \mathbf{x}') \leq V(l, \mathbf{x}).$$

*Since $V_i$ is a global Lyapunov functions with respect to $Var_i^S$ for $i = 1, 2$, we know that Constraint C4 holds for $V_i$:*

$$\forall \mathbf{x}, \mathbf{x}' \in \mathcal{X} \bullet \left(\mathbf{x} \models \varphi^{env}_{inv} \wedge \mathbf{x}^{C\prime} = \mathbf{x}^{C} \wedge \mathbf{x}^{I\prime}, \mathbf{x}^{I} \models \varphi^{env}_{dscr}\right) \Rightarrow V_i(l, \mathbf{x}') \leq V_i(l, \mathbf{x}).$$

*Since $\leq$ is compatible with addition, we can write*

$$\begin{aligned} \forall \mathbf{x}, \mathbf{x}' \in \mathcal{X} \bullet \quad & \left(\mathbf{x} \models \varphi^{env}_{inv} \wedge \mathbf{x}^{C\prime} = \mathbf{x}^{C} \wedge \mathbf{x}^{I\prime}, \mathbf{x}^{I} \models \varphi^{env}_{dscr}\right) \\ & \Rightarrow \sum_i V_i(l, \mathbf{x}') \leq \sum_i V_i(l, \mathbf{x}). \end{aligned}$$

*By choice of $V$, we conclude*

$$\begin{aligned} \forall \mathbf{x}, \mathbf{x}' \in \mathcal{X} \bullet \quad & \left(\mathbf{x} \models \varphi^{env}_{inv} \wedge \mathbf{x}^{C\prime} = \mathbf{x}^{C} \wedge \mathbf{x}^{I\prime}, \mathbf{x}^{I} \models \varphi^{env}_{dscr}\right) \\ & \Rightarrow V(l, \mathbf{x}') \leq V(l, \mathbf{x}). \end{aligned}$$

*This completes the proof because we have shown that all constraints are satisfied in all cases.* □

## 5.4 Advanced Driver Assistant System: A Case Study

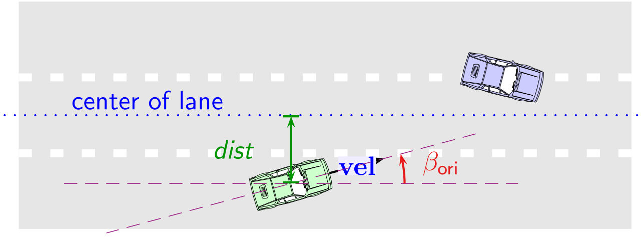


Figure 5.9: The plant: a car on the road

As an example application for the framework, we took a loosely coupled control application and designed an advanced driver assistance system (ADAS) [Dam+14; MHR17; DMR14]. The case study takes a car on the road as its basis. The parameters are

- the car's velocity $vel_{\text{cur}}$,
- the car's distance to the center of the lane $dist_{\text{cur}}$,
- the car's orientation relative to the direction of the lane $\beta_{\text{ori}}$.

The car can be controlled by

- the car's acceleration $acc$ with disturbance $s$,

- the car's change of the orientation $\beta_{\mathsf{steer}}$ which roughly corresponds to the effect of a steering wheel.

This view on the plant is visualized in Figure 5.9. We briefly introduce the main requirements on the system: The ADAS has to

**(Obj1)** maintain a centrifugal force comfortable for a driver,

**(Obj2)** bring and then keep the car on the center of its lane,

**(Obj3)** control the speed thereby considering driver requests for a certain speed value.

Although not formally introduced in this context, our framework makes use of interfaces to annotate properties of a controller. These annotations involve assumptions on the controller's environment and guarantees about a controller's service (on the plant) together with events which a controller is able to receive or to send. To avoid circular assume-guarantee reasoning, we require a controller to continuously maintain its guarantees while its assumptions are satisfied and even a short time after which its assumptions are violated. That way controllers have to be implemented robustly with respect to changes of their environment.

> **Convention 5.42 (Guarantees Do Have a Forward-Window.)**
> *In this thesis, in order to increase readability, we will omit this detail and whenever we say* a controller C guarantees $\varphi^{guara}$ given its assumption $\varphi^{env}$ holds, *we mean, that* C *indeed guarantees* $\varphi^{guara}$ *as long as* $\varphi^{env}$ *holds and for a duration* $\epsilon > 0$ *after which* $\varphi^{env}$ *has been violated.* ◁

Requiring a component to keep up its guarantees for a short duration after which the assumptions are violated is needed for the composition of safety properties. This slightly stronger requirement allows us to prevent circular reasoning when composing safety properties. However, this is not exploited during the composition of stability properties as our stability notion rely on (already established) safety assumptions only. Or stated differently, the framework does not support convergence assumptions. For example, we cannot handle the case where component A converges as long as component B converges.

In the framework, events are implemented using communication channels. Each *event* has the form $(\varphi^{prom}, \varphi^{assm}, \mathit{dueTime}, \mathit{duration})$ where

- $\varphi^{prom}$ is a predicate specifying a time-bounded promise,
- $\varphi^{assm}$ is a predicate specifying a time-bounded assumption,
- *dueTime* is a time duration before $\varphi^{prom}$ and $\varphi^{assm}$ become active, and
- *duration* is the duration after *dueTime* for which $\varphi^{prom}$ and $\varphi^{assm}$ are active.

On the sending side — where the event is called an *out-event* — $\varphi^{prom}$ specifies a time-bounded strengthened promise on the service of a component and $\varphi^{assm}$ is a time-bounded relaxed assumption on the environment. This allows a controller to notify other controllers about improved operation conditions where the sending controller might

- relax assumptions on its environment or
- strengthen its own service guarantees.

Sending an event means translating an out-event into an in-event. On the receiving side, the *in-event*'s $\varphi^{prom}$ is a time-bounded relaxation on the guarantee that the receiving component has to fulfill and $\varphi^{assm}$ is a time-bounded strengthened assumption that the receiving component may rely on. This allows for the receiving controller to get notified that the sending controller may

- tolerate more behavior within its environment. In this case the receiving controller is allowed to exhibit additional behavior. or
- provide stronger service guarantees. In this case the receiving controller is allowed to rely on these stronger guarantees.

This mechanism allows the controllers to dynamically negotiate assumptions and guarantees during runtime. Therefore, a more fine grained cooperation of controllers can be designed which is not possible with static assumptions and guarantees.

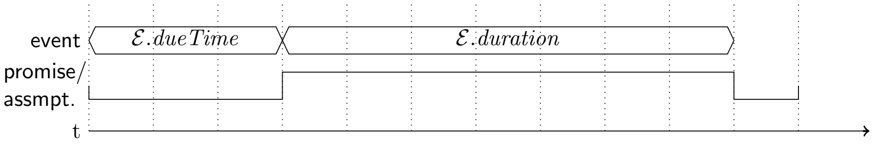


Figure 5.10: Visualization of a event's relationship between timings, promises, and relaxed assumptions. The first line of the figure shows the parts of the event (due time and duration) and the time interval they cover. The second line shows the when the event's associated relaxed assumption (resp. strengthened promise) is active (high) or inactive (low).

Graphically we annotate a transition with $\mathcal{E}?$ denoting the *reception of an event* and $\mathcal{E}!$ denoting the *emission of an event*. This roughly follows the notation used in communicating automata. However, our semantics allows to ignore incoming events and events might be emitted without reception. Since reception is optional, it has to hold that the base assumption $\varphi^{env}$ of a control component $\mathsf{CC} = (\mathsf{C}, \mathsf{P}, \varphi^{env})$ implies the relaxed event assumption $\varphi^{assm}$.

> **Note 5.43**
> *Using several communication channels one can encode finitely many different events with different event promises, assumptions, due times, and durations. Using a* best-fit *strategy one can make — discretized — promises and assumptions on continuous*

> *signals. However, to improve readability, we treat such a* family of events *as a single event with a finite valued payload* $(dueTime, duration, \dots)$. ◁

In the following, we briefly summarize our case study to demonstrate how our developed techniques work together in order to establish meaningful properties of a complex system. We present the plant, give a short overview on the architecture and sub-components of the ADAS, and inspect the assumptions and guarantees of the sub-components. Using the interfaces of the sub-components, we derive interfaces for composed components where assumptions and guarantees are aggregated. Here, some assumptions might already be satisfied by the guarantees of other components and we omit them from the composed interface. Guarantees can, however, be propagated to the composed interface and some guarantees might even be combined to obtain new guarantees.

The details have been published in [Dam+14; MHR17] and a summary of the initial conditions, invariant assumption, guaranteed safety, and guaranteed stability properties are given in the following. Along the way of presenting the case study, we highlight challenging aspects for state-of-the-art verification techniques and sketch techniques we applied and developed to cope with these particular aspects.

### 5.4.1 Plant

**Plant**

$$\dot{vel}_{\mathsf{cur}} = s \cdot acc \;\wedge\; \dot{\beta}_{\mathsf{ori}} = \beta_{\mathsf{steer}} - \beta_{\mathsf{street}} \;\wedge\; \dot{dist}_{\mathsf{cur}} = vel_{\mathsf{cur}} \cdot \sin(\beta_{\mathsf{ori}})$$

$$5 \leq vel_{\mathsf{cur}} \leq 50 \;\wedge\; -5^\circ \leq \beta_{\mathsf{ori}} \leq 5^\circ \;\wedge\; -10 \leq dist_{\mathsf{cur}} \leq 10$$

Figure 5.11: The automaton modeling the plant (CAR).

Our plant CAR, a simplified model of a car, is visualized in Figure 5.11. It has two controllable parameters: velocity and orientation. Our plant model, although diminutive, is far from trivial as its dynamics involves non-linearities — even non-polynomials. The plant is non-linear because of (1) the differential equation for the velocity $vel_{\mathsf{cur}}$ and (2) the differential equation for the distance to the middle of the lane $dist_{\mathsf{cur}}$. As we will see later, we need to perform reachability and stability analysis on the dynamic model. Since most verification tools cannot handle such kind of equations, we employed enclosures to make this model available for verification. In our case study, we used two different enclosures for addressing the differential equation of $dist_{\mathsf{cur}}$

**(Encl.1)** a linear overapproximations using an additive error $\dot{dist}_{\mathsf{cur}} \in \{m \cdot \beta_{\mathsf{ori}} + \underline{e}, m \cdot \beta_{\mathsf{ori}} + \overline{e}\}$ such that $\forall vel_{\mathsf{cur}}, \beta_{\mathsf{ori}} \bullet vel_{\mathsf{cur}} \cdot \sin(\beta_{\mathsf{ori}}) \in \{m \cdot \beta_{\mathsf{ori}} + \underline{e}, m \cdot \beta_{\mathsf{ori}} + \overline{e}\}$ and

**(Encl.2)** a linear overapproximation using a multiplicative error $\dot{dist}_{\mathsf{cur}} \in \{\underline{m} \cdot \beta_{\mathsf{ori}}, \overline{m} \cdot \beta_{\mathsf{ori}}\}$ such that $\forall vel_{\mathsf{cur}}, \beta_{\mathsf{ori}} \bullet vel_{\mathsf{cur}} \cdot \sin(\beta_{\mathsf{ori}}) \in \{\underline{m} \cdot \beta_{\mathsf{ori}}, \overline{m} \cdot \beta_{\mathsf{ori}}\}$.

Computing the additive error interval $E_a = [\underline{e}, \overline{e}]$ or the multiplicative error interval $E_m = [\underline{m}, \overline{m}]$ can be done efficiently since both $vel_{\mathsf{cur}}$ and $\beta_{\mathsf{ori}}$ are bounded. To reduce the absolute error the systems has been further partitioned into several regions each with a different error interval.[2]

### 5.4.2 ADAS: Architecture

The designed ADAS is composed of three sub-components, two controllers and a translation component. The two controllers are a velocity controller, VC, and a steering controller, SC. Although the motivation of our framework is distributed control taking into account message loss and latency, it can be used to modularize the design. In that sense, we have implemented the coordination of the steering controller SC and the velocity controller VC in a separate component, which is called the event translator ET. Note, that this modularized design is uniquely described in this thesis and has not been presented before. This separation led to more generalized controllers compared to the ones we described in [Dam+14; MHR17].

The velocity controller is responsible for its *local objective* of controlling the speed (Obj3), the steering controller is responsible for lane keeping (Obj2) and together, they have to accomplish the *global objective* of guaranteeing a comfortable centrifugal force (Obj1). The global objective can be achieved by a loose coupling. In that sense, maintaining a comfortable centrifugal force is achieved as long as a car cannot be arbitrarily fast in a narrow turn and a fast car may not decide to change its steering angle too abruptly. Thus, the smaller the steering angles and velocities are, the looser is the coupling between the two controllers. The exact relation is illustrated in Figure 5.12.

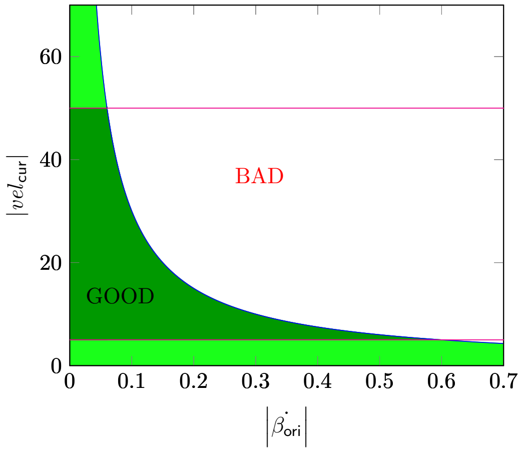


Figure 5.12: Velocity and steering control is coupled via the centrifugal force. A driver feels comfortable at centrifugal forces within the “GOOD” range.

Since keeping the car on the lane is more important than keeping the velocity high, the steering is given higher priority. Thus, the velocity controller might only reach the user’s desired velocity $vel_{\mathsf{des}}$ if appropriate. The coupling due to centrifugal force is expressed

[2] The details can be found in [Dam+14; MHR17].

by the following formula

$$|F_{\mathsf{cntrif}}| = \left| mass \cdot vel_{\mathsf{cur}} \cdot \dot{\beta_{\mathsf{ori}}} \right|.$$

Given a maximal change of orientation $\dot{\beta_{\mathsf{ori}}} = \beta_{\mathsf{steer}}$ and a maximal force $F_{\mathsf{comf}}$, a passenger with a mass of *mass* might feel comfortable, this connection allows us to compute an upper bound on the velocity $vel_{\mathsf{cur}}$, which we call the maximal comfortable velocity $vel_{\mathsf{comf}}$. In the case study, we use a mass *mass* of 80kg and maximal comfortable force $F_{\mathsf{comf}}$ of 240$^{\mathrm{kg \cdot m}}/_{\mathrm{s}^2}$ which leads to maximal acceleration of 3$^{\mathrm{m}}/_{\mathrm{s}^2}$ due to the centrifugal force.[3]

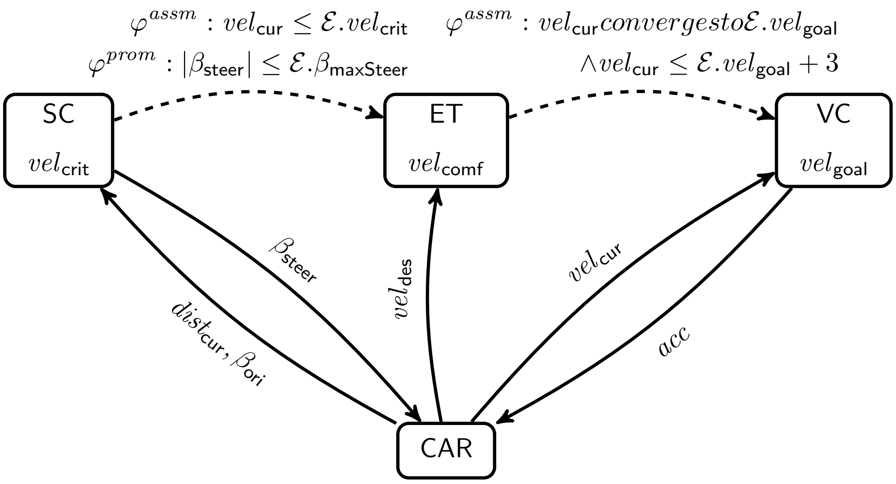

$\varphi^{assm} : vel_{\mathsf{cur}} \leq \mathcal{E}.vel_{\mathsf{crit}}$
$\varphi^{prom} : |\beta_{\mathsf{steer}}| \leq \mathcal{E}.\beta_{\mathsf{maxSteer}}$
$\varphi^{assm} : vel_{\mathsf{cur}} converges to \mathcal{E}.vel_{\mathsf{goal}}$
$\wedge vel_{\mathsf{cur}} \leq \mathcal{E}.vel_{\mathsf{goal}} + 3$
SC
$vel_{\mathsf{crit}}$
ET
$vel_{\mathsf{comf}}$
VC
$vel_{\mathsf{goal}}$
$\beta_{\mathsf{steer}}$
$vel_{\mathsf{des}}$
$vel_{\mathsf{cur}}$
$dist_{\mathsf{cur}}, \beta_{\mathsf{ori}}$
$acc$
CAR

Figure 5.13: Global System Architecture of the ADAS. Dashed arrows indicate event-based communication and normal arrows indicate reading of sensors and driving of actuators. The variables $\mathcal{E}.vel_{\mathsf{crit}}, \mathcal{E}.\beta_{\mathsf{maxSteer}}, \mathcal{E}.vel_{\mathsf{goal}}$ are placeholders representing the finitely many events and only part of the interface in order to facilitate representation.

