CARL
VON
OSSIETZKY
universität OLDENBURG

Fakultät II – Informatik, Wirtschafts- und Rechtswissenschaften
Department für Informatik

Decomposition of Stability Proofs for Hybrid Systems

Dissertation zur Erlangung des Grades eines Doktors der Ingenieurwissenschaften
(Dr.-Ing.)

vorgelegt von

Jens Oehlerking

Gutachter:

Prof. Dr.-Ing. Oliver Theel
Prof. Dr. Martin Fränzle

Datum der Verteidigung: 16.12.2011

First, I would like to thank the people from my group, the System Software and Distributed Systems Group of the University of Oldenburg, for providing me with a friendly and supportive work environment, and for spirited discussions. I'd also like to thank the members of the Networking and Telecommunications and Environmental Informatics groups for lunch time talk, cake, and crazy movies.

This thesis would not have been possible without the people within the AVACS research center. Dear colleagues in Oldenburg, Saarbrücken, Freiburg, and Prague, I am grateful for the inspiration and constructive feedback you provided during our regular meetings. Also, I would like to mention Robert Rucker, who programmed the hybrid system simulation engine that helped me create some of the images in this thesis.

The best thing about being young
is not knowing how stupid you are.
The worst thing about growing old
is forgetting how stupid you were
when you were young.

N. Senada

Abstract

The verification of hybrid systems, encompassing both discrete-time and continuous-time behavior, is a problem of rising importance. Hybrid behavior occurs wherever a digital system, operating in discrete time, interacts with a real-world environment, which evolves in continuous time. One desired property of hybrid systems is global asymptotic stability. A globally asymptotically stable system converges toward a pre-defined target state from everywhere in the state space. This property guarantees that the system is robust to temporary external disturbances, correcting their effects on its own accord. Stability proofs are usually conducted with the help of so-called Lyapunov functions, which act as generalized energy functions of the system. A Lyapunov function maps each of the possible system states onto an energy value, such that the energy decreases as the system evolves. The existence of such a function serves as a proof of global asymptotic stability. Furthermore, numerical methods for the computation of such functions exist, allowing for automated stability verification. While these methods work well for small-scale systems, they do, however, not scale up well to systems with large discrete state spaces that appear in many real-world applications. Therefore, it is desirable to conduct these computations in a decompositional manner, solving many small-scale problems instead of one large-scale problem.

This thesis bridges this gap by introducing an automatable decomposition methodology that works on hybrid automaton models. A hybrid automaton is viewed as a graph and decomposed into sub-components, for which Lyapunov function computation are conducted individually. The results of these computations are then combined to yield a stability proof for the entire system. These local proofs are not only more lightweight than the larger (and possibly intractable) standard proof, but also allow for the localization of the problem if the computation fails, and the compositional construction of complex stable hybrid automata. The decomposition takes place on two levels: strongly connected components and cycles of the automaton. Our results prove that strongly connected components can be analyzed completely separately. For the cycle-based second level, properties of Lyapunov functions are exploited to combine local proofs into a global proof.

Furthermore, the results are extended to the domain of probabilistic hybrid automata, which are a combination of hybrid systems and Markov processes. The decomposition results in this thesis are generalized to this setting, to allow for automatable decompositional stability proofs for probabilistic systems. For both non-probabilistic and probabilistic systems, the decompositional approach is also exploited to yield a set of rules for the structured construction of stable hybrid systems with complex discrete behavior. Furthermore, we derive a general method for the hierarchical component-based design of stabilizing hybrid controllers for a given plant, building on the decomposition results.

Kurzzusammenfassung

Die Verifikation hybrider Systeme mit sowohl zeitdiskretem als auch zeitkontinuierlichem Verhalten ist ein Problem mit wachsender Bedeutung. Hybrides Verhalten tritt immer dann auf, wenn ein digitales System mit einer sich zeitkontinuierlich entwickelnden realen Umgebung interagiert. Eine wünschenswerte Eigenschaft hybrider Systeme ist globale asymptotische Stabilität. Ein global asymptotisch stabiles System konvergiert von jedem Zustand im seinem Zustandsraum zu einem vordefinierten Zielzustand, womit das System in der Lage ist, die Effekte vorübergehender externer Störungen selbstständig zu korrigieren. Stabilitätsbeweise werden üblicherweise mittels so genannter Ljapunowfunktionen durchgeführt, die als eine generalisierte Energiefunktion des Systems fungieren. Eine Ljapunowfunktion bildet jeden möglichen Systemzustand auf einen Energiewert ab, so dass der Energiewert entlang der Systemläufe des Systems monoton sinkt, und die Existenz solche einer Funktion beweist globale asymptotische Stabilität. Des Weiteren existieren numerische Methoden, um solche Funktionen zu berechnen. Hiermit wird die automatische Verifikation der Stabilität hybrider Systeme ermöglicht. Diese Methoden funktionieren zwar zuverlässig auf kleinen Systemen, die Skalierbarkeit auf Systeme realistischer Größe ist aber in der Regel nicht gegeben. Aus diesem Grund ist es wünschensert, Stabilitätsverifikation dekompositionell durchzuführen, so dass das Lösen mehrerer kleiner Verifikationsprobleme an die Stelle des Lösens eines großen Verifikationsproblems treten kann.

Diese Arbeit schließt diese Lücke durch ein automatisierbares Dekompositionsverfahren auf hybriden Automaten, in dem die Automaten als Graphen aufgefasst und zerlegt werden. Auf den resultierenden Teilautomaten können dann die notwendigen Berechnungen einzeln durchgeführt werden, deren Resultate dann wieder zu einen Beweis der Stabilität des Gesamtsystems zusammengefügt werden können. Diese lokalen Berechnungen sind nicht nur einfacher handhabbar als der Standardbeweis, sondern erlauben auch eine Fehlerdiagnose, falls der Beweis fehlschlägt, und den kompositionellen Entwurf komplexer stabiler Automaten. Die Dekomposition findet auf zwei Ebenen statt: starke Zusammenhangskomponenten und Zyklen. Die Ergebnisse dieser Arbeit zeigen, dass starke Zusammenhangskomponenten vollständig unabhängig voneinander betrachtet werden können, während auf der Zyklenebene Eigenschaften von Ljapunowfunktionen für den Beweis genutzt werden können.

Des Weiteren werden die Ergebnisse generalisiert für probabilistische hybride Automaten, eine Kombination aus hybriden Automaten und Markowprozessen. Die Dekomposition wird auf diese Systemklasse übertragen, so dass automatische, dekompositionelle Stabilitätsbeweise auch für diese Systemklasse möglich werden. Sowohl für nicht-probabilistische als auch für probabilistische Systeme wird der Dekompositionsansatz zudem ausgenutzt, um eine Menge an Regeln für den strukturierten Entwurf stabiler hybrider System aufzustellen. Zudem wird hieraus ein Verfahren abgeleitet, welches den hierarchischen, komponentenbasierten Entwurf stabiler hybrider Regler für eine gegebene Regelstrecke ermöglicht.

Contents

1 Introduction

This thesis tackles a particularly difficult problem in the field of automatic verification, namely *global asymptotic stability* proofs of *hybrid systems*. Consider a classic control loop setup as given in Figure 1.1. This loop consists of a plant, which should be driven toward a desired value, and a controller communicating with the plant through sensors and actuators. The desired value, the *set point*, is selected externally. For instance, the controller could represent a heating system controlling the temperature of a room via some heating and cooling devices. In this example, the plant is a model of the room, and the controller measures and influences its temperature, while the set point is the desired temperature which is set by the (human) operator of the system. Additionally, external disturbances might influence the plant. Whenever such an external disturbance occurs, perturbing the room temperature, we now expect the controller to steer the temperature back to this set point after the disturbance has ceased. Furthermore, we expect this to happen in an acceptable time frame and also expect there is no significant overshoot or undershoot in temperature. Also, when the operator changes the set point, we require the temperature to converge toward this new value.

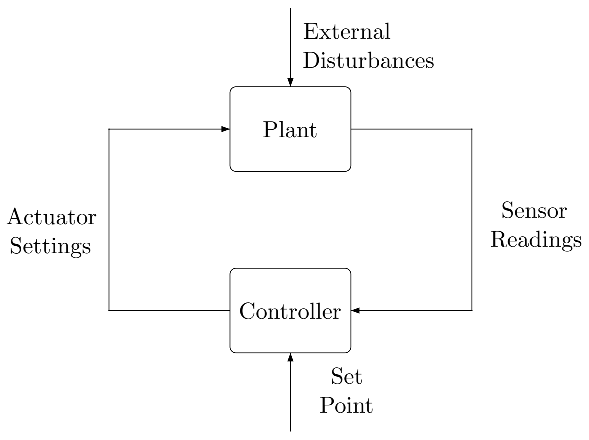


Figure 1.1: Classic Control Setup

In control theory, convergence to the set point is guaranteed by the system property called *asymptotic stability*. The property is defined with respect to a so-called *equilibrium point*, which in our example is the set point. Asymptotic stability implies converges to the equilibrium point in the absence of external disturbances. This is a property of the system as a whole, including both the plant and the controller, the *closed-loop* system. Figure 1.2(a) shows such a convergent trajectory, and Figure 1.2(b) shows a divergent trajectory of an unstable system. For a *globally asymptotically stable* system, we require that all trajectories of the system are of the type shown in Figure 1.2(a). Divergent, as well as oscillating trajectories must not exist in such a system. Here, the word “global” implies that this property holds for all initial states. The initial states can, for instance, be interpreted as the state after an external disturbance has perturbed the system or after a new set point has been chosen. In contrast, the property *local asymptotic stability* only guarantees such a behavior for some parts of the state space, which are often called *regions of attraction.* To make local stability properties useful in practice, the region of attraction must be sufficiently large.

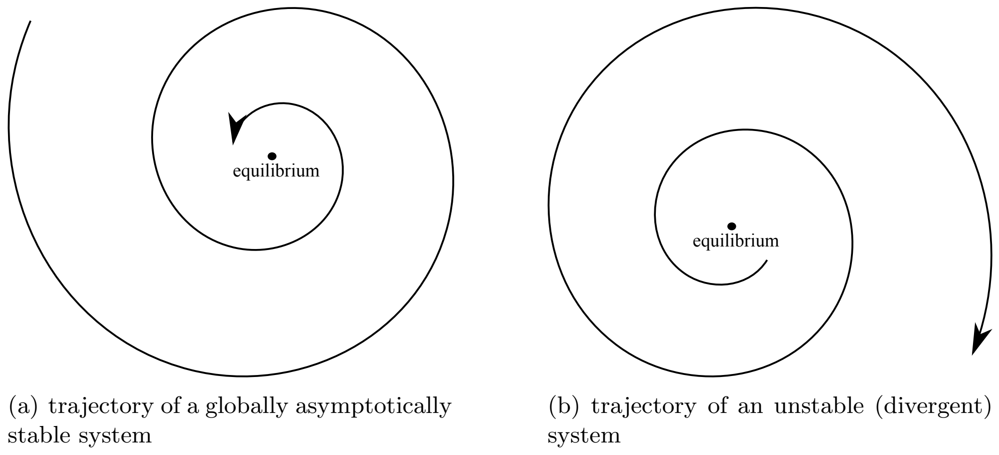

equilibrium
equilibrium
(a) trajectory of a globally asymptotically stable system
(b) trajectory of an unstable (divergent) system

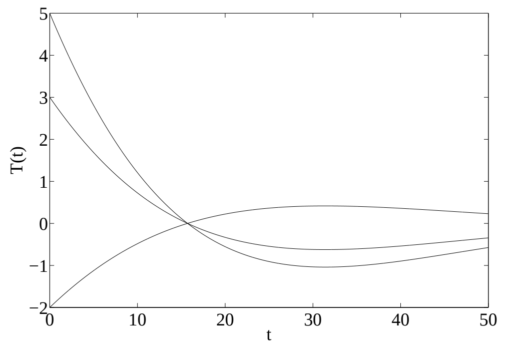
Figure 1.2: Trajectories of Stable and Unstable Systems

The simplest form of asymptotically stable temperature control loop is given by the differential equation $\dot{T} = -cT$, where T is the difference between the actual room temperature and the set point, and where c is a positive constant describing the convergence rate. This system will simply exponentially converge toward $T = 0$ for any initial state, as $x(t) = x(0)e^{-ct}$ is the solution of the differential equation. This means that the room temperature will always converge back to the set point, from any initial state. Therefore, the system described by this differential equation is globally asymptotically stable.

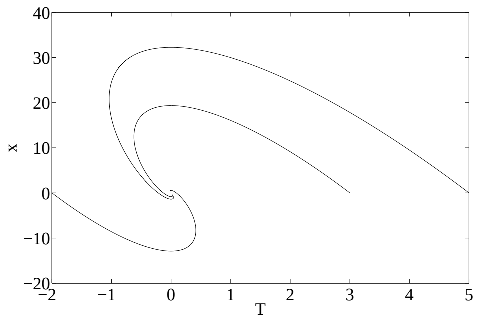
Of course, closed-loop models of real systems usually contain several variables which may interact in complex ways, making the analysis of this stability property much more complicated. For instance, a proportional-integral temperature controller could contain an additional internal variable x modeling the integral part, leading to a control loop described by the differential equations $\dot{T} = -c_1T - c_2x$ and $\dot{x} = T$, for some positive constants c_1 and c_2. For this system, some (basic) knowledge of control theory is required to decide whether it is stable for given constants c_1 and c_2. This type of controller can also consider the history of the temperature T via the integrator variable x. Example trajectories of such a system with $c_1 = 0.1$ and $c_2 = 0.005$ are given in Figure 1.3.

Since global asymptotic stability

- implies that any transient external disturbance on the system variables is eventually compensated,



(a) temperature vs. time



(b) temperature vs. integrator variable

Figure 1.3: Example Trajectories for the Proportional-integral Temperature Controller

- guarantees that, upon changing the set point of a system, the new set point is tracked properly with the system state converging towards it, and
- lends itself well to some very efficient analysis methods,

it is a central property of a control loop which is of great interest in the control community.

Clearly, asymptotic stability entails a conjunction of liveness properties: for any radius around the set point, we then know that the system state will eventually stay within this radius, if the system behaves as modeled. Since liveness properties imply "that something good eventually happens," and not necessarily with a bound on the overall time, they are inherently more difficult to prove than safety properties (i.e., properties implying that "something bad never happens"). This additional difficulty stems from the fact that one must ensure that there is a continuing progress in the system's behavior towards the goal (in our case, towards the equilibrium). How this progress is defined depends very much on the system itself. For instance, as indicated in Figure 1.3, the proportional-integral temperature controller can slightly overshoot the set point, so that the temperature might temporarily move away from the equilibrium. Thus, it is not sufficient to simply measure progress by the Euclidean distance to the set point. For safety properties, on the other hand, it is sufficient to prove that an action leading to something bad is forbidden at all times. Therefore, traditional verification approaches for discrete systems (e.g., model checking) generally have greater difficulties solving liveness problems than safety problems.

Another degree of difficulty in the problems addressed in this thesis stems from the class of systems that is considered. *Hybrid systems* are systems which operate on a continuous time line, but can exhibit both continuous evolution of some variables (usually given by ordinary differential equations or inclusions), as well as discrete updates of some variables. Such system models occur in fields like embedded system design, industrial automation, networked control, or biological systems, as well as many other application scenarios. For instance, a digital autopilot system has to interact with a continuously changing environment, and software controlling chemical processes has to deal with chemical reactions happening in real time. Historically, the analysis of continuous-time behavior lies within the domain of applied mathematics and control theory, while the models for discrete behavior usually stem from computer science. Hybrid systems lie on the boundary between these two worlds, combining continuous evolution of the system state with discrete events representing different modes of operation and logic based decision making.

The discrete behavior often takes the form of *discrete switches* between different control strategies. For instance, the temperature controller could implement "emergency behavior" for very high or very low temperatures. When one of these emergency situations occurs, then the controller would switch to a different differential equation for controlling the continuous variables, in this case the temperature.

One powerful hybrid system model consists of an augmented finite automaton, whose discrete states (also called *modes*) describe the discrete part of the hybrid system. Each discrete state is then mapped onto a differential equation (or inclusion), describing the evolution of the system's continuous variables while the system is in this particular discrete state, and optionally some invariant constraints that must hold while the automaton is in this discrete state. Discrete events are modeled as transitions from one discrete state to another one, and can be triggered by guard conditions which may fire if the continuous variables take a certain value (e.g., the temperature controller switches if the room temperature becomes too high), random experiments (i.e., probabilistic switches) or even completely non-deterministically (i.e., discrete events are seen as outside influences that can occur unpredictably). The guard conditions are often interpreted as "lazy," that is, if a guard becomes true the corresponding transition *can* be taken, but taking them is not required unless the current invariant condition is also violated. This model class is called *hybrid automaton* [Alur *et al.*, 1993]. More elaborate variants of hybrid automata also allow for *probabilistic transitions,* which can be used to represent discrete random events influencing the system, leading to *probabilistic hybrid automaton* models.

Figure 1.4 gives an example of a hybrid temperature control loop. It is given as a hybrid automaton. The three nodes describe three modes of operation, from left to right: a heating mode increasing the temperature with a constant rate of four (active for low temperatures $T \leq -10$), a mode capable of both heating and cooling with a linear factor $c = 0.4$ (active for temperatures $-10 \leq T \leq 10$), and a cooling mode decreasing the temperature with rate four (active for high temperatures $T \geq 10$). The arrows model the possible transitions between the modes and are labeled with guard conditions enabling the transitions. The arrows going into the states from below denote possible initial states of the system.

1.1 Stability Verification of Hybrid Systems

One important challenge is the verification of such systems — the formal proof of properties of a given model of a hybrid system. For hybrid system models, verification

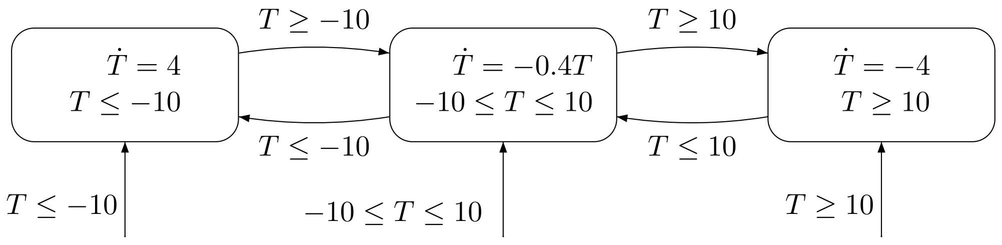


Figure 1.4: Hybrid Temperature Control Loop

approaches need to combine the knowledge of both worlds: Methods for discrete verification (for instance, model checking) must be augmented with methods for treating the continuous parts of the system, while methods from control engineering and calculus (e.g., solution concepts for differential equations, Lyapunov functions, frequency space analysis) need to be modified to allow for logic-based discrete switches.

In general, asymptotic stability properties for hybrid systems are undecidable, even for very simple classes of hybrid systems, for example linear systems with saturation [Blondel & Tsitsiklis, 1999, 2000; Blondel *et al.*, 2001b]. Direct methods of proving stability bear little promise, since one would need to prove a liveness property on uncountably many unboundedly long system trajectories. Therefore, the most widely taken approach is indirect: instead of verifying convergence of every single system trajectory, a state-based approach is used. For every state the system can enter (consisting both of a continuous and a discrete part), a function measuring an “energy value” of a state is computed, such that it always indicates progress.

These functions are termed *Lyapunov functions* [Lyapunov, 1907] (see Figure 1.5 for an illustration) and are a well known tool from control theory, originally used for stability proofs for purely continuous or purely discrete systems. Lyapunov functions can be pictured as a kind abstract “energy function” of the hybrid system. If global asymptotic stability is to be shown, this function must be chosen such that

- the only energy minimum is at the equilibrium point,
- the energy is strictly decreasing along all trajectories everywhere except at the equilibrium, and
- the energy grows unboundedly with growing distance from the equilibrium.

The existence of such a function implies global asymptotic stability of the system. Classic control theoretic results typically use only a single such function for the entire state space of a system. Therefore, the entire behavior of a system is abstracted away by a single function of this type, whose existence serves as a proof of global asymptotic

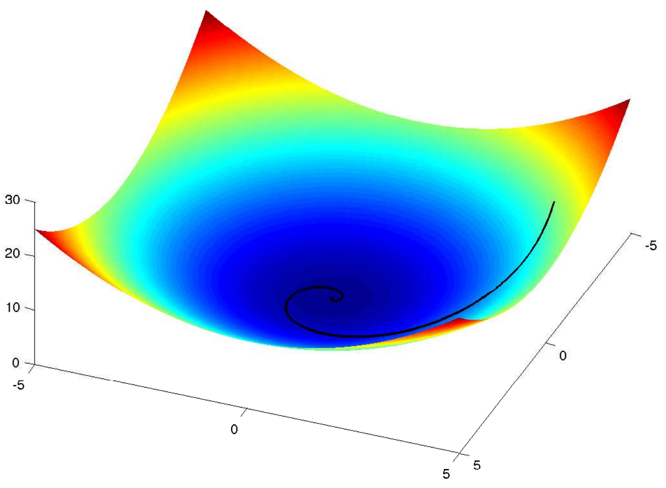


Figure 1.5: Quadratic Lyapunov Function and Convergent Trajectory

stability. Lyapunov functions can be seen as a generalization of similar concepts from the computer science community for proving liveness conditions, for example *ranking functions*, *variant functions* or *termination functions*.

For some classes of systems, it is possible to automatically identify such Lyapunov functions, thus conducting a proof of global asymptotic stability. For instance, for a single linear differential equation, this requires the solution of a system of linear equations, the *Lyapunov equation*. In this case the Lyapunov function is quadratic. For more complex systems, this process becomes increasingly difficult. A parametrized Lyapunov function template can be devised, with a number of free parameters. Finding adequate valuations for these parameters such that a Lyapunov function is obtained can then be viewed as a constraint system which must be solved. However, since automatic methods are not widely in use, the identification of a suitable function is still often left to the intuition of the engineer.

With the increased interest in hybrid systems in the 1990s, there were efforts to transport these results, which were originally conceived for purely continuous systems, to the hybrid domain [Branicky, 1998; Johansson & Rantzer, 1998; Pettersson, 1999; Hafstein, 2004; Lazar & Jokić, 2010]. The approach of employing a single Lyapunov function is not always suitable for the hybrid case. Especially if the dynamics in different modes of operation are radically different, one might not succeed in identifying a single closed-form Lyapunov function. This lead to the idea of employing multiple Lyapunov functions of a chosen parametric form for different modes of operation or different parts of the continuous state space [Branicky, 1998]. For instance, for a temperature controller, the control strategies for normal and emergency operation could be different enough such that there might be no closed-form Lyapunov function which can be derived from a template.

As it turned out, this multiple Lyapunov function approach results in a constraint system that can still be solved relatively efficiently. One method for the computation of such function results in constraint systems which can be viewed as special class of non-linear, convex optimization problems, namely *semidefinite programming* (SDP) problems [Johansson & Rantzer, 1998; Pettersson, 1999; Parrilo, 2003; Prajna & Papachristodoulou, 2003]. To compute a Lyapunov function in this manner, a parametrized function template is devised by hand (typically piecewise quadratic or piecewise polynomial), and the free parameters in the function are automatically filled with valid values by numerical software. In the hybrid case, more than one *local Lyapunov function* can be used (usually one per discrete mode, although sometimes even more are necessary), as long as certain additional conditions amongst the functions are satisfied. The problem can then be reduced to a non-linear but convex optimization problem that can be solved efficiently for small-to-medium sized problems.

The limited use of the semidefinite programming based approach in verification software is at least partly caused by the following facts:

- There is a significant increase in complexity with respect to the size of the hybrid model. The problem size grows linearly with the number of modes of operation and the number of possible mode transitions, leading to high-degree polynomial blowup in computation time.
- The numerical software can only provide approximate solutions, which in any case need to be checked by verifying that all constraints are actually satisfied (via eigenvalue computation) before they can be assumed to be correct. This is possible in an automated manner, but it requires extra effort.
- Numerical instability is a problem, since large-scale hybrid systems with complex discrete behavior have a tendency to produce poorly conditioned optimization problems, such that existing solutions are not necessarily found.

Nevertheless, these methods work very well for hybrid automata with simple discrete, but possibly complex continuous behavior.

Other Lyapunov methods are, for instance, based on linear optimization [Hafstein, 2004; Lazar & Jokić, 2010]. In this case, increased numerical stability is traded for a decrease in flexibility for the Lyapunov functions, which are usually restricted to piecewise linear functions. Therefore, these methods often require a relatively fine-grained partitioning of the state space. This somewhat limits the applicability to higher-dimensional state spaces, as the number of partitions can be expected to grow exponentially with the number of continuous variables. On the other hand, convex optimization-based approaches like semidefinite programming can often avoid this blowup because only coarser partitionings (or no partitionings at all) are needed, exploiting the larger search space for the functions.

Since it is difficult to prove asymptotic stability of a control loop, it is also difficult to *design* hybrid controllers maintaining this property. This is caused by the fact that local arguments cannot be easily combined into global arguments for asymptotic stability. Adding new modes of operation to an existing hybrid controller can easily break the stability property, and one needs to examine the interaction between the old controller and the new mode of operation to come to a conclusion about the stability of the resulting system. Since the SDP-based methods can only be used to prove asymptotic stability of a system as a whole, they are not particularly useful during such a design process. Ideally, one would want to have conditions on the original hybrid system and on the newly added mode of operation which are sufficient for stability of the composed system and able to actually prove stability for a large class of systems.

1.2 Contribution of this Thesis

This thesis focuses on stability proofs for hybrid systems via semidefinite programming based methods, striving to

- make the Lyapunov function computation methods more tractable in practice, and
- allowing the structured design of stable hybrid systems.

This is achieved by exploitation of the discrete structures of the system, resulting in the possibility of

- decomposition of large, possibly intractable proofs into several smaller, tractable sub-proofs,
- composition and transformation rules on hybrid systems such that the resulting systems are provably stable, and
- hierarchical, component based design of stable hybrid controllers.

While semidefinite programming is used as a method of choice for Lyapunov function computation in the scope of this thesis, the results are however general in the sense that they can be applied to *any* alternative method yielding Lyapunov functions for hybrid systems.

The decomposition takes place on two levels. First, a hybrid automaton can be decomposed into its strongly connected components, which can be treated completely separately. These separate sub-proofs per component still yield an overall proof of global asymptotic stability on the entire system. Figure 1.6 shows a hybrid automaton that is decomposed into its strongly connected components, as indicated by the dashed red lines.