The architecture of the ADAS is visualized in Figure 5.13. ET can be seen as a glue component that translates out-events from the steering controller to in-events for the velocity controller making use of the look-up table $vel_{\mathsf{comf}} = velFor(\beta_{\mathsf{maxSteer}})$. This look-up-table is derived from the above described formula for $F_{\mathsf{cntrif}}$ as $\frac{F_{\mathsf{comf}}}{mass \cdot \beta_{\mathsf{maxSteer}}}$ where $\beta_{\mathsf{maxSteer}}$ is an absolute bound on the steering, i. e., the first derivative of the orientation. The values returned on a *best-fit* strategy are given in 5.14. Additionally, the event translator ET has to consider the user-chosen velocity $vel_{\mathsf{des}}$ and the maximal velocity under which the steering controller SC is able to perform its service $vel_{\mathsf{crit}}$.

### 5.4.3 Steering Controller

The implementation of the steering controller SC is a *PI*-like controller as visualized in Figure 5.15. As the SC is in charge of lane keeping, we want it to steer the car such that

[3] We do not claim that these are realistic values.

| **Maximum Steering** ($\beta_{\mathsf{maxSteer}}$) | **Comfortable Velocity** ($vel_{\mathsf{comf}}$) |
|---|---|
| $\leq 0.3$ | 10 |
| $\leq 0.20$ | 15 |
| $\leq 0.12$ | 25 |
| $\leq 0.06$ | 50 |
| $\leq 0.05$ | 60 |

Table 5.14: Comfortable velocities for different absolute bounds on the steering as given by the look-up table $vel_{\mathsf{comf}} = \mathit{velFor}(\beta_{\mathsf{maxSteer}})$.

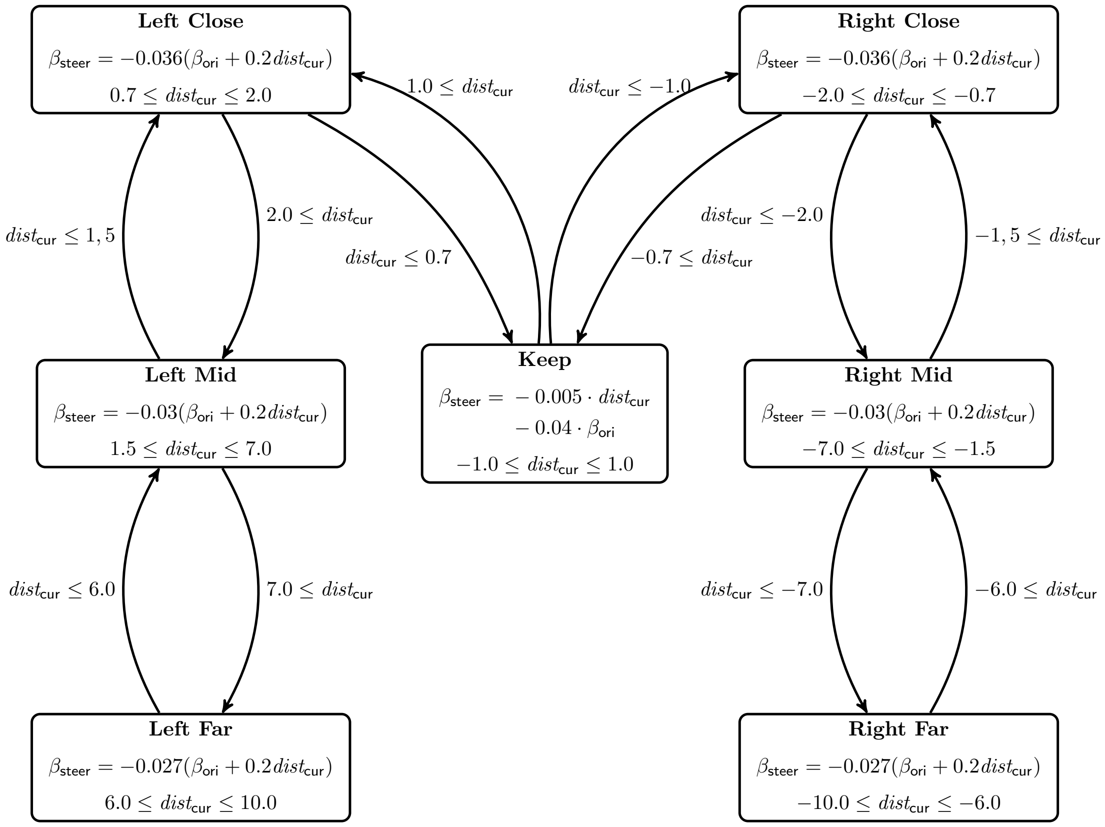

**Left Close**
$\beta_{\mathsf{steer}} = -0.036(\beta_{\mathsf{ori}} + 0.2 dist_{\mathsf{cur}})$
$0.7 \leq dist_{\mathsf{cur}} \leq 2.0$
$1.0 \leq dist_{\mathsf{cur}}$
$dist_{\mathsf{cur}} \leq -1.0$
**Right Close**
$\beta_{\mathsf{steer}} = -0.036(\beta_{\mathsf{ori}} + 0.2 dist_{\mathsf{cur}})$
$-2.0 \leq dist_{\mathsf{cur}} \leq -0.7$
$dist_{\mathsf{cur}} \leq 1,5$
$2.0 \leq dist_{\mathsf{cur}}$
$dist_{\mathsf{cur}} \leq 0.7$
$dist_{\mathsf{cur}} \leq -2.0$
$-0.7 \leq dist_{\mathsf{cur}}$
$-1,5 \leq dist_{\mathsf{cur}}$
**Left Mid**
$\beta_{\mathsf{steer}} = -0.03(\beta_{\mathsf{ori}} + 0.2 dist_{\mathsf{cur}})$
$1.5 \leq dist_{\mathsf{cur}} \leq 7.0$
**Keep**
$\beta_{\mathsf{steer}} = -0.005 \cdot dist_{\mathsf{cur}}$
$-0.04 \cdot \beta_{\mathsf{ori}}$
$-1.0 \leq dist_{\mathsf{cur}} \leq 1.0$
**Right Mid**
$\beta_{\mathsf{steer}} = -0.03(\beta_{\mathsf{ori}} + 0.2 dist_{\mathsf{cur}})$
$-7.0 \leq dist_{\mathsf{cur}} \leq -1.5$
$dist_{\mathsf{cur}} \leq 6.0$
$7.0 \leq dist_{\mathsf{cur}}$
$dist_{\mathsf{cur}} \leq -7.0$
$-6.0 \leq dist_{\mathsf{cur}}$
**Left Far**
$\beta_{\mathsf{steer}} = -0.027(\beta_{\mathsf{ori}} + 0.2 dist_{\mathsf{cur}})$
$6.0 \leq dist_{\mathsf{cur}} \leq 10.0$
**Right Far**
$\beta_{\mathsf{steer}} = -0.027(\beta_{\mathsf{ori}} + 0.2 dist_{\mathsf{cur}})$
$-10.0 \leq dist_{\mathsf{cur}} \leq -6.0$

Figure 5.15: Model of the steering controller (SC)

| Name | Property |
|---|---|
| **SC.A1** | Initially $dist_{\mathsf{cur}} \in [0,9] \wedge \beta_{\mathsf{ori}} \in [-2.6,0] \vee dist_{\mathsf{cur}} \in [9,0] \wedge \beta_{\mathsf{ori}} \in [0,2.6]$ holds. |
| **SC.A2** | It always holds that $vel_{\mathsf{cur}} \in [5,50]$. |
| **SC.A3** | If $\mathcal{E}_{out}$ is active, then $vel_{\mathsf{cur}} \leq \mathcal{E}_{out}.vel_{\mathsf{crit}}$ holds.<br>If no event is active, then $vel_{\mathsf{cur}} \leq vel_{\mathsf{safe}}$ with $vel_{\mathsf{safe}} = 10$ holds. |
| **SC.G1** | It always holds that $(\beta_{\mathsf{ori}}, dist_{\mathsf{cur}})$ converges to $\mathbf{0}$. |
| **SC.G2** | It always holds that $\beta_{\mathsf{ori}} \in [-5°, 5°]$. |
| **SC.G3** | It always holds that $dist_{\mathsf{cur}} \in [-10,10]$. |
| **SC.G4** | It always holds that $\beta_{\mathsf{steer}} \in [-0.3,0.3]$. |
| **SC.G5** | If $\mathcal{E}_{out}$ is active, then $\lvert\beta_{\mathsf{steer}}\rvert \leq \mathcal{E}_{out}.\beta_{\mathsf{maxSteer}}$ holds. |
| **SC.G6** | For all sendable events $\mathcal{E}_{out}$ holds $\mathcal{E}_{out}.vel_{\mathsf{crit}} \in \{10,\ldots,50\}$. |

Table 5.16: Snippet from the interface of the steering controller

it converges to the center of the lane while not leaving the lane. These properties are formalized in the controllers interface. A snippet of the interface is given in Table 5.16 which contains initial and global assumptions as well as guarantees.

The assumptions of the steering controller are either simple properties on the initial state **SC.A1** or safety properties **SC.A2** and **SC.A3**. While **SC.A2** is directly realizable as a control component's base assumption $\varphi^{env}_{\mathsf{SC}}$, **SC.A3** is an assumption involving events. **SC.A3** belongs to the same events as the guarantees **SC.G5** and **SC.G6**. Together, they are communicated based on the following operation modes:

(OP 1) in the base case, $vel_{\mathsf{crit}}$ is set to 10 resulting in the assumption $vel_{\mathsf{cur}} \leq 10$, which allows intensive steering actions where $\beta_{\mathsf{steer}}$ may be as large as $[-0.3, 0.3]$,

(OP 2) if at some point in time $vel_{\mathsf{crit}}$ is raised to 25, then an event is sent, that relaxes the assumption to $vel_{\mathsf{cur}} \leq 25$ while strengthening guarantee **SC.G4** by promising more limited steering actions in which $\beta_{\mathsf{steer}}$ may not exceed $[-0.12, 0.12]$, and

(OP 3) if at some point in time $vel_{\mathsf{crit}}$ is raised to 50, then an event $\mathcal{E}$ is sent, that relaxes the assumption even more to $vel_{\mathsf{cur}} \leq 50$ while the promise is strengthened to very smooth steering actions in which $\lvert\beta_{\mathsf{steer}}\rvert$ is less than 0.05,

In our case study, we require that there is a part of the SC which is responsible for sending the actual events according to the above rules. However, we did not require a special implementation but assumed that this part sets $vel_{\mathsf{crit}}$ and sends events following a strategy that

- observes the street in front of the car,
- overapproximatively predicts the behavior within a given time horizon *horizon*, and
- updates $vel_{\mathsf{crit}}$ and sends events appropriately.

Note that $vel_{\mathsf{crit}}$ is different from $vel_{\mathsf{comf}}$. $vel_{\mathsf{comf}}$ indicates an upper bound on the velocity that is convenient for the passengers of the car while $vel_{\mathsf{crit}}$ indicates an upper bound on the velocity under which the steering controller is able to keep the car on the lane. The quality of the prediction is important as sending truthful events is crucial for assumption **SC.A3** being satisfied all the time.

Property **SC.G1** is a prototypical instance of GAS-Asm. It states that the controller achieves convergence to a single equilibrium under assumptions even though controlling $(\beta_{\mathsf{ori}}, dist_{\mathsf{cur}})$ depends on the open input $vel_{\mathsf{cur}}$. Properties **SC.G2** to **SC.G4** are simple invariants describing the service of the car: keep the car on the lane while not exceeding maximal steering actions.

### Interface Satisfaction

In the following, we describe how interface satisfaction for the SC is established. Proving interface satisfaction means to prove that the steering controller establishes the guarantees **SC.G1** to **SC.G6** under the assumptions **SC.A1** to **SC.A3**.

Proving interface satisfaction faces two major problems

- most reachability tools — capable of safety verification — handle linear systems only and have difficulties handling infinite trajectories and
- automatic approaches for stability verification — such as Stabhyli— can only handle systems with regions and flows described by at most polynomials.

To overcome these problems, we have employed the enclosure techniques described in Section 5.4.1 and developed the unrolling algorithm presented in Section 4.3. In the context of the case study, we have manually applied the unrolling technique. Later on, due to the general applicability and value, we have implemented and used them in [MHT15; HM15; HMT15]. Table 5.16 summarizes the properties annotated at the interface of the steering controller's control component $\mathsf{CC}_{\mathsf{SC}} = (\mathsf{SC}, \mathsf{CAR}, \varphi^{env}_{\mathsf{SC}})$. To prove interface satisfaction, we exploited these four additional properties of the composite of the steering controller and the plant:

**time invariance** the controller stores no information about the past. The behavior depends on the current state only,

**symmetry** the controller behaves identical on the left and on the right side of the lane, which is reflected in the automaton,

**planarity** a reduced model of the plant and the steering controller can be obtained that is only two-dimensional (i. e., it has only two continuous variables),

**update-free** the reduced model involves only transitions whose updates are the identity, i. e., the valuations of variables are not changed during transitions.

To verify that property **SC.G1** is satisfied, we built the parallel composition of the plant and the steering controller, removed the differential equation for the velocity and replaced $vel_{\mathsf{cur}}$ with its whole range for the base case (OP 1), i. e., $vel_{\mathsf{cur}} \in [5, 10]$. The resulting automaton in visualized in Figure 5.18. For the resulting system, we can apply the usual techniques to obtain Lyapunov functions to prove convergence after additionally enclosing the sine by a linear differential inclusion and a multiplicative error (Encl.2). We have to repeat the procedure for OP 2 (i. e., $vel_{\mathsf{cur}} \in [5, 25]$) and OP 3 (i. e., $vel_{\mathsf{cur}} \in [5, 50]$) with appropriate restrictions of the operation range, e. g., we might assume that high velocities $vel_{\mathsf{cur}} \in [5, 50]$ are allowed only very close to the center of the lane $dist_{\mathsf{cur}} \in [-1, 1] \wedge \beta_{\mathsf{ori}} \in [-1, 1]$. The restricted automaton can be found in Figure 5.17. Note, that these restricted ranges can serve as the basis for the implementation of the event sending mechanism of the steering controller. As we did not implement that mechanism, we simply assumed the existence of an mechanism satisfying properties **SC.G5** and **SC.G6**.

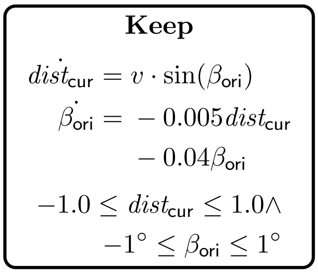

**Keep**
$\dot{dist}_{\mathsf{cur}} = v \cdot \sin(\beta_{\mathsf{ori}})$
$\dot{\beta}_{\mathsf{ori}} = -0.005 dist_{\mathsf{cur}} - 0.04\beta_{\mathsf{ori}}$
$-1.0 \leq dist_{\mathsf{cur}} \leq 1.0 \wedge$
$-1^\circ \leq \beta_{\mathsf{ori}} \leq 1^\circ$

Figure 5.17: Parallel composition of the plant and the steering controller for high velocities

A side-effect of **SC.G1** is that we can compute level sets from the Lyapunov functions which we exploited in the verification of properties **SC.G2**, **SC.G3**, and **SC.G4**. Here, we used the unrolling algorithm to step-by-step transform the hybrid automaton into an equivalent acyclic hybrid automaton. This procedure repeatedly splits the locations of the hybrid automaton until no further transition can be taken and no trajectory will ever hit the unsafe set. The termination relies on the knowledge of a safe set (a level set not intersecting the unsafe set) obtained from the before-mentioned Lyapunov functions.

Details on all verification steps have been published in [MHR17].

### 5.4.4 Velocity Controller

The VC is responsible of controlling the velocity according to the user's desired velocity and the maximal allowed velocity according to the behavior of the remainder of the ADAS. The implementation of this functionality is again split into two subautomata VC_PI and VC_ER. VC_PI is a classical $PI$-controller with saturations as visualized in Figure 5.19 and VC_ER is an event receiver and set-point generator. The event receiver VC_ER listens for incoming events communicating different values for $\mathcal{E}_{in}.vel_{\mathsf{goal}}$ and sets the local set-point $vel_{\mathsf{goal}}$ appropriately.[4] A possible implementation of the event receiver is visualized in Figure 5.20. The VC_PI's task is then to drive the current velocity $vel_{\mathsf{cur}}$

[4] It is actually listening for many events, because events do not have a payload and, therefore, we encode different ranges of $\mathcal{E}_{in}.vel_{\mathsf{goal}}$ using a multitude of events. However, throughout this thesis, we hide this technical detail and assume finite-valued payloads for the sake of readability.

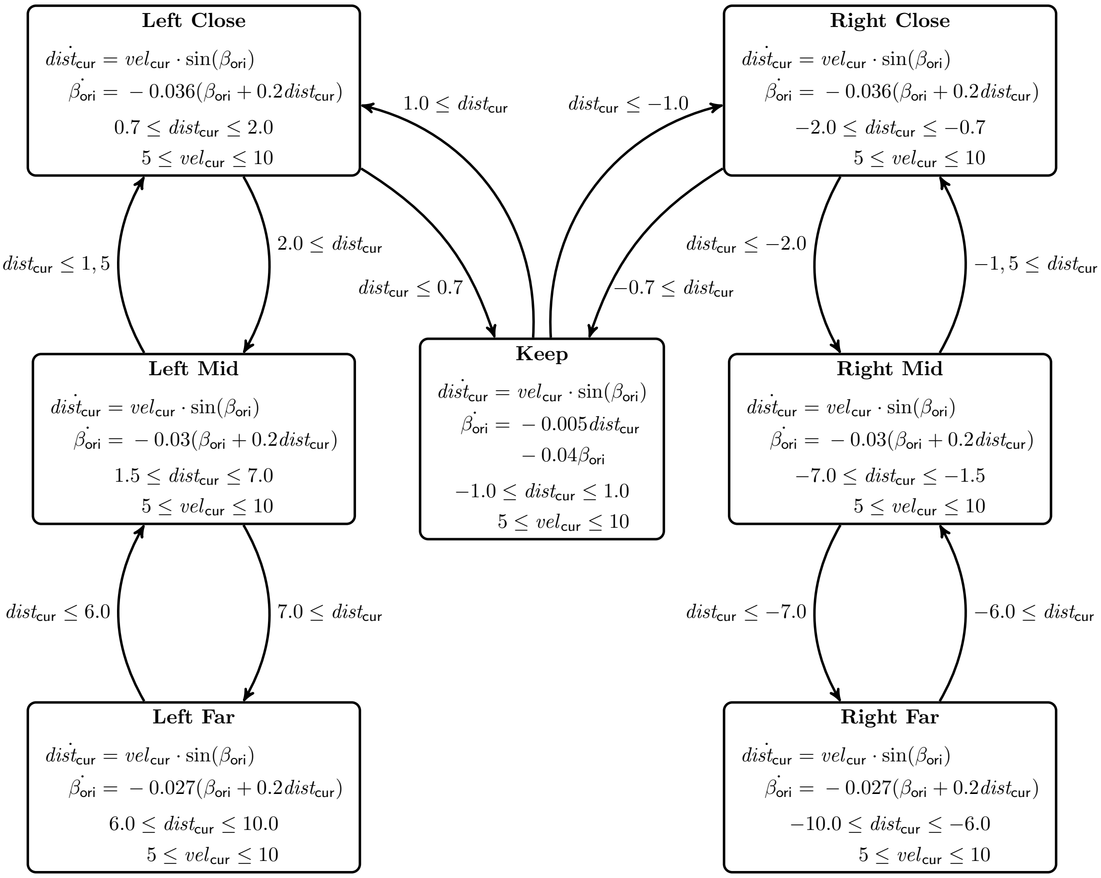

Left Close
$\dot{dist}_{\mathsf{cur}} = vel_{\mathsf{cur}} \cdot \sin(\beta_{\mathsf{ori}})$
$\dot{\beta}_{\mathsf{ori}} = -0.036(\beta_{\mathsf{ori}} + 0.2dist_{\mathsf{cur}})$
$0.7 \leq dist_{\mathsf{cur}} \leq 2.0$
$5 \leq vel_{\mathsf{cur}} \leq 10$
Right Close
$\dot{dist}_{\mathsf{cur}} = vel_{\mathsf{cur}} \cdot \sin(\beta_{\mathsf{ori}})$
$\dot{\beta}_{\mathsf{ori}} = -0.036(\beta_{\mathsf{ori}} + 0.2dist_{\mathsf{cur}})$
$-2.0 \leq dist_{\mathsf{cur}} \leq -0.7$
$5 \leq vel_{\mathsf{cur}} \leq 10$
$1.0 \leq dist_{\mathsf{cur}}$
$dist_{\mathsf{cur}} \leq -1.0$
$dist_{\mathsf{cur}} \leq 1,5$
$2.0 \leq dist_{\mathsf{cur}}$
$dist_{\mathsf{cur}} \leq 0.7$
$dist_{\mathsf{cur}} \leq -2.0$
$-0.7 \leq dist_{\mathsf{cur}}$
$-1,5 \leq dist_{\mathsf{cur}}$
Left Mid
$\dot{dist}_{\mathsf{cur}} = vel_{\mathsf{cur}} \cdot \sin(\beta_{\mathsf{ori}})$
$\dot{\beta}_{\mathsf{ori}} = -0.03(\beta_{\mathsf{ori}} + 0.2dist_{\mathsf{cur}})$
$1.5 \leq dist_{\mathsf{cur}} \leq 7.0$
$5 \leq vel_{\mathsf{cur}} \leq 10$
Keep
$\dot{dist}_{\mathsf{cur}} = vel_{\mathsf{cur}} \cdot \sin(\beta_{\mathsf{ori}})$
$\dot{\beta}_{\mathsf{ori}} = -0.005dist_{\mathsf{cur}}$
$-0.04\beta_{\mathsf{ori}}$
$-1.0 \leq dist_{\mathsf{cur}} \leq 1.0$
$5 \leq vel_{\mathsf{cur}} \leq 10$
Right Mid
$\dot{dist}_{\mathsf{cur}} = vel_{\mathsf{cur}} \cdot \sin(\beta_{\mathsf{ori}})$
$\dot{\beta}_{\mathsf{ori}} = -0.03(\beta_{\mathsf{ori}} + 0.2dist_{\mathsf{cur}})$
$-7.0 \leq dist_{\mathsf{cur}} \leq -1.5$
$5 \leq vel_{\mathsf{cur}} \leq 10$
$dist_{\mathsf{cur}} \leq 6.0$
$7.0 \leq dist_{\mathsf{cur}}$
$dist_{\mathsf{cur}} \leq -7.0$
$-6.0 \leq dist_{\mathsf{cur}}$
Left Far
$\dot{dist}_{\mathsf{cur}} = vel_{\mathsf{cur}} \cdot \sin(\beta_{\mathsf{ori}})$
$\dot{\beta}_{\mathsf{ori}} = -0.027(\beta_{\mathsf{ori}} + 0.2dist_{\mathsf{cur}})$
$6.0 \leq dist_{\mathsf{cur}} \leq 10.0$
$5 \leq vel_{\mathsf{cur}} \leq 10$
Right Far
$\dot{dist}_{\mathsf{cur}} = vel_{\mathsf{cur}} \cdot \sin(\beta_{\mathsf{ori}})$
$\dot{\beta}_{\mathsf{ori}} = -0.027(\beta_{\mathsf{ori}} + 0.2dist_{\mathsf{cur}})$
$-10.0 \leq dist_{\mathsf{cur}} \leq -6.0$
$5 \leq vel_{\mathsf{cur}} \leq 10$

Figure 5.18: Parallel composition of the plant and the steering controller for low velocities

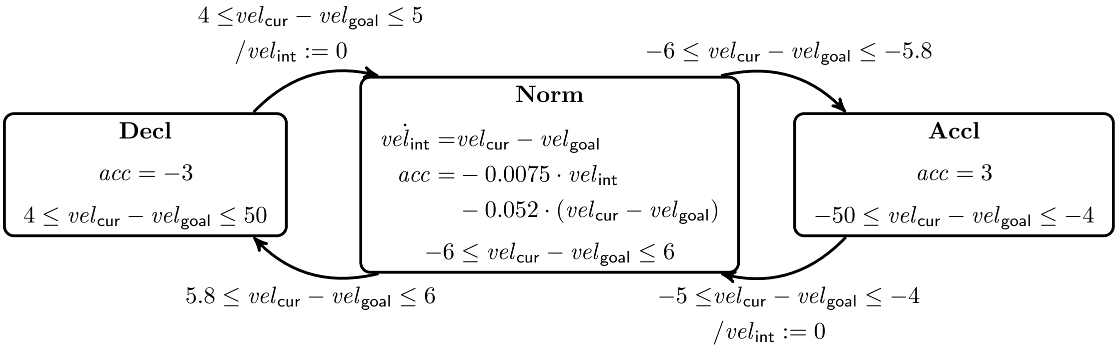

$4 \leq vel_{\mathsf{cur}} - vel_{\mathsf{goal}} \leq 5$
$/vel_{\mathsf{int}} := 0$
$-6 \leq vel_{\mathsf{cur}} - vel_{\mathsf{goal}} \leq -5.8$
Norm
$\dot{vel}_{\mathsf{int}} = vel_{\mathsf{cur}} - vel_{\mathsf{goal}}$
$acc = -0.0075 \cdot vel_{\mathsf{int}}$
$- 0.052 \cdot (vel_{\mathsf{cur}} - vel_{\mathsf{goal}})$
$-6 \leq vel_{\mathsf{cur}} - vel_{\mathsf{goal}} \leq 6$
Decl
$acc = -3$
$4 \leq vel_{\mathsf{cur}} - vel_{\mathsf{goal}} \leq 50$
Accl
$acc = 3$
$-50 \leq vel_{\mathsf{cur}} - vel_{\mathsf{goal}} \leq -4$
$5.8 \leq vel_{\mathsf{cur}} - vel_{\mathsf{goal}} \leq 6$
$-5 \leq vel_{\mathsf{cur}} - vel_{\mathsf{goal}} \leq -4$
$/vel_{\mathsf{int}} := 0$

Figure 5.19: Model of the velocity controller *PI*-controller (VC_PI).

towards the set-point $vel_{\mathsf{goal}}$. Due to peak-overshots, setting the set-point has to be done with care as the current velocity $vel_{\mathsf{cur}}$ might exceed $vel_{\mathsf{goal}}$. This peak-overshot has been identified to be less than 3m/s [DMR14]. Hence, $vel_{\mathsf{goal}}$ has to be adjusted depending on whether we

**decrease the setpoint** where $vel_{\mathsf{goal}}$ has to be set above the global minimally allowed velocity of 5m/s and

**increase the setpoint** where $vel_{\mathsf{goal}}$ has to be set below the maximally allowed velocity $vel_{\mathsf{max}}$, e.g. $\mathcal{E}_{in}.vel_{\mathsf{goal}} \leq vel_{\mathsf{max}} - 3$.

Note that the given implementation is rather pessimistic and the capabilities are very limited in the following sense:

- events are not queued in any form, i. e., if they are not yet usable, then they are dropped and
- events are not aggregated, i. e., any received event that has a duration lasting longer than the current remaining time, will overwrite the current set-point if the set-point is not decreased.

The last point — albeit very rudimentary — allows chaining events, i. e., events with a overlapping duration can be used to constantly keep the velocity high but short phases of very high velocities might be ignored.

The interface of the control component $\mathsf{CC}_{\mathsf{VC}} = (\mathsf{VC}, \mathsf{CAR}, \varphi^{env}_{\mathsf{VC}})$ is summarized in Table 5.21.

Like the steering controller the VC has assumptions on the initial state **VC.A1** stating that the initial velocity is not too high. Additionally, the safety assumptions **VC.A2** requires the payload of the events to have a fixed number of different values for $vel_{\mathsf{goal}}$ only.

Guarantees **VC.G1** to **VC.G3** describe the service of th VC where

- **VC.G1** and **VC.G2** describe the range of the velocity globally maintained by the VC in the base case, and depending on the received events,

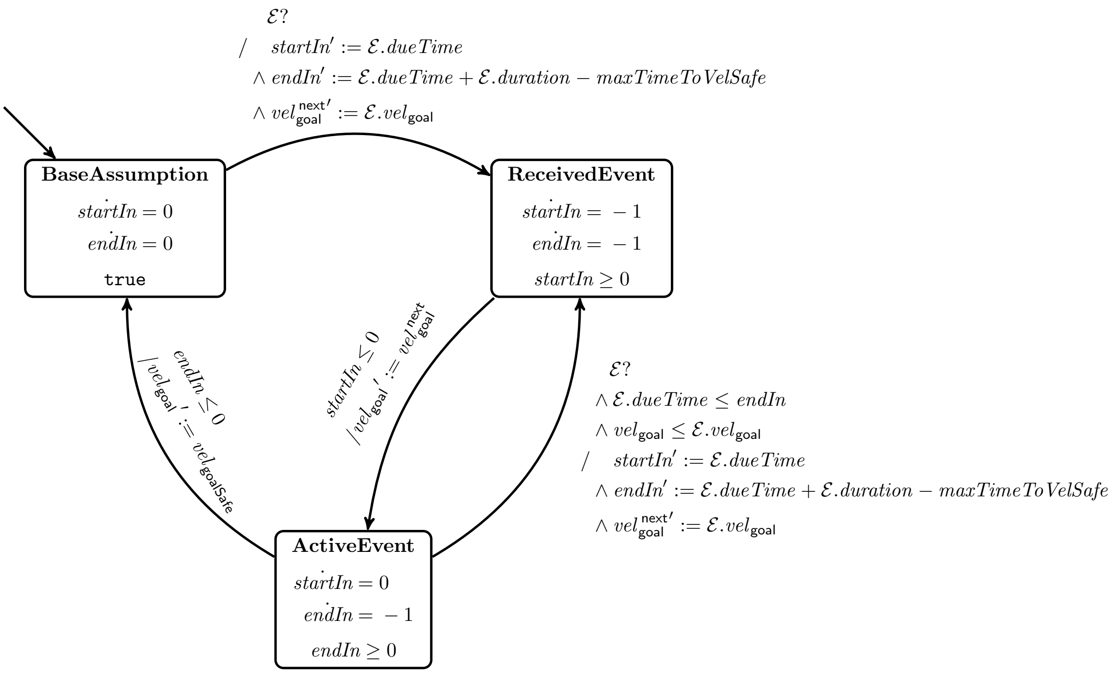

$\mathcal{E}?$
$/ \quad startIn' := \mathcal{E}.dueTime$
$\wedge\ endIn' := \mathcal{E}.dueTime + \mathcal{E}.duration - maxTimeToVelSafe$
$\wedge\ vel_{\mathsf{goal}}^{\mathsf{next}}{}' := \mathcal{E}.vel_{\mathsf{goal}}$
BaseAssumption
$\dot{startIn} = 0$
$\dot{endIn} = 0$
true
ReceivedEvent
$\dot{startIn} = -1$
$\dot{endIn} = -1$
$startIn \geq 0$
$endIn \leq 0$
$/ vel_{\mathsf{goal}}' := vel_{\mathsf{goalSafe}}$
$startIn \leq 0$
$/ vel_{\mathsf{goal}}' := vel_{\mathsf{goal}}^{\mathsf{next}}$
$\mathcal{E}?$
$\wedge\ \mathcal{E}.dueTime \leq endIn$
$\wedge\ vel_{\mathsf{goal}} \leq \mathcal{E}.vel_{\mathsf{goal}}$
$/ \quad startIn' := \mathcal{E}.dueTime$
$\wedge\ endIn' := \mathcal{E}.dueTime + \mathcal{E}.duration - maxTimeToVelSafe$
$\wedge\ vel_{\mathsf{goal}}^{\mathsf{next}}{}' := \mathcal{E}.vel_{\mathsf{goal}}$
ActiveEvent
$\dot{startIn} = 0$
$\dot{endIn} = -1$
$endIn \geq 0$

Figure 5.20: Model of the velocity controller event receiver (VC_ER), which implements a simple strategy for exploiting chains of incoming events.

- **VC.G3** describes the convergence of the velocity and how the set point depends on the received events.

Property **VC.G3** is actually a multitude of $\varphi^{con}$-conGAS properties each encoded with a condition

$$\varphi^{con}_{\star} \equiv (vel_{\mathsf{goal}} = \star) \wedge (\mathsf{flow} \Rightarrow \dot{vel}_{\mathsf{goal}} = 0) \wedge (\neg\mathsf{flow} \Rightarrow vel_{\mathsf{goal}}{}' = vel_{\mathsf{goal}})$$

for every possible target velocity $\mathcal{E}_{in}.vel_{\mathsf{goal}}$, i. e., $\star \in \{8, \dots, 47\}$. This gives 39 different equilibrium points.

### Interface Satisfaction

In the following, we describe how interface satisfaction for the VC is established.

Since $\mathcal{E}_{in}.vel_{\mathsf{goal}}$ is not controlled by the VC and the actual value of $\mathcal{E}_{in}.vel_{\mathsf{goal}}$ is the subject of the events that the VC may receive, the velocity controller has to rely on assumptions on the environment in order to set its internal set-point $vel_{\mathsf{goal}}$ appropriately. Having only finitely many values for $\mathcal{E}_{in}.vel_{\mathsf{goal}}$ allows us to prove the properties of the velocity controller individually for every possible value including the fallback value $vel_{\mathsf{goal}} = vel_{\mathsf{goalSafe}} = 8$ (in which case the maximum velocity shall be below $vel_{\mathsf{safe}} = 10$ after at most $maxTimeToVelSafe$ time).

| Name | Property |
|---|---|
| **VC.A1** | Initially $vel_{\mathsf{cur}} \in [5, 10]$ holds. |
| **VC.A2** | For all receivable events $\mathcal{E}_{in}$ holds $\mathcal{E}_{in}.vel_{\mathsf{goal}} \in \{8, \ldots, 47\}$. |
| **VC.G1** | It always holds that $vel_{\mathsf{cur}} \in [5, 50]$. |
| **VC.G2** | If $\mathcal{E}_{in}$ is active, then $vel_{\mathsf{cur}} \in [5, \mathcal{E}_{in}.vel_{\mathsf{goal}} + 3]$ holds.<br>Otherwise, it holds that $vel_{\mathsf{cur}} \in [5, vel_{\mathsf{safe}}]$ with $vel_{\mathsf{safe}} = 10$. |
| **VC.G3** | If $\mathcal{E}_{in}$ is active then $vel_{\mathsf{cur}}$ converges to $\mathcal{E}_{in}.vel_{\mathsf{goal}}$ for $\min(0, \mathcal{E}_{in}.duration - maxTimeToVelSafe)$ long.<br>Otherwise, $vel_{\mathsf{cur}}$ converges to $vel_{\mathsf{goalSafe}} = 8$. |

Table 5.21: Snippet from the interface of the velocity controller

As mentioned before, property **VC.G3** consists of multiple $\varphi^{con}$-conGAS sub-properties (one for each receivable event). Each individual $\varphi^{con}$-conGAS property can be verified using a parallel composition with the induced environmental automaton (cf. Lemma 5.19) corresponding to the phases of active events.