The second level of decomposition then takes place within the strongly connected components. Refer to Figure 1.7 as an example of a hybrid automaton which is decomposed into a number of cyclic sub-components, with the "slicing" being indicated by the dashed lines. The discrete states which are bisected by these lines act as the connection points between the different sub-components. In this case, the sub-components are *cycles* of the automaton. Cycles are a convenient choice for the sub-components as then no fixed point computations are needed for the decomposition. Using the decomposition theorems in this thesis, these cycles each result in separate constraint systems, which can

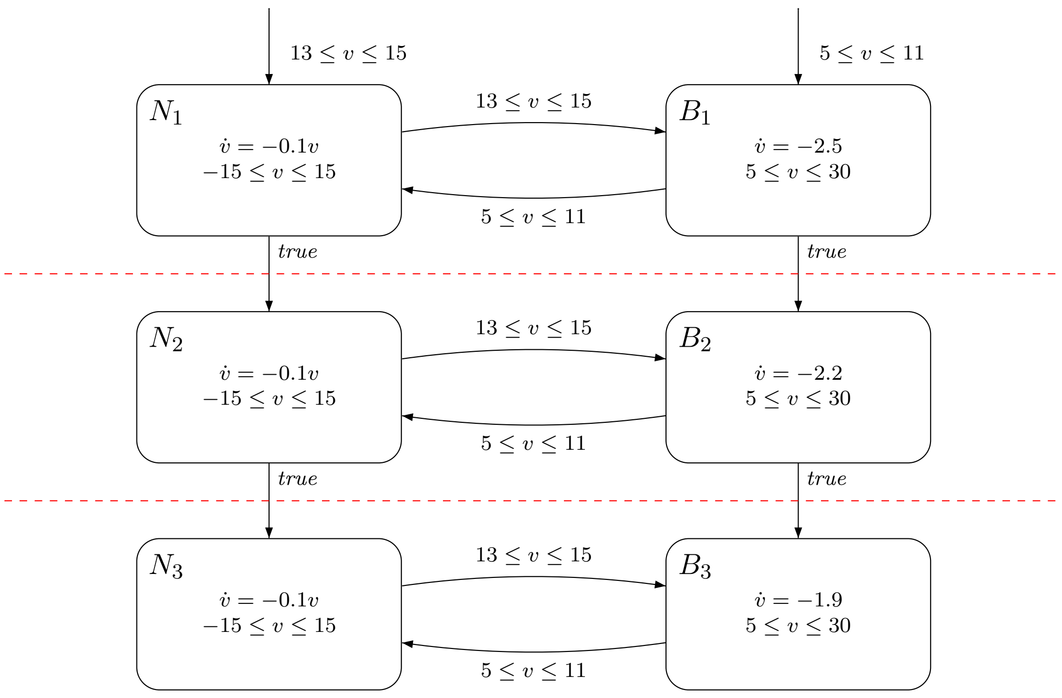


Figure 1.6: Decomposition of a Hybrid Automaton into Strongly Connected Components

be solved one by one. However, in this case, some information from previous per-cycle computation needs to be taken into account for its neighbor cycles, in order to arrive at an overall stability proof. This information takes the shape of Lyapunov function sets for the intersection nodes which are used to guarantee "compatibility" of the computations for the different cycles.

These two levels of decomposition are also applied to *probabilistic hybrid automata*, which is possible in a relatively straightforward manner. The property shown in this case is essentially a form of "stability with probability one" that allows for non-convergent trajectories as long as their probability mass is zero. For this class of systems, a third level of decomposition is possible, based on abstracting the probabilistic hybrid automaton into a *Markov decision process*.

We then also utilize the core decompositional arguments to formulate rules for the *composition* and *modification* of globally asymptotically stable hybrid automata, such that the stability property is maintained. Taking the compositionality paradigm one step further, we also outline how this type of composition can be employed for library-based design of hybrid controllers stabilizing a given plant. This approach allows for component-based *hierarchical* construction of controllers. For each component, *information hiding* is in place, such that only the stability-relevant information is visible on a component's interface. This information takes the form of Lyapunov functions.

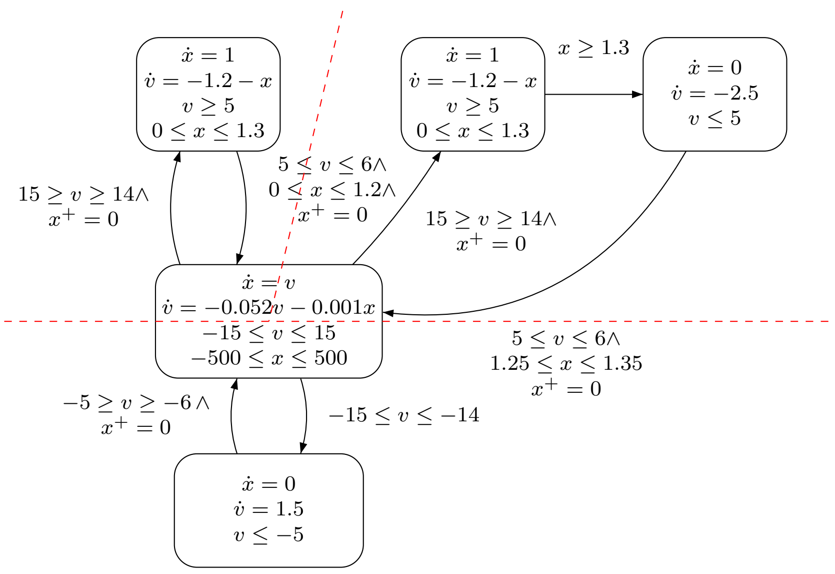


Figure 1.7: Cyclic Decomposition of a Hybrid Automaton

While results on stability proof decomposition of the continuous-time part of the hybrid system (e.g., through input-to-state stability [Heemels *et al.*, 2007] or small gain theorems [Liberzon & Nešić, 2006]) already existed at the time of writing, discrete decomposition of stability proofs was not systematically researched. This thesis aims to fill this gap and establishes a method for "slicing" hybrid automata with complex discrete structures into easier to handle sub-automata, as well as a composition rule set as an aid in the design process. In the end, this kind of slicing eases both the analysis and the synthesis of stable hybrid systems, as stability of a hybrid automaton is notoriously hard to see even for an experienced engineer. This is especially true when the discrete structures are complex. Eventually, the aim is to provide the engineer with methods that can be used during the design process to identify new behavior that can be added to an automaton without breaking stability. The hope is that the results given in this thesis pave the way for a structured design process that does not require the designer to understand the intricacies of hybrid automata. Instead, it would be sufficient to periodically ask a verification tool (that can provide quick answers since it only works on a sub-automaton) and follow a set of simple rules.

The main contributions of this thesis are the following:

- We provide a number of theorems for the decomposition of stability proofs into a number of smaller sub-proofs. These theorems are based on Lyapunov function computation, but they are not restricted to a particular computation method for these functions. The decomposition is graph-based and works on the hybrid automaton's discrete structure.

- We give algorithms which can be used to exploit these theorems in a fully automatic verification tool. These algorithms work by interpreting a hybrid automaton as a graph and by successively conducting local proofs on parts of the automaton. Parts of the automaton for which such a proof has taken place are then removed, resulting in a step-by-step reduction procedure.

- We transfer the decomposition results to probabilistic hybrid systems, hybrid systems with probabilistic transitions. Again, graph-based decomposition can be conducted on this class of systems in a similar manner.
- We present a set of composition and transformation rules on hybrid automata that preserve stability. These rules are based on the decomposition theorems and can be used for synthesis of stable hybrid systems, both non-probabilistic and probabilistic.
- We outline a framework for hierarchical, component-based design of globally asymptotically stable hybrid systems based on Lyapunov functions. These results are again based on the decomposition results and allow hierarchical design of hybrid controllers. Information hiding takes place within each controller component, so that only the information directly required for a stability proof is visible on the component's interface.

The questions to be answered in this thesis include:

- How can a stability proof for a hybrid system be split into sub-proofs that can be conducted one by one, still yielding a proof for the whole system?
- Are there cases where this decomposition is complete or can even be stronger than standard Lyapunov analysis?
- Given two hybrid systems which are known to be stable, how can we make sure that a new system obtained by switching between the two is also stable, re-using as much information as possible?
- If we introduce probabilistic behavior into a hybrid system, can we guarantee stability with a certain probability without having to re-do the complete proof?

The author's key publications related to the results presented in this thesis are:

- [Oehlerking & Theel, 2009a], presenting the basic decomposition methods given in Chapter 4,
- [Oehlerking & Theel, 2009b], extending these decomposition methods to the domain of probabilistic hybrid systems and stochastic stability properties, as discussed in Chapter 5,
- [Damm *et al.*, 2010], discussing component-based structured design of stable and safe hybrid systems, with the stability part being discussed in Chapter 6.

The hope is that these results narrow the gap between theory and practice, eventually making it possible to supply reliable software tools for automated stability/liveness verification of hybrid systems, as a complement for the existing verification tools, which can already tackle safety proofs reasonably well.

1.3 Thesis Outline

After this introduction, in Chapter 2 gives a detailed discussion of the current state of the art in stability verification for hybrid systems, including contributions from the computer science, mathematics and control engineering domains. A special focus is on Lyapunov related methods, as these are central to this thesis.

Chapter 3, contains an in-depth recapitulation of existing results in stability verification via Lyapunov function computation. This chapter lays the formal groundwork for the remainder of the thesis, including definitions of the hybrid system model, global asymptotic stability and several types of Lyapunov functions. Most importantly, the linear matrix inequality (LMI) based method for Lyapunov function computation is described in detail.

Chapter 4 then contains the first major contribution of the thesis: a decompositional framework for stability analysis based. The decomposition theorems presented in this chapter have been geared toward linear matrix inequality based Lyapunov function computations, but can in general be used with any method capable of computing such functions. For LMI based methods, algorithms are then described which allow the automatic execution of such a decompositional analysis.

Chapter 5 contains the second major contribution of the thesis: the transfer of the results from Chapter 4 to the stochastic domain. There, the hybrid automaton definition is first extended to deal with probabilistic transitions. The standard Lyapunov results, as well as the decomposition algorithm are then transferred to this domain, allowing automated stability analysis also for stochastic models.

In Chapter 6, these analysis techniques are then turned into synthesis rules for globally asymptotically stable hybrid systems, both in the non-stochastic and the stochastic domain. The result is a rule set that can be used to incrementally construct a globally asymptotically stable hybrid system, without the necessity to verify the system as a whole. Furthermore, a component-based design approach supporting information hiding is proposed, which can also deal with delayed switching between components. This design approach allows the hierarchical construction of a stabilizing controller for a given plant, based on the decomposition results from Chapter 4.

Finally, Chapter 7 gives a conclusion and a brief overview on open problems and possible extensions of the decomposition.

2 Stability Verification of Hybrid Systems – The State of the Art

This chapter gives an overview on results in the area of hybrid system stability verification and the state of the art at the time of writing. First, hybrid system models in general are discussed in Section 2.1. Then, an overview of the different stability definitions in the literature is given in Section 2.2. Section 2.3 describes previous research concerned with Lyapunov function theorems for the non-hybrid and the hybrid case, respectively. This also includes a discussion of methods for the automatic computation of such functions. Section 2.4 gives an overview of non-Lyapunov based methods for stability proofs available in the literature. Finally, decomposition of stability proofs is discussed in Section 2.5.

Note that this chapter is not intended to present an in-depth description of the methods, but instead provides a brief overview of related work. Key definitions, theorems and methodologies that are relevant within the scope of this thesis are described in detail in Chapter 3.

2.1 Hybrid System Models

Hybrid systems are systems consisting of a combination of continuous and discrete behavior. Hybrid system models must therefore include formalisms to adequately represent either type of behavior. This section gives an overview of various classes of hybrid models that are of importance for design and analysis today. Generally, these models require one formalism (such as ordinary/partial/algebraic/stochastic differential equations or inclusions) for the continuous part and another formalism (automata, logical structures, switch planes, Markov chains) for the discrete part. To obtain models for hybrid systems, these formalisms need to be combined in some manner.

One way of viewing a hybrid system, particularly from the control-theoretic viewpoint, is as a so-called *switched system* [Liberzon, 2003]. This type of hybrid control loop model was first proposed by Witsenhausen [1966]. Here, the system is viewed as a feedback control loop with a continuous-time plant and a continuous-time controller (see Fig. 2.1). However, the controller is not one single component, but instead consists of a collection of sub-controllers, only one of which is connected to the plant at any time. The plant and the sub-controllers are usually modelled as systems of ordinary differential equations or inclusions. The switching between the controllers can be modelled non-deterministically via external input [Daafouz *et al.*, 2002], randomly via a probability distribution [Dimarogonas & Kyriakopoulos, 2004; Chatterjee & Liberzon, 2006], or deterministically by selecting a fixed switching strategy depending on the system state [Pettersson &

Lennartson, 1996; Prajna & Papachristodoulou, 2003]. Deterministic strategies are often chosen to minimize a given cost function, for instance by achieving the fastest possible convergence. Switching strategies are often represented by *switch planes* in the state space which trigger a change in discrete state when they are crossed [Pettersson & Lennartson, 1996; Prajna & Papachristodoulou, 2003]. While this model is close to the control-theoretic view of a feedback control system (and many hybrid systems actually represent feedback control of some sort), expressiveness of the discrete model is limited. For instance, lazy “can”-switches (as opposed to urgent “must”-switches) are not expressible. Furthermore, discrete updates to continuous variables (e.g., variable resets) are often not included in these types of models.

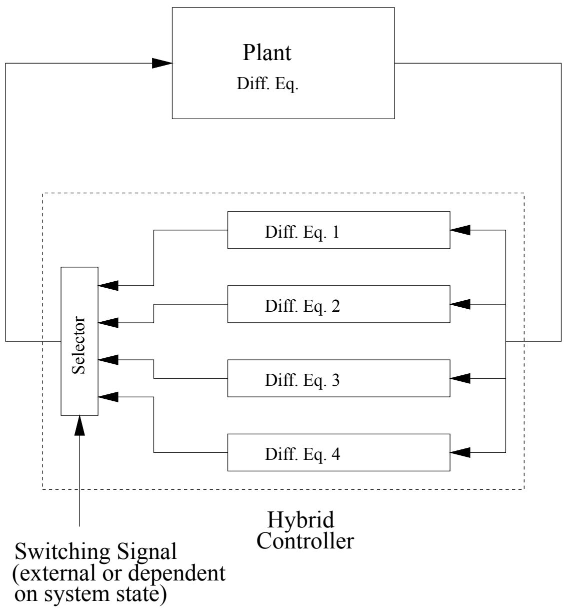


Figure 2.1: Illustration of a Switched System Model

Another hybrid system model sees the system as a differential equation or inclusion with a discontinuous right-hand side. In this type of system, the discrete state is not explicitly modelled, but is derived from the continuous state. This means, that, in general, each discrete state is associated with a sub-set of the state space. A differential equation or inclusion is considered active once the continuous state is in its area of responsibility. Therefore, it is not directly possible to model phenomena like hysteresis, where the same continuous state is associated with different discrete states, depending on the history of the trajectory. Also here, discrete updates of continuous variables are usually not modelled. An example of a differential equation with discontinuous right-hand side is for example

$$\dot{x} = \begin{cases} -1, \text{ if } x > 0 \\ 0, \text{ if } x = 0 \\ 1, \text{ if } x < 0 \end{cases}$$

See [Cortés, 2008] for a detailed discussion of such systems, with many adaptions of results from standard non-hybrid systems with continuous right-hand sides. A drawback of this modeling approach is the relative inexpressiveness with respect to discrete structures of the system.

If such a differential equation or inclusion with discontinuous right-hand side also allows for discrete updates of continuous variables, then this is related to the notion of *impulsive systems* [Hespanha *et al.*, 2008]. Impulsive systems are differential equations allowing for sudden (impulsive) changes of variable values. Here, the discrete state is modelled as just another (continuous) system variable which happens to stay constant when no switching occurs. Switches are implicitly defined as the points in time when these variables do change. Impulsive differential equation systems can, for instance, be of the form

$$\begin{aligned} \dot{x} &= f(t, x(t)) \text{ for } t \neq \{t_1, \ldots, c_n\} \\ x^+ &= g(t, x(t)) \text{ for } t \in \{t_1, \ldots, c_n\} \end{aligned}$$

By allowing both discontinuities and impulsive behavior in differential equations or inclusions, a larger class of hybrid systems can be covered.

On the other end of the spectrum are automaton based models. Since automata are naturally capable of expressing the discrete component of a system's behavior, they have to be enriched to deal with the continuous dynamics. This leads to the concept of *hybrid automata* [Alur *et al.*, 1993]. Here, the nodes (corresponding to discrete states) and edges (corresponding to discrete state transitions) of the automaton are labeled with differential equations or inequalities and guard conditions on the continuous variables, respectively. The differential equation/inequality associated with a node describes how the continuous state evolves when its current discrete state corresponds to the node. The guard conditions are usually first-order predicates over the continuous variables, describing when a transition can or must be taken (depending on the semantics of the particular model). Hybrid automata can also be augmented with input/output notions to facilitate parallel compositions [Lynch *et al.*, 2003]. The class of hybrid automaton models has been used extensively by the computer science community and forms the basis for many safety and reachability verification approaches (for example, [Henzinger & Rusu, 1998; Frehse, 2008]). Hybrid automaton models can also be extended to deal with systems containing discrete random experiments, leading to so-called probabilistic hybrid automata [Sproston, 2000].

One particular hybrid automaton model will be used in this thesis and is defined in Section 3.2. An extended model with probabilistic switching will also be employed in Chapter 5.

Other models for hybrid systems replace differential equations by partial differential equations [Bayen *et al.*, 2004], differential algebraic equations [Mitchell & Susuki, 2008], or stochastic differential equations [Cassandras & Lygeros, 2007] in place of the differential equations or inclusions. A related modeling paradigm proposes the use of programs instead of automata for the basic discrete structure. *Hybrid programs* [Platzer & Quesel, 2008] enrich program structures with means of expressing continuous evolution and non-determinism.

For use with discrete-time analysis tools, purely discrete-time equivalents of many of these models exist, called *discrete-time hybrid systems* [Ferrari-Trecate *et al.*, 2002; Bemporad *et al.*, 2002; Feng, 2002]. For instance, differential equations can be replaced by difference equations. These purely discrete models are often the result of some discretization of a continuous-time hybrid automaton. While these systems are strictly speaking not hybrid (despite the somewhat misleading name) but purely time-discrete, their structures are similar to the hybrid systems they approximate, with some (discrete) variables taking over the function of the continuous-time variables. However, these discrete variables can then still take values in the real numbers, leading to a potentially uncountable state space. These purely discrete systems can then, for instance, be analyzed with discrete-time tools, like discrete-time model checkers. Of course, great care must be taken to avoid fatal modeling errors because of inadequate sampling rates.

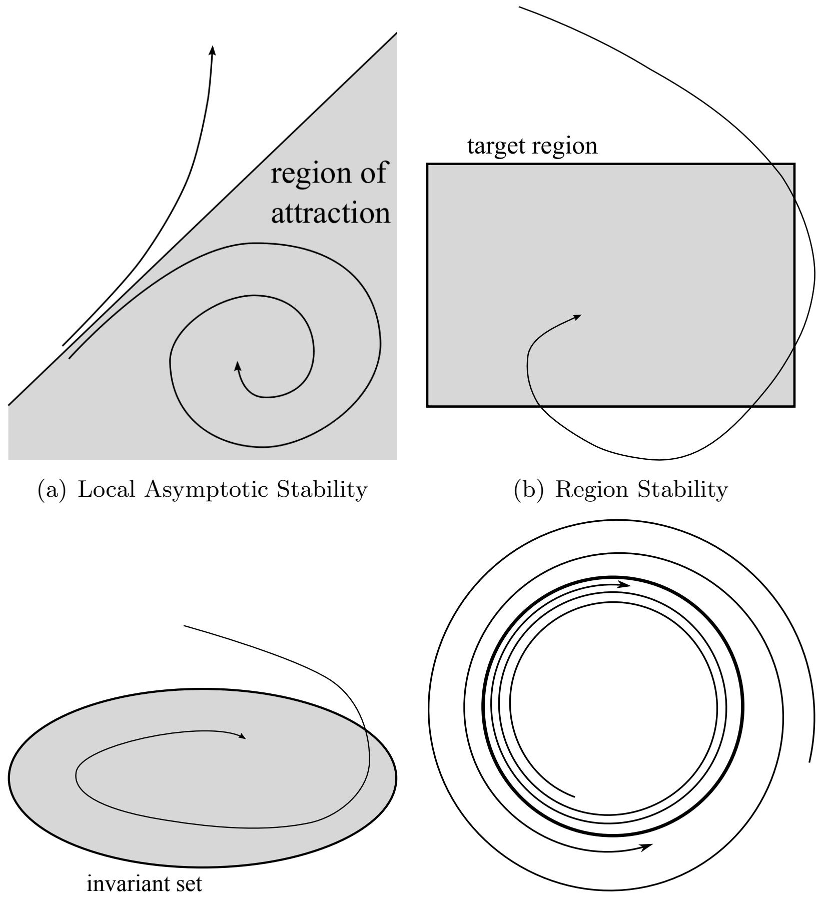
2.2 Stability Properties

This section summarizes different stability definitions for feedback control systems. While all of these definitions describe properties that imply some sort of robustness to disturbances, different system models and different application contexts require a variety of stability notions. Figure 2.2 illustrates some of the key stability definitions discussed below.

2.2.1 Lyapunov Stability Properties

Some very commonly used stability definitions for autonomous systems (systems without explicitly modeled external input) are subsumed under the term *Lyapunov stability* [Lyapunov, 1907]. The basic notions are *stability* and *attractivity*. These are usually defined with respect to an *equilibrium state* of the system, a state where the time derivatives of all continuous variables are zero. Such an equilibrium state is therefore stationary in the sense that it is never left by any system behavior allowed by the model. Without loss of generality, this equilibrium state can be assumed to be the origin of the state space. A differential equation or inclusion with a different equilibrium can always be transformed into one where the equilibrium is at the origin by a simple translation of the coordinate system.

Stability is a boundedness condition, implying that any bound on the distance to the equilibrium is respected for all times, if one chooses the initial state close enough. Attractivity requires the convergence of each trajectory to the origin. If a system is both stable and attractive, it is *asymptotically stable*. Stability and attractivity can be defined *globally*, for all initial states, or *locally*, only for some initial states. For linear differential equations, local (asymptotic) stability with respect to some open set



(a) Local Asymptotic Stability (b) Region Stability

(c) Invariant Set Stability (d) Limit Cycle Stability

Figure 2.2: Stability Notions

containing the equilibrium always implies global (asymptotic) stability. In this case, the proof is simple: show that the system matrix is Hurwitz. The system matrix is the matrix A in the vector notation $\dot{x} = Ax$, where x is the system state vector. The matrix A is Hurwitz if and only if the real parts of all eigenvalues are negative, which is equivalent to global asymptotic stability. Truly local stability properties are usually associated with a so-called *region of attraction* (see Figure 2.2(a)), defining a set of initial states from which convergence is guaranteed [Chesi, 2004; Ratschan & She, 2010; Tan & Packard, 2008]. In case of non-linear or hybrid system dynamics, there can also be multiple equilibria, each with its own region of attraction. If a non-linear system is linearized in the origin and the linearized system is globally asymptotically stable, then the original system is locally asymptotically stable with some basin of attraction under come continuity assumptions. A stronger property that can be shown with the help of some Lyapunov-based proof methods is *exponential stability*, implying at least exponentially fast convergence. This property is tied to quadratic Lyapunov functions and can be used to estimate the convergence rate of an asymptotically stable system [Pettersson, 1999]. These stability definitions can be stated both for continuous time (and therefore for hybrid systems, which operate in continuous time) and discrete time (and therefore discrete-time hybrid systems). The notion of *global asymptotic stability* will be formally defined in Section 3.3 and acts as the property to be shown by the decompositional methods in Chapter 4. As these stability definitions are part of the control systems canon, they are discussed in most basic textbooks on the subject [Khalil, 1996; Sontag, 1998; Liberzon, 2003].

For systems with input signals, similar stability properties can be defined. Generally, these properties require some condition on the input signal and imply a stability-like property on either the system's state or an output signal. Some common stability properties here are *input-to-state stability (ISS)* and *input-to-output stability (IOS)* [Khalil, 1996]. If a system is ISS (IOS), then boundedness of the input signal implies boundedness of the system state (output signal), and convergence of the input signal implies convergence of the system state (output signal). This class of properties can also be shown by Lyapunov-type analysis [Cai & Teel, 2005; Heemels *et al.*, 2007; Hespanha *et al.*, 2008].

2.2.2 Generalized Stability Properties

Another type of stability property relaxes the convergence requirement from a single target state to an entire target set of states. This is termed *stability with respect to an invariant set* [Khalil, 1996; Ye *et al.*, 1998; Cai *et al.*, 2008; Ratschan & She, 2010] or *region stability* [Podelski & Wagner, 2006]. In the former case, the target set needs to be invariant, that is, it cannot be left, once entered (see Figure 2.2(c)). Proofs for this type of stability property can again be conducted via Lyapunov-like arguments, together with LaSalle's invariance principle [Khalil, 1996]. Informally speaking, these arguments require a decreasing function measuring the distance to an internal point of the target set. The region stability approach is more flexible in the sense that it also allows the system state to leave the target again, as long as this happens only finitely long (see Figure 2.2(b)). There are no constraints on the allowed behaviors within such an invariant set region, once it has been reached by a trajectory. However, in practice, systems are usually designed with a specific behavior in mind, even if they do not converge to a single state. For these cases, stability can also be generalized to convergence toward a behavior of the system, and not a state. One example is the so-called *limit cycle stability* (see Figure 2.2(d), where a system with inherently periodic behavior is required to converge toward some *limit cycle* [Khalil, 1996].

This limit cycle represents state set, so that, when it is reached, all system trajectories will follow a prescribed cyclic behavior, for instance continuing to move in a loop for all future times. In this case, every trajectory of the system (or every trajectory starting within a sub-set of the state space, if it is interpreted as a local property) is required to converge to the limit cycle, eventually approximating the cyclic behavior arbitrarily close. In other words, convergence is not defined with respect to a state, but a periodic trajectory. This notion of stability, while different from Lyapunov stability, can still be partly shown with similar methods. For instance, so-called *Poincaré maps* [Khalil, 1996] can be used to construct a system whose asymptotic stability implies the stability of a limit cycle of the original system. This is done by problem-specific sampling of the trajectories. For example, all time instances where a trajectory crosses a hyperplane could comprise the sequence of sample points, and convergence of the sample point sequence on the hyperplane then implies stability with respect to some limit cycle. This stability notion has received considerable interest in the control community [Rubensson & Lennartsson, 2000; Hiskens, 2001; Simić, 2002; Girard, 2003; Gonçalves, 2005], but there has not yet been a strong push toward automated proofs, at the time of writing.

2.2.3 Stochastic Stability Properties

For stochastic systems, many variations of stability properties exist, usually consisting of a non-stochastic stability notion which is relaxed probabilistically. Since stability definitions can be endowed with probabilism in various manners, the result is a large collection of possible definitions, which are often not uniformly named in the literature.

Since global asymptotic stability is the conjunction of global stability (i.e., a boundedness condition) and global attractivity (i.e., convergence to an equilibrium state), probabilities can be introduced to both of these sub-properties individually. For example, for stability, three notions of stochastic stability based on convergence notions for random variables can be defined: *almost sure stability, stability in the mean, stability in probability* [Loparo & Feng, 1996]. Similar definitions are possible for the attractivity property.

For instance, *global stability in probability* means that, for any ϵ-ball around the equilibrium and any positive probability p, if a trajectory starts close enough to the equilibrium, it can be guaranteed that it stays within the ϵ-ball forever with probability p. On the other hand, the stronger notion of *almost sure global stability* is equivalent to requiring that this is the case with probability 1 [Kozin, 1972]. As one can see, there are a multitude of ways of introducing probability measures into stability definitions, making it especially important to precisely define stochastic stability properties.

For some results on stability in probability without the convergence property, see [Dimarogonas & Kyriakopoulos, 2004], or for results on almost sure asymptotic stability for systems under random switching via Lyapunov-like functions see [Chatterjee & Liberzon, 2007].

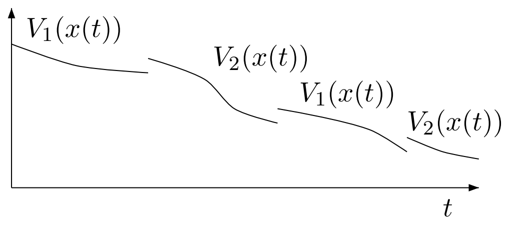
2.3 Lyapunov Functions

This section discusses Lyapunov function based results for stability proofs and their automation potential. First, we deal with results providing Lyapunov function theorems for stability proofs. Next, a survey of automatic computation methods based on these results is given. Finally, we discuss work dealing with the utilization of Lyapunov-like functions also for safety proofs. Again, this section is only intended to give a brief overview of the literature. In-depth discussion of results that are directly relevant to the thesis then follows in Chapter 3.

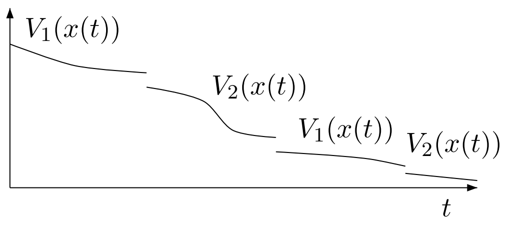
2.3.1 Lyapunov Theorems

The basic idea of employing an energy-like function to indirectly prove stability, also known as *Lyapunov's second method*, dates back to observations on mechanical systems [Lyapunov, 1907] and has since become part of the control systems canon. The basic Lyapunov function theorems for the non-hybrid case can be found in almost any basic textbook covering non-linear control theory (e.g., [Sontag, 1998, p. 218] and [Khalil, 1996]). In the literature, there are variants for asymptotic and non-asymptotic Lyapunov stability [Pettersson, 1999], differential inclusions, also with discontinuous right-hand sides [Cortés, 2008], local stability properties, stability with respect to invariant sets [Cai *et al.*, 2008], input-to-state stability [Heemels *et al.*, 2007], or stochastic stability [Loparo & Feng, 1996]. More recently, there also have been efforts to relax the Lyapunov function conditions by allowing the function values to actually increase over a timespan, as long as this is compensated in the long run [Ahmadi & Parrilo, 2008]. Completeness results guaranteeing the existence of Lyapunov functions (also known as *converse theorems*) have also been derived, covering special classes of systems. For instance, it is very well known that there exists a quadratic Lyapunov function for any asymptotically stable linear differential equation. In this case, the result is even constructive, allowing for the computation of the function via the so-called *Lyapunov equation* [Sontag, 1998, p. 213]. Such converse theorems for hybrid systems are briefly discussed later in this section.

For hybrid systems, most classical Lyapunov function results are not directly applicable, as they usually require continuity of the vector field as a prerequisite. However, many adaptions of Lyapunov function theorems to the hybrid case exist [Pettersson & Lennartson, 1996; Johansson & Rantzer, 1998; Branicky, 1998; Chatterjee & Liberzon, 2006; Cai *et al.*, 2008; Cortés, 2008]. A simple approach is to use a common Lyapunov function for all discrete modes [Liberzon, 2003; Vu & Liberzon, 2005]. If no discrete updates of continuous states are possible, then this actually proves asymptotic stability *for all possible switching behaviors* between the discrete modes of the system. In other words, for such a system the usage of a common Lyapunov function means that all knowledge about the discrete structures of the system is discarded. For this reason, this approach is restrictive, as many stable example systems do not allow for a common Lyapunov function for all modes [Johansson & Rantzer, 1998]. An alternative approach, pioneered by Branicky [1994], allows the use of one Lyapunov function per mode. Here, in addition to the standard Lyapunov function conditions, the *entry point sequence* for each mode needs to have non-increasing energy values, that is, the sequence of continues states attained by the trajectory upon entering a new discrete mode (see Figure 2.3(a)). To facilitate the computation of Lyapunov functions, often a stronger condition is used: non-increasingness of the Lyapunov function over time. In this case, it is required that the new Lyapunov function value after the switching (and possible application of a discrete update of the continuous variables) is not larger than the value before the switching [Pettersson & Lennartson, 1996; Pettersson, 1999] (see Figure 2.3(b)). Clearly, this con-



(a) LF with non-increasing entry points



(b) strictly non-increasing LF

Figure 2.3: Piecewise Lyapunov Functions over Time

dition implies the one given by Branicky. It does, however, have the advantage that the Lyapunov conditions can be formulated completely independently of individual trajectories, only using state based (and not time based) constraints. This approach can also be extended to allow for more than one Lyapunov function per mode (or, equivalently, a discontinuous Lyapunov function), each covering only one part of a partitioned state space [Pettersson & Lennartson, 1996; Pettersson, 1999; Oehlerking *et al.*, 2007].

For systems with inputs, similar Lyapunov theorems exist, for instance based on so-called *input-to-state stability (ISS) Lyapunov functions* [Heemels *et al.*, 2007]. Alternatively, constrained inputs can simply be collapsed into a differential inclusion, resulting in an autonomous system with non-deterministic dynamics which already factor in any outside disturbances.

In most cases, equivalents of the results for the discrete-time domain exist, transferred in a straightforward manner from the discrete-time domain [Kalman & Bertram, 1960; Feng, 2002; Rubensson & Lennartsson, 2000]. Essentially, there is a negativity constraint on the difference of Lyapunov function values between a state and its predecessor state, instead of on the time derivative of the Lyapunov function, yielding a concept that is similar to a ranking or termination function.

Converse theorems for Lyapunov functions have always been of great interest within the control community, and some results have also been obtained for hybrid systems. Only very recently, a general converse theorem guaranteeing the existence of Lyapunov functions for globally asymptotically stable hybrid systems has been presented [Cai *et al.*, 2007, 2008]. For some important earlier results concerning local existence of Lyapunov functions for locally asymptotically stable hybrid systems, see, for instance, [Ye *et al.*, 1998]. However, these results are not directly amenable for automation, that is, they do not provide an actual algorithm for obtaining the functions. For specific types of systems and Lyapunov functions, specialized constructive theorems exist, for example for piecewise affine systems and piecewise quadratic Lyapunov functions [Pettersson & Lennartson, 1997] or for twice continuously differential equations and piecewise linear Lyapunov functions [Hafstein, 2004]. However, since stability of even simple hybrid system classes is undecidable [Blondel *et al.*, 2001a,b], one can in practice only hope for constructive converse theorems for limited classes of systems and limited types of Lyapunov functions. In the hybrid domain, these converse theorems are still only of limited practical relevance, also because the computation of an existing Lyapunov function might be too costly. Also, if an algorithm fails to identify a Lyapunov function for a hybrid system, this is does usually not disprove stability.

2.3.2 Automatic Computation of Lyapunov Functions

While Lyapunov function theory has been well established for manual proofs of stability properties, automatic computation of such functions for hybrid systems was effectively not researched until the mid-1990s. The most common methods for such computations employ linear and non-linear optimization techniques. Here, parametrized function templates are used, and constraints on the free parameters formulated, such that the resulting function has the desired Lyapunov property. Often the functions are only defined piecewise, that is, different function templates are used for different parts of a partitioned hybrid state space. Numerical algorithms are then used to obtain a solution fulfilling all constraints, yielding the desired Lyapunov function. If the Lyapunov function template allows for only (piecewise) linear functions, then this problem can be mapped onto a (possibly very large) linear optimization problem [Hafstein, 2004; Lazar & Jokić, 2010], based on *Farkas' lemma* or the *mean value theorem*. However, quadratic Lyapunov function templates are more commonly used, because such functions allow for symmetries that are useful in countering state space explosion with respect to the number of continuous states in the system. Quadratic Lyapunov function templates, in general, require less partitioning and therefore scale better to systems with high-dimensional state spaces. Furthermore, quadratic functions allow the formulation of the Lyapunov function constraints as so-called *linear matrix inequalities* [Boyd *et al.*, 1994], which result in non-linear, but still convex, optimization problems [Pettersson & Lennartson, 1996; Johansson & Rantzer, 1998; Pettersson, 1999]. These problems can be solved by descent-like interior point optimization methods in polynomial time [Boyd & Vandenberghe, 2004; Nesterov & Nemirovskii, 1994]. This LMI formulation is based on the Positivstellensatz and can guarantee positiveness/negativeness of functions without having to examine any individual value within their range. The software tools that are available for the solution of this kind of problem are discussed in depth in Section 3.6.1.

Pettersson & Lennartson [1996] employed the so-called $\mathcal{S}$-procedure [Yakubovich, 1977] to formulate LMI problems that only represent *local* conditions on a part of the state space. With this technique, a family of local Lyapunov functions for parts of the state space can be computed, such that they prove stability for the entire system. Here, the LMI problems can become rather large because additional free parameters need to be added to the optimization problem for each local Lyapunov functions, as well as auxiliary variables required for the $\mathcal{S}$-procedure. Therefore, one has to take care to keep the partitionings as coarse as possible in order to maintain tractability of the problem in practice. Johansson & Rantzer [1998] proposed a restricted, but computationally more benign variant of a multiple Lyapunov function approach. There, the individual local Lyapunov functions are not completely independent, but related by a fixed scaling matrix that is derived from the partitioning. This effectively reduces the solution space, but results in numerically simpler optimization problems.

However, direct application of LMI methods is limited to (piecewise) quadratic Lyapunov functions and linear/affine dynamics. To lift these restrictions, the *sums-of-squares decomposition* [Parrilo, 2003; Parrilo & Jadbabaie, 2007; Parrilo & Lall, 2003; Peet *et al.*, 2006; Prajna & Papachristodoulou, 2003; Papachristodoulou & Prajna, 2005; Papachristodoulou, 2004] can be applied to permit the analysis of non-affine systems and the search for non-quadratic Lyapunov functions. The basic idea is an (in general conservative but sound) substitution of non-quadratic terms in the constraints by quadratic constraints, resulting again in LMIs. This approach can also be combined with the $\mathcal{S}$-procedure or the various concepts for multiple Lyapunov functions.

One might also think about using techniques from model checking/satisfiability checking for the computation of Lyapunov functions. However, Lyapunov function conditions inherently contain alternating quantifiers as they are of the type:

$$\exists p \in \mathbb{R}^n : \forall x \in \mathbb{R}^n : f(p, x) \geq 0,$$

where f is a function to $\mathbb{R}$ which is parameterized in p. This type of formula is hard to verify, and for instance beyond the capabilities of satisfiability checkers such as HySAT [Fränzle *et al.*, 2007]. One can view the LMI formulation as a convenient way of eliminating the "$\forall$"-quantifier, turning the problem into one that is purely existential by exploiting the very specific nature of the problem. As of now, only limited attempts to compute Lyapunov-like functions via symbolic methods have been undertaken (see, for example, [Gulwani & Tiwari, 2008], which replaces inequality by equality constraints).

2.3.3 Identification of Stabilizing Controllers

In the domain of control theory, it is important to differentiate between the concept of "stability" and the concept of "stabilization" or "stabilizability." While stability is a property of an entire control loop, comprising of both plant and controller, stabilizability is a property of just the plant. If there exists a controller that can be used in a feedback loop to stabilize an inherently unstable plant, the plant is stabilizable. The process of identifying such a controller is known as "stabilization" of the plant. The stabilization problem can also be solve using Lyapunov function methods, by identifying *control Lyapunov functions* for the plant that imply constraints on admissible stabilizing controllers. However, this problem is much more complex than proving stability, since the search space (comprising both the free parameters in the Lyapunov function and the controller) is in generally non-convex. Instead of LMI-based methods, stabilization problems can be solved with *bilinear matrix inequalities*, which are unfortunately much more difficult to solve than LMIs [Zhai *et al.*, 2003]. To remedy this problem, relaxations of these non-convex problems have been devised, see for instance [Prajna & Jadbabaie, 2004; Lazar & Jokić, 2010]. However, conservatism remains an issue in case of such relaxations. A related problem is the synthesis of a controller which steers a system trajectory to a designated part of the state space [Habets *et al.*, 2006].

2.3.4 Lyapunov Functions as Barrier Certificates

Lyapunov-like methods have also been employed for proofs of safety properties. Since the energy value of standard Lyapunov functions is generally non-increasing over time, the set of all reachable states from one initial state can be restricted to those states that have a lower energy value. When quadratic Lyapunov functions are used, the result is an ellipsoidal over-approximation of the reachable set. Additionally, sums-of-squares techniques can also be employed to deal with complex dynamics. A contour line of a Lyapunov-like function used in this manner is also referred to as a *barrier certificate* [Prajna & Jadbabaie, 2004; Prajna & Rantzer, 2005], and the function itself is also called *criticality function* [Damm *et al.*, 2007], as it indicates the distance to a critical state in an abstract manner. Barrier certificates are special cases of differential invariants in the sense of Platzer & Clarke [2008].

The computational techniques for this type of analysis are essentially the same as for stability proofs, with some relaxations to the Lyapunov properties. For instance, it is not necessary to require that the function has a global minimum at the origin, as actual convergence is not part of the proof obligation. However, using this technique, safety and stability can also be shown together, re-using the Lyapunov functions from the stability proof for the safety proof [Damm *et al.*, 2010].

2.4 Other Methods for Stability Proofs

Previous work on stability verification for hybrid systems that does not directly rely on Lyapunov functions has been relatively sparse. Also, proofs of unbounded-horizon convergence require some way of tracking "progress," so that such methods usually contain Lyapunov-type arguments in some quantity. One approach by Podelski & Wagner [2006] uses sampling of trajectories, such that stability with respect to a target region in the state space is implied by the finiteness of the sampling point sequence. However, this finiteness property is again a termination problem, which is solved through the computation of ranking functions. Ranking functions can again be seen as a special case of Lyapunov functions.

Another method not directly using Lyapunov functions employs joint spectral radii, that is, the largest eigenvalues of a matrix obtained by multiplying the system matrices of a discrete-time switched system in any order [Parrilo & Jadbabaie, 2007]. Here, the goal is a proof of contractiveness by calculating an upper bound to the spectral radius.

Another different view uses the gains associated with individual modes of a switched linear system and combines them to yield a stability proof [Langerak *et al.*, 2003]. This approach exhibits strong parallels to the theory of piecewise quadratic Lyapunov functions [Langerak & Poldermans, 2005] and is not directly applicable to richer classes of systems.

If there is an explicit upper bound on the time for convergence (which usually means that there also must be a bounded state space), then convergence is a reachability problem. The question whether a target set S is reachable in t_0 time units is then equivalent to the question whether the predicate $\neg S \wedge t > t_0$ is unsatisfiable. This simpler class of problems can therefore be solved by tools like PHAVer [Frehse, 2008], although complexity and the quality of reachable set over-approximations remain an issue, especially with long time horizons and rich dynamics.

2.5 Decompositional Verification and Compositional Design

In general, decompositional techniques for stability proofs of hybrid systems have, at the time of writing, not received wide attention. However, some interesting results are summarized in this section. One possible approach is decomposition along the continuous variables of a system, that is, grouping a system into connected sub-systems, each with their own differential equations. These sub-systems are then connected via input-output relations. Such block diagrams are a standard notation in control theory, and widely exploited to show properties for connected linear (non-hybrid) systems. There have been some attempts of transferring these types of results to the hybrid domain. Viewing a system as a number of interconnected components with input-output relations can exploited for *small gain theorems*. These theorems allow the composition of two sub-systems, each with input and output signal, in a feedback loop, such that the resulting system is stable. This is achieved by imposing constraints on the *gains* of the two systems, that is, the level of "amplification" of the input signal they cause on the output. In recent years, several small-gain theorems for hybrid systems have been presented. [Laila & Nešić, 2003; Liberzon & Nešić, 2006; Nešić & Liberzon, 2005]. However, such gains remain hard to compute for hybrid systems, so that these ideas are not yet widely used in verification. Input-to-state stability as described above can also be exploited for this kind of decomposition [Heemels & Weiland, 2008; Dashkovskiy *et al.*, 2008].

A second possibility is decomposition in the discrete domain, for instance the graph structure of a hybrid automaton. However, decomposition of stability proofs along this axis has, to the best of the author's knowledge, not been systematically conducted. This kind of decomposition is especially crucial, since real-life digital controllers can easily have hundreds of discrete states, and this system size can simply not be tackled without any notion of decomposition. Therefore, one goal of this thesis is systematic decomposition of hybrid systems (given as hybrid automata) on its discrete states.

Compositional design methodologies for hybrid systems are already supported through tools like Matlab Simulink and Stateflow, but such tools do not currently support a structured design process leading to a verifiable system. Another formalism for expressing sequential as well as parallel formalisms are HyCharts [Grosu & Stauner, 2002], which provide a graphical notation for hierarchical hybrid system design. Compositional design methodologies for safe hybrid systems are for instance based on o-minimal hybrid automata [Casagrande *et al.*, 2008], hybrid I/O automata [Frehse *et al.*, 2004] (in both cases for parallel composition) or differential invariants [Platzer & Clarke, 2008] (for transition composition). However, these methodologies do not cover stability properties.

As the decomposition of stability proofs employing the graph structure of hybrid automata was not sufficiently researched until now, this thesis contributed to the state of the art by contributing to closing this gap. In addition to this, the results will also be used to derive rules for structured design, such that both the analysis and the design of stable hybrid automata is eased considerably. We will continue in Chapter 3 by describing in detail how stability proofs can be automated and then introduce the general decomposition procedure in Chapter 4.

3 Lyapunov Function Computation for Hybrid Systems

This chapter gives the basic definitions, theorems and verification methods for global asymptotic stability proofs. First, some basic notation is introduced in Section 3.1. Then, the hybrid system notion used in the remainder of this thesis, namely hybrid automata with differential inclusions, is defined formally in Section 3.2. This includes a discussion of solution concepts of differential inclusions and hybrid phenomena like Zeno behavior. Next, global asymptotic stability is formally defined in Section 3.3, together with Lyapunov functions. The application of Lyapunov functions for safety proofs via barrier certificates is then discussed in Section 3.4. Next, LMI-based Lyapunov function computation methods are reviewed in detail in Section 3.5, followed by a discussion of the underlying numerical methods in Section 3.6. Finally, a detailed cruise control example of an automatic Lyapunov function computation is given in Section 3.7. The chapter is then concluded by a summary in Section 3.8.

This chapter is intended to provide the groundwork for the decompositional analysis in the following chapter. The verification methods in this chapter are “monolithic” in the sense that discrete structures are not explicitly exploited. Instead, the problem of identifying a Lyapunov function for a system results in a single and possibly large LMI problem. The decompositional treatment then follows in Chapter 4, building on the “monolithic” results.

3.1 Notation

This section gives some basic definitions and notations that are used extensively in this thesis.

Definition 3.1 (Sets)**.** Let $\mathbb{R}$ be the set of real numbers and let $\mathbb{R}^+$ be the set of all non-negative real numbers. For a set X, $\mathcal{P}(X)$ defines the *power set* of X, that is, the set of all sub-sets of X.

Definition 3.2 (Matrix and Vector Notation)**.** For a matrix $M \in \mathbb{R}^{m \times n}$, the matrix $M^T \in \mathbb{R}^{n \times m}$ denotes its *transpose*. For a column vector $v \in \mathbb{R}^{n \times 1}$, $v^T \in \mathbb{R}^{1 \times n}$ denotes the row vector resulting from the matrix transposition. Individual entries of vectors and matrices are identified by subscripts in parentheses, such that $v_{(i)}$ is the i-th element of vector v and $M_{(i,j)}$ is the entry in the i-th row and j-th column of M. For a row vector $v^T \in \mathbb{R}^{1 \times n}$ and a column vector $w \in \mathbb{R}^{n \times 1}$, $\langle v^T \middle| w \rangle$ denotes the *scalar product*. For a square matrix $M \in \mathbb{R}^{n \times n}$, $tr(M) = \sum_{i=1}^{n} m_{(i,i)}$ denotes its *trace*.

Definition 3.3 (Norms)**.** For a $p > 1$, define the *p-norm* of x as

$$||x||_p = \left(\sum_i |x_{(i)}|^p\right)^{1/p}.$$

The *Euclidean norm* of x is the special case $p = 2$, that is,

$$||x|| = \sqrt{\sum_i |x_{(i)}|^2}.$$

Define the *infinity norm* of x as

$$||x||_\infty = \max_i |x_{(i)}|.$$

Definition 3.4 (Closed Ball)**.** Define $B(x, \epsilon) \subseteq \mathbb{R}^n, x \in \mathbb{R}^n, \epsilon > 0$ as the *closed ball* around x with radius ϵ, $B(x, \epsilon) = \{y \in \mathbb{R}^n | \, ||y - x|| \leq \epsilon\}$.

Definition 3.5 (Sequence)**.** Finite or infinite sequences will be denoted with parentheses (e.g., (m_i)) when referring to the sequence itself, and the i-th element of the sequence m will be written m_i, without parentheses.

Definition 3.6 (Convex and Conic Sets and Hulls)**.** A set $X \in \mathbb{R}^n$ is called *convex*, if

$$x, y \in X \implies \forall \lambda \in [0, 1] : \lambda x + (1 - \lambda)y \in X,$$

implying that when two points lie in X then so does the line segment connecting them. A set $X \in \mathbb{R}^n$ is called *conic*, if

$$x, y \in X \implies \forall \lambda_1, \lambda_2 \geq 0 \text{ with } \lambda_1 + \lambda_2 > 0 : \lambda_1 x + \lambda_2 y \in X,$$

implying that the cone spanned by any two points in X (without its "tip") is still in X.

Next, we define the convex and conic hulls. The *convex hull* of a set of points $x_i \in \mathbb{R}^n$ is defined as

$$\mathit{convex}(\{x_1, \ldots, x_m\}) = \left\{ x \in \mathbb{R}^n \middle| \exists \lambda_i \geq 0 : \sum_i \lambda_i = 1 \wedge x = \sum_i \lambda_i x_i \right\}.$$

The *conic hull* is defined as

$$\mathit{cone}(\{x_1, \ldots, x_m\}) = \left\{ x \in \mathbb{R}^n \middle| \exists \lambda_i \geq 0 : (\exists i : \lambda_i > 0) \wedge x = \sum_i \lambda_i x_i \right\}.$$

Note that this definition of the conic set and hull is non-standard in the sense that we exclude the "tip of the cone," the value $x = 0$. This is achieved by explicitly requiring that one λ_i is strictly positive. In a similar manner we also define the convex and the conic hull of sets of functions. Let $f_i : \mathbb{R}^n \to \mathbb{R}^n, 1 \leq i \leq m$, be a family of real-valued functions. The *convex hull* of the functions f_i is the set

$$convex(\{f_1, \dots, f_m\}) := \left\{ \sum_i \lambda_i f_i \middle| (\forall i : \lambda_i \geq 0) \land \left(\sum_i \lambda_i = 1 \right) \right\}.$$

The *conic hull* is

$$cone(\{f_1, \dots, f_m\}) := \left\{ \sum_i \lambda_i f_i \middle| (\forall i : \lambda_i \geq 0) \land (\exists i : \lambda_i > 0) \right\}.$$

Remark 3.1 (Predicate Notation of Sets)**.** We will frequently use first-order predicates to describe sets over the reals. For instance, the predicate $x \leq 6 \land y \leq 0$ could be used to represent the set $\{x, y \in \mathbb{R} \mid x \leq 6 \land y \leq 0\}$. For ease of reading, the terser predicate notation is sometimes used in place of a verbose set notation, if there is no danger of confusion.

3.2 Modeling Hybrid Systems

This section defines the hybrid system formalism as it is used in this thesis. Hybrid systems are modelled as hybrid automata, which are finite automata whose nodes and edges are labeled reflecting the system's continuous behavior. This model has the advantage that hybrid systems can be seen as automata, and ultimately as a special type of graph. The decomposition techniques described in Chapter 4 make extensive use of this graph structure.

However, before these systems can be defined, a discussion of solution concepts for differential equations and inclusions is necessary. Differential inclusions will be attached to the nodes of the automaton to describe the continuous evolution of the hybrid system.

3.2.1 Solutions to Differential Inclusions

A *solution* is a continuous-time behavior that is permitted by a differential equation or inclusion. A solution to a *differential equation* $\dot{x} = f(x), x \in \mathbb{R}^n$, is a differentiable function $x : \mathbb{R}^+ \to \mathbb{R}^n$, such that $\dot{x}(t) := \frac{dx}{dt}(t) = f(x(t))$ for all t. In contrast, *differential inclusions* are of the form $\dot{x} \in F(x)$, where $F : \mathbb{R}^n \to \mathcal{P}(\mathbb{R}^n)$ maps each x onto a whole set of values for $\dot{x}$. According to the *Carathéodory solution concept* [Cortés, 2008], at almost all given times t (that is, all but some singular time instants), the time derivative $\dot{x}(t)$ must lie within the set $F(x)$. Differential inclusions are therefore a more general model for continuous dynamics than differential equations, since they can also model dynamics with uncertainties resulting from only inexact knowledge of the system to be modelled. Furthermore, standard differential equations are also covered by this model, if $F(x)$ is a singleton set for each x. All the results in this thesis apply to systems with differential inclusions, unless otherwise noted. The formal definition is given next.

Definition 3.7 (Differential Inclusion)**.** A *differential inclusion* is a set inclusion in the form $\dot{x} \in F(x), x \in \mathbb{R}^n$, where $F : \mathbb{R} \to \mathcal{P}(\mathbb{R}^n)$ is a set-valued function.

In order to formally define solutions of differential inclusions, we first need to define absolutely continuous functions.

Definition 3.8 (Absolute Continuity)**.** A function $f : I \to \mathbb{R}^n, I \subseteq \mathbb{R}$ is called *absolutely continuous*, if for all compact sub-intervals J of I and for all $\epsilon > 0$ there exists a $\delta > 0$, such that for all finite sequences of pairwise disjoint intervals $(x_i, y_i) \subseteq J$ the following holds:

$$\sum_i |x_i - y_i| < \delta \implies \sum_i ||f(x_i) - f(y_i)|| < \epsilon.$$

For differential inclusions, we define solutions according to the *Carathéodory solution concept* as follows.

Definition 3.9 (Carathéodory Solution of a Differential Inclusion)**.** A *(Carathéodory) solution* to a differential inclusion $\dot{x} \in F(x)$ is an absolutely continuous function $x : \mathbb{R}^+ \to \mathbb{R}^n$ such that $\dot{x}(t) \in F(x)$ for almost all t.