The properties **VC.G1** and **VC.G2** are simple safety properties whose verification can again be simplified using level sets obtained from the verification of Property **VC.G3**, similar to what we have done in Section 4.3. However, the proof relies on the value of $maxTimeToVelSafe$ and we need to ensure that $maxTimeToVelSafe$[5] is long enough for the velocity controller to drive $vel_{\mathsf{cur}}$ into a sufficiently small neighborhood around $vel_{\mathsf{goalSafe}}$. A very conservative value can be derived from the stability certificate, e. g., by asking for exponential stability instead of asymptotic stability in the definition of Theorem 5.23. A tighter value can be derived using the unrolling technique from Section 4.3. To summarize, the main observations for the proof are

- **for the upper bound on $vel_{\mathsf{cur}}$:** if there is no subsequent event, then VC_ER sets $vel_{\mathsf{goal}}$ back to $vel_{\mathsf{goalSafe}}$ at least $maxTimeToVelSafe$ before the last event becomes inactive and
- **for the lower bound on $vel_{\mathsf{cur}}$:** if VC_ER updates $vel_{\mathsf{goal}}$, then this takes into account the maximum peak-overshot on $vel_{\mathsf{cur}}$ as given before (see decreasing and increasing the setpoint on page 176).

**Example 5.44**
*Figure 5.22 shows one possible trajectory of the velocity controller. Initially the velocity is* 10 *also the critical velocity is* 10*, no event is active, and the initial set-point is* 8*. After* 20 *seconds an event becomes active notifying about the increases of* $vel_{\mathsf{crit}}$ *to* 25 *and — in order to take the peak-overshot into account — the set-point* $vel_{\mathsf{goal}}$ *is set to* 22*. After* 100 *seconds the critical velocity and the set-point are again increased to* 50 *and* 47*, respectively. After approximately* 140 *seconds, the set-point is reset to* 8 *because no subsequent event has been received. Approximately*

[5] In [DMR14], $maxTimeToVelSafe$ is a function mapping velocity to the time needed to converge back to a small neighborhood around $vel_{\mathsf{safe}}$.

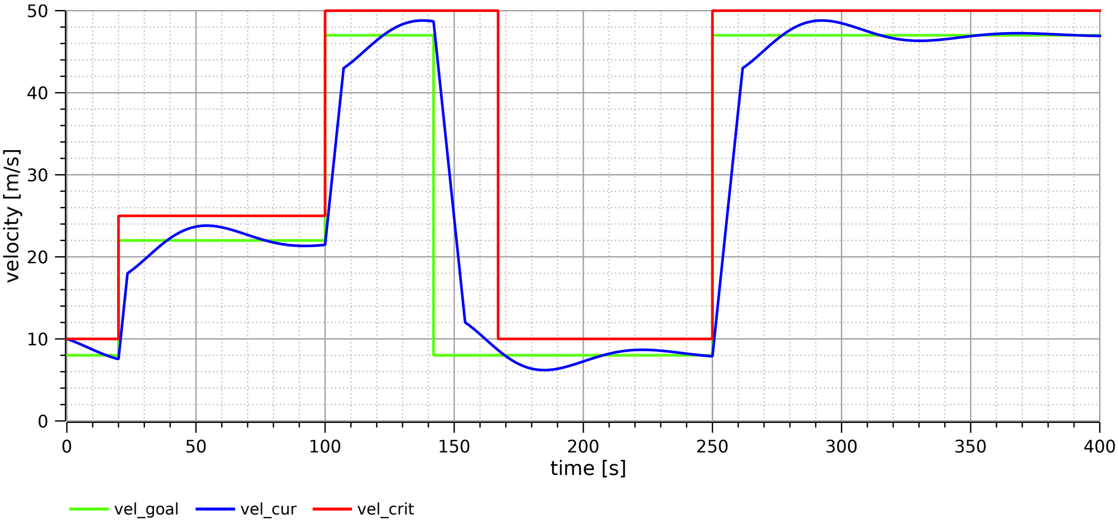


Figure 5.22: Example trajectory of the velocity controller.

> 25 *seconds later, the critical velocity is also back to the default value just slightly after the velocity reached the default range, i. e.,* $vel_{\mathsf{cur}} \in [5, 10]$. *After* 250*, finally, another event becomes active and the critical velocity is increased to* 50 *and the set-point to* 47 *again. The peak-overshoot is properly respected such that the critical velocity is never exceeded.* ◁

### 5.4.5 Event Translation

The event translator which acts like a glue component by relaying events from the steering controller including $\mathcal{E}_{in}.vel_{\mathsf{crit}}$ and $\mathcal{E}_{in}.\beta_{\mathsf{maxSteer}}$ to the velocity controller by sending events including $\mathcal{E}_{out}.vel_{\mathsf{goal}}$. For computing appropriate values for $vel_{\mathsf{goal}}$, the glue component chooses the actual event according to the function

$$\begin{aligned} \mathcal{E}_{out}.vel_{\mathsf{goal}} &= velFor(\mathcal{E}_{in}, vel_{\mathsf{des}}) \\ &= \max(\min(vel_{\mathsf{des}}, velFor(\mathcal{E}_{in}.\beta_{\mathsf{maxSteer}}), \mathcal{E}_{in}.vel_{\mathsf{crit}}) - 3, vel_{\mathsf{goalSafe}}) \end{aligned}$$

that actually takes the minimum of (1) the user-chosen velocity $vel_{\mathsf{des}}$, (2) the comfortable velocity $vel_{\mathsf{comf}}$ that the look-up-table $velFor(\cdot)$ returns for the promised range of $\beta_{\mathsf{maxSteer}}$ associated with the incoming event, (3) the critical velocity $vel_{\mathsf{crit}}$ as specified by the assumption of associated incoming event, and, finally, subtracts 3 — accounting for a potential peak-overshoot — from the minimum if the value is more than $vel_{\mathsf{goalSafe}}$ and $vel_{\mathsf{goalSafe}}$ otherwise. The event translator sends the target velocity $\mathcal{E}_{out}.vel_{\mathsf{goal}}$ only in cases where it is not less than the value in the base assumption $vel_{\mathsf{goalSafe}}$. The interface is summarized in Table 5.23.

Assumption **ET.A1** (resp. **ET.A2**) restricts the possible values for the user-chosen desired velocity (resp. the communicated critical velocity) to finitely many values from a certain range. Assumption **ET.A3** restricts the range of the change of the orientation

| Name | Property |
|---|---|
| **ET.A1** | It always holds that $vel_{\mathsf{des}} \in \{10, \ldots, 45\}$. |
| **ET.A2** | For all receivable events $\mathcal{E}_{in}$ holds $\mathcal{E}_{in}.vel_{\mathsf{crit}} \in \{10, \ldots, 50\}$. |
| **ET.A3** | If $\mathcal{E}_{in}$ is active, then $\lvert\beta_{\mathsf{steer}}\rvert \leq \mathcal{E}_{in}.\beta_{\mathsf{maxSteer}}$ holds.<br>Otherwise, it holds that $\lvert\beta_{\mathsf{steer}}\rvert \leq 0.3$. |
| **ET.A4** | If $\mathcal{E}_{out}$ is active, then $vel_{\mathsf{cur}} \leq \mathcal{E}_{out}.vel_{\mathsf{goal}} + 3$ holds.<br>Otherwise, it holds that $vel_{\mathsf{cur}} \leq vel_{\mathsf{safe}} = 10$. |
| **ET.G1** | For all events $\mathcal{E}_{out}$ holds $\mathcal{E}_{out}.vel_{\mathsf{goal}} \in \{8, \ldots, vel_{\mathsf{des}} - 3\}$. |
| **ET.G2** | It always holds that $vel_{\mathsf{cur}} \leq velFor(\lvert\beta_{\mathsf{steer}}\rvert)$. |
| **ET.G3** | If $\mathcal{E}_{in}$ is active, then $vel_{\mathsf{cur}} \leq \mathcal{E}_{in}.vel_{\mathsf{crit}}$ holds.<br>Otherwise, it holds that $vel_{\mathsf{cur}} \leq vel_{\mathsf{safe}}$ with $vel_{\mathsf{safe}} = 10$. |

Table 5.23: Snippet from the interface for the event translator ET

$\beta_{\mathsf{steer}}$, globally and depending on the received events. The assumption **ET.A4** describes the expected service that a velocity controller has to maintain a restricted range for the velocity unless a larger range was communicated via events.

Guarantee **ET.G1** simply states that $\mathcal{E}_{out}.vel_{\mathsf{goal}}$ may only take finitely many different values and is chosen such that is does not exceed the velocity chosen by the driver of the car $vel_{\mathsf{des}}$. Guarantees **ET.G2** and **ET.G3** describe the event translator's service. The service is to choose the target velocity appropriate for exceeding neither the velocity the driver might experience comfortable nor the velocity which is critical as communicated by the steering controller.

An exemplary implementation for the event translator is given in Figure 5.24.

### Interface Satisfaction

In the following, we sketch how interface satisfaction for the event translator can be established.

Guarantees **ET.G2** and **ET.G3** are actually involving an argument on events which is "events are truthful". The incoming and outgoing events are listed in Table 5.25. The incoming events combine assumption **ET.A3** and guarantee **ET.G3**. They can be read as: if it holds that $\lvert\beta_{\mathsf{steer}}\rvert \leq \mathcal{E}_{in}.\beta_{\mathsf{maxSteer}}$ where $\mathcal{E}_{in}.\beta_{\mathsf{maxSteer}}$ is the — via in-event $\mathcal{E}_{in}$ — communicated upper bound on the absolute change of the orientation $\lvert\beta_{\mathsf{steer}}\rvert$, then $vel_{\mathsf{goal}}$ is set appropriate for the communicated value of $vel_{\mathsf{crit}}$. This can be proven together with assumption **ET.A4**:

- **an event $\mathcal{E}_{in}$ has been received and an event $\mathcal{E}_{out}$ was send**: in this case, it holds that $vel_{\mathsf{cur}} \leq \mathcal{E}_{out}.vel_{\mathsf{goal}} + 3 \leq velFor(\mathcal{E}_{in}.\beta_{\mathsf{maxSteer}}) \leq velFor(\lvert\beta_{\mathsf{steer}}\rvert)$[6],
- **an event $\mathcal{E}_{in}$ has been retrieved and no event was send**: in this case, it holds that $vel_{\mathsf{cur}} \leq vel_{\mathsf{safe}} \leq velFor(\mathcal{E}_{in}.\beta_{\mathsf{maxSteer}}) \leq velFor(\lvert\beta_{\mathsf{steer}}\rvert)$, and

[6] Note, that if $\lvert\beta_{\mathsf{steer}}\rvert \leq \beta_{\mathsf{maxSteer}}$, then $velFor(\beta_{\mathsf{maxSteer}}) \leq velFor(\lvert\beta_{\mathsf{steer}}\rvert)$ due to the reciprocal proportion of $\lvert\beta_{\mathsf{steer}}\rvert$ and $vel_{\mathsf{cur}}$.

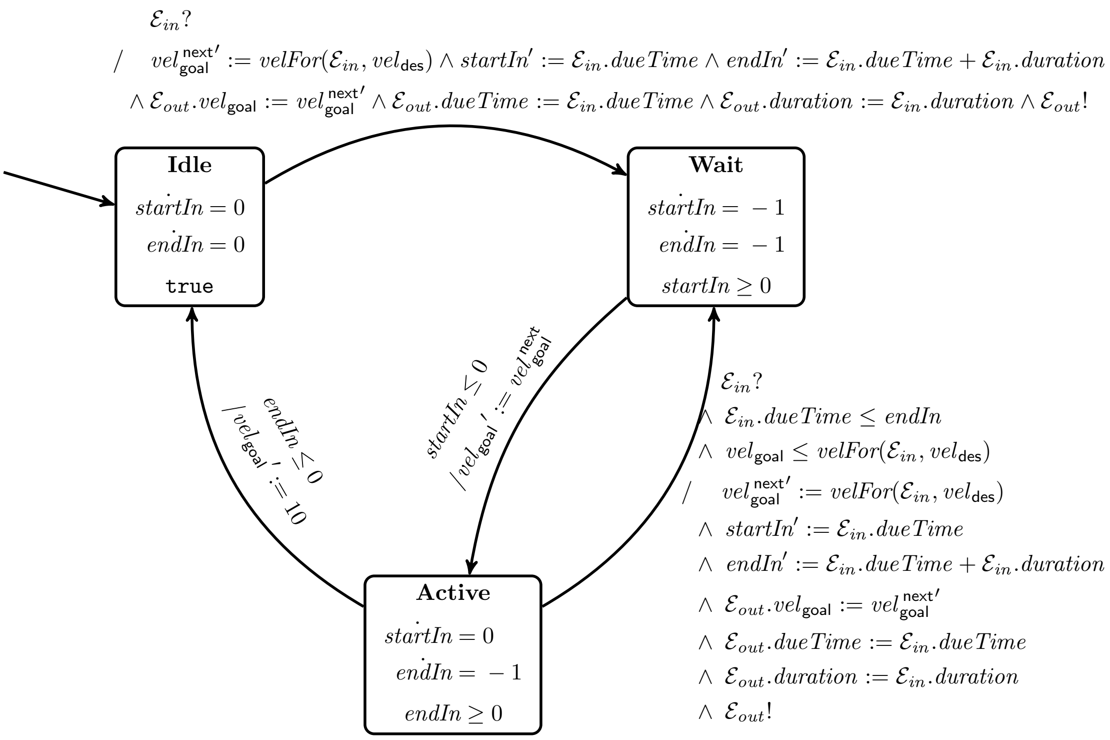

$\mathcal{E}_{in}?$
$/ \quad vel_{\mathsf{goal}}^{\mathsf{next}\prime} := velFor(\mathcal{E}_{in}, vel_{\mathsf{des}}) \wedge startIn' := \mathcal{E}_{in}.dueTime \wedge endIn' := \mathcal{E}_{in}.dueTime + \mathcal{E}_{in}.duration$
$\wedge\, \mathcal{E}_{out}.vel_{\mathsf{goal}} := vel_{\mathsf{goal}}^{\mathsf{next}\prime} \wedge \mathcal{E}_{out}.dueTime := \mathcal{E}_{in}.dueTime \wedge \mathcal{E}_{out}.duration := \mathcal{E}_{in}.duration \wedge \mathcal{E}_{out}!$
Idle
$\dot{startIn} = 0$
$\dot{endIn} = 0$
true
Wait
$\dot{startIn} = -1$
$\dot{endIn} = -1$
$startIn \geq 0$
Active
$\dot{startIn} = 0$
$\dot{endIn} = -1$
$endIn \geq 0$
$endIn \leq 0$
$/ vel_{\mathsf{goal}}' := 10$
$startIn \leq 0$
$/ vel_{\mathsf{goal}}' := vel_{\mathsf{goal}}^{\mathsf{next}}$
$\mathcal{E}_{in}?$
$\wedge\, \mathcal{E}_{in}.dueTime \leq endIn$
$\wedge\, vel_{\mathsf{goal}} \leq velFor(\mathcal{E}_{in}, vel_{\mathsf{des}})$
$/ \quad vel_{\mathsf{goal}}^{\mathsf{next}\prime} := velFor(\mathcal{E}_{in}, vel_{\mathsf{des}})$
$\wedge\, startIn' := \mathcal{E}_{in}.dueTime$
$\wedge\, endIn' := \mathcal{E}_{in}.dueTime + \mathcal{E}_{in}.duration$
$\wedge\, \mathcal{E}_{out}.vel_{\mathsf{goal}} := vel_{\mathsf{goal}}^{\mathsf{next}\prime}$
$\wedge\, \mathcal{E}_{out}.dueTime := \mathcal{E}_{in}.dueTime$
$\wedge\, \mathcal{E}_{out}.duration := \mathcal{E}_{in}.duration$
$\wedge\, \mathcal{E}_{out}!$

Figure 5.24: Model of the event translator (ET).

- **no event has been retrieved and therefore no event was send**: in this case, it holds that $vel_{\mathsf{cur}} \leq vel_{\mathsf{safe}} \leq velFor(0.3) \leq velFor(|\beta_{\mathsf{steer}}|)$.

Thus, in all cases, it holds that $vel_{\mathsf{cur}} \leq velFor(|\beta_{\mathsf{steer}}|)$ and, therefore, **ET.G2** is satisfied.

Guarantee **ET.G3**, additionally, states that $vel_{\mathsf{goal}}$ is always chosen such that $vel_{\mathsf{cur}}$ does not exceed the critical velocity, as communicated by the steering controller, by incorporating a safety margin of 3 to allow for slight overshots.

## 5.4.6 Composition

During the composition, we aggregate assumptions and guarantees. That is, some assumptions are already satisfied by the other component and we omit them from the composite interface and some guarantees might be combined to obtain other guarantees.

### The Composed Interface of the Event Translator and the Velocity Controller

The composition of the event translator ET and the velocity controller VC satisfies the common interface summarized in Table 5.26. In the following, we briefly reason why this is the case.