Remark 3.2. Absolute continuity implies that $x(t)$ is differentiable almost everywhere. This means abrupt changes of direction for $x(t)$ are possible, as long this behavior is restricted to a null set of times (i.e., a set of measure zero). For instance, a solution of the differential inclusion $\dot{x} \in [-1, 1]$ can abruptly change its time derivative between $\dot{x} = 1$ and $\dot{x} = -1$ on a null set. Moreover, absolute continuity of $x(t)$ implies that for the Lebesgue integral

$$x(t) = x(0) + \int_0^t \dot{x}(\tau) d\tau$$

for all t, so that differential inclusions can be cast into an integral form similar to differential equations.

Under certain assumptions, the existence and the uniqueness of a solution to a differential equation or inclusion can be guaranteed. The following two theorems serve as examples of such conditions. A variety of results requiring slightly different preconditions exists in the literature [Cortés, 2008]. One result uses the notion of *upper semicontinuity*, which is an extension of the standard continuity definition on real-valued functions to set-valued functions like $F(x)$.

Definition 3.10 (Upper Semicontinuity of Set-valued Functions)**.** A function $F : \mathbb{R}^n \to \mathcal{P}(\mathbb{R}^n)$ is called *upper semicontinuous*, if for all $x \in \mathbb{R}^n$ and $\epsilon > 0$ there exists a $\delta > 0$ such that $F(y) \subseteq \{y_1 + y_2 \in \mathbb{R}^n | \ y_1 \in F(x) \wedge y_2 \in B(0, \epsilon)\}$ for all $y \in B(x, \delta)$.

Under some additional conditions, upper semicontinuity of the right hand side of a differential inclusion implies the existence of at least one solution from every initial value $x_0 \in \mathbb{R}^n$.

Theorem 3.1 (Existence of Solutions [Cortés, 2008])**.** Let $\dot{x} \in F(x)$ be a differential inclusion. If $F(x)$ is non-empty, convex, closed, upper semicontinuous, and

$$\exists c > 0 : \forall x \in \mathbb{R}^n : \forall y \in F(x) : ||y|| \leq c(1 + ||x||),$$

then there is a solution to $\dot{x} \in F(x)$ for every initial value $x(0) \in \mathbb{R}^n$.

In order to show the uniqueness of a solution to differential inclusion, an additional Lipschitz-like property is required, similar to Lipschitz continuity in the *Picard-Lindelöf theorem* for differential equations. One such property is *one-sided Lipschitz continuity*, which is defined as follows.

Definition 3.11 (One-sided Lipschitz Continuity)**.** A function $F : \mathbb{R}^n \to \mathcal{P}(\mathbb{R}^n)$ is called *one-sided Lipschitz-continuous*, if there exists a scalar $c > 0$, such that for almost every $x_1, x_2 \in \mathbb{R}^n$:

$$\forall y_1 \in F(x_1), y_2 \in F(x_2) : (x_1 - x_2)^T (y_1 - y_2) \leq c(||x_1 - x_2||).$$

This condition then gives us a uniqueness result on solutions of differential inclusions.

Theorem 3.2 (Uniqueness of Solutions [Cortés, 2008])**.** Let $\dot{x} \in F(x)$ be a differential inclusion, where $F(x)$ is non-empty, convex, closed, upper semicontinuous, and one-sided Lipschitz-continuous, then $\dot{x} \in F(x)$ has at most one solution for every initial value $x_0 \in \mathbb{R}^n$.

For differential inclusions for which $F(x)$ is just a singleton set for all x, we will usually use the standard differential equation notation $\dot{x} = f(x)$ instead of $\dot{x} \in F(x)$. The stability analysis methods presented in this thesis can deal with initial conditions that permit no solutions, as well as with initial conditions that permit several (or even infinitely many) solutions. The results simply apply to all solutions that exist. For a discussion on criteria for the existence and uniqueness of solutions for differential inclusions, we refer to [Cortés, 2008; Frankowska & Aubin, 2009].

Next, we give two examples for differential inclusions, namely a control loop modelling a simple cruise control system and a model of a battery.

Example 3.1 (Proportional-Integral Speed Controller)**.** Consider a cruise control application consisting of a plant P modeling a vehicle and a hybrid controller C. The controller C influences the acceleration of the vehicle, so that the acceleration signal a is an output of C and an input to P. Additionally, assume that the reaction of the vehicle to the acceleration signal a is not precise. Instead, the vehicle can only guarantee to keep its actual acceleration within 2.5% of the value requested by the controller. This is modelled by another input to P, a relative disturbance value s. Therefore, the plant dynamics are given by

$$\begin{aligned} \dot{v} &= s \cdot a \\ s &\in [0.975, 1.025] \end{aligned}$$

The controller has the task of bringing the value of v toward 0. Here v can be seen as modeling the velocity differential between the current velocity and a desired set point. The controller consists of the differential equation $\dot{x} = v$ and the equality $a = -0.001x - 0.052v$, where x is the variable keeping track of the integral of v.

After eliminating the variables a and s, the result is a differential inclusion for the closed-loop system:

$$\begin{aligned} \dot{v} &\in convex(-0.000975x - 0.0507v, -0.001025x - 0.0533v) \\ \dot{x} &= v \end{aligned}$$

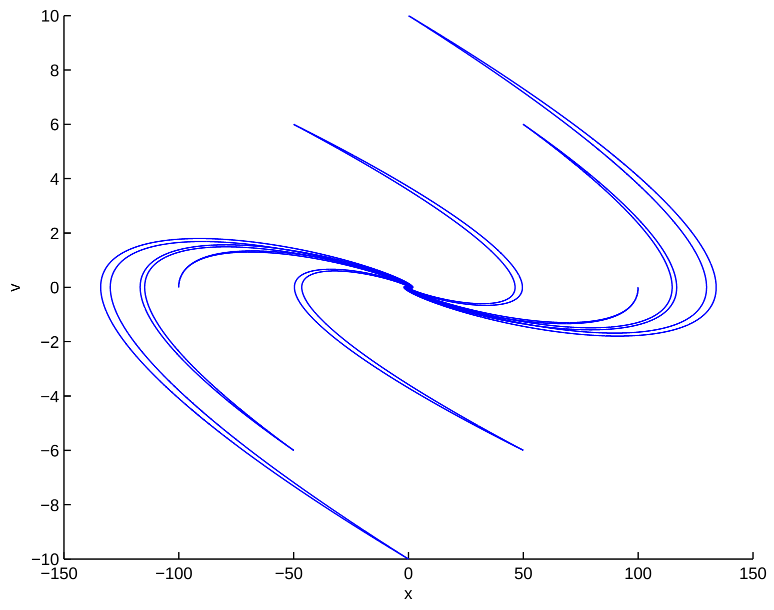


Figure 3.1: Simulation Runs for Example 3.1

Regardless of how the bounded disturbance s (now modelled by a differential inclusion) behaves within its bounds, the system will converge to $v = x = 0$. In fact, the system is globally asymptotically stable. Figure 3.1 gives example plots for this system. For each initial state, two possible trajectories are plotted, representing the extremal dynamics in the convex hull on the right hand side of the differential inclusion for $\dot{v}$.

Example 3.2 (Kinetic Battery Model [Manwell & McGowan, 1993])**.** One example for a differential inclusion is the model of a battery. There are two continuous variables in the system: the *bound charge* b and the *available charge* a. Energy can be drained from the available charge a (as long as $a > 0$). As the available charge decreases, energy from the bound charge b is made available. However, this does not happen instantaneously. Instead, the transition rate between bound and available charge depends on the difference between the two charges. The larger the difference, that is, the more the available charge has been depleted, the higher the rate. If the energy that is drained from the battery is not constant, but assumed to vary between two bounds E_L and E_H, then the battery dynamics can be described by the differential inclusion

$$\begin{bmatrix} \dot{a} \\ \dot{b} \end{bmatrix} \in F\left(\begin{bmatrix} a \\ b \end{bmatrix}\right) := \left\{ \begin{bmatrix} -E + k(b-a) \\ -k(b-a) \end{bmatrix} \middle| E_L \leq E \leq E_H \right\}$$

for some fixed $E_L, E_H, k > 0$. The energy E is subtracted from the available charge a, and the transfer rate between the bound and the available charge is given by $k(b-a)$. If E were zero, then the system would eventually converge to $b = a$, with the total charge $a + b$ remaining constant at all times.

Since this differential inclusion fulfills all conditions of Theorem 3.1, this implies that there exists at least one solution for each initial state However, $F(x)$ is not one-sided Lipschitz-continuous, and there are infinitely many possible trajectories for every initial state.

3.2.2 Hybrid Automata

With a definition of differential inclusions and a solution concept for the continuous dynamics, it is possible to define hybrid automata and their solutions. Here, the solution concept is based on *two-dimensional time.* Essentially, this means we keep track both the progress of continuous time (one dimension) and the number of switches (the other dimension) separately. The advantage lies in some extra expressiveness: we can deal with multiple sequential switches at the same time instant more easily. The syntax is based on a finite automaton, for which we attach general differential inclusions (as defined above) to the discrete states (the modes of operation of the hybrid automaton). Furthermore, invariant sets are also attached to the modes, describing conditions that must hold while a mode is active. The edges can be labeled with guard conditions (also modelled as sets) and updates on the continuous states which are applied once an edge is taken.

Definition 3.12 (Hybrid Automaton)**.** A *hybrid automaton H* is a tuple

$$(\mathcal{M}, \mathcal{S}, \mathcal{V}, \mathcal{T}, \mathit{Flow}, \mathit{Inv}, \mathit{Init}),$$

where

- $\mathcal{M}$ is a finite set of *modes*,
- $\mathcal{S} = \mathbb{R}^{|\mathcal{V}|}$ is the *continuous state space*,
- $\mathcal{V}$ is the set of *continuous variables*, with each variable corresponding to a coefficient of the vectors $x \in \mathcal{S}$, in some fixed order,
- $\mathcal{T}$ is a set of *transitions* given as tuples (m_1, m_2, G, U), where
  - $m_1 \in \mathcal{M}$ is the *source mode*,

  - $m_2 \in \mathcal{M}$ is the *target mode*,
  - $G \subseteq \mathcal{S}$ is the closed *guard set*,
  - $U : \mathcal{S} \to \mathcal{S}$ is the *update function* for the continuous state variables,

- *Flow* : $\mathcal{M} \to [\mathcal{S} \to \mathcal{P}(\mathcal{S})]$ is the *flow function*, mapping each mode onto a set-valued function which in turn maps each $x \in \mathcal{S}$ onto a closed sub-set of $\mathcal{S}$, which is taken as the right-hand side of a differential inclusion $\dot{x} \in \mathit{Flow}(m)(x)$,

- *Inv* : $\mathcal{M} \to \mathcal{P}(\mathcal{S})$ is the *invariant function*, mapping each mode onto a closed sub-set of the continuous state space, and

- *Init* $\subseteq \mathcal{M} \times \mathcal{S}$ is the closed set of combinations of *initial discrete and continuous states*.

All initial, invariant, flow, and guard sets are always assumed to be closed in this model. Concerning the kind of stability analysis presented in this thesis, this is not a serious restriction. For the Lyapunov function computation methods, open sets would have to be over-approximated by enclosing closed sets. In other words, any proof of stability we obtain using these Lyapunov-based methods for a system with some open sets also implies the same property for the system obtained by replacing the open sets by their closures. Therefore, to simplify matters, it is helpful to assume closedness already in beforehand.

Remark 3.3. Hybrid automata will often be depicted graphically in the scope of this thesis. Such a representation consists of the underlying graph, labeled with invariants, flows, guards, and updates. While invariants $\mathit{Inv}(m)$ and guards G are defined as sets of states, we will usually label the nodes and edges with predicates describing these sets. For instance, a node label $a \geq 5$ means that the invariant is the set $\{x \in \mathcal{S} | x_{(j)} \geq 5\}$, where the variable $a \in \mathcal{V}$ corresponds to the j-th coefficient in x. If more than one predicate appears in one node or on one edge, then the invariant/guard is represented by the *conjunction* of the predicates. If no update function is given on an edge, we assume the identity function $U(x) = x$. Flows are given by differential equations or inclusions. Initial states will be represented by arrows pointing to discrete modes, but not originating from a node. These "initial edges" are labeled with a predicate specifying the conditions on the continuous variables under which a system trajectory is allowed to start in this particular mode. The *Init* set is then represented by the disjunction of the predicates implied by all these initial edges. For example, two initial edges, one pointing to mode m_1 with predicate $x \leq 5$, and one pointing to mode m_2 with predicate $x \geq 5$ represent the initial set $\{(m, s) | (m = m_1 \wedge x \leq 5) \vee (m = m_2 \wedge x \geq 5)\}$.

Definition 3.13 (Solutions of Hybrid Automata)**.** Let H be a hybrid automaton. Let (t_i) be a possibly infinite sequence $t_i \in \mathbb{R}^+ \cup \{\infty\}$, representing the *switching times* of the system such that

- $t_0 = 0$,

- $t_i \geq t_{i-1}$ for all $i > 0$, and
- only the final element of a finite sequence (t_i) may have the value ∞.

A *solution* of the hybrid automaton H is a possibly infinite sequence of tuples (m_i, x_i), with

- $m_i \in \mathcal{M}$ and
- $x_i : [t_i, t_{i+1}] \to \mathcal{S}$ (or $x_i : [t_i, t_{i+1}) \to \mathcal{S}$ in case $t_{i+1} = \infty$) a function which is absolutely continuous on $[t_i, t_{i+1}]$ (or $[t_i, t_{i+1})$, respectively),
- (t_i) is a switching time sequence of infinite length if (m_i, x_i) is of infinite length, and one element longer than (m_i, x_i) otherwise,

such that:

1. $(m_0, x_0(0)) \in \mathit{Init}$,
2. for all $t_i \leq t < t_{i+1} : x_i(t) \in \mathit{Inv}(m_i)$,
3. for all $t_i \leq t < t_{i+1} : \dot{x}_i(t) \in \mathit{Flow}(m)(x_i(t))$ for almost all t,
4. for all $t_{i+1}, i > 0$ which are not the final element of (t_i), there exists a mode transition $(m, \tilde{m}, G, U)$, such that
   a) $lim_{t \uparrow t_{i+1}} x_i(t) \in G$,
   b) $x_{i+1}(t_i) = U(x_i(t_i))$,
   c) $m_i = m$,
   d) $m_{i+1} = \tilde{m}$,
5. if (t_i) is infinite, then $\lim_{i \to \infty} t_i = \infty$.

A solution is called *infinite* if (t_i) is infinite and diverges to infinity or if (t_i) is finite and its final element is ∞. It is called *finite* otherwise. The sequence (m_i) is called the *mode sequence* corresponding to the solution.

Definition 3.14 (Discrete, Continuous, and Hybrid State)**.** For a given solution and a time $t \geq 0$, the value of the m_i with the uniquely determined index i such that $t_i \leq t < t_{i+1}$ is called the *discrete state* at time t and is denoted by $m(t)$. Similarly, the function value $x_i(t)$ such that $t_i \leq t < t_{i+1}$ is the *continuous state* at time t and is denoted by $x(t)$. Furthermore, the state before an update is defined as $x(t_{i+1}^-) := \lim_{t \uparrow t_{i+1}} x(t)$. Any function $x : I \to \mathcal{S}$ with $I = [0, r], r \in \mathbb{R}^+$ or $I = \mathbb{R}^+$ such that $x(t)$ is the continuous state at time t of a solution of H is called a *trajectory* of the system. The tuple $(m(t), x(t))$, as a function of t, defines the *hybrid trajectory* of H, giving the *hybrid state* at time t. A hybrid trajectory is called *extendable*, if it is the prefix of another hybrid trajectory and *non-extendable* otherwise.

Essentially, a trajectory is the evolution of the hybrid automaton's continuous state. Note that trajectories are often discontinuous, as discontinuities can be introduced through the update functions. Asymptotic stability talks about trajectories, disregarding the discrete states the system goes through.

Note that, whenever a discrete update of the continuous state occurs at time t via the update function, the continuous state at time t is defined as the new state after the update. If there are multiple mode switches at the same time instant t, $m(t)$ is the mode the system reaches after all such switches, and $x(t)$ is the continuous state after all corresponding discrete updates have been applied.

Example 3.3 (Hybrid Kinetic Battery Model)**.** Figure 3.2 shows a hybrid automaton model of a battery that is recharged at a constant rate, if the total charge falls below a threshold.

The *discharge* mode is very similar to the differential inclusion from Example 3.2. In addition to the differential inclusion, the invariant set $\{(a, b) \mid a + b \geq 5 \wedge a \geq 0\}$ has been added to the mode by simply labeling it with corresponding predicates. According to its invariant, this mode must be left once the total charge $a + b$ drops below five or a becomes negative.

The *charge* mode governs the battery recharge with some constant rate R. The energy gain is added to the available charge (and potentially converted into bound charge, if it is note needed immediately). The invariants for the charge mode only require the two charges to remain positive. Therefore, the system can remain in charge mode for a potentially unbounded time (maximum capacity of the battery is not modelled here).

There are two transitions between these two modes. As soon as the total charge drops below six, the battery *can* go into charge mode. Note that it *must* recharge as soon as the total charge drops below five, as the invariant set of the left hand side mode is left at that point, forcing the mode transition. Therefore, between a total charge of five and six, there is a non-deterministic choice of whether to take the transition or not.

The second transition leads back from the charge mode and may be taken, once the total charge is at least eight again.

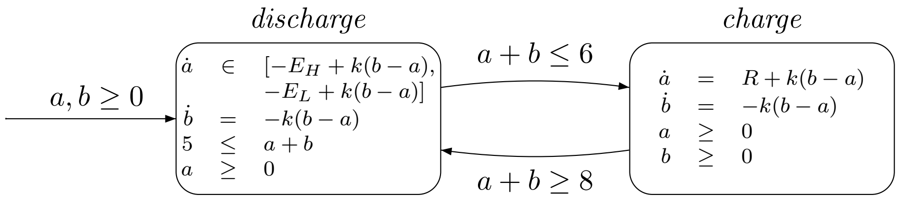


Figure 3.2: Hybrid Kinetic Battery Model

A phase plot of the system, with parameters $E_L = 0.8, E_H = 1.2, R = 1$ and $k = 1$, and initial values $a = 3$ and $b = 0$ is pictured in Figure 3.3. Note how the effect of the uncertainty introduced by the differential inclusion is visible on the top part of the "loop," which jitters considerably. For the purpose of this simulation, a new value for the non-deterministic part in the differential inclusion for $\dot{a}$ in the discharge mode was

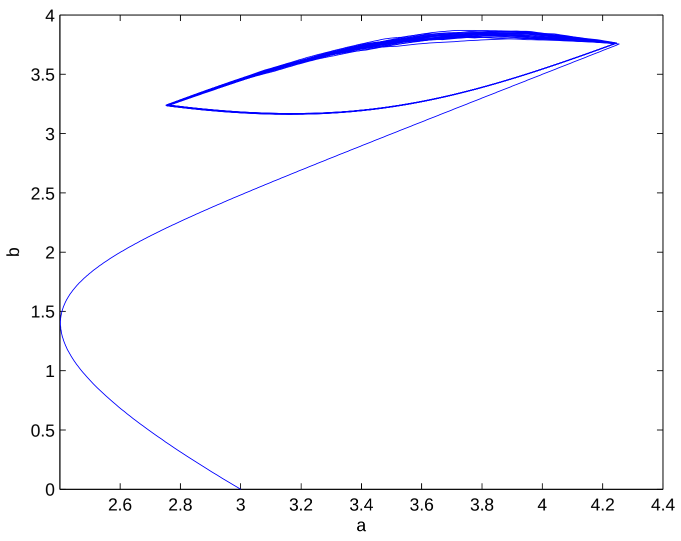


Figure 3.3: Phase Plot of Battery Model

selected randomly every 0.1 time units, and each transition was taken as soon as possible.

Even though it may seem as if this system stabilizes with respect to some periodic behavior (or a region in the state space), this is not the case, since the location of the "loop" depends on the initial state of the system. The term "stability" is generally used when there is one specific point/area of convergence, regardless of the initial state.

Infinitely many switches at the same time instant t do not result in valid solutions of the hybrid automaton as defined in Definition 3.13, since the sequence t_i would have to be infinite and convergent to t, which would contradict Condition 5. Sequences (m_i, x_i) violating Condition 5 of Definition 3.13 (and therefore not being a valid solution of the hybrid automaton), but fulfilling all other conditions are termed *Zeno*. Condition 5 can therefore be interpreted as a restriction of the class of behaviors that are considered a solution to non-Zeno behaviors.

We permit the occurrence of Zeno behavior (i.e., the existence of such non-solution sequences) in the hybrid automata that are considered for stability analysis. However, since solutions of hybrid automata are per definition free of Zeno behavior (due to Condition 5), and since stability analysis will be applied to solutions only, it will simply be ignored in the analysis.

Note that infinitely fast switching in a hybrid system is often not desired in the first place. If it is intentional (e.g., sliding mode control), the Zeno behavior can, for instance, be eliminated by "covering" the part the state space that exhibits Zeno behavior (e.g., a sliding surface) by a new mode with an associated differential inclusion that explicitly models the sliding behavior. This approach is related to the *Filippov solution concept* [Zolezzi, 2002], which eliminates sliding mode Zeno behavior by applying this idea. The following example is a modified version of the battery model from Example 3.3, exhibiting Zeno behavior.

Example 3.4 (Battery Model with Zeno Behavior)**.** The battery model in Figure 3.4 differs from Example 3.3 only in one guard. Recharging may now be stopped when the total charge is at least six instead of eight. This introduces possible Zeno behavior. If the total charge reaches six, it is possible to switch to the charge mode. However, as the guard of the reverse transition is also true at this moment, the system may switch back again immediately. This cycle can be repeated infinitely often, without any time passing, leading to an infinite, but convergent sequence of switching times. We do, however, not consider this case a solution of the hybrid automaton, so that this possible behavior would be ignored in any subsequent analysis.

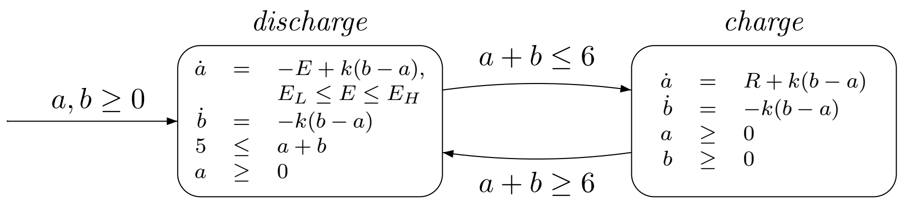


Figure 3.4: Kinetic Battery Model with Zeno Behavior

The next example shows a useful non-Zeno variant of the battery model that permits finite trajectories.

Example 3.5 (Battery Model with Non-extendable Finite Trajectories)**.** Figure 3.5 shows a model of a battery that permits non-extendable finite hybrid trajectories whenever the available charge a reaches 0. When this happens, the invariant of the left hand side state is invalidated, but the guard of the outgoing transition is not satisfied. Such a model makes sense, if it is seen as a part of a larger automaton obtained through decomposition. For instance, this automaton could actually be a sub-automaton of a larger model that contains special handling for the case $a = 0$, by an outgoing transition leaving the sub-automaton from the left hand side node. This is the reason why such models are useful, and explicitly considered in the scope of this thesis.

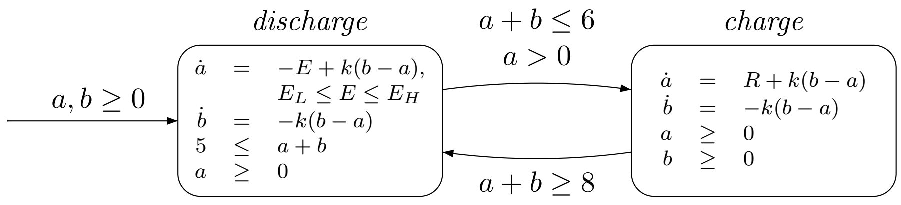


Figure 3.5: Kinetic Battery Model with Non-extendable Finite Trajectories

3.3 Global Asymptotic Stability and Lyapunov Functions

This section gives a formal definition of the stability property that is central to this thesis: global asymptotic stability. This definition is independent of the hybrid nature of a system and applicable to any dynamical system operating in continuous time. Next, Lyapunov functions are formally introduced, first for systems with purely continuous dynamics and then for hybrid systems.

3.3.1 Global Asymptotic Stability

The state $x_e \in \mathbb{R}^n$ the system is supposed to converge to is also called the system's *equilibrium state*. A system given by a single continuous differential inclusion $\dot{x} \in F(x), x \in \mathbb{R}^n$ cannot be globally asymptotically stable unless $F(x_e) = \{0\}$, that is the equilibrium state is never left, once entered. By convention, we assume that the equilibrium state the system is expected to converge to is always at 0. Of course, in practical applications it is often the case that the system is supposed to converge to some other continuous state. A cruise controller, for instance, might be expected to force the velocity v to converge to a given target value v_0, representing a *set point* of the system. In this case, the equilibrium can be artificially shifted to 0 by variable substitution. If the set point is x_e, the variable x can be replaced by a new variable $\bar{x} = x - x_e$. The result is a new differential equation in $\bar{x}$, whose right hand sides are shifted accordingly. The equilibrium will be $\bar{x} = 0$, which is equivalent to $x = x_e$. Moreover, global asymptotic stability of a system with respect to an equilibrium at 0 implies global asymptotic stability for any set point, as long as the control strategy is independent of the actual set point. This is the case if the control strategy can be formulated only in terms of $\bar{x}$. Therefore, without loss of generality, the equilibrium state can assumed to be 0 for stability analysis. This is particularly important for the Lyapunov function computation methods, because the algorithms generally assume an equilibrium at 0.

If a system has an equilibrium state at 0, then *global asymptotic stability* can be defined as follows.

Definition 3.15 (Global Asymptotic Stability [Khalil, 1996])**.** A continuous-time dynamic system H is called *globally stable (GS)* if for all $\epsilon > 0$ there exists a $\delta > 0$ such that for all (finite or infinite) trajectories $x(\cdot)$ of H and for all times t for which $x(t)$ is defined:

$$||x(0)|| < \delta \implies ||x(t)|| < \epsilon.$$

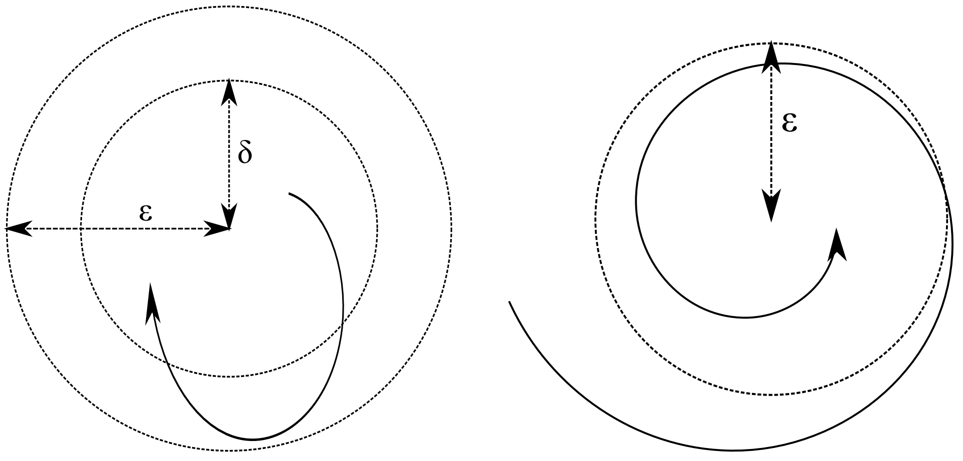
H is called *globally attractive (GA)* if for all infinite trajectories $x(t)$ of H

$$\lim_{t\to\infty} x(t) = 0, \text{ that is, } \forall \epsilon > 0 : \exists t_0 \geq 0 : \forall t > t_0 : ||x(t)|| < \epsilon.$$

A system that is both globally stable and globally attractive is called *globally asymptotically stable (GAS)*.