Note, that property **VC.G3** refers to an internal event which is neither intuitive nor allowed in our framework and thus has to be omitted from the interface of the composition.

| Event Type | Assumption | Guarantee |
|---|---|---|
| in | $\lvert\beta_{\mathsf{steer}}\rvert \leq \mathcal{E}_{in}.\beta_{\mathsf{maxSteer}} = 0.3$ | $vel_{\mathsf{cur}} \leq \mathcal{E}_{in}.vel_{\mathsf{crit}} = 10$ |
| in | $\lvert\beta_{\mathsf{steer}}\rvert \leq \mathcal{E}_{in}.\beta_{\mathsf{maxSteer}} = 0.12$ | $vel_{\mathsf{cur}} \leq \mathcal{E}_{in}.vel_{\mathsf{crit}} = 25$ |
| in | $\lvert\beta_{\mathsf{steer}}\rvert \leq \mathcal{E}_{in}.\beta_{\mathsf{maxSteer}} = 0.05$ | $vel_{\mathsf{cur}} \leq \mathcal{E}_{in}.vel_{\mathsf{crit}} = 50$ |
| out | $vel_{\mathsf{cur}} \leq \mathcal{E}_{out}.vel_{\mathsf{goal}} + 3 = 25$ | none |
| out | $vel_{\mathsf{cur}} \leq \mathcal{E}_{out}.vel_{\mathsf{goal}} + 3 = 45$ | none |

Table 5.25: Receivable and send events from the event translator ET.

| Name | Property |
|---|---|
| **VC.A1** | Initially that $vel_{\mathsf{cur}} \in [5, 10]$ holds. |
| **ET.A1** | It always holds that $vel_{\mathsf{des}} \in \{10, \dots, 45\}$. |
| **ET.A2** | For all receivable events $\mathcal{E}_{in}$ holds $\mathcal{E}_{in}.vel_{\mathsf{crit}} \in [10, 50]$. |
| **ET.A3** | If $\mathcal{E}_{in}$ is active, then $\lvert\beta_{\mathsf{steer}}\rvert \leq \mathcal{E}_{in}.\beta_{\mathsf{maxSteer}}$ holds. |
| **VC.G1** | It always holds that $vel_{\mathsf{cur}} \in [5, 50]$. |
| **VC.G3** | If the internal event $\mathcal{E}_{int}$ is received and active, then $vel_{\mathsf{cur}}$ converges to $\mathcal{E}_{int}.vel_{\mathsf{goal}}$ for $\min(0, \mathcal{E}_{int}.duration - maxTimeToVelSafe)$ long. Otherwise $vel_{\mathsf{cur}}$ converges to $vel_{\mathsf{goalSafe}} = 8$. |
| **ET.G2** | It always holds that $vel_{\mathsf{cur}} \leq velFor(\lvert\beta_{\mathsf{steer}}\rvert)$. |
| **ET.G3** | If $\mathcal{E}_{in}$ is active, then $vel_{\mathsf{cur}} \leq \mathcal{E}_{in}.vel_{\mathsf{crit}}$ holds. Otherwise, it holds that $vel_{\mathsf{cur}} \leq vel_{\mathsf{safe}}$ with $vel_{\mathsf{safe}} = 10$. |

Table 5.26: Snippet from the composed interface of the event translator and the velocity controller

This is due to the fact that (a) internal events are hidden and more severe (b) reception of an event is not guaranteed. However, the guarantee is actually fulfilled by the composite and it is a lack of expressiveness of the current definition of interfaces that prevents one from annotating this property. This lack will be subject of future extensions. One possible extension is to (a) specify the conditions under which events are sent and (b) include communication channels that guarantee delivery of events or limit the number of lost events.

### Interface Composability of the Event Translator and the Velocity Controller

The interface composition is valid since

- guarantee **ET.G1** satisfies assumption **VC.A2**, i. e., for all receivable events $\mathcal{E}_{in}$, it holds that $\mathcal{E}_{in}.vel_{\mathsf{goal}} \in \{8, \dots, 47\}$.
- guarantee **VC.G2** satisfies assumption **ET.A4**, i. e., if $\mathcal{E}_{out}$ is active, then $vel_{\mathsf{cur}} \leq \mathcal{E}_{out}.vel_{\mathsf{goal}} + 3$ holds. Otherwise, it holds that $vel_{\mathsf{cur}} \leq vel_{\mathsf{safe}} = 10$.
- All other assumptions are passed to the composed interface.

### The Composed Interface of the ADAS

Finally, a composition with the steering controller allows to derive the interface given in Table 5.27 for the ADAS.

| Name | Property |
|---|---|
| **VC.A1** | Initially that $vel_{\mathsf{cur}} \in [5, 10]$ holds. |
| **ET.A1** | It always holds that $vel_{\mathsf{des}} \in \{10, \ldots, 45\}$. |
| **SC.A1** | Initially that $dist_{\mathsf{cur}} \in [0, 9] \wedge \beta_{\mathsf{ori}} \in [-2.6, 0] \vee dist_{\mathsf{cur}} \in [9, 0] \wedge \beta_{\mathsf{ori}} \in [0, 2.6]$ holds. |
| **VC.G1** | It always holds that $vel_{\mathsf{cur}} \in [5, 50]$. |
| **VC.G3** | If the internal event $\mathcal{E}_{int}$ is received and active, then $vel_{\mathsf{cur}}$ converges to $\mathcal{E}_{int}.vel_{\mathsf{goal}}$ for $\min(0, \mathcal{E}_{int}.duration - maxTimeToVelSafe)$ long. Otherwise, $vel_{\mathsf{cur}}$ converges to $vel_{\mathsf{goalSafe}} = 8$. |
| **ET.G2** | It always holds that $vel_{\mathsf{cur}} \leq velFor(\lvert\beta_{\mathsf{steer}}\rvert)$. |
| **SC.G1** | It always holds that $(\beta_{\mathsf{ori}}, dist_{\mathsf{cur}})$ converges to $\mathbf{0}$. |
| **SC.G2** | It always holds that $\beta_{\mathsf{ori}} \in [-5°, 5°]$. |
| **SC.G3** | It always holds that $dist_{\mathsf{cur}} \in [-10, 10]$. |
| **SC.G4** | It always holds that $\beta_{\mathsf{steer}} \in [-0.3, 0.3]$. |

Table 5.27: Snippet from the composed interface of the ADAS

### Interface Composability with the Steering Controller

The interface composition is valid since

- guarantee **VC.G1** satisfies assumption **SC.A2**, i. e., it always holds that $vel_{\mathsf{cur}} \in [5, 50]$.
- guarantee **ET.G3** satisfies assumption **SC.A3**, i. e., if $\mathcal{E}_{out}$ is active, then $vel_{\mathsf{cur}} \leq \mathcal{E}_{out}.vel_{\mathsf{crit}}$ holds. Otherwise it holds that $vel_{\mathsf{cur}} \leq vel_{\mathsf{safe}}$ with $vel_{\mathsf{safe}} = 10$.
- guarantee **SC.G5** satisfies assumption **ET.A3**, i. e., if $\mathcal{E}_{out}$ is active, then $\lvert\beta_{\mathsf{steer}}\rvert \leq \mathcal{E}_{out}.\beta_{\mathsf{maxSteer}}$ holds.
- guarantee **SC.G6** satisfies assumption **ET.A2**, i. e., for all sendable events $\mathcal{E}_{out}$ holds $\mathcal{E}_{out}.vel_{\mathsf{crit}} \in \{10, \ldots, 50\}$.

Observe, that only assumptions on the activation (initial state) and user demands are left in the final interface of the compositionally constructed ADAS.

### Summary on the composed ADAS

We have composed an advanced driver assistant system that under some assumptions on the activation (e. g., not too far from the center of the lane) and user demand (e. g., not too low or high desired velocity) guarantees the following set of objectives:

Obj1 “Maintain a centrifugal force comfortable for a driver” corresponds to property **ET.G2**,

Obj2 “Bring and then keep the car on the center of its lane” corresponds to property **SC.G1**,

Obj3 “Control the speed also considering driver requests for a certain speed value” corresponds to property **VC.G3**.

Establishing the objectives naïvely on the parallel composition of the hybrid I/O automata requires to construct an hybrid I/O automaton that has roughly

$$\begin{aligned} &|Loc_{\mathsf{VC}} \times Loc_{\mathsf{ET}} \times Loc_{\mathsf{SC}} \times Loc_{\mathsf{SC\_EG}} \times D(velFor) \times D(vel_{\mathsf{des}})| \\ =&\, 9 \cdot 3 \cdot 7 \cdot 3 \cdot 5 \cdot 35 \\ =&\, 99225 \end{aligned}$$

locations where $D(velFor) = 5$ is the domain of the value of the look-up table $velFor$, $D(vel_{\mathsf{des}}) = 35$ is the domain of the user’s desired velocity, and the number of locations of SC_EG is assumed to be at least three. Which is to the best of the author’s knowledge far from being tractable for any stability verification technique. Indeed, it would require finding 99225 Lyapunov functions where each function is represented by a quadratic template with at least nine free parameters not even counting the extra parameters for $\mathcal{S}$-Procedure terms, SOS decomposition, and parameters of the $\mathcal{K}^\infty$ functions.

## 5.5 Summary

In this chapter, we have presented a framework for designing safe and stable hybrid systems. In Section 5.2, we have sketched preliminary work on a framework for the sequential (transition) composition where a single plant is controlled by a single controller at a time but control is passed between different controllers. In Section 5.3, this framework has been extended to parallel composition where different aspects of a single plant are controlled by different controllers but each aspect is controlled at any time. Applicability and usefulness have been demonstrated by a case study in Section 5.4.

Together, both operators, the sequential composition together with the parallel composition, allow for

1. sequentially designing controllers that control a distinct aspect of a plant and achieves a certain service, and
2. use the newly introduced parallel composition operator to compose the control components obtained from the sequential composition.

To describe the service of a controller, two new stability notions have been introduced:

- *global asymptotic stability under assumptions* allowing to describe that a controller stabilizes a plant towards a certain equilibrium point given that the environment always satisfies certain assumptions and

- *conditional global asymptotic stability* allowing to describe that a controller stabilizes a plant towards a certain equilibrium point whenever a certain condition is meet.

Capturing the behavior of the environment using (conditional) assumptions allows to verify the properties of the controllers in very local and isolated settings. We have also shown in what cases controllers are composable in a way that preserves the sub-components properties. In the framework, we have carefully formalized interfaces and composability constraints such that

- interfaces carry just enough details such that

- composability can be decided based on the information annotated at the interface and

- composite interfaces including requirements on the environment as well as delivered services of composed components can be derived based on the information annotated at the interface.

Such a verification in localized settings is appealing since it facilitates reuse and replacement of sub-components, improves the ability for humans to inspect faulty parts of the design, and enables the use of automatic verification methods that otherwise struggle from the size of the component.

We have demonstrated the potential of our framework using a case study on an advanced driver assistant system (ADAS) where two controllers and a translator component achieve both local and common objectives. We have given the implementations of the controllers, the translation component, and snippets of their interfaces. We have sketched

- how to locally prove the controller's properties,

- how to prove composability, and

- how to derive the interface of the composed controller.

The number of locations of the composed controller is at least 99225 while the size of every sub-components is not more than seven. Proving the same interface for the composed controller is not possible with contemporary tools. This shows that our framework has the potential to scale well with the size of the controller to-design and allows to handle very large systems assuming that a proper decomposition can be found.

CHAPTER 6

# Conclusion and Future Work

This chapter gives a conclusion, highlights the main results, and states possible future work.

This thesis addressed the automatic verification of global asymptotic stability via Lyapunov functions in the context of hybrid automata modeling hybrid systems. Global asymptotic stability is a property that is composed of two sub-properties (1) attractivity and (2) stability. The former ensures that the state of the system converges to a so-called equilibrium point and the latter ensures that the distance to the equilibrium point is bounded in terms of the initial state. The verification of these properties is undecidable even for simple hybrid systems, such as saturated linear systems and proofs cannot easily be derived from arguments local to every subsystem. Even worse, direct methods — like inspecting every trajectory or bundle of trajectories — are usually neither capable of verification nor falsification of asymptotic stability properties because this would require to analyze infinitely many trajectories that are not even bounded in length. However, indirect methods that establish state-based arguments are promising. One such argument can be obtained by a standard tool from control theory called Lyapunov theory which includes the search for Lyapunov functions. These functions assign each (hybrid, i. e., discrete and continuous) state of the system a value such that the value strictly decreases while the system evolves with the only minimum being at the equilibrium. If this function can be found, then the system is asymptotically stable. When searching for such a Lyapunov function, we have to take into account the interplay of the different dynamics in different locations of the system as well as the transitions between the locations, i. e., it cannot easily be conclude from arguments local the locations.

## 6.1 Conclusion

In order to be able to address the verification of hybrid systems with size and complexity as found in real-life applications, a key enabler is the ability to apply (semi-)automatic verification tools. Until the start of this work, there was no such tool which automatically applied Lyapunov theory-based stability verification to hybrid systems with polynomial flows, guards, and invariants. In fact, Stabhyli is the first tool to automatize all steps from translating a hybrid automaton into constraints until interpreting the result of numerical solvers. One of the insights gained is that choosing the right representation for a hybrid system is crucial for the successful certification of stability. Multiple reasons where discovered and addressed (1) numerical issues, (2) implicit information, and

(3) missing support in designing hybrid system. To counteract (1), we have equipped Stabhyli with numerical and symbolic methods to reject invalid candidate solutions obtained from a numerical solver and heuristics to optimize the constraints. To address (2), we found that successful verification depends on the representation of the hybrid system. For example locations which are connected via cyclic transitions whose guards are overlapping, require the Lyapunov functions to assign the same value to all hybrid states in the intersection. Due to the overlapping guards, it is possible that a trajectory might switch to any of the locations at any time, hence, they have to have equal Lyapunov function values. For such a situation, we have implemented a detection and resolution technique for such implicit equalities in the constraint system. However, the experiments lead to the insight that it is also worth looking into techniques that automatically rewrite the hybrid system in a way that separates the region induced by the intersection into a new location. For this new location the invariant has to be the intersection and the flow would be a differential inclusion that encloses the flow of the involved locations.

A second technique addressing (2), is to unroll the hybrid automaton. This yields an equivalent representation of the hybrid system whose stability can be decided by inspecting all modes and all transitions in isolation. Furthermore, the transitions of this alternative representation also have refined guards. For some systems obtaining such refined guards is essential as otherwise the transition constraint will require the Lyapunov function to decrease for all states that satisfy the guard including those which are unreachable.

A third technique addressing (2), is to alter the structure of the hybrid automaton's underlying graph. This technique is suitable for systems with a dense graph structure and reduces the number of decomposition steps at the drawback of introducing possible spurious runs.

Finally, to address (3) the framework for component-based design of safe and stable hybrid system by Damm et al. has been extended to include parallel composition. Our approach tackles distributed control and exploits realistic assumptions about the inability of individual controllers to influence each without time passing. The fact that under these assumptions local safety and stability properties of subsystems are inherited by the composed system, support the common belief that realistic assumptions enable us to handle more complex systems.

Together, the techniques and implemented tools that were developed in this work allow us to automatically verify stability of hybrid system or to design stable hybrid systems. This includes cases where a naïve application of the Lyapunov theorem will not be successful and adaptations of the representation of the hybrid system or the inclusion of extra knowledge into the constraints is needed.

## 6.2 Outlook and Future Work

Even though Stabhyli was planned to be a prototype tool, it is already possible to use it for the automatic derivation of stability proofs in practice. This application is the most evident achievement of this work. A number of examples of varying size and complexity have been proven stable throughout this thesis and they show the usability of Stabhyli. However, size and complexity are still challenging and handling hybrid systems as found in industry without compositional approaches or other preprocessing is not promising. Therefore, the results of this work should primarily be used for future research with respect to more transformations techniques and tighter integration of verification methods into the design process. Future work can be done in two directions:

- **practical extensions**, such as (1) implementing and comparing techniques to exploit special shapes within hybrid automata and therefrom generated constraint systems or (2) adding and comparing backend solvers, are enabled through the modular implementation of Stabhyli and

- we started to integrate automatic verification of safety and stability and to develop design principles and frameworks for the construction of safe and stable hybrid systems. As the results show that this is a promising field, further **theoretical research** allowing us to exploit more knowledge from safety verification in the stability verification and vice versa, will allow handling even more complex systems.

### Extension of Stabhyli

At the moment, Stabhyli supports only two backends for solving semidefinite programs. The two numeric solvers are SDPA and CSDP. Integrating and comparing other — possibly newer and possibly better — numerical solvers like SCS or SDPB will increase the practical usability of Stabhyli. Due to the fact that some solvers perform better on one or the other problem, this would increase the chance of successfully computing Lyapunov functions — especially in the case where Stabhyli queries all solvers independently. This could also include symbolic techniques such as cylindrical algebraic decomposition possibly combined with numerical solvers. Some initial experiments with QEPCAD [Bro03] and Redlog [DS97] show that for small systems, e. g., one location and few variables, Lyapunov functions can be computed at the drawback of a dramatically increased runtime. In [She+13; SX14], She analyzed local asymptotic stability of polynomial dynamical system. They presented an approach that uses a sufficient condition for the existence of a Lyapunov function which can be represented as a semi-algebraic set. This set is then successively under-approximated and, finally, a sample point is computed to show non-emptiness of the set. If such a sample point exists, then this proves the existence of a Lyapunov function. It seems worth integrating this approach into Stabhyli.

Another extension is to add backends that search for Lyapunov functions by relaxations other than SOS decomposition with SDP. One approach is the work of Sassi et al. which employs linear relaxations [Sas+15] based on linear combinations of Bernstein polynomials. An alternative — which is enabled by this work — is to use the technique for the generation of polyhedral underapproximations of quadrics presented in Section 8 and [HM15; HMT15] to obtain a parameterized polyhedron for the parameterized quadratic constraint system that results from applying the SOS decomposition. This allows using techniques for solving linear constraints which are known to be less prone to numerical issues.