In a nutshell, global stability is a boundedness condition (see Figure 3.6(a)). For any bound ϵ on the distance to the equilibrium state 0, by bounding the possible initial states with the δ-ball, we only obtain trajectories that stay within the ϵ-ball. In other words, by allowing only small enough deviations from 0, represented by initial states, it is always possible to stay within a bound on all states, such that "small" disturbances only lead to "small" errors in the long run.

Global attractivity covers the actual convergence to the desired state (see Figure 3.6(b)). This property can be viewed as a conjunction of infinitely many liveness properties: For every epsilon-ball $B(0, \epsilon)$ (the dashed circle) around the origin, every trajectory will eventually enter $B(0, \epsilon)$ and never leave it again. Note that convergence does usually not mean that the equilibrium state is actually reached. It is sufficient to approach it infinitely close. This means that a globally attractive system is also region stable with respect to any region containing the equilibrium in its interior.



(a) Global Stability (b) Global Attractivity

Figure 3.6: Global Asymptotic Stability

The acronym "GAS" is used both as a noun and as an adjective in this thesis, meaning either "globally asymptotically stable" or "global asymptotic stability."

Note that the basic definitions of stability and attractivity neither constrain the ϵ-deviation nor the convergence rate. It is, however, possible to quantify these values as a side product of Lyapunov-function-based verification. Convergence rates for the attractivity property are discussed in Section 3.5.4.

For hybrid systems, global asymptotic stability talks only about trajectories, representing the evolution of a system's *continuous* state over time. The discrete behavior of a system only influences asymptotic stability indirectly, as mode switchings force the system to follow different differential inclusions, leading to different trajectories. Only the continuous state is required to converge to 0. In fact, it is easy to construct GAS systems for which the discrete state does not converge at all, but oscillates forever. A simple system with this behavior is given in Example 3.6.

In the literature, global asymptotic stability for hybrid systems is also sometimes defined with the additional stipulation that there do not exist any non-extendable finite trajectories. The notion of GAS used in this thesis does not require this assumption and is also sometimes termed *preasymptotic stability* [Cai *et al.*, 2008].

As a counterpart to global stability, there also exist definitions of local stability properties, where stability and attractivity are only fulfilled for initial states $x(0)$ that lie within certain sub-sets, called *regions of attraction*. This thesis focuses on global stability properties, but adaptions for the local case are possible in many cases by adequate restriction of a system's invariants.

Example 3.6 ([Johansson & Rantzer, 1998])**.** Consider the hybrid system given in Figure 3.7. The two discrete modes are combined by a switching strategy that results in convergence to the equilibrium from any initial state. This example illustrates that it is in general not a necessary condition that the Euclidean distance to the equilibrium decreases monotonically.

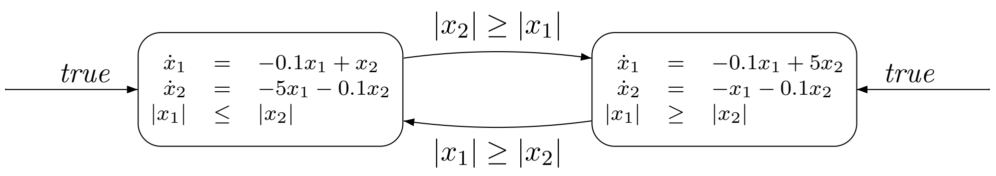

$|x_2| \geq |x_1|$
true
$\dot{x}_1 = -0.1x_1 + x_2$
$\dot{x}_2 = -5x_1 - 0.1x_2$
$|x_1| \leq |x_2|$
$\dot{x}_1 = -0.1x_1 + 5x_2$
$\dot{x}_2 = -x_1 - 0.1x_2$
$|x_1| \geq |x_2|$
true
$|x_1| \geq |x_2|$

Figure 3.7: Globally Asymptotically Stable Two-mode System

As can be seen in the plot of an example trajectory given in Figure 3.8, this distance will actually sometimes increase for some time. Lyapunov functions, as they are defined in the following section, provide a generalization of such a distance measure that can also be applied to this system.

Sometimes, we only want a system to be GAS for some of its variables. For instance, some system variables might only have a helper role inside a controller, not representing any physical properties. If some system variables do not need to fulfill the stability property, then we call this *GAS with respect to a sub-set of variables*, as defined below.

Definition 3.16 (Global Asymptotic Stability with Respect to a Sub-set of Variables)**.** Let H be a hybrid automaton, and let $\mathcal{V}'$ be a sub-set of the variables of H. For a system state vector $x \in \mathcal{S}$, let $x_{|\mathcal{V}'}$ be the sub-vector only containing the values of variables in $\mathcal{V}'$. A continuous-time dynamic system H is called *globally stable (GS) with respect to* $\mathcal{V}'$ if for all $\epsilon > 0$ there exists a $\delta > 0$ such that for all functions $x_{|\mathcal{V}'}(\cdot)$ and for all t for which $x_{|\mathcal{V}'}(t)$ is defined we have

$$||x(0)|| < \delta \implies ||x_{|\mathcal{V}'}(t)|| < \epsilon.$$

H is called *globally attractive (GA) with respect to* $\mathcal{V}'$ if for all functions $x_{|\mathcal{V}'}(t)$, we have

$$\lim_{t\to\infty} x_{|\mathcal{V}'}(t) = 0, \text{ that is, } \forall \epsilon > 0 : \exists t_0 \geq 0 : \forall t > t_0 : ||x_{|\mathcal{V}'}(t)|| < \epsilon,$$

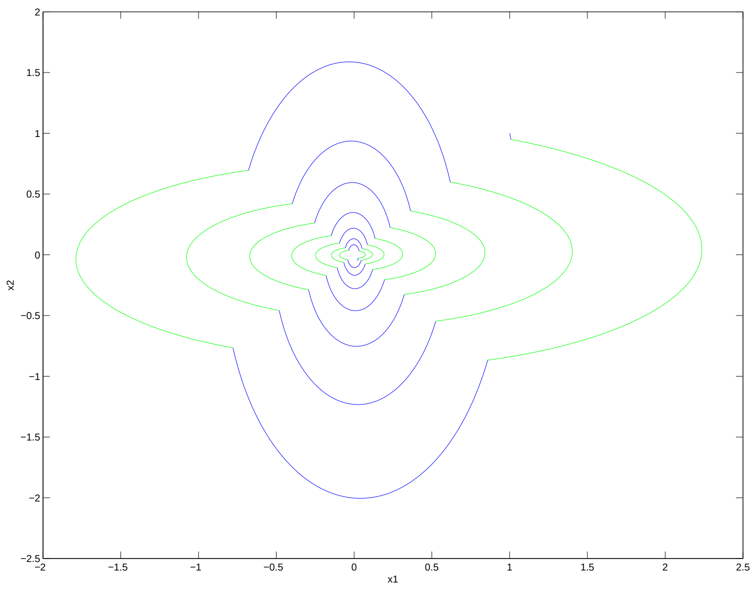


Figure 3.8: Example Trajectory for the System Given in Figure 3.7

where 0 is the origin of $\mathbb{R}^{|\mathcal{V}'|}$. A system that is both globally stable with respect to $\mathcal{V}'$ and globally attractive with respect to $\mathcal{V}'$ is called *globally asymptotically stable (GAS) with respect to* $\mathcal{V}'$.

Note that, for the global stability property, the bound on the initial state $x(0)$ considers all variables while the bound on the overall reachable states only refers to variables in $\mathcal{V}'$. This is because variables in $\mathcal{V}'$ will usually still have dependencies with variables outside $\mathcal{V}'$. If there is no bound on the initial state of those other variables, then it is usually not possible to prove a bound on the overall states of variables in $\mathcal{V}'$.

3.3.2 Lyapunov Functions

Throughout this thesis, Lyapunov functions are used to prove the global asymptotic stability property given by Definition 3.15. In this section, the theorems for conducting such a proof are presented. They allow the conclusion that the existence of a Lyapunov function does indeed imply global asymptotic stability. See Figure 3.9 for an image of a Lyapunov function with a superimposed trajectory (in black) which converges and along which the Lyapunov function value is strictly decreasing. First, systems consisting only of a single differential inclusion are covered (comparable to hybrid automata with just one discrete mode, no transitions, and an invariant *true*), and then the Lyapunov function definition is extended to hybrid automata. For this purpose, another definition using individual local functions for the discrete modes of the automaton is given.

First, we define *class* K^∞ *functions*, which are in turn used to define Lyapunov func-

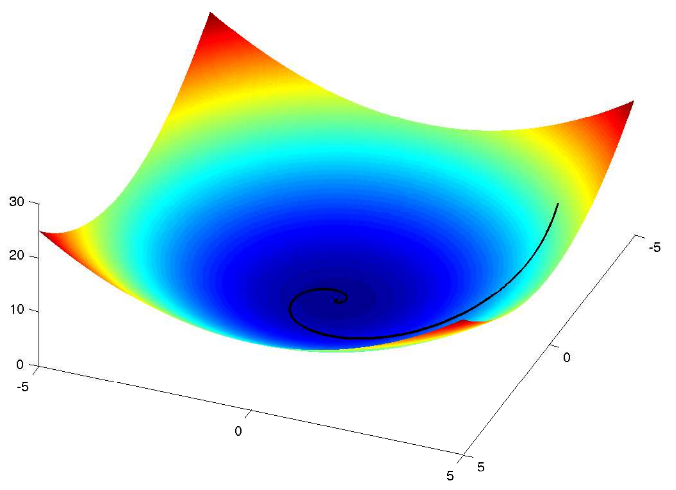


Figure 3.9: Quadratic Lyapunov Function and Convergent Trajectory

tions.

Definition 3.17 (Class K^∞ Functions [Khalil, 1996]). A function $f : \mathbb{R}^+ \to \mathbb{R}^+$ is of *class* K^∞ if and only if:

- $f(0) = 0$,
- f is strictly monotonically increasing, and
- $\lim_{x\to\infty} f(x) = \infty$.

Theorem 3.3 (Lyapunov Functions [Pettersson, 1999; Cortés, 2008]). Let $\dot{x} \in F(x), x \in \mathbb{R}^n$, be a differential inclusion. If there exists a continuously differentiable function $V : \mathbb{R}^n \to \mathbb{R}$, called *Lyapunov function*, such that

(1) there exist class K^∞ functions f_1 and f_2 such that for all $x \in \mathbb{R}^n$ $f_1(||x||) \le V(x) \le f_2(||x||)$,

(2) there exists a class K^∞ function f_3 such that for all $x \in \mathbb{R}^n$: $\dot{V}(x) \le -f_3(||x||)$, where $\dot{V}(x) := \sup \left\{ \langle \frac{dV}{dx}(x) \mid y \rangle | y \in F(x) \right\}$,

then the continuous-time system given by the differential inclusion is GAS.

Here, $\frac{dV}{dx}(x)$ denotes the gradient of V at x, given as a row vector. The *time derivative* $\dot{V}$ of V is defined as the supremum of the scalar product of this gradient (describing the rate of change of V with respect to x) and any behavior allowed by the differential inclusion, given as a column vector (describing the rate of change of x with respect to time).

We require continuous differentiability for the Lyapunov functions since compositions of continuously differentiable and absolutely continuous functions are again absolutely continuous. This means that $V(x(t))$ is also absolutely continuous, which is instrumental for concluding GAS of the system, as the evolution of $V(x(t))$ can then be cast into an integral form as in Remark 3.2.

Remark 3.4 (Global Exponential Stability [Pettersson, 1999])**.** A special case of a class K^∞ function is $f(x) = \alpha x$ for some $\alpha > 0$. If the class K^∞ functions f_1, f_2, and f_3 are of this type in the above theorem, then one can furthermore conclude that all system trajectories converge with an exponential convergence rate. This is called *exponential stability*. This restricted notion of stability has the advantage that the convergence rate can be used to estimate the convergence time from any starting state to any target region around the equilibrium. The computational method that will be described in Section 3.5 generally show exponential stability.

If $F(x)$ is a singleton set for all x, then the system can be represented as a differential equation $\dot{x} = f(x)$. In this case, condition (2) reduces to

(2') *there exists a class K^∞ function f_3 such that for all $x \in \mathbb{R}^n$: $\dot{V}(x) \leq -f_3(||x||)$, where $\dot{V}(x) := \left\langle \frac{dV}{dx}(x) \mid f(x)\right\rangle$.*

For a system consisting only of a single differential inclusion, we need to identify a function that is positive everywhere but in the origin, where it is required to be zero, as it is bounded from above by a class K^∞ function. Furthermore, Condition (2) stipulates that the function decreases along every solution of the differential inclusion, except in the origin. This implies convergence to the origin.

To extend this theorem to hybrid systems, it is necessary to account for the discontinuities introduced by the switching. A simple way of achieving this is by imposing additional conditions on the Lyapunov functions, such that the discrete updates triggered by the transitions cannot lead to an increase of the Lyapunov function value. This leads to the concept of *common Lyapunov functions*. Here, all discrete modes of hybrid automaton share the same Lyapunov function.

Theorem 3.4 (Common Lyapunov Functions [Pettersson, 1999])**.** Let H be a hybrid automaton according to Definition 3.12. If there exists a continuously differentiable function $V : \mathcal{S} \to \mathbb{R}$, called a *common Lyapunov function*, such that

(1) there exist class K^∞ functions f_1 and f_2 such that for all $x \in \bigcup_{m \in \mathcal{M}} \mathit{Inv}(m)$: $f_1(||x||) \leq V(x) \leq f_2(||x||)$,

(2) for each $m \in \mathcal{M}$ there exists a class K^∞ function f_3 such that for all $x \in \mathit{Inv}(m)$: $\dot{V}_m(x) \leq -f_3(||x||)$, where $\dot{V}_m(x) := \sup\left\{\left\langle \frac{dV}{dx}(x) \mid y\right\rangle \middle| y \in \mathit{Flow}(m)(x)\right\}$,

(3) for each mode transition $(m_1, m_2, G, U) \in \mathcal{T} : x \in G \implies V(U(x)) \leq V(x)$,

then H is globally asymptotically stable.

Note that Condition (3) is trivially true in case all updates U are the identity function. In this case, the guards G of the transitions are of no consequence, which in turn implies that the hybrid automaton H is GAS *regardless of the switching strategy* that is employed. This illustrates the conservatism of the common Lyapunov function approach: For this class of systems, common Lyapunov functions can only be identified if the system would be stable under any switching strategy.

For instance, the system given in Example 3.6 is globally asymptotically stable, but this cannot be shown by a common Lyapunov function. The reason for this is the existence of possible alternative switching strategies that would not result in a stable hybrid system. For such cases, refined notions of Lyapunov functions for hybrid systems are needed.

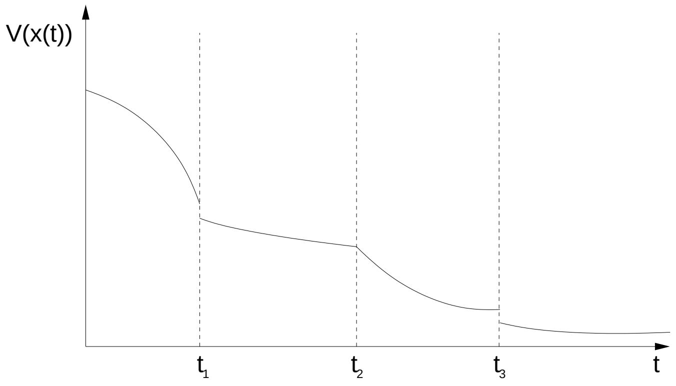


Figure 3.10: Piecewise Continuous Lyapunov Function over Time

A common approach here is to allow for multiple local Lyapunov functions for the different discrete modes (see Figure 3.10) [Branicky, 1998]. The family of these local Lyapunov functions then forms the global Lyapunov function. Each of the local functions fulfills Conditions (1) and (2) of Theorem 3.4, whenever the mode m can be active, that is whenever the current continuous state x is allowed by its invariant $Inv(m)$. Furthermore, upon switching from one discrete mode to another, the local Lyapunov function value of the source mode may not be lower than that of the target mode. Discrete switchings between modes, each with its own Lyapunov function, must not increase the global Lyapunov function value. Under these circumstances, stability of a hybrid system given by a hybrid automaton can be shown by identifying a family of suitable local Lyapunov functions, one function per mode.

Theorem 3.5 (Discontinuous Lyapunov Functions [Pettersson, 1999])**.** Let H be a hybrid automaton according to Definition 3.12. If for each $m \in \mathcal{M}$ there exists a continuously differentiable $V_m : \mathcal{S} \to \mathbb{R}$ such that

(1) for each $m \in \mathcal{M}$ there exist class K^∞ functions f_1 and f_2 such that for all $x \in Inv(m)$: $f_1(||x||) \leq V_m(x) \leq f_2(||x||)$,

(2) for each $m \in \mathcal{M}$ there exists a class K^∞ function f_3 such that for all $x \in Inv(m)$:

$\dot{V}_m(x) \leq -f_3(||x||)$, where $\dot{V}_m(x) := \sup \left\{ \langle \frac{dV}{dx}(x) | y \rangle | y \in \mathit{Flow}(m)(x) \right\}$,

(3) for each mode transition $(m_1, m_2, G, U) \in \mathcal{T} : x \in G \implies V_{m_2}(U(x)) \leq V_{m_1}(x)$,

then the hybrid automaron H is globally asymptotically stable. The function V_m is called a *local Lyapunov function (LLF)* of H for mode m. The family of $V_m, m \in \mathcal{M}$ forms a function $V(x, m) : \mathbb{R}^n \times \mathcal{M} \to \mathbb{R}$, where $V(x, m) = V_m(x)$. This function $V(x, m)$ is called a *global Lyapunov function (GLF)* of H.

The theorem given in [Pettersson, 1999] deals with differential equations in place of differential inclusions. However, the proof given in [Pettersson, 1999, p. 87] can be applied with only very minor changes (i.e., replacing continuously differentiable solutions for differential equations with absolutely continuous Carathéodory solutions) for the more general case of Theorem 3.5. Theorems 3.3 and 3.4 are simply special cases of Theorem 3.5.

Example 3.7. For the system from Example 3.6, it is now possible to prove stability via Lyapunov functions. A possible discontinuous Lyapunov function is given as $V_{m_1}(x_1, x_2) = 5x_1^2 + x_2^2$ and $V_{m_2}(x_1, x_2) = x_1^2 + 5x_2^2$ [Johansson & Rantzer, 1998].

Sometimes, even one local continuous Lyapunov function per mode might not be enough. In this case, it is possible to allow for a discontinuous Lyapunov function even within a single node. This can be achieved by partitioning its invariant set into several sub-sets, each with its own (continuous) Lyapunov function. Theorems for this case are variants of Theorem 3.5, where the invariants *Inv*(m) are replaced by these sub-sets. Additionally, one needs information on the possible transition directions between sub-sets, to formulate inequality conditions in the vein of Condition (3). Another way of viewing this refinement is as a re-formulation of the hybrid automaton. It is possible to split a mode of a hybrid automaton into several new modes, each covering a part of the original mode's invariant set, and each inheriting the continuous dynamics of the old mode. The transitions between the new modes must at least include all transitions between the partition sets which trajectories could have taken in the original system. In this case, all trajectories of the old system will also be trajectories of the new system, so that stability of the latter implies stability of the former. This usage of multiple Lyapunov functions for a single mode is directly equivalent to conducting stability analysis corresponding to Theorem 3.5 (i.e., using one Lyapunov function per mode) for the new system. However, such a mode splitting must be performed very carefully to be of any use: Brute-force splitting will lead to an exponential growth in complexity which renders the problem of identifying Lyapunov function intractable in practice very quickly. See [Oehlerking *et al.*, 2007] for a detailed discussion. A special case of node splitting will also be employed in Chapter 4 to facilitate decomposition of hybrid automata.

Lyapunov functions (either continuous or discontinuous) for any given hybrid automaton are closed under conic combination.

Theorem 3.6 (Cones of Lyapunov Functions)**.** *Let H be a hybrid automaton and, for each $1 \leq i \leq k$, let V^i be a GLF for H. Then, all functions in cone$(\{V_i \mid 1 \leq i \leq k\})$ are also global Lyapunov functions for H.*

This theorem follows directly from the Definition 3.5, since also class $\mathcal{K}^\infty$ functions form a convex cone. It is straightforward to see that both the sum of two Lyapunov functions and the product of a Lyapunov function and a positive scalar are again Lyapunov functions.

Remark 3.5. The fact that the "tip of the cone" is not included in Definition 3.6 is motivated by Theorem 3.6 stating that all conic combinations of (global or local) Lyapunov functions are again (global or local) Lyapunov functions for the same system or mode, except for the constant zero function. Clearly, the function $f(x) = 0$ cannot be a Lyapunov function in any case, since it does not decrease strictly along any trajectory.

The property expressed by Theorem 3.6 has a very important consequence: Since conic hulls are also always convex, the problem of identifying a Lyapunov function for a given system is always convex, and therefore solvable by convex optimization. This property will also play a central role in the decomposition methods in Chapter 4.

One special case is a system where only some continuous variables are required to converge to 0. The corresponding stability property, *GAS with respect to a sub-set of variables*, has already been given in Definition 3.16. In this case, we have the option to choose a Lyapunov function that does not depend on the non-convergent variables at all. Consequently, the Lyapunov theorem can be relaxed as follows.

Theorem 3.7 (Lyapunov Functions for a Sub-set of System Variables)**.** Let H be a hybrid automaton according to Definition 3.12 and let $\mathcal{V}' \subset \mathcal{V}$. Let $x_{|\mathcal{V}'}$ be the sub-vector of $x \in \mathcal{S}$ only containing values of variables in $\mathcal{V}'$. If for each $m \in \mathcal{M}$ there exists a set of variables $\mathcal{V}_m$ with $\mathcal{V}' \subseteq \mathcal{V}_m \subseteq \mathcal{V}$ and a continuously differentiable function $V_m : \mathcal{S} \to \mathbb{R}$ such that

(1) for each $m \in \mathcal{M}$ there exist class K^∞ functions f_1 and f_2 such that for all $x \in Inv(m)$: $f_1(||x_{|\mathcal{V}_m}||) \leq V_m(x) \leq f_2(||x_{|\mathcal{V}_m}||)$,

(2) for each $m \in \mathcal{M}$ there exists a class K^∞ function f_3 such that for all $x \in Inv(m)$: $\dot{V}_m(x) \leq -f_3(||x_{|\mathcal{V}_m}||)$, where $\dot{V}_m(x) := \sup\left\{\left\langle \frac{dV}{dx}(x) \mid y\right\rangle \middle| y \in Flow(m)(x)\right\}$,

(3) for each mode transition $(m_1, m_2, G, U) \in \mathcal{T} : x \in G \implies V_{m_2}(U(x)) \leq V_{m_1}(x)$,

then the hybrid system H is globally asymptotically stable with respect to $\mathcal{V}'$.

The difference to Theorem 3.5 lies in the fact that V_m is no longer required to grow with increasing absolute value of variables outside $\mathcal{V}'$. In particular, we can now also have Lyapunov functions which do not depend on these variables at all, which was impossible in Theorem 3.5. However, it can still be desirable to have a these variables explicitly appear in the Lyapunov function term, especially, if they have a noticeable effect on variables that do lie in $\mathcal{V}'$. In general, any variables that have an effect on the behavior of the variables on $\mathcal{V}'$ in mode m should be included in $\mathcal{V}_m$. If, however, a variable is not needed at all in mode m and if does not change in mode m (for instance, because it only plays a role in some other modes), then there is no need to force it to appear in $\mathcal{V}_m$. The following Example 3.8 is such a case where a variable is not needed in a mode and its derivative is therefore set to zero. In this case, it would not be possible at all to find a LLF for this mode unless this variable is excluded from $\mathcal{V}_m$, as it remains unchanged over time.

With the help of Theorems 3.3, 3.4, 3.5, and 3.7, it is possible to conduct a proof of global asymptotic stability by identifying suitable Lyapunov functions. This proof can be conducted by hand, if one manages to find a suitable Lyapunov function (or a family of local Lyapunov functions) for the system. However, for complex systems, this is usually a complicated and time-consuming task, requiring a lot of insight into the structure and the behavior of the system. Therefore, it is very desirable to automate this process. It is, of course, not possible to search the entire space of functions, so we must restrict the search space to a parametrized sub-space. The next section gives an overview about the linear-matrix-inequality-based methods for Lyapunov function computation that are used as building blocks for decompositional proofs in this thesis. In theory, these methods can be applied to a large class of systems, but in many cases, their usefulness is undermined by numerical issues that still make it challenging to find solutions in practice. These drawbacks will be discussed in Section 3.5.

The following example illustrates the use of Lyapunov functions for stability proofs.

Example 3.8 (Cruise Control Example)**.** Consider an extension of the proportional-integral controller of Example 3.1 on page 31. The plant remains the same as before, but the controller is now hybrid. The controller C is modelled as a hybrid system consisting of a proportional-integral (PI) controller sub-system that is active if v is near 0, and two sub-systems that model so-called *saturations*, that is, "cut-offs" for maximal and minimal acceleration. A PI-controller is a linear system and would therefore violate comfort requirements of maximal acceleration and deceleration for large velocity differentials. This is remedied by the inclusion of the two additional modes. Each of the three modes of operation has a different "area of responsibility" in the state space, which is reflected in its particular invariant. The modes are as follows:

- A deceleration mode m_1, setting $a = -2$, which models a constant braking effort: This mode is responsible for situations when the vehicle is considerably too fast, and therefore associated with the invariant $5 \leq v$.
- a PI-control mode m_2, with an additional internal variable x, representing the integrator part (i.e., $\dot{x} = v$), and setting $a = -0.001x - 0.052v$: This mode is responsible for the smooth approach to $v = 0$, and therefore associated with the invariant $-15 \leq v \leq 15$.
- a acceleration mode m_3, setting $a = 1.5$, which models a constant maximal acceleration: This mode is responsible for situation when the vehicle is considerably too slow, and therefore associated with the invariant $-5 \geq v$.