### Extensions of the Transformation Techniques

For the technique relaxing hybrid automata with a dense control structure to hybrid automata with a sparse control structure, it would be interesting to investigate a tighter coupling of the decomposition and our relaxation approach. A first step will be to not discard the progress made by the decomposition but reuse the "gained knowledge." Doing so will greatly reduce the computational effort.

For the unrolling technique one could try to exploit further knowledge that can be derived from the Lyapunov theorem. In this sense, we can obtain an outer approximation of the system's behavior by exploiting the Lyapunov functions and $\mathcal{K}^\infty$ functions of the Lyapunov theorem (see Theorem 2.31). Until now, we have constructed polyhedral underapproximations of the level sets of the Lyapunov functions. Using a similar construction one could also obtain an overapproximation of the level set. For such an overapproximation holds that the system state is always contained in the polyhedron. Second, using the Lyapunov theorem, we can compute the slowest rate of decrease of the Lyapunov function's value. This rate can then be used to shrink the overapproximating polyhedron while the systems evolves and it is still guaranteed that the system's state is contained in the polyhedron. This gives us a very rough view on the evolution of the system. Nevertheless, this overapproximation can be computed very efficiently. Such an efficient way of exploring reachable states might then be used to guide a more precise and computationally intensive reach set computation.

### Extensions of the Framework for the Design of Safe and Stable Hybrid Systems

Most obviously, the framework presented in Chapter 5 misses tool support. We have already implemented the stability related proof obligations that arise during the sequential composition. But at the current state supplying the evidence for the parallel composition is a manual task even if the case-study has shown that certain steps can be handed over to automatic tools like STABHYLI and SOAPBOX.

Another extension would be to extend the framework towards dissipativity or passivity. Dissipativity generalizes Lyapunov theory to handle systems with inputs and outputs. In contrast to our work this does not rely on invariant assumption on the environment but on incorporating vanishing terms on the inputs. This allows to describe that a system stabilizes as long as its input stabilizes [EA06]. McCourt and Antsaklis have already demonstrated that it is possible to derive storage functions — which are the Lyapunov function equivalent in dissipativity — automatically in [MA14].

Another challenging task is to extend the framework such that sequential and parallel composition can be nested arbitrarily. To allow this, the interfaces have to be extended to also include information that allows to arbitrate the processes of passing the control from one component to multiple other components and vice versa. This is not trivial because it has to be guaranteed that every actuator is driven by exactly one component at all times and transferring control has to be synchronized. However, such an extension would allow us to give immediate feedback and even derive stability and safety proofs during the design of complex systems.

Despite these future work, the techniques and results presented in this thesis together with the prototypical implementation Stabhyli show and push applicability of automated stability verification even for large and complex systems. Having a sequential composition operator to build control components that can be further composed with other control components in parallel, enables engineers to build libraries of components and reuse them in new applications. Several extensions to the automated certification of stability (as well as safety) properties or even obtaining feedback in case a proof failed, supports in optimizing and releasing trustworthy products.

# Bibliography

### Own Peer-Reviewed Publications

[MT13a] Eike Möhlmann and Oliver E. Theel. “Stabhyli: A Tool for Automatic Stability Verification of Non-Linear Hybrid Systems”. In: *Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control (part of CPS Week), HSCC'13*. Ed. by Calin Belta and Franjo Ivancic. Philadelphia, Pennsylvania, USA: ACM, Apr. 2013, pp. 107–112. ISBN: 978-1-4503-1567-8. URL: http://doi.acm.org/10.1145/2461328.2461347.

[MT13b] Eike Möhlmann and Oliver E. Theel. “Towards Automatic Detection of Implicit Equality Constraints in Stability Verification of Hybrid Systems”. In: *Proceedings of the 1th Congreso Nacional de Ingeniería Informática / Aplicaciones Informáticas y de Sistemas de Información, CoNaIISI 2013*. Ed. by Marcelo M. Marciszack, Roberto M. Muñoz, and Mario A. Groppo. Córdoba, Córdoba, Argentina: Red de Carreras de Ingeniería Informática / Sistemas de Informatión (RIISIC), Nov. 2013. URL: http://conaiisi.frc.utn.edu.ar/PDFsParaPublicar/1/schedConfs/1/104-504-1-DR.pdf.

[DMR14] Werner Damm, Eike Möhlmann, and Astrid Rakow. “Component Based Design of Hybrid Systems: A Case Study on Concurrency and Coupling”. In: *Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control (part of CPS Week), HSCC'14*. Ed. by Martin Fränzle and John Lygeros. Berlin, Berlin, Germany: ACM, Apr. 2014, pp. 145–150. ISBN: 978-1-4503-2732-9. DOI: 10.1145/2562059.2562120. URL: http://doi.acm.org/10.1145/2562059.2562120.

[MT14] Eike Möhlmann and Oliver E. Theel. “Towards Counterexample-Guided Computation of Validated Stability Certificates for Hybrid Systems”. In: *Proceedings of the 2nd Congreso Nacional de Ingeniería Informática / Aplicaciones Informáticas y de Sistemas de Información, CoNaIISI 2014*. Ed. by Marcelo M. Marciszack, Roberto M. Muñoz, and Mario A. Groppo. Vol. 2. San Luis, Argentina: Red de Carreras de Ingeniería Informática / Sistemas de Información (RIISIC), Nov. 2014. URL: http://www.informatik.uni-oldenburg.de/~eikemoe/pubs/CoNaIISI_123.pdf.

[HM15] Willem Hagemann and Eike Möhlmann. “Inscribing H-Polyhedra in Quadrics using a Projective Generalization of Closed Sets”. In: *Proceedings of the 27th Canadian Conference on Computational Geometry, CCCG 2015*. Kingston, Ontario, Canada: Queen's University, Ontario, Canada, Aug. 2015. URL: http://research.cs.queensu.ca/cccg2015/CCCG15-papers/07.pdf.

[MHT15] Eike Möhlmann, Willem Hagemann, and Oliver E. Theel. “Hybrid Tools for Hybrid Systems - Proving Stability and Safety at Once”. In: *Formal Modeling and Analysis of Timed Systems - 13th International Conference, FORMATS 2015, Madrid, Spain, September 2-4, 2015, Proceedings*. Vol. 9268. Lecture Notes in Computer Science. Springer, Sept. 2015, pp. 222–239. ISBN: 978-3-319-22974-4. DOI: 10.1007/978-3-319-22975-1_15. URL: http://dx.doi.org/10.1007/978-3-319-22975-1_15.

[MT15] Eike Möhlmann and Oliver E. Theel. “Breaking Dense Structures: Proving Stability of Densely Structured Hybrid Systems”. In: *Proceedings of the 4th International Workshop on Engineering Safety and Security Systems, ESSS 2015, Oslo, Norway, June 22, 2015.* Ed. by Jun Pang, Yang Liu, and Sjouke Mauw. Vol. 184. Electronic Proceedings in Theoretical Computer Science. Open Publishing Association, June 2015, pp. 49–63. DOI: 10.4204/EPTCS.184.4. URL: http://dx.doi.org/10.4204/EPTCS.184.4.

[MHR17] Eike Möhlmann, Willem Hagemann, and Astrid Rakow. “Verifying a PI Controller using SoapBox and Stabhyli”. In: *ARCH@CPSWeek 2016, 3rd International Workshop on Applied Verification for Continuous and Hybrid Systems, Vienna, Austria.* Ed. by Goran Frehse and Matthias Althoff. Vol. 43. EPiC Series in Computing. originally submitted and accepted for ARCH14. EasyChair, 2017, pp. 115–125. URL: http://www.easychair.org/publications/paper/Verifying_a_PI_Controller_using_SoapBox_and_Stabhyli.

### Own Technical Reports

[Dam+14] Werner Damm, Willem Hagemann, Eike Möhlmann, and Astrid Rakow. *Component Based Design of Hybrid Systems: A Case Study on Concurrency and Coupling*. Reports of SFB/TR 14 AVACS 95. http://www.avacs.org. SFB/TR 14 AVACS, Feb. 2014. URL: http://www.avacs.org/fileadmin/Publikationen/Open/avacs_technical_report_095.pdf.

[HMT15] Willem Hagemann, Eike Möhlmann, and Oliver E. Theel. *Hybrid Tools for Hybrid System: Proving Safety and Stability at once (Extended Version)*. Reports of SFB/TR 14 AVACS 108. http://www.avacs.org. SF-B/TR 14 AVACS, Sept. 2015. URL: http://www.avacs.org/fileadmin/Publikationen/Open/avacs_technical_report_108.pdf.

[DMR16] Werner Damm, Eike Möhlmann, and Astrid Rakow. *A Design Framework for Concurrent Hybrid System Controllers with Safety and Stability Annotations*. Reports of SFB/TR 14 AVACS 105. http://www.avacs.org. SF-B/TR 14 AVACS, Apr. 2016. URL: http://www.avacs.org/fileadmin/Publikationen/Open/avacs_technical_report_105.pdf.

### References

[AL91] Martín Abadi and Leslie Lamport. "An Old-Fashioned Recipe for Real Time". In: *Real-Time: Theory in Practice, REX Workshop, Mook, The Netherlands, June 3-7, 1991, Proceedings*. Ed. by J. W. de Bakker, Cornelis Huizing, Willem P. de Roever, and Grzegorz Rozenberg. Vol. 600. Lecture Notes in Computer Science. Springer, 1991, pp. 1–27. ISBN: 3-540-55564-1. DOI: 10.1007/BFb0031985. URL: http://dx.doi.org/10.1007/BFb0031985.

[Aba+08] Alessandro Abate, Maria Prandini, John Lygeros, and Shankar Sastry. "Probabilistic Reachability and Safety for Controlled Discrete Time Stochastic Hybrid Systems". In: *Automatica* 44.11 (2008), pp. 2724–2734. DOI: 10.1016/j.automatica.2008.03.027. URL: http://dx.doi.org/10.1016/j.automatica.2008.03.027.

[Alu+95] Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A. Henzinger, Pei-Hsin Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. "The Algorithmic Analysis of Hybrid Systems". In: *Theor. Comput. Sci.* 138.1 (1995), pp. 3–34. DOI: 10.1016/0304-3975(94)00202-T. URL: http://dx.doi.org/10.1016/0304-3975(94)00202-T.

[Alu+92] Rajeev Alur, Costas Courcoubetis, Thomas A. Henzinger, and Pei-Hsin Ho. "Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems". In: *Hybrid Systems*. Ed. by Robert L. Grossman, Anil Nerode, Anders P. Ravn, and Hans Rischel. Vol. 736. Lecture Notes in Computer Science. Springer, 1992, pp. 209–229. ISBN: 3-540-57318-6. DOI: 10.1007/3-540-57318-6_30. URL: http://dx.doi.org/10.1007/3-540-57318-6_30.

[14] *American Control Conference, ACC 2014, Portland, OR, USA, June 4-6, 2014*. IEEE, 2014. ISBN: 978-1-4799-3272-6. URL: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6849600.

[ADG03] Eugene Asarin, Thao Dang, and Antoine Girard. "Reachability Analysis of Nonlinear Systems Using Conservative Approximation". English. In: *Hybrid Systems: Computation and Control*. Springer Berlin Heidelberg, 2003, pp. 20–35. ISBN: 978-3-540-00913-9. DOI: 10.1007/3-540-36580-X_5.

[ADG07] Eugene Asarin, Thao Dang, and Antoine Girard. "Hybridization methods for the analysis of nonlinear systems". English. In: *Acta Informatica* 43.7 (2007), pp. 451–476. ISSN: 0001-5903. DOI: 10.1007/s00236-006-0035-7.

[Asa+12] Eugene Asarin, Venkatesh Mysore, Amir Pnueli, and Gerardo Schneider. "Low dimensional hybrid systems - decidable, undecidable, don't know". In: *Inf. Comput.* 211 (2012), pp. 138–159. DOI: 10.1016/j.ic.2011.11.006. URL: http://dx.doi.org/10.1016/j.ic.2011.11.006.

[Bar+11] Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli. “CVC4”. In: *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV’11)*. Snowbird, UT: Springer-Verlag, 2011, pp. 171–177. ISBN: 978-3-642-22109-5. URL: http://dl.acm.org/citation.cfm?id=2032305.2032319.

[BT07] Clark Barrett and Cesare Tinelli. “CVC3”. In: *Proceedings of the 19th International Conference on Computer Aided Verification (CAV’07)*. Ed. by Werner Damm and Holger Hermanns. Vol. 4590. Lecture Notes in Computer Science. Berlin, Germany. Springer-Verlag, July 2007, pp. 298–302.

[BI13] Calin Belta and Franjo Ivancic, eds. *Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control, HSCC 2013, April 8-11, 2013, Philadelphia, PA, USA*. ACM, 2013. ISBN: 978-1-4503-1567-8.

[BBB07] Alberto Bemporad, Antonio Bicchi, and Giorgio C. Buttazzo, eds. *Hybrid Systems: Computation and Control, 10th International Workshop, HSCC 2007, Pisa, Italy, April 3-5, 2007, Proceedings*. Vol. 4416. Lecture Notes in Computer Science. Springer, 2007. ISBN: 978-3-540-71492-7.

[Ben+08] L. Benvenuti, A. Ferrari, E. Mazzi, and A. L. Sangiovanni Vincentelli. “Hybrid Systems: Computation and Control: 11th International Workshop, HSCC 2008, St. Louis, MO, USA, April 22-24, 2008. Proceedings”. In: ed. by Magnus Egerstedt and Bud Mishra. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. Chap. Contract-Based Design for Computation and Verification of a Closed-Loop Hybrid System, pp. 58–71. ISBN: 978-3-540-78929-1. DOI: 10.1007/978-3-540-78929-1_5. URL: http://dx.doi.org/10.1007/978-3-540-78929-1_5.

[Bha07] Rajendra Bhatia. *Positive Definite Matrices*. Princeton Series in Applied Mathematics. Princeton University Press, 2007. ISBN: 9780691129181. URL: http://www.jstor.org/stable/j.ctt7rxv2.

[BPT12] G. Blekherman, P. Parrilo, and R. Thomas. *Semidefinite Optimization and Convex Algebraic Geometry*. Ed. by Grigoriy Blekherman, Pablo A. Parrilo, and Rekha R. Thomas. Philadelphia, PA: Society for Industrial and Applied Mathematics, 2012. DOI: 10.1137/1.9781611972290. eprint: http://epubs.siam.org/doi/pdf/10.1137/1.9781611972290. URL: http://epubs.siam.org/doi/abs/10.1137/1.9781611972290.

[Blo+00] Vincent D. Blondel, Olivier Bournez, Pascal Koiran, and John N. Tsitsiklis. “The Stability of Saturated Linear Dynamical Systems Is Undecidable”. In: *STACS 2000, 17th Annual Symposium on Theoretical Aspects of Computer Science, Lille, France, February 2000, Proceedings*. Ed. by Horst Reichel and Sophie Tison. Vol. 1770. Lecture Notes in Computer Science. Springer, 2000, pp. 479–490. ISBN: 3-540-67141-2. DOI: 10.1007/3-540-46541-3_40. URL: http://dx.doi.org/10.1007/3-540-46541-3_40.

[BT99] Vincent D. Blondel and John N. Tsitsiklis. "Complexity of stability and controllability of elementary hybrid systems". In: *Automatica* 35.3 (1999), pp. 479–489. DOI: 10.1016/S0005-1098(98)00175-7. URL: http://dx.doi.org/10.1016/S0005-1098(98)00175-7.

[Bog+14] Sergiy Bogomolov, Goran Frehse, Marius Greitschus, Radu Grosu, Corina S. Pasareanu, Andreas Podelski, and Thomas Strump. "Assume-Guarantee Abstraction Refinement Meets Hybrid Systems". In: *Hardware and Software: Verification and Testing - 10th International Haifa Verification Conference, HVC 2014, Haifa, Israel, November 18-20, 2014. Proceedings*. Ed. by Eran Yahav. Vol. 8855. Lecture Notes in Computer Science. Springer, 2014, pp. 116–131. ISBN: 978-3-319-13337-9. DOI: 10.1007/978-3-319-13338-6_10. URL: http://dx.doi.org/10.1007/978-3-319-13338-6_10.

[BMP10] Sergiy Bogomolov, Corina Mitrohin, and Andreas Podelski. "Composing Reachability Analyses of Hybrid Systems for Safety and Stability". In: *Automated Technology for Verification and Analysis - 8th International Symposium, ATVA 2010, Singapore, September 21-24, 2010. Proceedings*. Ed. by Ahmed Bouajjani and Wei-Ngan Chin. Vol. 6252. Lecture Notes in Computer Science. Springer, 2010, pp. 67–81. ISBN: 978-3-642-15642-7. DOI: 10.1007/978-3-642-15643-4_7. URL: http://dx.doi.org/10.1007/978-3-642-15643-4_7.

[Bor99] Brian Borchers. "CSDP, a C Library for Semidefinite Programming." In: *Optimization Methods and Software* 10 (1999), pp. 613–623.

[Boy+94] S. Boyd, L.E. Ghaoui, E. Feron, and V. Balakrishnan. *Linear Matrix Inequalities in System and Control Theory*. Studies in Applied Mathematics. Society for Industrial and Applied Mathematics, 1994. ISBN: 9780898714852. URL: http://books.google.de/books?id=A06mvmmIKUkC.

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, Mar. 2004. ISBN: 0521833787. URL: http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20%5C&path=ASIN/0521833787.

[Bra94] M. S. Branicky. "Stability of switched and hybrid systems". In: *Proceedings of 1994 33rd IEEE Conference on Decision and Control*. Vol. 4. Dec. 1994, 3498–3503 vol.4. DOI: 10.1109/CDC.1994.411688.

[Bra98] M. S. Branicky. "Multiple Lyapunov functions and other analysis tools for switched and hybrid systems". In: *IEEE Transactions on Automatic Control* 43.4 (Apr. 1998), pp. 475–482. ISSN: 0018-9286. DOI: 10.1109/9.664150.

[Bro03] Christopher W. Brown. "QEPCAD B: A Program for Computing with Semi-algebraic Sets Using CADs". In: *SIGSAM Bull.* 37.4 (Dec. 2003), pp. 97–108. ISSN: 0163-5824. DOI: 10.1145/968708.968710. URL: http://doi.acm.org/10.1145/968708.968710.

[CTG06] C. Cai, A. R. Teel, and R. Goebel. "Results on relaxation theorems for hybrid systems". In: *Proceedings of the 45th IEEE Conference on Decision and Control*. Dec. 2006, pp. 276–281. DOI: 10.1109/CDC.2006.377007.

[CGT08] Chaohong Cai, Rafal Goebel, and Andrew R. Teel. "Smooth Lyapunov Functions for Hybrid Systems Part II: (Pre)Asymptotically Stable Compact Sets". In: *IEEE Trans. Automat. Contr.* 53.3 (2008), pp. 734–748. DOI: 10.1109/TAC.2008.919257. URL: http://dx.doi.org/10.1109/TAC.2008.919257.

[Che05] Guanrong Chen. "Stability of Nonlinear Systems". In: *Encyclopedia of RF and Microwave Engineering*. John Wiley & Sons, Inc., 2005. ISBN: 9780471654506. DOI: 10.1002/0471654507.eme413. URL: http://dx.doi.org/10.1002/0471654507.eme413.

[CK03] A. Chutinan and B.H. Krogh. "Computational techniques for hybrid system verification". In: *Automatic Control, IEEE Transactions on* 48.1 (2003), pp. 64–75. ISSN: 0018-9286. DOI: 10.1109/TAC.2002.806655.

[Col75] George E. Collins. "Hauptvortrag: Quantifier elimination for real closed fields by cylindrical algebraic decomposition". In: *Automata Theory and Formal Languages, 2nd GI Conference, Kaiserslautern, May 20-23, 1975.* Ed. by H. Barkhage. Vol. 33. Lecture Notes in Computer Science. Springer, 1975, pp. 134–183. ISBN: 3-540-07407-4. DOI: 10.1007/3-540-07407-4_17. URL: http://dx.doi.org/10.1007/3-540-07407-4_17.

[Cui07] Pieter J. L. Cuijpers. "On Bicontinuous Bisimulation and the Preservation of Stability". In: *Hybrid Systems: Computation and Control, 10th International Workshop, HSCC 2007, Pisa, Italy, April 3-5, 2007, Proceedings.* Ed. by Alberto Bemporad, Antonio Bicchi, and Giorgio C. Buttazzo. Vol. 4416. Lecture Notes in Computer Science. Springer, 2007, pp. 676–679. ISBN: 978-3-540-71492-7. DOI: 10.1007/978-3-540-71493-4_59. URL: http://dx.doi.org/10.1007/978-3-540-71493-4_59.

[DX15] Liyun Dai and Bican Xia. "Smaller SDP for SOS decomposition". English. In: *Journal of Global Optimization* 63.2 (2015), pp. 343–361. ISSN: 0925-5001. DOI: 10.1007/s10898-015-0300-9. URL: http://dx.doi.org/10.1007/s10898-015-0300-9.

[Dam+10] Werner Damm, Henning Dierks, Jens Oehlerking, and Amir Pnueli. "Towards Component Based Design of Hybrid Systems: Safety and Stability". In: *Essays in Memory of Amir Pnueli*. 2010, pp. 96–143.

[Dam+05] Werner Damm, Angelika Votintseva, Alexander Metzner, Bernhard Josko, Thomas Peikenkamp, and Eckard Böde. "Boosting re-use of embedded automotive applications through rich components". In: *Proceedings of Foundations of Interface Technologies* 2005 (2005).

[DMT10] Thao Dang, Oded Maler, and Romain Testylier. “Accurate Hybridization of Nonlinear Systems”. In: *Hybrid Systems: Computation and Control.* Stockholm, Sweden: ACM, 2010, pp. 11–20. ISBN: 978-1-60558-955-8. DOI: 10.1145/1755952.1755956.

[DH88] James H. Davenport and Joos Heintz. “Real Quantifier Elimination is Doubly Exponential”. In: *J. Symb. Comput.* 5.1/2 (1988), pp. 29–35. DOI: 10.1016/S0747-7171(88)80004-X. URL: http://dx.doi.org/10.1016/S0747-7171(88)80004-X.

[DB08] Leonardo De Moura and Nikolaj Bjørner. “Z3: An Efficient SMT Solver”. In: *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems.* TACAS’08/ETAPS’08. Budapest, Hungary: Springer-Verlag, 2008, pp. 337–340. ISBN: 978-3-540-78799-0. URL: http://dl.acm.org/citation.cfm?id=1792734.1792766.

[DS97] Andreas Dolzmann and Thomas Sturm. “REDLOG: Computer Algebra Meets Computer Logic”. In: *SIGSAM Bull.* 31.2 (June 1997), pp. 2–9. ISSN: 0163-5824. DOI: 10.1145/261320.261324. URL: http://doi.acm.org/10.1145/261320.261324.

[DM12] Parasara Sridhar Duggirala and Sayan Mitra. “Lyapunov abstractions for inevitability of hybrid systems”. In: *Hybrid Systems: Computation and Control (part of CPS Week 2012), HSCC’12, Beijing, China, April 17-19, 2012.* Ed. by Thao Dang and Ian M. Mitchell. ACM, 2012, pp. 115–124. ISBN: 978-1-4503-1220-2. DOI: 10.1145/2185632.2185652. URL: http://doi.acm.org/10.1145/2185632.2185652.

[EA06] Christian Ebenbauer and Frank Allgöwer. “Analysis and design of polynomial control systems using dissipation inequalities and sum of squares”. In: *Computers & Chemical Engineering* 30.10-12 (2006), pp. 1590–1602. DOI: 10.1016/j.compchemeng.2006.05.014. URL: http://dx.doi.org/10.1016/j.compchemeng.2006.05.014.

[Fis97] Gerd Fischer. *Lineare Algebra.* 10th ed. Grundkurs Mathematik. Braunschweig: Vieweg Verlag, Sept. 1997. ISBN: 3-528-77217-4.

[Frä+07] Martin Fränzle, Hardi Hungar, Christian Schmitt, and Boris Wirtz. *HLang: Compositional Representation of Hybrid Systems via Predicates.* Reports of SFB/TR 14 AVACS 20. ISSN: 1860-9821, http://www.avacs.org. SFB/TR 14 AVACS, July 2007.

[Fre+11] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. “SpaceEx: Scalable verification of hybrid systems”. In: *Computer Aided Verification.* Vol. 6806. Lecture Nodes in Computer Science. Springer Berlin Heidelberg, 2011, pp. 379–395.

[Fuj+07] Katsuki Fujisawa, Kazuhide Nakata, Makoto Yamashita, and Mituhiro Fukuda. "SDPA Project : Solving Large-Scale Semidefinite Programs". In: *Journal of the Operations Research Society of Japan* 50.4 (Dec. 2007), pp. 278–298. ISSN: 04534514. URL: http://ci.nii.ac.jp/naid/110006532053/en/.

[Gal09] Jean Gallier. *Notes on Convex Sets, Polytopes, Polyhedra, Combinatorial Topology, Voronoi Diagrams and Delaunay Triangulations.* Tech. rep. 650. University of Pennsylvania Department of Computer and Information Science, 2009. URL: http://repository.upenn.edu/cis_reports/650.

[Gao+10] Sicun Gao, M. Ganai, F. Ivancic, A Gupta, S. Sankaranarayanan, and E.M. Clarke. "Integrating ICP and LRA solvers for deciding nonlinear real arithmetic problems". In: *Formal Methods in Computer-Aided Design (FMCAD'10), 2010.* Oct. 2010, pp. 81–89.

[Gir05] Antoine Girard. "Reachability of Uncertain Linear Systems Using Zonotopes". In: *Hybrid Systems: Computation and Control.* Vol. 3414. Lecture Nodes in Computer Science. Springer Berlin Heidelberg, 2005, pp. 291–305. ISBN: 978-3-540-25108-8. DOI: 10.1007/978-3-540-31954-2_19.

[Hag14] Willem Hagemann. "Reachability Analysis of Hybrid Systems Using Symbolic Orthogonal Projections". English. In: *Computer Aided Verification.* Vol. 8559. Lecture Nodes in Computer Science. Springer Berlin Heidelberg, 2014, pp. 406–422.

[Hag15] Willem Hagemann. "Symbolic orthogonal projections: a new polyhedral representation for reachability analysis of hybrid systems". PhD thesis. Saarland University, 2015. URL: http://scidok.sulb.uni-saarland.de/volltexte/2015/6304/.

[HTP05] Esfandiar Haghverdi, Paulo Tabuada, and George J. Pappas. "Bisimulation relations for dynamical, control, and hybrid systems". In: *Theor. Comput. Sci.* 342.2-3 (2005), pp. 229–261. DOI: 10.1016/j.tcs.2005.03.045. URL: http://dx.doi.org/10.1016/j.tcs.2005.03.045.

[Hen96] Thomas A. Henzinger. "The Theory of Hybrid Automata". In: *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, USA, July 27-30, 1996.* IEEE Computer Society, 1996, pp. 278–292. ISBN: 0-8186-7463-6. DOI: 10.1109/LICS.1996.561342. URL: http://dx.doi.org/10.1109/LICS.1996.561342.

[Hen+98] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. "What's Decidable about Hybrid Automata?" In: *J. Comput. Syst. Sci.* 57.1 (1998), pp. 94–124. DOI: 10.1006/jcss.1998.1581. URL: http://dx.doi.org/10.1006/jcss.1998.1581.

[Ito+11] Hiroshi Ito, Zhong-Ping Jiang, Sergey N. Dashkovskiy, and Björn S. Rüffer. “A Small-Gain Theorem and Construction of Sum-Type Lyapunov Functions for Networks of iISS Systems”. In: *Proc. IEEE American Contr. Conf.* 2011, pp. 1971–1977.

[Ito+13] Hiroshi Ito, Zhong-Ping Jiang, Sergey Dashkovskiy, and Björn S. Rüffer. “Robust stability of networks of iISS systems: Construction of sum-type Lyapunov functions”. In: *IEEE Trans. Autom. Control* 58.5 (May 2013), pp. 1192–1207. DOI: 10.1109/TAC.2012.2231552.

[JCK07] Christian Jansson, Denis Chaykin, and Christian Keil. “Rigorous Error Bounds for the Optimal Value in Semidefinite Programming”. In: *SIAM J. Numerical Analysis* 46.1 (2007), pp. 180–200. DOI: 10.1137/050622870. URL: http://dx.doi.org/10.1137/050622870.

[Jha+07] Sumit Kumar Jha, Bruce H. Krogh, James E. Weimer, and Edmund M. Clarke. “Reachability for Linear Hybrid Automata Using Iterative Relaxation Abstraction”. In: *Hybrid Systems: Computation and Control, 10th International Workshop, HSCC 2007, Pisa, Italy, April 3-5, 2007, Proceedings.* Ed. by Alberto Bemporad, Antonio Bicchi, and Giorgio C. Buttazzo. Vol. 4416. Lecture Notes in Computer Science. Springer, 2007, pp. 287–300. ISBN: 978-3-540-71492-7. DOI: 10.1007/978-3-540-71493-4_24. URL: http://dx.doi.org/10.1007/978-3-540-71493-4_24.

[Kel14] Christopher M. Kellett. “A compendium of comparison function results”. In: *Mathematics of Control, Signals, and Systems* 26.3 (2014), pp. 339–374. ISSN: 1435-568X. DOI: 10.1007/s00498-014-0128-8. URL: http://dx.doi.org/10.1007/s00498-014-0128-8.

[Kha96] Hassan K. Khalil. *Nonlinear Systems.* 1st ed. Upper Saddle River, (N.J.): Prentice Hall, 1996. ISBN: 9780132280242. URL: https://books.google.de/books?id=qiBuQgAACAAJ.

[Kha00] Hassan K. Khalil. *Nonlinear Systems.* 1st ed. Pearson Education, 2000. ISBN: 9780131227408. URL: https://books.google.de/books?id=v%5C_BjPQAACAAJ.

[KV00] Alexander B. Kurzhanski and Pravin Varaiya. “Ellipsoidal Techniques for Reachability Analysis”. English. In: *Hybrid Systems: Computation and Control.* Vol. 1790. Lecture Nodes in Computer Science. Springer Berlin Heidelberg, 2000, pp. 202–214. ISBN: 978-3-540-67259-3. DOI: 10.1007/3-540-46430-1_19.

[LG09] Colas Le Guernic and Antoine Girard. “Reachability Analysis of Hybrid Systems Using Support Functions”. In: *Computer Aided Verification.* Vol. 5643. Lecture Nodes in Computer Science. Springer Berlin Heidelberg, 2009, pp. 540–554.

[Li+09] Li Li, Kai-Duo He, Ming Gu, and Xiao-Yu Song. “Equality Detection for Linear Arithmetic Constraints”. In: *Journal of Zhejiang University SCIENCE A* 10 (2009), pp. 1784–1789.

[Lib03] Daniel Liberzon. *Switching in Systems and Control.* Systems & Control: Foundations & Applications. Springer Science & Business Media, 2003. ISBN: 9780817642976.

[LSW94] Yuandan Lin, E. Sontag, and Yuan Wang. “Recent results on Lyapunov-theoretic techniques for nonlinear stability”. In: *American Control Conference, 1994.* Vol. 2. June 1994, 1771–1775 vol.2. DOI: 10.1109/ACC.1994.752377.

[Löf04] J. Löfberg. “YALMIP : A Toolbox for Modeling and Optimization in MATLAB”. In: *Proceedings of the 13th Conference on Computer-Aided Control System Design (CACSD'04).* Taipei, Taiwan, 2004.

[Lya07] Aleksandr M. Lyapunov. “Problème général de la stabilité du movement”. fre. In: *Annales de la faculté des sciences de Toulouse* 9 (1907), pp. 203–474. URL: http://eudml.org/doc/72801.

[LSV03] Nancy A. Lynch, Roberto Segala, and Frits W. Vaandrager. “Hybrid I/O automata”. In: *Inf. Comput.* 185.1 (2003), pp. 105–157. DOI: 10.1016/S0890-5401(03)00067-1. URL: http://dx.doi.org/10.1016/S0890-5401(03)00067-1.

[MD 95] B. Reznick M.D. Choi T.Y. Lam. “Sums of squares of real polynomials”. In: K-theory and Algebraic Geometry: Connections with Quadratic Forms and Division Algebras 58.Bd. 58 (1995). Ed. by American Mathematical Society, pp. 103–126. ISSN: 0082-0717. URL: https://books.google.de/books?id=OedDnQEACAAJ.

[MA14] Michael J. McCourt and Panos J. Antsaklis. “SOS methods for demonstrating dissipativity for switched systems”. In: *American Control Conference, ACC 2014, Portland, OR, USA, June 4-6, 2014.* IEEE, 2014, pp. 3936–3941. ISBN: 978-1-4799-3272-6. DOI: 10.1109/ACC.2014.6858757. URL: http://dx.doi.org/10.1109/ACC.2014.6858757.

[MT00] Ian Mitchell and Claire Tomlin. “Level Set Methods for Computation in Hybrid Systems”. In: *Hybrid Systems: Computation and Control.* 2000, pp. 310–323. DOI: 10.1007/3-540-46430-1_27. URL: http://dx.doi.org/10.1007/3-540-46430-1_27.

[Nuz+10] Pierluigi Nuzzo, Alberto Alessandro Angelo Puggelli, Sanjit A. Seshia, and Alberto L. Sangiovanni-Vincentelli. *CalCS: SMT Solving for Non-linear Convex Constraints.* Tech. rep. UCB/EECS-2010-100. EECS Department, University of California, Berkeley, June 2010. URL: http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-100.html.

[Oeh11] Jens Oehlerking. “Decomposition of Stability Proofs for Hybrid Systems”. PhD thesis. Carl von Ossietzky University of Oldenburg, Department of Computer Science, Oldenburg, Germany, 2011.

[OBT07] Jens Oehlerking, Henning Burchardt, and Oliver E. Theel. “Fully Automated Stability Verification for Piecewise Affine Systems”. In: *Proceedings of the 10th international conference on Hybrid systems: computation and control (HSCC’07)*. 2007, pp. 741–745.

[OT09] Jens Oehlerking and Oliver E. Theel. “Decompositional Construction of Lyapunov Functions for Hybrid Systems”. In: *Proceedings of the 12th International Conference on Hybrid Systems: Computation and Control (HSCC’09)*. 2009, pp. 276–290.

[Pap+13] A. Papachristodoulou, J. Anderson, G. Valmorbida, S. Prajna, P. Seiler, and P. A. Parrilo. *SOSTOOLS: Sum of squares optimization toolbox for MATLAB*. `http://arxiv.org/abs/1310.4716`, 2013.