To arrive at a hybrid automaton model for the system, one has to build a closed-loop model subsuming plant and controller dynamics. For each mode, the flow function and invariant set are simply constructed from the conjunction of the plant dynamics and the

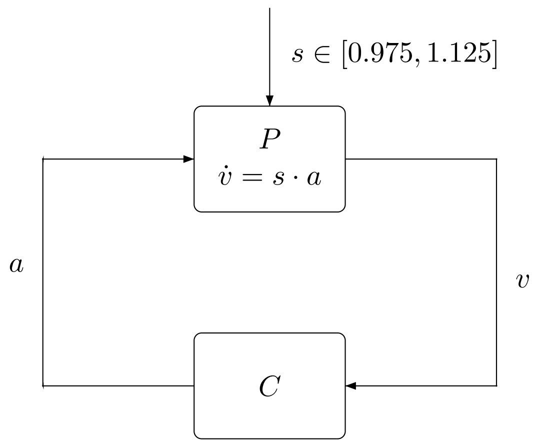


Figure 3.11: Cruise Controller Setup

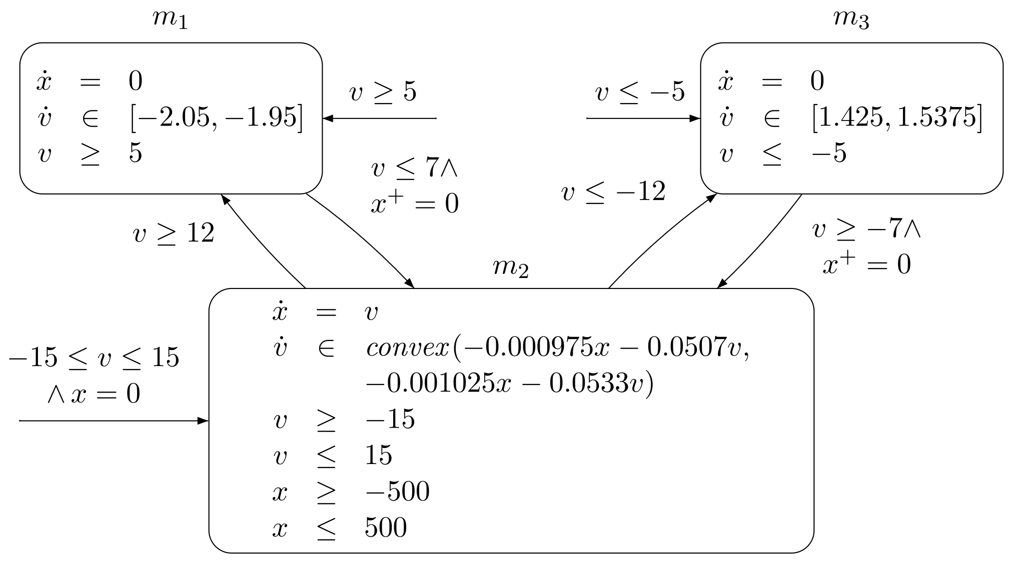

m_1
$\dot{x} = 0$, $\dot{v} \in [-2.05, -1.95]$, $v \geq 5$
$v \geq 5$
m_3
$\dot{x} = 0$, $\dot{v} \in [1.425, 1.5375]$, $v \leq -5$
$v \leq -5$
$v \leq 7 \wedge x^+ = 0$
$v \leq -12$
$v \geq 12$
$v \geq -7 \wedge x^+ = 0$
m_2
$\dot{x} = v$
$\dot{v} \in convex(-0.000975x - 0.0507v, -0.001025x - 0.0533v)$
$v \geq -15$
$v \leq 15$
$x \geq -500$
$x \leq 500$
$-15 \leq v \leq 15 \wedge x = 0$

Figure 3.12: Cruise Control Automaton

controller dynamics of the mode in question. Also, a suitable switching strategy should be selected. This leads to the hybrid automaton model as given in Figure 3.12. Note that some additional constraints on the integrator variable x have been imposed, namely $-500 \leq x \leq 500$. This additional invariant can be exploited for the stability analysis and does not reduce the number of possible trajectories, as states with $|x| > 500$ are not reachable from the initial states in any case. This can, for instance, be verified with barrier certificate techniques (see Section 3.4).

Note that the variables s and a, which did not appear on any left-hand sides of differential equations or inclusions, have been eliminated to arrive at a simpler description. The variable s has simply been replaced by the interval that bounds it, and the acceleration a is equal to $\dot{v}$. The transitions have been chosen to avoid chattering (i.e., transitions back and forth between modes in quick succession). This is achieved by a hysteresis-like switching strategy that also prevents Zeno behavior.

Proving global asymptotic stability of this system with respect to equilibrium point $v = 0$ and a sub-set of variables $\mathcal{V}' = \{v\}$ requires us to identify a Lyapunov function V_{m_i} for each of the three modes, such that all conditions of Theorem 3.7 are fulfilled.

For V_{m_2}, since v directly depends on x, we set $\mathcal{V}_m = \{x, v\}$. One example function fulfilling Conditions (1) and (2) of 3.7 is

$$V_{m_2}(v, x) = 46.7455x^2 + 1101.9vx + 20729v^2.$$

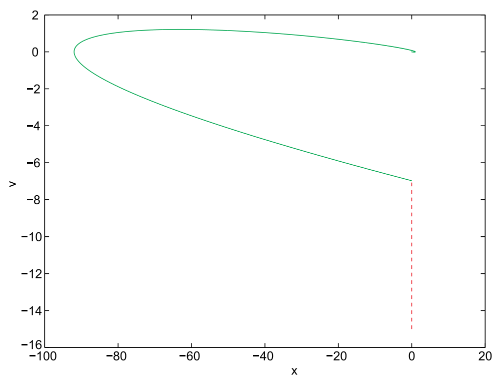
For the modes m_1 and m_3, the variable x is of no consequence, as it remains unchanged. Therefore, we set $\mathcal{V}_{m_1} = \mathcal{V}_{m_3} = \{v\}$. We can pick any quadratic function in v with positive coefficient, for example

$$V_{m_1}(v, x) = V_{m_3} = v^2.$$

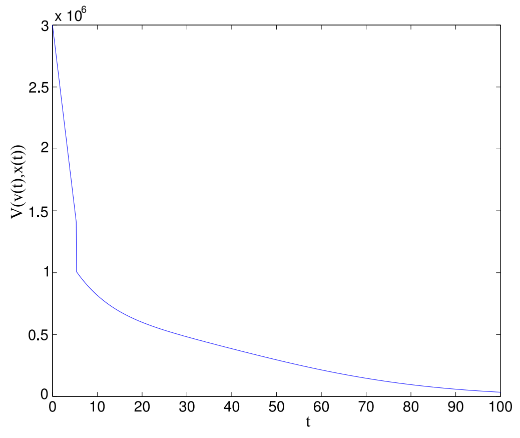
These Lyapunov functions need not depend on x, since the value of variable x is of no consequence for the behavior of variable v, and since x will be reset to 0, once we return to mode m_2. These functions fulfill Condition (1) on $v \geq 5$ and $v \leq -5$, and the time derivative $\dot{V}$ is given as $-4v$ for mode m_1 and $3v$ for mode m_3, respectively. Both of these functions fulfill Condition (2) on the invariant of the corresponding mode.

However, one can easily see that Condition (3) is not satisfied for all transitions of the hybrid automaton for these LLFs. Computing Lyapunov functions separately for all modes, as we have done, is not guaranteed to produce a global Lyapunov function. To obtain functions V_{m_i} which do not have this problem, we can, for instance, choose $V_{m_1} = 200000v$ and $V_{m_3} = -200000v$. In this case, also Condition (3) is satisfied.

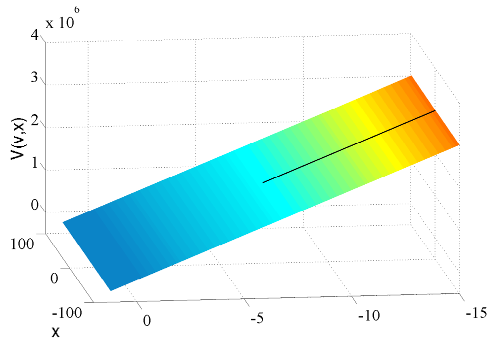
Figure 3.13 gives visualizations of the global Lyapunov function obtained in this manner. In Figure 3.13(a), an example trajectory of the system is shown, starting at $v = -15$ in mode m_3. The red/dashed part of the trajectory shows the time spent in m_3. Eventually, the system switches to mode m_2, which is shown by the green/solid part of the trajectory, and converges to $v = x = 0$. Figure 3.13(b) shows the evolution of the global Lyapunov function value over time. Note that the function is strictly decreasing over time along the trajectory. As soon as the switch to m_2 occurs (around time $t = 5$), there is a discontinuity in the overall global Lyapunov function, which is visible as a sudden drop of the Lyapunov function value. The Lyapunov function value eventually converges to 0. Figures 3.13(c) and 3.13(d) show the Lyapunov functions for modes m_3 and m_2, respectively. Superimposed on the function surfaces is the segment of the example trajectory belonging to the particular mode. The Lyapunov function for mode m_1 is not shown, as it is simply the mirror image of V_{m_3} along the plane $v = 0$, and the example trajectory does not enter the mode m_1.



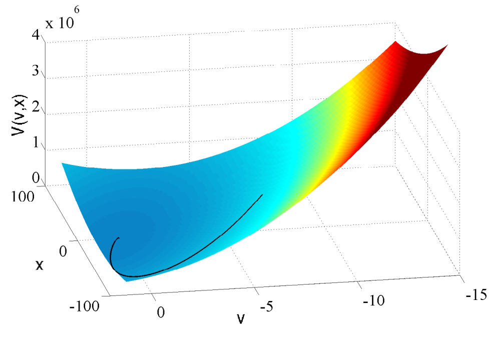
(a) two-dimensional trajectory plot



(b) Lyapunov function over time



(c) Lyapunov function plot for m_3



(d) Lyapunov function plot for m_2

Figure 3.13: Cruise Control System: Example Trajectory and Lyapunov Functions

3.4 Lyapunov Functions as Barrier Certificates

A concept that is related to Lyapunov functions is the so-called *barrier certificate* [Prajna & Jadbabaie, 2004]. If one is interested in proving safety (i.e., the non-reachability of a pre-specified set of undesirable states), then a function with Lyapunov-like properties can be employed. The basic idea is to find a function fulfilling a relaxed version of the Lyapunov condition on the time derivative. To be exact, a function V such that $\dot{V}(x) \leq 0$ is required, and if this is the case, then we know that V is always monotonically decreasing over time, but not necessarily strictly decreasing. If, at time t, we have $V(x(t)) \leq c$ for some positive c, then we know that all states x with $V(x) > c$ will remain unreachable for all future times.

Therefore, if we have such a function V, we can prove safety if we identify a value c such that $V(x_0) \leq c$ for all initial states and such that $V(x) > c$ for all unsafe states. See Figure 3.14 for an illustration.

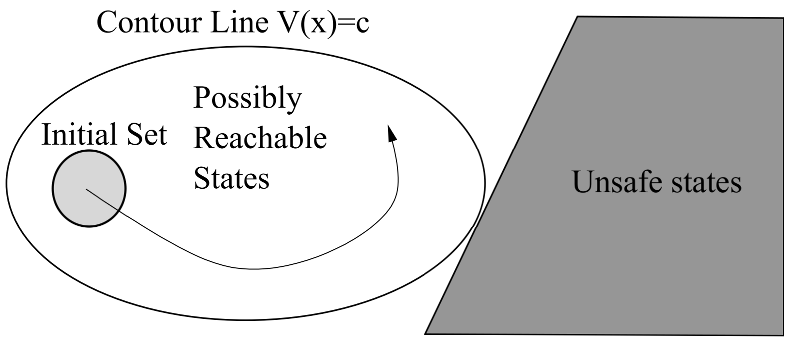


Figure 3.14: Barrier Certificate Separating Initial and Unsafe States

Since the conditions on Lyapunov functions (as in Theorem 3.3) are stronger than those on barrier certificate functions, every Lyapunov function can also be used to conduct a safety proof with respect to some unsafe region, allowing a joint proof of stability and safety. The only additional effort needed to conduct the safety proof consists of identifying a suitable constant c.

In the same vein as for discontinuous Lyapunov functions, discontinuous functions can also serve as barrier certificates. The conservative approach is to require that function V does not increase upon switching from one mode to another (as in Theorem 3.5). However, it is also sufficient to require that, upon switching from m_1 to m_2, the set of states the system can attain after the switch is again separated from the unsafe region by a contour line $V(m_2, x) = c_{m_2}$ for the target mode. When taking this approach, each mode m may have its own threshold c_m, separating all possible initial states of the mode (taking into account all incoming transitions) from the unsafe set.

In other words, V may increase, as long as this increase is not so severe that the safety requirement can be violated at some point. See Fig. 3.15 for an illustration of an admissible behavior of the function V over time. The dotted lines depict the thresholds c_m which guarantee that the system is still safe. Since these threshold might differ depending on the active mode, they may change upon every mode switch. The value of the function V, represented by the solid lines, must not increase while a mode is active. Upon a mode switch, however, function V is allowed to increase, as long as the threshold for the new mode is not exceeded.

The following section introduces methods for the automatic computation of Lyapunov functions. These methods can also be applied for the computation of barrier certificates with only minor changes [Prajna & Jadbabaie, 2004].

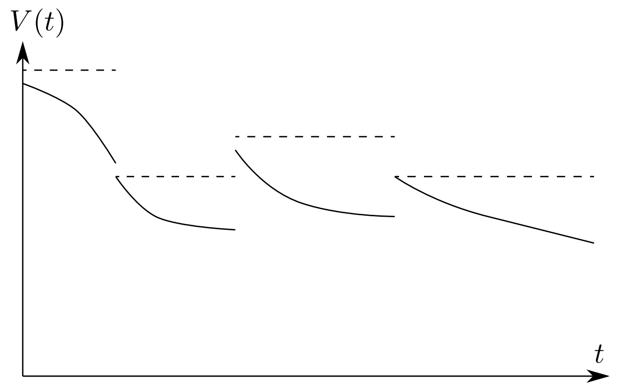


Figure 3.15: Admissible Behavior of a Barrier Certificate Function V over Time

3.5 Computing Lyapunov Functions

This section details the actual computation of parametrized Lyapunov functions for certain classes of hybrid systems. These computations are based on *linear matrix inequalities* (LMI) [Boyd *et al.*, 1994], which can be used to formulate the Conditions (1)-(3) of Theorem 3.5 as conditions on matrices. These inequalities contain unknowns that represent the free parameters of a Lyapunov function template. For the standard LMI approach to stability proofs, such templates are quadratic functions of the form $V(x) = x^T P x$, where P is a symmetric matrix of free parameters, for example

$$P = \begin{bmatrix} p_1 & p_2 & p_3 \\ p_2 & p_4 & p_5 \\ p_3 & p_5 & p_6 \end{bmatrix}$$

for a three dimensional system, with the p_i being the parameters. These free parameters will be bound to fixed values, such that a global Lyapunov function according to Theorem 3.5 results. This problem can be mapped onto a class of non-linear, convex optimization problems called *semidefinite programming (SDP)* problems, which can in turn be solved by numerical algorithms.

The approaches presented in this section have one common characteristic: they look at the hybrid system in a monolithic fashion. All parts of the system (i.e., differential inclusions, invariant, guards, updates) are mapped onto corresponding linear matrix inequality constraints. The resulting constraint system is then converted into *one* SDP problem, which is handed over to a solver. If the solver is able to find a solution (i.e., values for the parameters, such that Conditions (1)-(3) of Theorem 3.5 are all fulfilled), then it will return valuations for the free parameters. The computed Lyapunov function can then be used as a certificate of global asymptotic stability. However, the methods discussed below have their limitations, which is a main motivation for the decompositional methods which are the central focus of this thesis. These limitations will be discussed in detail in Section 3.6.3. The decomposition methods that are presented in Chapter 4 split the problem of computing a Lyapunov function into *several local* LMI problems.

In order to conduct Lyapunov function computations, a few restrictions need to be formulated. The first one is that the differential inclusions need to be *convex*.

Definition 3.18 (Convex Differential Inclusion)**.** A differential inclusion $\dot{x} \in F(x)$ is called *convex*, if there exist m functions $f_i : \mathbb{R}^n \to \mathbb{R}^n, 1 \leq i \leq m$, such that for all x

$$F(x) = \{y \mid y \in \mathit{convex}(\{f_1(x), \ldots, f_m(x)\})\}\,.$$

A convex differential inclusion has the property that all the behaviors it allows lie in a convex set, given by finitely many corner points. At any point in time, the system may behave according to any differential equation $\dot{x} = f_i(x)$, or any differential equation corresponding to a convex combination of the functions f_i.

Note that the use of convex differential inclusions is more restrictive than the use of general differential inclusions $\dot{x} \in F(x)$. However, convex differential inclusions are still sufficient to model most of the uncertain dynamics that arise in hybrid system applications. For instance, bounded disturbances on the system dynamics can be modelled in this fashion, in the same manner as in Example 3.8. If a disturbance does not result in a convex right-hand side of a differential inclusion, then it is also usually possible to over-approximate the system behavior in a convex manner.

Secondly, we need to restrict the type of functions that are allowed as corner points of the cone, that is, the functions f_i. Direct application of LMI based methods requires that these functions are either linear or affine. The same restriction applies to the update functions. However, with the sums-of-squares decomposition [Parrilo, 2003] (see Section 3.5.3), this restriction can be circumvented to a certain degree. Note that an affine function $f : \mathbb{R}^n \to \mathbb{R}^n$ can always be written in the form $f(x) = Ax + b$, where A is an $n \times n$-matrix and b is a vector of size n.

Thirdly, the Lyapunov function template is of the form $V = x^T P x$, and therefore the computed Lyapunov functions are all quadratic. Again, the sums-of-squares decomposition allows for more general classes of Lyapunov functions, most notably higher-degree polynomials.

3.5.1 Linear Matrix Inequalities

The first step of a Lyapunov function computation involves the formulation of the Lyapunov function conditions as a *linear matrix inequality (LMI)*. To formally define LMIs, we first need to define the *positive semidefiniteness operator* on matrices.

Definition 3.19 (Semidefiniteness)**.** A function $f : \mathbb{R}^n \to \mathbb{R}$ is called *positive semidefinite*, if $f(x) \geq 0$ for all x. A function f is called *negative semidefinite* if $-f$ is positive semidefinite. A matrix M is positive (negative) semidefinite, if the quadratic function $f(x) = x^T M x$ is positive (negative) semidefinite. This is denoted by $M \succeq 0$ ($M \preceq 0$).

Semidefiniteness is a useful property: even though it is a global condition that holds for all x, semidefiniteness of a quadratic function $V = x^T P x$ can be checked without explicitly looking at any specific values of $V(x)$. Instead, semidefiniteness can be decided by looking at the matrix P only, for instance by computing its eigenvalues. If the real parts of all eigenvalues of P are non-negative, then P is positive semidefinite, and therefore also the function V (see Section 3.6.4).

Moreover, the semidefiniteness operator "$\succeq$" is a conic operator that can be used to define optimization problems that are still convex and can be solved with relative efficiency [Boyd & Vandenberghe, 2004]. One can informally see this as taking linear optimization problems and replacing the "$\geq$" operator on scalars with the "$\succeq$" operator on matrices. *Linear matrix inequalities* are a way of formalizing such optimization problems.

Definition 3.20 (Linear Matrix Inequality [Boyd *et al.*, 1994])**.** A *linear matrix inequality (LMI)* is an inequality of the form

$$F_0 + \sum_{i=1}^{m} \lambda_i F_i \succeq 0,$$

where $\lambda_i \in \mathbb{R}, 1 \leq i \leq m$ and the F_i, $0 \leq i \leq m$ are symmetric matrices in $\mathbb{R}^{n \times n}$. The λ_i are the *free variables* of the LMI. A *solution* to an LMI is a valuation of the free variables λ_i such that the inequality is fulfilled.

Multiple linear matrix inequalities can always be collapsed into one inequality. For instance, the inequalities $\lambda M_1 \succeq 0$ and $\lambda M_2 \succeq 0$, with $\lambda \in \mathbb{R}$, if they should be solved simultaneously, are equivalent to the LMI

$$\begin{bmatrix} \lambda M_1 & 0 \\ 0 & \lambda M_2 \end{bmatrix} \succeq 0.$$

Using this type of diagonal block structure, any family of LMIs can be viewed as a single LMI. Therefore, there is no need to differentiate between systems of LMIs and a single LMI. Both terms will be used interchangeably from this point on.

Instead of writing down the left hand side of an LMI as sums of products of a scalar and a matrix, shorthand notations are also frequently used. For instance, the inequality $A^T X + XA \succeq 0$, where X is a symmetric matrix of unknowns, and A is a (not necessarily symmetric) matrix with fixed entries, can be expanded into an LMI by extracting all entries of X as scalars λ_i. Since this matrix product notation is much more concise, it will occur frequently in the scope of this thesis.

3.5.2 LMIs for Stability Proofs

To prove global asymptotic stability, the Lyapunov conditions of Theorems 3.3, 3.4, 3.5, and 3.7 can be formulated as linear matrix inequalities. For the non-hybrid case of Theorem 3.3 the LMI constraint system look as follows.

Theorem 3.8 (LMI for Asymptotic Stability of Linear, Convex Differential Inclusions [Boyd *et al.*, 1994, p.62])**.** Let $f_i(x) = A_i x$ be a set of linear functions, forming a convex differential inclusion $\dot{x} \in F(x)$. Denote as I the $n \times n$ identity matrix. Then, $\dot{x} \in F(x)$ is globally asymptotically stable if and only if there exists an $\alpha > 0$, such that the following LMI has a solution:

$$\text{Find } P \in \mathbb{R}^{n\times n} \text{ and } \alpha \in \mathbb{R}, \text{ such that}$$
$$\begin{array}{rcl} P - \alpha I & \succeq & 0 \\ \text{for all } i : A_i^T P + P A_i + I & \preceq & 0 \end{array}$$

The variable α has been introduced to model the class K^∞ function f_1 from Theorem 3.3. This is equivalent to setting $f_1(||x||)$ to $x^T(\alpha I)x = \alpha||x||^2$. The constraint $V(x) \leq f_2(||x||)$ is not needed here, since such a function f_2 will always exists for a quadratic V. Without loss of generality, the class K^∞ function f_3 has been set to $||x||^2$. No additional scalar variable is needed since Lyapunov functions form a convex cone, so that the solution can always be linearly scaled as desired.

Note that the f_i are all assumed linear, and not affine. This is no restriction, because a single convex differential inclusion including an affine, but not linear function f_i cannot be globally asymptotically stable in 0. One possible trajectory would always follow this f_i, and since $f_i(0) \neq 0$, it can not converge to 0.

As opposed to most other Lyapunov theorems, especially concerning hybrid systems, the existence of a solution to this LMI is both a necessary and a sufficient condition for global asymptotic stability. If the differential inclusion is in fact a differential equation $\dot{x} = f(x)$, then each P solving the LMI is the solution of the Lyapunov equation of the form $A^T P + PA = -Q$ for some positive definite matrix Q [Khalil, 1996].

Theorem 3.8 exploits the fact that, whenever a function V is a Lyapunov function for an entire family of differential equations $\dot{x} = f_i(x)$ (with respect to the same class $\mathcal{K}^\infty$ functions), it is also a Lyapunov function for the convex differential inclusion defined by the f_i. The Lyapunov Condition (2) from Theorem 3.3 is closed under convex combination of the dynamics. If $\left\langle \frac{dV}{dx}(x) \mid f_1(x) \right\rangle$ and $\left\langle \frac{dV}{dx}(x) \mid f_1(x) \right\rangle$ fulfill Condition (2), then so does $\left\langle \frac{dV}{dx}(x) \mid \lambda f_1(x) + (1-\lambda) f_2(x) \right\rangle = \lambda \left\langle \frac{dV}{dx}(x) \mid f_1(x) \right\rangle + (1-\lambda) \left\langle \frac{dV}{dx}(x) \mid f_2(x) \right\rangle$, if $0 \leq \lambda \leq 1$. Therefore, as long as the continuous variables evolve according to some convex combination of the functions f_i, the Lyapunov function V will decrease along all trajectories, guaranteeing GAS. For this reason, it is sufficient to show that Condition (2) is fulfilled only for the finitely many "corner point" functions f_i.

In the scope of a hybrid automaton, LMI methods can also be applied to *affine* convex differential inclusions, that is, differential inclusions for which the "corner point" dynamics are of the form $\dot{x} = A_i x + b_i$. Note that this also includes dynamics with constant right hand sides, which occur often in hybrid system models, for example when modeling timeouts or saturation. This trick consists of the introduction of another system variable, which represents the constant 1. In other words, we define the vector

$$\bar{x} = \begin{bmatrix} x \\ 1 \end{bmatrix},$$

and use this new vector $\bar{x}$ in the LMI. In this manner, a differential equation $\dot{x} = Ax+b$ is simply represented as

$$\dot{\bar{x}} = \begin{bmatrix} A & b \\ 0 & 0 \end{bmatrix} \bar{x},$$

and the parametric term $\bar{x}^T \bar{P} \bar{x}, \bar{P} \in \mathbb{R}^{(n+1)\times(n+1)}$, can be used to represent functions with quadratic, linear, and constant parts.

In order to extend these results for the computation of Lyapunov functions to hybrid automata, one major restriction must be overcome: LMI conditions are always global, or, in other words, positive semidefiniteness is a condition on *all* values of x. When dealing with hybrid automata, guards and invariants usually restrict the set of states where something specific can happen. For instance, there is no need to prove Lyapunov conditions for states where the invariant does not hold, as no continuous evolution is possible from such a state anyway. Therefore, a method for expressing Lyapunov conditions only for *some* states is needed. This gap between globality and locality is bridged by the $\mathcal{S}$-procedure, [Yakubovich, 1977]. The $\mathcal{S}$-procedure is a relaxation that can be used to represent local positiveness conditions by global positiveness conditions.

Theorem 3.9 ($\mathcal{S}$-procedure [Boyd *et al.*, 1994, p.23])**.** Let $f_i : \mathbb{R}^n \to \mathbb{R}, 0 \leq i \leq m$. If there exist $\lambda_i \geq 0, 1 \leq i \leq m$, such that

$$\forall x \in \mathbb{R}^n : f_0(x) - \sum_{i=1}^{m} \lambda_i f_i(x) \geq 0,$$

then, for all $x \in \mathbb{R}^n$,

$$\forall x \in \{y \in \mathbb{R}^n \mid \forall i, 1 \leq i \leq m : f_i(y) \geq 0\} : f_0(x) \geq 0$$

Moreover, if $m = 1$, and the function f_1 is quadratic, then the two conditions are equivalent.

The $\mathcal{S}$-procedure allows the replacement of a locality condition, given as a number of positiveness conditions $\forall i, 1 \leq i \leq m : f_i(y) \geq 0$, by a subtractive term. If we want to show positiveness of some function f_0 on a set where all these positiveness conditions hold, then we can also show global positiveness of f_0 minus this additional term instead. In the simplest case, when there is just a single quadratic positiveness condition $f_1 \geq 0$, this transformation is exact.