[PL03] Pablo A. Parrilo and Sanjay Lall. “Semidefinite Programming Relaxations and Algebraic Optimization in Control”. In: *Eur. J. Control* 9.2-3 (2003), pp. 307–321. DOI: `10.3166/ejc.9.307-321`. URL: `http://dx.doi.org/10.3166/ejc.9.307-321`.

[Pet99] Stefan Pettersson. “Analysis and Design of Hybrid Systems”. PhD thesis. CTH, Gothenburg, Sweden, 1999.

[PW06] Andreas Podelski and Silke Wagner. “Model Checking of Hybrid Systems: From Reachability Towards Stability”. In: *Hybrid Systems: Computation and Control, 9th International Workshop, HSCC 2006, Santa Barbara, CA, USA, March 29-31, 2006, Proceedings*. Ed. by João P. Hespanha and Ashish Tiwari. Vol. 3927. Lecture Notes in Computer Science. Springer, 2006, pp. 507–521. ISBN: 3-540-33170-0. DOI: `10.1007/11730637_38`. URL: `http://dx.doi.org/10.1007/11730637_38`.

[PW07a] Andreas Podelski and Silke Wagner. “A Sound and Complete Proof Rule for Region Stability of Hybrid Systems”. In: *Hybrid Systems: Computation and Control, 10th International Workshop, HSCC 2007, Pisa, Italy, April 3-5, 2007, Proceedings*. Ed. by Alberto Bemporad, Antonio Bicchi, and Giorgio C. Buttazzo. Vol. 4416. Lecture Notes in Computer Science. Springer, 2007, pp. 750–753. ISBN: 978-3-540-71492-7. DOI: `10.1007/978-3-540-71493-4_76`. URL: `http://dx.doi.org/10.1007/978-3-540-71493-4_76`.

[PW07b] Andreas Podelski and Silke Wagner. “Region Stability Proofs for Hybrid Systems”. In: *Formal Modelling and Analysis of Timed Systems (FORMATS’07)*. 2007, pp. 320–335.

[PW07c] Andreas Podelski and Silke Wagner. "Region Stability Proofs for Hybrid Systems". In: *Formal Modeling and Analysis of Timed Systems, 5th International Conference, FORMATS 2007, Salzburg, Austria, October 3-5, 2007, Proceedings*. Ed. by Jean-François Raskin and P. S. Thiagarajan. Vol. 4763. Lecture Notes in Computer Science. Springer, 2007, pp. 320–335. ISBN: 978-3-540-75453-4. DOI: 10.1007/978-3-540-75454-1_23. URL: http://dx.doi.org/10.1007/978-3-540-75454-1_23.

[PW98] Victoria Powers and Thorsten Wörmann. "An algorithm for sums of squares of real polynomials". In: *Journal of Pure and Applied Algebra* 127.1 (1998), pp. 99–104. ISSN: 0022-4049. DOI: http://dx.doi.org/10.1016/S0022-4049(97)83827-3. URL: http://www.sciencedirect.com/science/article/pii/S0022404997838273.

[Pra12] Pavithra Prabhakar. "Foundations for approximation based analysis of stability properties of hybrid systems". In: *Proceedings of the 50th Annual Allerton Conference on Communication, Control, and Computing*. 2012, pp. 1602–1609. DOI: 10.1109/Allerton.2012.6483412. URL: http://dx.doi.org/10.1109/Allerton.2012.6483412.

[Pra+13] Pavithra Prabhakar, Parasara Sridhar Duggirala, Sayan Mitra, and Mahesh Viswanathan. "Hybrid Automata-Based CEGAR for Rectangular Hybrid Systems". In: *Verification, Model Checking, and Abstract Interpretation, 14th International Conference, VMCAI 2013, Rome, Italy, January 20-22, 2013. Proceedings*. Ed. by Roberto Giacobazzi, Josh Berdine, and Isabella Mastroeni. Vol. 7737. Lecture Notes in Computer Science. Springer, 2013, pp. 48–67. ISBN: 978-3-642-35872-2. DOI: 10.1007/978-3-642-35873-9_6. URL: http://dx.doi.org/10.1007/978-3-642-35873-9_6.

[PDV12] Pavithra Prabhakar, Geir E. Dullerud, and Mahesh Viswanathan. "Preorders for reasoning about stability". In: *Proceedings of the 15th International Conference on Hybrid Systems: Computation and Control (HSCC'12)*. 2012, pp. 197–206. DOI: 10.1145/2185632.2185662. URL: http://doi.acm.org/10.1145/2185632.2185662.

[PLM13] Pavithra Prabhakar, Jun Liu, and Richard M. Murray. "Pre-orders for reasoning about stability properties with respect to input of hybrid systems". In: *Proceedings of the International Conference on Embedded Software (EMSOFT'13)*. 2013, pp. 1–10. DOI: 10.1109/EMSOFT.2013.6658602. URL: http://dx.doi.org/10.1109/EMSOFT.2013.6658602.

[PS13] Pavithra Prabhakar and Miriam Garcia Soto. "Abstraction Based Model-Checking of Stability of Hybrid Systems". In: *Proceedings of the 25th International Conference on Computer Aided Verification (CAV'13)*. 2013, pp. 280–295. DOI: 10.1007/978-3-642-39799-8_20. URL: http://dx.doi.org/10.1007/978-3-642-39799-8_20.

[PS14] Pavithra Prabhakar and Miriam Garcia Soto. "An algorithmic approach to stability verification of polyhedral switched systems". In: *American Control Conference, ACC 2014, Portland, OR, USA, June 4-6, 2014*. IEEE, 2014, pp. 2318–2323. ISBN: 978-1-4799-3272-6. DOI: 10.1109/ACC.2014.6859056. URL: http://dx.doi.org/10.1109/ACC.2014.6859056.

[PS15] Pavithra Prabhakar and Miriam Garcia Soto. "Foundations of Quantitative Predicate Abstraction for Stability Analysis of Hybrid Systems". In: *Verification, Model Checking, and Abstract Interpretation - 16th International Conference, VMCAI 2015, Mumbai, India, January 12-14, 2015. Proceedings.* Ed. by Deepak D'Souza, Akash Lal, and Kim Guldstrand Larsen. Vol. 8931. Lecture Notes in Computer Science. Springer, 2015, pp. 318–335. ISBN: 978-3-662-46080-1. DOI: 10.1007/978-3-662-46081-8_18. URL: http://dx.doi.org/10.1007/978-3-662-46081-8_18.

[PS16] Pavithra Prabhakar and Miriam Garcia Soto. "Hybridization for Stability Analysis of Switched Linear Systems". In: *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control, HSCC 2016, Vienna, Austria, April 12-14, 2016*. Ed. by Alessandro Abate and Georgios E. Fainekos. ACM, 2016, pp. 71–80. ISBN: 978-1-4503-3955-1. DOI: 10.1145/2883817.2883840. URL: http://doi.acm.org/10.1145/2883817.2883840.

[PV13] Pavithra Prabhakar and Mahesh Viswanathan. "On the decidability of stability of hybrid systems". In: *Proceedings of the 16th international conference on Hybrid systems: computation and control, HSCC 2013, April 8-11, 2013, Philadelphia, PA, USA*. Ed. by Calin Belta and Franjo Ivancic. ACM, 2013, pp. 53–62. ISBN: 978-1-4503-1567-8. DOI: 10.1145/2461328.2461339. URL: http://doi.acm.org/10.1145/2461328.2461339.

[PP03] S. Prajna and A. Papachristodoulou. "Analysis of Switched and Hybrid Systems - Beyond Piecewise Quadratic Methods". In: *American Control Conference, 2003. Proceedings of the 2003*. Vol. 4. June 2003, 2779–2784 vol.4. DOI: 10.1109/ACC.2003.1243743.

[RS10] Stefan Ratschan and Zhikun She. "Providing a Basin of Attraction to a Target Region of Polynomial Systems by Computation of Lyapunov-Like Functions". In: *SIAM J. Control and Optimization* 48.7 (2010), pp. 4377–4394. DOI: 10.1137/090749955. URL: http://dx.doi.org/10.1137/090749955.

[Rez78] Bruce Reznick. "Extremal PSD forms with few terms". In: *Duke Math. J.* 45.2 (June 1978), pp. 363–374. DOI: 10.1215/S0012-7094-78-04519-2. URL: http://dx.doi.org/10.1215/S0012-7094-78-04519-2.

[SDP12] Alberto L. Sangiovanni-Vincentelli, Werner Damm, and Roberto Passerone. "Taming Dr. Frankenstein: Contract-Based Design for Cyber-Physical Systems". In: *Eur. J. Control* 18.3 (2012), pp. 217–238. DOI: 10.3166/ejc.18.217-238. URL: http://dx.doi.org/10.3166/ejc.18.217-238.

[SDI08] Sriram Sankaranarayanan, Thao Dang, and Franjo Ivančić. “Symbolic Model Checking of Hybrid Systems Using Template Polyhedra”. In: *Tools and Algorithms for the Construction and Analysis of Systems*. Springer Berlin Heidelberg, 2008, pp. 188–202. ISBN: 978-3-540-78799-0. DOI: 10.1007/978-3-540-78800-3_14.

[Sas+15] Mohamed Amin Ben Sassi, Sriram Sankaranarayanan, Xin Chen, and Erika Ábrahám. “Linear relaxations of polynomial positivity for polynomial lyapunov function synthesis”. In: *IMA Journal of Mathematical Control and Information* (2015), dnv003.

[Sei13] Peter Seiler. “SOSOPT: A Toolbox for Polynomial Optimization”. In: *CoRR* abs/1308.1889 (2013). URL: http://arxiv.org/abs/1308.1889.

[She+13] Zhikun She, Haoyang Li, Bai Xue, Zhiming Zheng, and Bican Xia. “Discovering polynomial Lyapunov functions for continuous dynamical systems”. In: *J. Symb. Comput.* 58 (2013), pp. 41–63. DOI: 10.1016/j.jsc.2013.06.003. URL: http://dx.doi.org/10.1016/j.jsc.2013.06.003.

[SX14] Zhikun She and Bai Xue. “Discovering Multiple Lyapunov Functions for Switched Hybrid Systems”. In: *SIAM J. Control and Optimization* 52.5 (2014), pp. 3312–3340. DOI: 10.1137/130934313. URL: http://dx.doi.org/10.1137/130934313.

[Stu98] Bernd Sturmfels. “Polynomial equations and convex polytopes”. In: *American Mathematical Monthly* 105.10 (1998), pp. 907–922.

[Tar51] Alfred Tarski. “A Decision Method for Elementary Algebra and Geometry”. In: *Bulletin of the American Mathematical Society* 59 (1951). Ed. by Robert McNaughton.

[TT13] Gerald Teschl and Susanne Teschl. *Mathematik für Informatiker: Band 1: Diskrete Mathematik und Lineare Algebra*. 4th ed. Springer-Verlag Berlin Heidelberg, 2013, p. 522. ISBN: 978-3-642-37971-0. DOI: 10.1007/978-3-642-37972-7.

[War62] Stephen Warshall. “A Theorem on Boolean Matrices”. In: *Journal of the ACM* 9 (1962), pp. 11–12.

[Wei88] Volker Weispfenning. “The Complexity of Linear Problems in Fields”. In: *J. Symb. Comput.* 5.1/2 (1988), pp. 3–27. DOI: 10.1016/S0747-7171(88)80003-8. URL: http://dx.doi.org/10.1016/S0747-7171(88)80003-8.

[Wil94] J.H. Wilkinson. *Rounding Errors in Algebraic Processes*. Dover Publications, Incorporated, 1994. ISBN: 9780486679990. URL: http://books.google.de/books?id=yFogU9Ot-qsC.

[YMH98] Hui Ye, A. N. Michel, and Ling Hou. “Stability theory for hybrid dynamical systems”. In: *IEEE Transactions on Automatic Control* 43.4 (Apr. 1998), pp. 461–474. ISSN: 0018-9286. DOI: 10.1109/9.664149.

[Zsc15] Philipp Zschoche. “Automatisierte Erkennung und Eliminierung impliziter Äquivalenzen in quantifizierten nicht-linearen Ungleichungssystemen zur Stabilitätsverifikation hybrider Systeme mittels Lyapunov-Theorie”. Bachelor’s Thesis. University of Oldenburg, 2015. URL: https://www.uni-oldenburg.de/informatik/svs/lehre/abschlussarbeiten/2015/automatisierte-erkennung-und-eliminierung-impliziter-aquivalenzen-in-quantifizierten-nicht-linearen-ungleichungssystemen-zur-stabilitatsverifikation-hybrider-systeme-mittels-lyapunov-theorie/.

# Glossary

**conditional global asymptotic stability** An modified version of globally asymptotically stable that needs to be satisfied only in periods of time where a certain condition is satisfied. 141

**flow** A flow of a location denotes how the valuations of the continuous variables are allowed to evolve. The flow is usually described as differential equations or differential inclusions. 18, 120

**global asymptotic stability under assumptions** An modified version of globally asymptotically stable that needs to be satisfied only for trajectories that always satisfy a certain assumption. 136

**global exponential stability** Like globally asymptotically stable but additionally the decay bounded by an exponential function. 24

**global asymptotic stability** A system that is globally asymptotically stable satisfies that (1) starting close to the equilibrium, the system will remain close to the equilibrium point, (2) eventually converge to the equilibrium point. 24, 145

**guard** A guard denotes when a transition is enabled, i. e., a trajectory is allowed to take the transition. 18, 120

**hybrid I/O automaton** An extension of a hybrid automaton with internal and external variables as well as actions or events to allow communication within a network of hybrid I/O automata. 120

**hybrid automaton** A hybrid automaton is a formalism to model hybrid systems. It captures the discrete behavior in form of an automaton and continues behavior in form of differential equations or inclusions. The vertices of the automaton are called locations and are annotated with invariants and differential equations or inclusions. The transitions of the automaton are annotated with guards and updates. 17

**hypothetical syllogism** A rule of inference.

$$\text{(Hypothetical Syllogism)}\ \frac{\mathrm{P} \Rightarrow \mathrm{Q} \qquad \mathrm{Q} \Rightarrow \mathrm{R}}{\mathrm{P} \Rightarrow \mathrm{R}}$$

Sometimes also called “Modus Barbara”. 155, 156

**invariant** An invariant of a location denotes when a location is enabled, i. e., a trajectory is allowed to stay in the location and time evolves. 18, 120

**linear matrix inequality** A convex constraint stating that a linear combination of matrices has to be positive semidefinite. 36

**location** A vertex of a hybrid automaton. 17, 18, 120

**positive semidefinite** A symmetric real matrix $M$ is positive (semi-)definite iff for every non-zero column vector $\mathbf{x}$ holds that the scalar $\mathbf{x}^T M\mathbf{x} > 0$ (resp. $\mathbf{x}^T M\mathbf{x} \geq 0$). A function $f : D \rightarrow \mathbb{R}$ is positive (semi-)definite iff $f(0) = 0$ and $f(\mathbf{x}) > 0$ (resp. $f(\mathbf{x}) \geq 0$)for every $x \in D \setminus \{0\}$. 14

**semidefinite program** A convex optimization problem minimizing a linear objective function s.t. while a linear combination of matrices has to be positive semidefinite. 37

**sum-of-squares** A polynomial is a sum-of-squares iff it can be rewritten as a sum of squared terms. This implies that the polynomial is positive semidefinite. 34

**transition** An edge of a hybrid automaton. 18, 120

**update** An update might change the valuation of certain continues variables when a trajectory takes a transition. 18, 120

**variables** A variable is an entity taking values. It is differentiated between continuous variables $v \in \mathit{Var}$ taking real values and the discrete location variable $l$ that is location valued. A vector of valuations for each continuous variable is called a continuous state. A valuation for the discrete location variable is called a discrete state. A pair of a continuous state and a discrete state is called the hybrid state. The set of all possible values of the variables is call the continuous state space. 17

# Acronyms

**ADAS** advanced driver assistance system

**ADF** autonomous driving function

**BNF** Backus–Naur form

**CEGAR** counterexample guided abstraction refinement

**conGAS** conditional global asymptotic stability. *Glossary:* conditional global asymptotic stability

**CPS** cyber-physical system

**DNF** disjunctive normal form. *Glossary:* disjunctive normal form

**FOL** first order logic

**GAS** global asymptotic stability. *Glossary:* global asymptotic stability

**GAS-Asm** global asymptotic stability under assumptions. *Glossary:* global asymptotic stability under assumptions

**GES** global exponential stability. *Glossary:* global exponential stability

**GLF** global Lyapunov function

**HA** hybrid automaton. *Glossary:* hybrid automaton

**HAL** hybrid automaton language

**HIOA** hybrid I/O automaton. *Glossary:* hybrid I/O automaton

**HS** hybrid system. *Glossary:* hybrid system

**LF** Lyapunov function

**LFP** Lyapunov function projections

**LLF** local Lyapunov function

**LMI** linear matrix inequality. *Glossary:* linear matrix inequality

**LP** linear program

**LPoP** linear polynomial optimization problem

**NP** non-deterministic polynomial time

**NRA** non-linear real arithmetic

**PD** positive definite. *Glossary:* positive semidefinite

**PI-controller** PI-controller. *Glossary:* PI-controller

**PSD** positive semidefinite. *Glossary:* positive semidefinite

**SAT** boolean satisfiability problem

**SCC** strongly connected component

**SDP** semidefinite program. *Glossary:* semidefinite program

**SDP solver** semidefinite program solver. *Glossary:* semidefinite program

**semidefinite programming** semidefinite programming. *Glossary:* semidefinite program

**SLLF** single-location Lyapunov function

**SMT** satisfiability modulo theory

**SOS** sum-of-squares. *Glossary:* sum-of-squares

# List of Figures and Tables