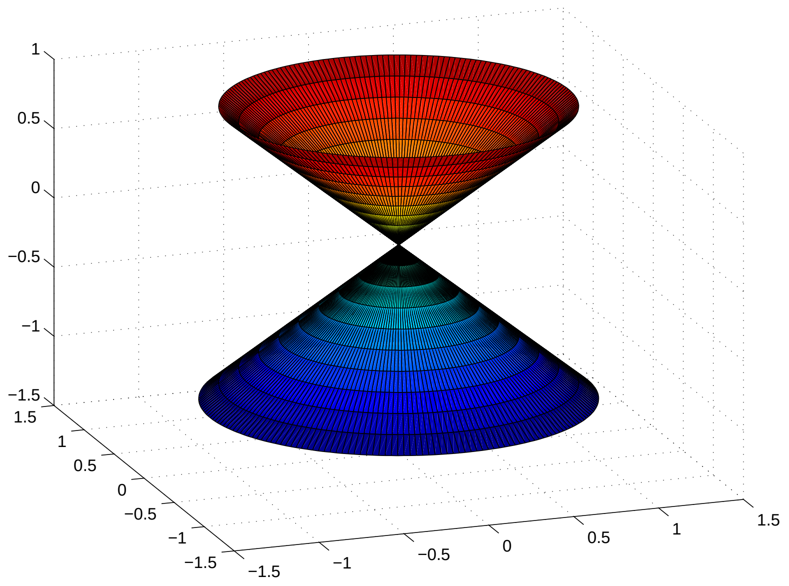
Regions of the state space that can be represented by such a single condition in this exact case (combined with the substitution trick described above) are *conic sections* (see Fig. 3.16), including conic shapes and ellipsoids. For instance, if $f_0(x) \geq 0$ is to hold for all x with $||x|| \leq 1$, then this region can be represented by a quadratic function $f_1 = 1 - \sum_{i=1}^{m} x_{(i)}^2$. The local condition

$$\forall x \in \left\{ y \in \mathbb{R}^n \;\middle|\; 1 - \sum_{i=1}^{m} x_{(i)}^2 \geq 0 \right\} : f_0(x) \geq 0$$

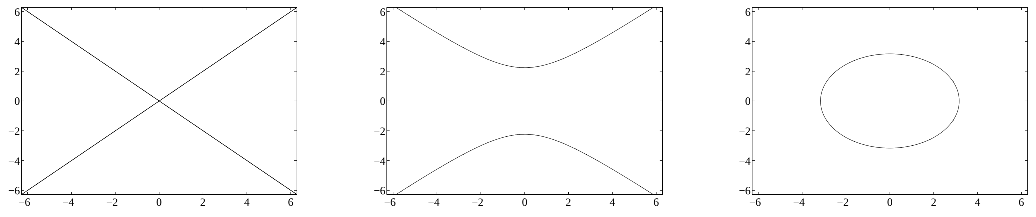
is then equivalent to the global condition that there exists a $\lambda \geq 0$ such that

$$\forall x \in \mathbb{R}^n : f_0(x) - \lambda \left(1 - \sum_{i=1}^{m} x_{(i)}^2 \right) \geq 0.$$

This constraint can then be expressed as an LMI that is equivalent to the original problem, if the function f_0 is also quadratic.



(a) Three-dimensional Double Cone



(b) Central Vertical Cut (c) Off-Center Vertical Cut (d) Horizontal Cut

Figure 3.16: Quadratic $\mathcal{S}$-procedure: Possible Conic Sections in Three Dimensions

For other region shapes, the functions f_i must be chosen to over-approximate the region. In this case, it becomes useful to use more than one function f_i, with the region itself lying in the intersection of their positive sets (see Figure 3.17).

Since the λ_i are free variables eventually to be assigned valuations by the LMI solver, using multiple functions f_i gives extra degrees of freedom for finding a solution. One way

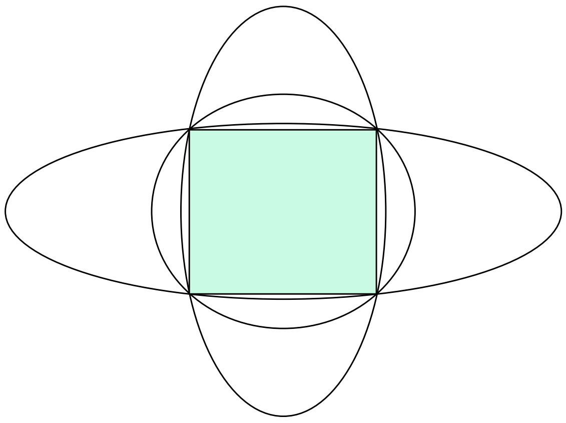


Figure 3.17: Multiple Quadratic $\mathcal{S}$-procedure Functions for a Single Set

to view this is to interpret the sum $\sum_{i=1}^{m} \lambda_i f_i(x)$ as a linear combination of the shapes represented by the functions f_i. Through the free variables λ_i, the solver can effectively select any over-approximation of the region that is a linear combination of the f_i. In theory, it is therefore helpful to give as many f_i as possible, as this can only increase the size of the the solution space by allowing possibly better approximations of a region. In practice, one must however take care not to overburden the solver with the extra variables λ_i, and not to introduce numerical instability by including badly conditioned matrices. How to arrive at quadratic $\mathcal{S}$-procedure functions f_i for various shapes of regions is discussed in detail in [Pettersson, 1999]. For instance, general polytopes can be represented by a number of quadratic functions, each representing the product of two constraining hyperplane constraints. Box-shaped regions can be represented by a single function representing an ellipsoid over-approximation.

With the help of the $\mathcal{S}$-procedure, LMIs for Lyapunov function computation for hybrid automata can now be stated. The $\mathcal{S}$-procedure is used to exploit the information given by the guards and invariants of the hybrid automaton, and restrict conditions appropriately.

Theorem 3.10 (LMI for Asymptotic Stability of Hybrid Automata with Convex, Affine Differential Inclusions [Pettersson, 1999])**.** Let $\epsilon > 0$ and let H be a hybrid automaton. Let the $Q_j^m \in \mathbb{R}^{(n+1)\times(n+1)}$ be matrices, such that for every mode $m \in \mathcal{M}$,

$$x \in \mathit{Inv}(m) \implies [x, 1]\, Q_j^m \begin{bmatrix} x \\ 1 \end{bmatrix} \geq 0,$$

and let the $R_j^e \in \mathbb{R}^{(n+1)\times(n+1)}$ be matrices, such that for every transition $e \in \mathcal{T}$ with guard set G

$$x \in G \implies [x, 1]\, R_j^e \begin{bmatrix} x \\ 1 \end{bmatrix} \geq 0.$$

Assume that $\mathit{Flow}(m)$ is a convex, affine differential inclusion for each m, spanned by a set of functions $f_i^m(x) = A_i^m x + b_i^m$. Furthermore, assume that, for each transition e, the associated update function is affine with $U(x) = A^e x + b^e$. Define

$$\tilde{I} = \begin{bmatrix} I & 0 \\ 0 & 0 \end{bmatrix},$$

where I is the $n \times n$ identity matrix. If the LMI problem

Find $P_m \in \mathbb{R}^{(n+1)\times(n+1)}$ and $\alpha, \beta, \mu_j^m, \nu_j^m, \eta_j^m, \vartheta_j^e \in \mathbb{R}$, such that

$$\begin{aligned}
\alpha - \epsilon &\succeq 0 && (3.1)\\
\beta - \epsilon &\succeq 0 && (3.2)\\
\text{for all } m, j : \mu_j^m, \nu_j^m, \eta_j^m &\succeq 0 && (3.3)\\
\text{for all } e, j : \vartheta_j^e &\succeq 0 && (3.4)\\
\text{for all } m : P_m - \sum_j \mu_j^m Q_j^m - \tilde{I} &\succeq 0 && (3.5)\\
\text{for all } m : P_m + \sum_j \nu_j^m Q_j^m - \beta \tilde{I} &\preceq 0 && (3.6)\\
\text{for all } m, i : \begin{bmatrix} (A_i^m)^T & 0 \\ (b_i^m)^T & 0 \end{bmatrix} P_m + P_m \begin{bmatrix} A_i^m & b_i^m \\ 0 & 0 \end{bmatrix} + \sum_j \eta_j^m Q_j^m + \alpha \tilde{I} &\preceq 0 && (3.7)\\
\text{for all } e = (m_1, m_2, G, A^e x + b^e) : & && \\
P_{m_1} - \begin{bmatrix} (A^e)^T & 0 \\ (b^e)^T & 1 \end{bmatrix} P_{m_2} \begin{bmatrix} A^e & b^e \\ 0 & 1 \end{bmatrix} - \sum_j \vartheta_j^e R_j^e &\succeq 0 && (3.8)
\end{aligned}$$

has a solution, then the hybrid automaton H is GAS.

Theorem 3.10 is a combination of a result for discontinuous systems with differential inclusions [Cortés, 2008, p.63] and the LMI theorem for hybrid systems with differential equations [Pettersson, 1999, p.97]. The LMI Constraints (3.1) to (3.4) are all one-dimensional and therefore only non-negativeness conditions on $\mathbb{R}$. Constraints (3.5) to (3.8) directly correspond to the conditions in Theorem 3.5. Here, the matrices Q_j^m and R_j^e are $\mathcal{S}$-procedure terms representing invariants and guards, respectively. If a solution to the LMI is found, then the local Lyapunov function for each discrete mode m is given by

$$V_m(x) = [x, 1]\, P_m \begin{bmatrix} x \\ 1 \end{bmatrix}.$$

The family of functions V_m forms the global Lyapunov function, proving global asymptotic stability as per Theorem 3.5.

Additionally, LMI problems can also contain linear *objective functions*, linear functions on the free variables (both scalars and entries of matrix variables) that are to be maximized/minimized. Possible uses for these functions are the computation of the best possible convergence rate estimate (see Section 3.5.4) or optimization wih respect to the variables of certain LLFs for the decompositional methods of Chapter 4.

Note that the matrix $\tilde{I}$ is used here to represent the class K^∞ function $f(||x||)) = ||x||^2$. As per Remark 3.4, this means that the system is not only GAS, but globally exponentially stable if a solution is found. Therefore, the method cannot be used to prove GAS for systems that converge slower than exponentially. However, such systems can often be converted into exponentially stable systems by assuming overall bounds on the continuous system states. This will also be illustrated for the detailed example presented in Section 3.7.

If only GAS with respect to a sub-set of variables $\mathcal{V}'$ is of interest, then the matrix $\tilde{I}$ in Constraints (3.5), (3.6) and (3.7) should be replaced by a matrix containing a 1 in each diagonal entry corresponding to a variable in $\mathcal{V}_m$.

For modes m with $0 \in \mathit{Inv}(m)$, the LLF cannot contain any constant or linear parts, as $V_m(0)$ is required to be zero. For such modes, the last row and column of P_m can therefore be set to zero in beforehand, reducing the number of free variables. It also advisable to do this for numerical reasons, as SDP solvers will usually not be able to return an exact zero solution for these values. Therefore, the returned "solution" for P_m would be slightly invalid in this case.

An alternative to the representation of right hand sides of differential inclusions by polytopic sets is the so called *full-block $\mathcal{S}$-procedure*, which can represent differential inclusions with ellipsoid right hand sides. Doing so results in an LMI which is comparable in size to an LMI for two corner points for the polytopic set. Depending on the differential inclusion, it can be beneficial to choose this approach over the one presented above. An application of this alternative approach can for instance be found in [Donkers *et al.*, 2009].

3.5.3 The Sums-of-squares Decomposition

The LMI problem in Theorem 3.10 can be used to compute piecewise quadratic Lyapunov functions for piecewise affine hybrid systems. If the system in question is not piecewise affine, or if a non-quadratic Lyapunov function is required to prove stability of a system, then this method cannot be used directly. However, it is possible to transform the problem of computing of such Lyapunov functions into an LMI problem with the help of the *sums-of-squares (SOS) decomposition* [Papachristodoulou & Prajna, 2002]. The basic idea is to substitute higher degree polynomial (or even transcendental) terms in the Lyapunov constraints by low-degree polynomial terms, such that the existence of a solution for the transformed system implies the existence of a Lyapunov function for the original system.

For instance, if we want to show that the function

$$f(x) = x^4 - x^3 + x^2$$

is non-negative, we can substitute x^2 with a new variable z, obtaining a new function

$$\tilde{f}(x,z) = z^2 - xz + x^2 = [x,z]P \begin{bmatrix} x \\ z \end{bmatrix}$$

with

$$P = \begin{bmatrix} 1 & -0.5 \\ -0.5 & 1 \end{bmatrix}.$$

It is known that the semidefiniteness of P implies the non-negativity of f [Parrilo, 2003]. Therefore, this type of substitution allows us to reduce the Lyapunov conditions on a non-quadratic function to a semidefiniteness problem of a matrix, making the positiveness problem above amenable for solution with LMI methods. Note that, for the decomposition of polynomials into sums of squares of polynomials, the opposite implication is not true. In some special cases, namely polynomials in one variable, any quadratic polynomials (trivially, since they do not need any substitution at all), and quartic polynomials in two variables, equivalence holds. This question of equivalence for the more general case of rational functions is also known as *Hilbert's 17th problem* [Reznick, 2000].

The sums-of-squares decomposition can also be employed in conjunction with the $\mathcal{S}$-procedure, so that it is possible to represent different region shapes by non-quadratic functions. Furthermore, $\mathcal{S}$-procedure-like additive terms can be used to exploit the implicit equality constraints induced by the substitution. For instance, the additional constraint that $z = x^2$ in the above example can be modelled by introducing an unbounded new variable μ, and augmenting the constraint $P \succeq 0$ to:

$$\begin{bmatrix} 1 & -0.5 & 0 \\ -0.5 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \mu \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -0.5 \\ 0 & -0.5 & 0 \end{bmatrix} \succeq 0$$

Some SDP solvers (like SeDuMi [Romanko *et al.*, 1999]) also support equality constraints directly, making this construction unnecessary. See Section 3.6.1 for a detailed list of software tools.

3.5.4 Estimating Convergence Times

While the existence of a Lyapunov function for a dynamic system is sufficient to guarantee global asymptotic stability, this proof does not directly provide a bound on the convergence rate of the system. In practice, one is usually not only interested in convergence to some point (as guaranteed by the attractivity property), but the system state should also be within some distance to the equilibrium in a certain time. It is possible to arrive at estimates for the convergence rate of the system by looking at the convergence rate of the Lyapunov function. In a nutshell, one can view the behavior of the Lyapunov function over time as a trajectory of a one-dimensional dynamic system, the only dimension being the Lyapunov function value. This simple system is globally asymptotically stable only if this is the case for the original system. A similar parallel can be established between the convergence rates of the two systems. This is done by characterizing sub-sets of the state space of the original system by their Lyapunov function values. Because the convergence rate of a Lyapunov function can be easily encoded in LMI problems, doing so provides a convenient method for obtaining upper bounds on convergence times to designated target sets.

It is possible to directly obtain an upper bound on the convergence rate of the system from a solution to the LMI problem (3.1)-(3.8). This is due to the facts that such a solution guarantees exponentially fast convergence, and that it is possible to derive this exponential convergence rate. Define $\alpha' = \alpha/\beta$. Together with Constraint (3.7), we then know that

$$\dot{V}_m(x) \leq -\alpha||x||^2 \leq -(\alpha/\beta)V_m(x) = -\alpha' V_m(x)$$

This inequality can be viewed as a linear differential inclusion in a single variable V. Together with Condition (3.8), which guarantees non-increasingness of the function V V during mode switches, the solution of this differential inclusion implies that for any trajectory $x(t)$:

$$V(x(t)) \leq e^{-\alpha' t} V(x(0)).$$

Therefore, V converges exponentially to 0, with rate α'. However, this rate α' can be somewhat conservative, especially if the dynamics are non-linear. Essentially, the rate α' represents the slowest exponential convergence rate of all modes of the hybrid system.

Better convergence rate estimates can usually be obtained by encoding the inequality $\dot{V} \leq -\alpha' V$ directly into the LMI. To achieve this, Constraint (3.7) can be replaced by

$$\text{for all } m, i : \begin{bmatrix} (A_i^m)^T & 0 \\ (b_i^m)^T & 0 \end{bmatrix} P_m + P_m \begin{bmatrix} A_i^m & b_i^m \\ 0 & 0 \end{bmatrix} + \sum_j \eta_j^m Q_j^m + \alpha' P_m \preceq 0 \quad (3.9)$$

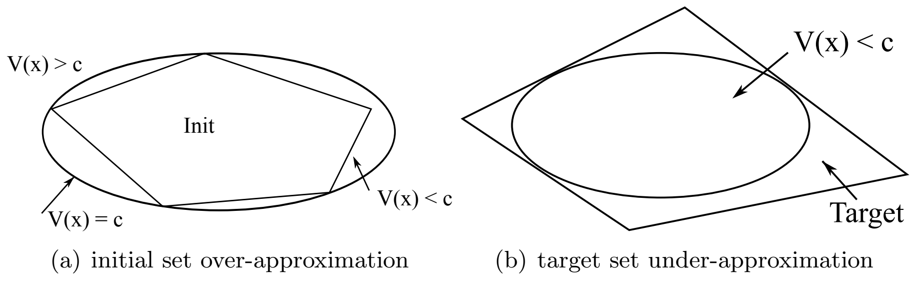
It is also possible to select a suitable objective function for the convex optimization problem corresponding to the LMI, such that the matrices P_m guaranteeing the optimal convergence rate estimates can be found. This is achieved by forcing the optimization algorithm to maximize the variable α' [Pettersson, 1999].

Note that, even in case the optimal value of α' is identified, the actual convergence rate of the system might be much higher (and the convergence time lower) than the computed bound, if it is impossible to stay in the slowest mode for all t. The switching logic is not exploited when computing the upper bound, and generally a worst case approach is taken.

In order to convert the convergence rate information given by the variable α' into convergence time information for a given set of initial and target states, some simple additional computation steps are necessary. First, we need an upper bound on the Lyapunov function values for the set of initial states *Init*. This problem is equivalent to identifying the minimal Lyapunov function value c, such that

$$x \in \mathit{Init} \implies V(x) \leq c.$$

If the function V is quadratic, non-negative and zero only at the origin, the sub-level sets of V are ellipsoids and therefore convex. If *Init* is a bounded polytope, then checking this inequality for a given c can simply be achieved by checking whether $V(x_i) < c$ for



(a) initial set over-approximation (b) target set under-approximation

Figure 3.18: Representing Initial and Target Sets by Lyapunov Function Contour Lines

all corner points x_i of *Init* (see Figure 3.18(a)). This turns the problem of finding the minimal c into a simple one-dimensional optimization problem.

For the target set *Target*, we need to make a slightly more complicated computation. We need to identify a scalar c, such that

$$V(x) \leq c \implies x \in \mathit{Target}.$$

This problem is equivalent to finding a suitable ellipsoid that lies entirely inside the target region (see Figure 3.18(b)). For quadratic Lyapunov functions and bounded polytopic target sets, the problem of finding the such a maximum value of c can also be cast into an optimization problem. In this case, there is an LMI representation of the problem (see [Boyd *et al.*, 1994, p.70] for a detailed discussion).

If we know an upper bound $c_1 > 0$ for $V(x), x \in \mathit{Init}$, and a c_2, such that $V(x) \leq c_2 \implies x \in \mathit{Target}$, this allows us to bound the maximum time Δ it can take for any trajectory starting in *Init* to converge to *Target* by the inequality

$$\Delta \leq \frac{1}{\alpha'} \ln\left(\frac{c_1}{c_2}\right).$$

If tighter estimates for convergence times are desired, one can, for instance, exploit dwell time information (i.e., the time spent in each mode by the trajectories), or conduct reachability analysis, viewing the time as another system variable with differential equation $\dot{t} = 1$. However, the latter approach does not have the generality of rate-based analysis, since it only allows the computation of convergence time with respect to *one* initial set and *one* target set, whereas the computation of a convergence rate allows the simple computation of convergence times for all initial and target sets, as long as the possible Lyapunov function values on these sets can be bounded, as discussed above.

3.6 Numerical Solution of LMI Problems

Having obtained LMI representations of Lyapunov function computation problems, as given above, the remaining step is the solution of these LMI problems. Since LMI problems always have a convex solution set, descent-like methods can be used to find such a solution. This section gives an overview of such techniques and the possible pitfalls one must seek to avoid. First, we give an overview of the different software packages for LMI specification and solution that were available at the time of writing. Then, a brief summary of the methods employed by these tools is given. Finally, practical issues relating to the numerical stability of these methods are discussed. In particular, we focus on how to assure sufficient accuracy of the results, which is crucial if the results of the computations are to be used for automatic verification.

3.6.1 Available Software

LMI problems can be solved with help of so-called *semidefinite programming (SDP)* solvers. A variety of free SDP tools have been developed, including the Matlab-based solvers SeDuMi and SDPT3 [Toh *et al.*, 1999], and the standalone solvers CSDP [Borchers, 1999], DSDP [Benson *et al.*, 2000] and SDPA [Yamashita *et al.*, 2003]. Most of these tools have seen updates in recent years. SDPA's sparse matrix input format continues to be supported by other tools, for example CSDP. Additionally, the commercial solvers PENSDP and PENNON [Kočvara & Stingl, 2003] are available. VSDP [Jansson, 2006] is a MATLAB wrapper around the solvers SDPT3 and SDPA that computes rigorous error bounds on the solution of an SDP problem, also taking floating point arithmetic errors into account. For Matlab, various tools for the direct specification of LMI exist, which in turn call one of the solvers listed above. These include the Multiparametric Toolbox (MPT) and Yalmip [Löfberg, 2004]. Figure 3.19 gives the current version and website of the different tools at the time of writing.

Tool	Version	Website
CSDP	6.10	`https://projects.coin-or.org/Csdp/`
DSDP	5.8	`http://www-unix.mcs.anl.gov/DSDP/`
MPT	2.6.3	`http://control.ee.ethz.ch/~mpt/`
PENSDP	2.2	`http://www.penopt.com/`
PENNON	0.9	`http://www.penopt.com/`
SDPA	7.3.1	`http://sdpa.indsys.chuo-u.ac.jp/sdpa/`
SDPT3	4.0	`http://www.math.nus.edu.sg/~mattohkc/sdpt3.html`
SeDuMi	1.3	`http://sedumi.ie.lehigh.edu`
SOSTOOLS	2.03	`http://www.cds.caltech.edu/sostools/`
VSDP	0.1	`http://www.ti3.tu-harburg.de/jansson/vsdp/`
YALMIP	20101122	`http://control.ee.ethz.ch/~joloef/yalmip.php`

Figure 3.19: SDP Tools and Websites

3.6.2 Brief Outline of Semidefinite Programming Algorithms

The solution set to any LMI problem always forms a convex set. Therefore, standard convex optimization techniques can be employed to approximate a solution to an LMI that optionally is minimal with respect to some linear objective function. This section gives a brief overview about the general techniques used for solving convex optimization problems in general and LMI/SDP problems in particular. For a detailed and rigorous discussion we refer to [Boyd & Vandenberghe, 2004] and [Nesterov & Nemirovskii, 1994]. The methods employed by any individual solver software (see Figure 3.19) might also vary slightly and contain further optimizations. However, the purpose of this section is not an exhaustive discussion, but a general overview. Information about the actual algorithm used for a particular solver is usually available in its documentation.

In a nutshell, convex optimization methods are descent-based. A sequence of points in the state space is computed, such that the value of the objective function and/or the degree of violation of the LMI constraints decreases in every step. As opposed to linear optimization, SDP solvers usually employ *interior point methods* instead of surface-based methods. This means that the sequence of points that is computed will not only lie on the solution set's surface, but also pass through its interior. It is notable that interior point methods result in polynomial time complexity with respect to the size of the constraint system [Nesterov & Nemirovskii, 1994]. There are algorithms for which time complexity for a single iteration step lies in $O(n^3)$ with respect to the number of variables as well as with respect to the number of constraints, while the number of iteration steps can be expected to grow logarithmically in either case.

An important concept for the solution is *Lagrangian duality*. The *Lagrangian* of an optimization problem is a function which is obtained from its objective function by adding weighted terms representing the degree of violation of the constraints. These weights are the so-called *Lagrangian multipliers.* With the help of this concept, a *dual problem* for any SDP problem can be constructed. The original (primal) problem minimizes the objective function with respect to the constraints, and is of the form

$$\begin{array}{rl} \text{minimize} & c^T x \\ \text{with respect to} & M_0 + \sum_{i=1}^{n} x_{(i)} M_i \succeq 0, \end{array}$$

where the M_i are matrices over the reals. The dual problem has the Lagrangian multipliers as unknowns and maximizes the infimum of the Lagrangian over all x with respect to a positiveness condition over the multipliers. It is of the form

$$\begin{array}{rl} \text{maximize} & tr(M_0 Y) \\ \text{with respect to} & tr(M_i) + c_{(i)} = 0 \text{ for all } 1 \leq i \leq n \\ & Y \succeq 0 \end{array}$$

where Y is a matrix representing the Lagrangian multipliers. As long as the solution set to the primal problem has a non-empty interior, *strong duality* holds, that is, the solutions of the primal and dual problems coincide. Moreover, feasible points of the primal problem (i.e., point fulfilling the constraints) always give an upper bound on the optimal value of $c^T x$, while feasible points of the dual problem always give a lower bound. Therefore, by solving the primal and dual problems simultaneously, accuracy bounds on the results can be derived and used as stopping conditions. The difference between the current upper and lower bounds is called the *duality gap* and can be used to measure the quality of a solution returned by the SDP solver.

However, the computation of a suitable descent direction for the primal and dual problems is in general not easy, since the optimization problem is non-linear. To this end, the solution of systems of linear equations are required, for instance, to do the equivalent of a *Newton step*. Progress is unfortunately not necessarily guaranteed if the problem is badly conditioned. In particular, this occurs if the matrix representing the linear equation system turns out to be singular or near-singular. The likelihood of such problems tends to increase with the dimensionality of the optimization problem. Once a direction has been established, a *line search* is conducted in that direction, determining the next point of the sequence. Essentially, this involves picking the point on a line in the descent direction that minimizes the objective function value (or maximizes the Lagrangian). If it is not possible to find a suitable new point with a lower objective function value, then the algorithm will terminate unsuccessfully.

3.6.3 Numerical Issues

Due to the nature of SDP algorithms and the fact that all computations rely on floating point numbers, there are some inherent problems that must be overcome, if one wants to use these tools for rigorous verification. We will now outline these problems and mention how they are dealt with in the scope of this thesis.

Successful termination does not always mean a useful result: SDP algorithms usually terminate when the infeasibility measures of the primal and dual problems and the duality gap are all sufficiently small. Here, "sufficiently small" means that the error is below a certain threshold relative to the computed solution. Depending on the numerical conditioning of the problem, this may or may be not sufficient. For instance, if a computed solution contains both very large (e.g., 10^6) and very small values (e.g., 10^{-6}), then a relative accuracy that would normally be deemed acceptable (e.g., 10^{-12}) might mean that the errors on the small values might still be of magnitude $10^6 \cdot 10^{-12} = 10^{-6}$ which is sometimes not enough to produce reliable results. Such problems can sometimes be avoided by re-scaling the problem (i.e., eliminating either the high or the low values). Nevertheless, it is always advisable to double-check all solutions that are returned by an SDP solver. Such *a posteriori* checking is described in Section 3.6.4.

Termination with failure does not always mean a useless result: On the other hand, SDP solvers can report failure for many reasons. If the problem is not found to be downright infeasible, then the solvers will often be able to compute a solution with reduced accuracy, despite reporting failure (i.e., the result is interpreted as a failure because a desired accuracy could not be reached). If this is the case, the result might still be salvaged after double-checking whether all of the LMI conditions are satisfied.

Implicit equality constraints can cause problems: If, in a hybrid system, two modes are connected by two transitions, one in each direction, and these transitions have the same guard set and no discrete update, then the corresponding Lyapunov functions are theoretically required to be equal on the guard set. This is mirrored in the LMI by two constraints modeling inequality conditions in both directions, resulting in an implicit equality condition. Naturally, numerical algorithms will usually be unable to exactly fulfill both these constraints, as this would require floating point numbers that are identical. For such cases, a special treatment is needed to keep the results numerically stable. See Section 4.5 on how to deal with this problem. A special case occurs when computing a Lyapunov function $\tilde{x}^T P \tilde{x}$ for a mode whose invariant contains the origin. In this case, the Lyapunov function cannot contain a constant or linear part (as no class K^∞ functions could bound it from above in that case). Therefore, the entries in the last row and column of P can be set to 0 in beforehand. In this case, if no convergence rate estimate is needed, even Condition (3.2) can theoretically be dropped, since the existence of a suitable value of β can always be guaranteed.

Reduction of the number of variables might improve numerical stability: In theory, the addition of further $\mathcal{S}$-procedure terms to an LMI can only lead to a larger solution space. In practice, however, the size of the LMI (both in terms of the number of constraints and the number of variables) is an important consideration. Additional $\mathcal{S}$-procedure terms (especially if they contain ill-conditioned matrices) carry the risk of causing numerical problems which make it impossible to find any solution at all. Therefore, it is useful to remove all non-necessary terms from the LMI. In particular, this includes terms carrying the implicit equality constraint that their multiplier is zero.

Large, irregular LMI problems are much more likely to attract numerical problems: Small-scale LMI problems can be solved reliably, unless they are very ill-conditioned. Larger, irregular LMI problems, especially those corresponding to Lyapunov function computation for a hybrid system with many modes, will however run an increasing danger of encountering one of the above problems. This makes LMIs like the one given in Theorem 3.10 hard to solve in practice, once the hybrid system becomes more complex. Direct computations for hybrid systems with more than a handful of discrete states are difficult, unless their structure is very regular. This is a main motivation for the decompositional methods in Chapter 4, which lead to small, local LMIs.

If the SDP solver fails, then there is little information on what caused the problem: Unless the problem was found to be infeasible by the solver, the failure of an SDP solver does not give further information on how to remedy this situation, for instance by amending the controller behavior. Naturally, the larger the LMI problem, the more problematic diagnosis becomes. It could be the case that the problem is badly scaled, containing both very large and very small matrix entries, or that the problem is infeasible or marginally feasible to begin with. Again, decompositional methods as described in Chapter 4 are helpful, since local computations support problem detection.

Optimal solutions with respect to linear objective functions will lie on the boundary of the feasible set: Since SDP solvers numerically approximate solutions to the SDP, this means that the "solution" returned under these circumstances will often lie outside the actual feasible set, but just barely. To obtain a real solution that robustly fulfills the LMI constraints, additional measures are needed. This problem is also addressed by the decompositional methods in Chapter 4, by computing a set of Lyapunov function instead of only one function. This allows for the easy computation of robust solutions to the corresponding LMI by averaging several "extremal" solutions.

3.6.4 Checking SDP Results

As discussed in the previous section, SDP algorithms are not always entirely robust in the sense that also false positives can be returned, that is, "solutions" which in fact do not fulfill the LMI constraints. This section discusses means of countering this problem.

Since the results of an SDP computation provide valuations for all free variables in the LMI, a simple approach is a posteriori checking of the constraints. By inserting the values returned by the software, and multiplying out the matrices for each LMI constraint, one arrives at a single matrix that is supposed to be positive (or negative) semidefinite. We will now give methods for checking definiteness of matrices. First, we need to define the *minors* of a matrix, which can be used for an equivalent and robustly checkable formulation of (semi-)definiteness.

Definition 3.21 (Minors)**.** Let $M \in \mathbb{R}^{n \times n}$ be a matrix. Let $I \subset \{1, \dots, n\}$ be a non-empty set of column/row indices for M. The determinants of all matrices that can be obtained from M by keeping only the rows and columns in such a set I are called *principal minors* of M. If I is of the form $\{1, \dots, m\}, 1 \leq m \leq n$, then a principal minor is called *leading principal minor*.

We now give a basic equivalence theorem that can be used to double-check the values returned by SDP solver software.

Theorem 3.11 (Checking Semidefiniteness [Bhatia, 2007])**.** Let $M \in \mathbb{R}^{n \times n}$ be a symmetric matrix. Then, the following statements are equivalent.

1. $M \succeq 0$.
2. All eigenvalues of M are non-negative.
3. All principal minors of M are non-negative.

Therefore, a posteriori checking of semidefiniteness conditions involves either the numeric computation of matrix eigenvalues, which can be done efficiently up to machine precision, or the computation of the principal minors, which can be done in exact arithmetic, but is only feasible for very small matrices. This can be done separately for each LMI constraint, so that the size of matrices to be considered generally depends linearly on the size of the continuous state space (after the sums-of-squares transformation, if applicable).

For example, LMI front-end YALMIP provides a *checkset* routine which automatically carries out a semidefiniteness test on all matrices obtained in this manner. This also includes the scalar constraints, which result in 1×1 matrices. For all such matrices, the smallest eigenvalue is computed, and if it is non-negative, then the matrix is positive semidefinite (modulo possible inaccuracies in the eigenvalue computation, which are usually negligible).

If this is not the case, but the negative eigenvalues are very close to zero, then the problem can be remedied. This occurs relatively frequently due to the inherent inaccuracies caused by the numerical approach. If the offending values are related to $\mathcal{S}$-procedure constraints in the LMI from Theorem 3.10, then we can simply set them to zero and re-run the solver. If they are related to the scalar variables α or β, then α or β will be slightly smaller than ϵ, which is not a problem, as long as either variable remains strictly positive. In this case no additional measures are needed. If we obtain small negative eigenvalues for any of the Constraints (3.5), (3.6), or (3.7), then this is also not a major problem, as we have used matrices $\tilde{I}$ or multiples thereof to robustify the LMI problem. A slightly negative eigenvalue therefore only means possible inaccuracies in the values of α and β, which can affect possible convergence rate estimates, but not the overall stability result. If such inaccuracies occur in Constraint (3.8), however, some care must be taken. Here, a negative eigenvalue close to zero means that there is a slight increase of the Lyapunov function value upon switching. To remedy this, another positive slack variable multiplied with matrix $\tilde{I}$ could be introduced, or the Lyapunov functions for the two offending modes could explicitly be set equal on the switch set (see, for example, [Johansson & Rantzer, 1998] or the discussion in Section 4.5.2).

3.7 Example

Consider the hybrid system from Example 3.8. This system was shown to be GAS. However, it does not converge exponentially fast, as the convergence rates for the modes m_1 and m_3 do not depend linearly on the distance to the equilibrium (i.e., the class K^∞ cannot be linear as required by the LMI approach). In order to convert this system in to one that is exponentially stable, we need to add a global invariant to the system which is conjoined with all mode invariants and guards. Since, in reality, such bounds will almost always exist, this is not a major drawback. In this case, we choose the global invariant

$$-20 \leq v \leq 20 \wedge -500 \leq x \leq 500.$$

The mode dynamics are represented as

$$\dot{x} \in convex(\{A_1^{m_i} x + b_1^{m_i}, A_2^{m_i} x + b_2^{m_i}\})$$

for mode m_i with the following matrices and vectors:

$$A_1^{m_1} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, b_1^{m_1} = \begin{bmatrix} 0 \\ 1.425 \end{bmatrix}$$

$$A_2^{m_1} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, b_2^{m_1} = \begin{bmatrix} 0 \\ 1.5375 \end{bmatrix}$$

$$A_1^{m_2} = \begin{bmatrix} 0 & 1 \\ -0.000975 & -0.0507 \end{bmatrix}, b_1^{m_2} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$A_2^{m_2} = \begin{bmatrix} 0 & 1 \\ -0.001025 & -0.0533 \end{bmatrix}, b_2^{m_2} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$A_1^{m_3} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, b_1^{m_3} = \begin{bmatrix} 0 \\ -1.95 \end{bmatrix}$$

$$A_2^{m_3} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, b_2^{m_3} = \begin{bmatrix} 0 \\ -2.05 \end{bmatrix}$$

For the invariant sets $Inv(m_1), Inv(m_2)$, and $Inv(m_3)$ for the three modes, the following $\mathcal{S}$-procedure matrices can be derived according to the procedure from [Pettersson, 1999] (rounded to six digits). Each matrix represents an ellipsoid set over-approximating the box-shaped mode invariants.

$$\begin{aligned} Q^{m_1} &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & -0.008899 & -0.111111 \\ 0 & -0.111111 & -0.388889 \end{bmatrix} \\ Q^{m_2} &= \begin{bmatrix} 0.000002 & 0 & 0 \\ 0 & -0.002222 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ Q^{m_3} &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & -0.008899 & 0.111111 \\ 0 & 0.111111 & -0.388889 \end{bmatrix} \end{aligned}$$

The guards for the four transitions e_1, e_2, e_3, and e_4 can be represented by the following matrices. Again, the matrices represent ellipsoids.

$$\begin{aligned} R^{e_1} &= \begin{bmatrix} -0.000002 & 0 & 0 \\ 0 & -0.5 & -3 \\ 0 & -3 & -17 \end{bmatrix} \\ R^{e_2} &= \begin{bmatrix} -0.000002 & 0 & 0 \\ 0 & -0.222222 & -3 \\ 0 & -3 & -39.5 \end{bmatrix} \\ R^{e_3} &= \begin{bmatrix} -0.000002 & 0 & 0 \\ 0 & -0.5 & 3 \\ 0 & 3 & -17 \end{bmatrix} \\ R^{e_4} &= \begin{bmatrix} -0.000002 & 0 & 0 \\ 0 & -0.222222 & 3 \\ 0 & 3 & -39.5 \end{bmatrix} \end{aligned}$$

Recall that the variable sets $\mathcal{V}_{m_1}$ and $\mathcal{V}_{m_3}$ were chosen to contain only v, since x remains constant in these modes. Therefore, define the matrix J, which will be used to represent the class K^∞ functions for modes m_1 and m_3 as follows.

$$J = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

For mode m_2, $\mathcal{V}_{m_2}$ must contain both x and v, therefore the matrix

$$\tilde{I} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

takes this role. For this system, we can now formulate the LMI problem whose solution yields a GLF proving GAS with respect to the variable v (and exponentially fast convergence). The constraint system looks as follows:

Find $P_1, P_2, P_3 \in \mathbb{R}^{3\times 3}$ and
$\alpha, \beta, \mu^{m_1}, \mu^{m_2}, \mu^{m_3}, \nu^{m_1}, \nu^{m_2}, \nu^{m_3}, \eta_1^{m_1}, \eta_2^{m_1}, \eta_1^{m_2}, \eta_2^{m_2}, \eta_1^{m_3}, \eta_2^{m_3}, \vartheta^{e_1}, \vartheta^{e_2}, \vartheta^{e_3}, \vartheta^{e_4} \in \mathbb{R}$,
such that

$$
\begin{aligned}
\alpha - \epsilon &\succeq 0 \\
\beta - \epsilon &\succeq 0 \\
\mu^{m_1}, \mu^{m_2}, \mu^{m_3}, \nu^{m_1}, \nu^{m_2}, \nu^{m_3}, \eta_1^{m_1}, \eta_2^{m_1}, \eta_1^{m_2}, \eta_2^{m_2}, \eta_1^{m_3}, \eta_2^{m_3} &\succeq 0 \\
\vartheta^{e_1}, \vartheta^{e_2}, \vartheta^{e_3}, \vartheta^{e_4} &\succeq 0 \\
P_1 - \mu^{m_1} Q^{m_1} - J &\succeq 0 \\
P_2 - \mu^{m_2} Q^{m_2} - \tilde{I} &\succeq 0 \\
P_3 - \mu^{m_3} Q^{m_3} - J &\succeq 0 \\
P_1 + \nu^{m_1} Q^{m_1} - \beta J &\preceq 0 \\
P_2 + \nu^{m_2} Q^{m_2} - \beta \tilde{I} &\preceq 0 \\
P_3 + \nu^{m_3} Q^{m_3} - \beta J &\preceq 0 \\
\begin{bmatrix} (A_1^{m_1})^T & 0 \\ (b_1^{m_1})^T & 0 \end{bmatrix} P_1 + P_1 \begin{bmatrix} A_1^{m_1} & b_1^{m_1} \\ 0 & 0 \end{bmatrix} + \eta_1^{m_1} Q^{m_1} + \alpha J &\preceq 0 \\
\begin{bmatrix} (A_2^{m_1})^T & 0 \\ (b_2^{m_1})^T & 0 \end{bmatrix} P_1 + P_1 \begin{bmatrix} A_2^{m_1} & b_2^{m_1} \\ 0 & 0 \end{bmatrix} + \eta_2^{m_1} Q^{m_1} + \alpha J &\preceq 0 \\
\begin{bmatrix} (A_1^{m_2})^T & 0 \\ (b_1^{m_2})^T & 0 \end{bmatrix} P_2 + P_2 \begin{bmatrix} A_1^{m_2} & b_1^{m_2} \\ 0 & 0 \end{bmatrix} + \eta_1^{m_2} Q^{m_2} + \alpha \tilde{I} &\preceq 0 \\
\begin{bmatrix} (A_2^{m_2})^T & 0 \\ (b_2^{m_2})^T & 0 \end{bmatrix} P_2 + P_2 \begin{bmatrix} A_2^{m_2} & b_2^{m_2} \\ 0 & 0 \end{bmatrix} + \eta_2^{m_2} Q^{m_2} + \alpha \tilde{I} &\preceq 0 \\
\begin{bmatrix} (A_1^{m_3})^T & 0 \\ (b_1^{m_3})^T & 0 \end{bmatrix} P_3 + P_3 \begin{bmatrix} A_1^{m_3} & b_1^{m_3} \\ 0 & 0 \end{bmatrix} + \eta_1^{m_3} Q^{m_3} + \alpha J &\preceq 0 \\
\begin{bmatrix} (A_2^{m_3})^T & 0 \\ (b_2^{m_3})^T & 0 \end{bmatrix} P_3 + P_3 \begin{bmatrix} A_2^{m_3} & b_2^{m_3} \\ 0 & 0 \end{bmatrix} + \eta_2^{m_3} Q^{m_3} + \alpha J &\preceq 0 \\
P_{m_1} - \begin{bmatrix} (A^{e_1})^T & 0 \\ (b^{e_1})^T & 1 \end{bmatrix} P_{m_2} \begin{bmatrix} A^{e_1} & b^{e_1} \\ 0 & 1 \end{bmatrix} - \vartheta^{e_1} R^{e_1} &\succeq 0 \\
P_{m_2} - \begin{bmatrix} (A^{e_2})^T & 0 \\ (b^{e_2})^T & 1 \end{bmatrix} P_{m_1} \begin{bmatrix} A^{e_2} & b^{e_2} \\ 0 & 1 \end{bmatrix} - \vartheta^{e_2} R^{e_2} &\succeq 0 \\
P_{m_3} - \begin{bmatrix} (A^{e_3})^T & 0 \\ (b^{e_3})^T & 1 \end{bmatrix} P_{m_2} \begin{bmatrix} A^{e_3} & b^{e_3} \\ 0 & 1 \end{bmatrix} - \vartheta^{e_3} R^{e_3} &\succeq 0 \\
P_{m_2} - \begin{bmatrix} (A^{e_4})^T & 0 \\ (b^{e_4})^T & 1 \end{bmatrix} P_{m_3} \begin{bmatrix} A^{e_4} & b^{e_4} \\ 0 & 1 \end{bmatrix} - \vartheta^{e_4} R^{e_4} &\succeq 0
\end{aligned}
$$

Turning over this constraint system to the SDP solver CSDP [Borchers, 1999] leads to a solution with accuracy of 10 decimal places after 19 iterations and 0.13 seconds on a 2x2GHz CPU. The results for the LLFs are as follows (rounded to six digits and normalized):

$$P_{m_1} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2.165988 & -1.596379 \\ 0 & -1.596379 & -0.362564 \end{bmatrix}$$

$$P_{m_2} = \begin{bmatrix} 0.004218 & 0.013248 & 0 \\ 0.013248 & 2.553457 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$P_{m_3} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2.158351 & 1.621242 \\ 0 & 1.621242 & -0.294953 \end{bmatrix}$$

Note that the values that are given as zero are only zero after rounding. However, the value in the lower right of matrix P_{m_2} must be exactly zero if P_{m_2} is to represent a valid LLF. Therefore, it can be advantageous to re-run the solver, setting these values to exactly zero in beforehand to arrive at more accurate results. The same applies to P_{m_1} and P_{m_3}, where the first row and column consists only of zero entries.

3.8 Summary

In this chapter, a hybrid system model, stability notions, Lyapunov function techniques, and methods for the automatic computation of Lyapunov functions from the literature have been introduced. The hybrid system model is based on automata, which are enriched with differential inclusions and constraints on the continuous variables to produce a model combining the expressivity of both domains. Stability proofs via Lyapunov functions can be conducted automatically for this type of system model by employing linear-matrix-inequality-based methods. These methods result in a number of constraints, which are simultaneously solved with numerical methods. The solution of such a constraint system yields a Lyapunov function acting as a certificate of global asymptotic stability. However, this type of analysis:

- does not exploit any special knowledge we have about the discrete structure of the automaton,
- frequently runs into numerical problems for complex systems,
- does not give useful feedback in case of failure, and
- cannot easily be exploited for system design, as it is difficult to directly derive useful design rules for stable hybrid automata.

Therefore, the following chapter focus on the possibilities of exploiting the discrete structures of the automaton to address these issues. The backbone of the analysis will still be LMI-based, but it will be shown how some extra purely discrete reasoning can greatly reduce the complexity of the individual LMIs to be solved. This reasoning takes the form of decomposition: partitioning hybrid automata into sub-automata, each associated with a smaller LMI problem. The decomposition is structured such that still a proof of global asymptotic stability for the entire system can be obtained. To this end, we give theorems allowing us to re-compose local proofs of stability on sub-automata again into a global proof.

4 Decompositional Stability Analysis

This chapter contains one of the main contributions of this thesis: a decompositional framework for automatic Lyapunov function computation for hybrid automata, based on the discrete structure of the automaton which are interpreted as a directed graph.

First, the difficulties and the benefits of discrete-state-based decomposition of hybrid automata are discussed in Section 4.1. The graph-theoretic groundwork required for the decomposition is then presented in Section 4.2. Section 4.3 contains the first level of decomposition, based on the strongly connected components of the underlying graph. Several decomposition theorems are presented and proved, together with a discussion on how these theorems can be exploited in practice. The key observation on this level of decomposition is that it can be conducted losslessly (i.e., a decompositional proof is possible for all systems for which a monolithic proof is possible) and even permits attractivity proofs for systems which are not GAS in some cases.

In Section 4.4, the second decomposition step inside the strongly connected components is conducted. The theorems presented in this section can generally deal with different types of further sub-partitionings, but from a practical standpoint focus on a decomposition into simple cycles. This decomposition is driven by Lyapunov-function-based reasoning, conservatively under-approximating sets of Lyapunov functions by simpler representations, namely conic polytopes. The required computations can take the form of linear matrix inequalities, but the results are general enough to potentially allow the use of different types of algorithms for the Lyapunov function computation.

This is followed by a discussion on how to efficiently and reliably conduct Lyapunov function computations on simple cycles, which form the bottom level of decomposition, in Section 4.5. Some special structures in the underlying graph are identified and exploited for LMI-based computations. Most notably, cases where the use of separate local Lyapunov functions for different adjacent modes does not result in an increased solution space are identified. In this case, it is possible to cut down the size of the LMI problems losslessly by employing common or dependent Lyapunov functions for such modes. Furthermore, reachability checks can be conducted at this stage, to identify non-traversable cycles which can subsequently be ignored.

A detailed walkthrough of an example system, a cruise controller with complex braking model, is then given in Section 4.6. The system models a proportional-integral velocity controller with a saturation mode modeling maximal acceleration and two types of brakes. These types of brakes can be interpreted as a service brake resulting in a relatively small deceleration, and therefore meeting comfort requirements, and an emergency brake resulting in a stronger deceleration. Each brake is modeled such that the full braking effect cannot be achieved immediately. Instead, there is a gradual increase of braking power until the maximum is reached. In total, the system consists of six modes of operation with linear and constant differential equations, and non-deterministic transitions modeling an entire set of possible switching strategies. The theorems and algorithms of Sections 4.3, 4.4, and 4.5 are applied to this system explained in detail based on this example. The result is a decompositional stability proof for the system.

Then, in Section 4.7, methods for the iterative refinement of the approximations are discussed. First, we discuss refinement algorithms for two intersecting cycles of the automaton. Two types of algorithms are given: an exhaustive algorithm and a heuristic-based approach which is not complete, but much more efficient in most cases. Then, these results for two cycles are extended to entire strongly connected components. Finally, Section 4.8 concludes the chapter.

4.1 Decomposing Hybrid Automata

Decomposition techniques of (not necessarily hybrid) systems along their continuous variables have a long tradition in the control community, and can be interpreted as decomposing a system into blocks that run in parallel (e.g., manifesting themselves as the block diagrams commonly used in control theory and by tools like Matlab Simulink). For instance, results on input-to-state stability or small-gain theorems, with adaptions to the hybrid domain, exist [Laila & Nešić, 2003; Liberzon & Nešić, 2006; Nešić & Liberzon, 2005; Heemels & Weiland, 2008], and can be used to break down stability proof obligations per block. Nevertheless, these techniques are hard to use for verification, unless the individual blocks are only loosely coupled. Generally, much information is lost if one tries to separate continuous variables that are strongly interrelated, making stability proofs often impossible in such cases.

In contrast, decomposition along the discrete axis, resulting in a sequential composition of different sub-systems, has not been significantly exploited for stability analysis in the control community. One reason is that this is general difficult to do, since stability properties, and especially convergence, often depend on the interplay of the various modes of operation. This leads to difficulties in dividing a proof into per-mode obligations. Nevertheless, having methods for applying this type of decomposition is highly desirable. The motivation is threefold:

- For hybrid automata with many discrete states, the LMI problem given in Theorem 3.10 is complex and there is a high likelihood of running into numerical problems making it impossible to solve. Therefore, the automatic computation of a solution is often impossible with satisfactory accuracy, or requires some manual re-scaling of parts of the automaton in order to avoid numerical problems. Decompositional analysis results in LMI problems that are local on parts of the automaton, spanning less modes and hence avoiding such problems in many cases. Instead of handing over all proof obligations directly to the SDP solver, exploitation of discrete structures can greatly increase the chance of obtaining a robust result from the solver for large automata.

- Standard LMI-based analysis cannot easily be used for construction of stable hybrid automata, since the interplay between different parts of the automaton, which causes stability or instability, is not directly visible in the model. Instead, the interdependencies between the different discrete states are presented to the SDP solver. However, the solver will generally not make explicit use of the discrete structures of the automaton. In contrast to this standard monolithic analysis, a decompositional approach can also be used for composition, by deriving verifiable conditions that imply "composability" of two automata, such that the resulting automaton is still stable. With the help of compositional reasoning, incremental construction of stable hybrid automata becomes possible. Both from a theoretical and practical perspective, decomposition results can provide insight into what "makes or breaks" a particular system's stability property. Unlike many safety properties, whether or not a system is stable is difficult to see even for the experienced engineer.

- Many realistic hybrid system models (e.g., autopilot systems) will have many more discrete modes than continuous variables, making decomposition along the discrete states more promising. Furthermore, with the help of additional modes, it is also possible to approximate the behavior of continuous variables that are difficult to handle, for instance because of non-linearities. This can for example be done by piecewise linear approximation, where each newly introduced segment of the state space with linear dynamics can again be interpreted as an additional mode.

Stability is in general not preserved under transition composition of hybrid automata, as the following examples show.

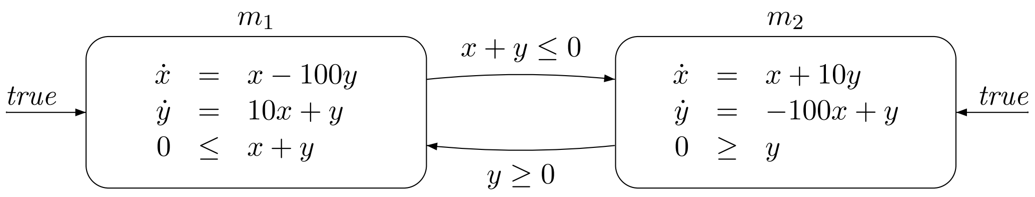
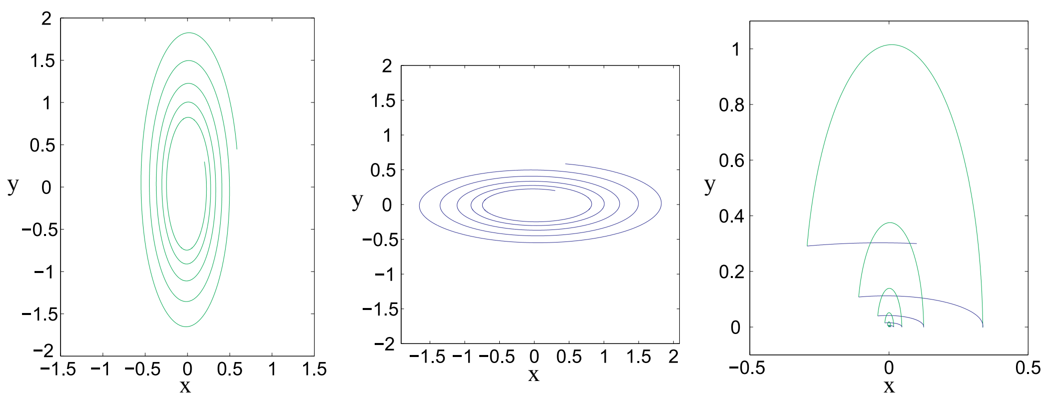


Figure 4.1: Stable System with Two Unstable Differential Equations

Example 4.1 (Unstable Differential Equations Resulting in a Stable Hybrid System)**.** Consider a hybrid system H_1 (taken from [Pettersson, 1999]), given by the the hybrid automaton in Figure 4.1. The differential equations of both individual modes m_1 and m_2 are individually *not* globally asymptotically stable. This can for instance be verified by checking that their system matrices are not Hurwitz (i.e., some eigenvalues have positive real parts). However, as a whole, the hybrid system H_1 is globally asymptotically stable. This is achieved by a smart switching between the two unstable modes that is enforced by the guards and invariants. This shows that non-stability of individual differential equations is not necessarily preserved under transition composition, as there might nevertheless exist switching strategies stabilizing the system. See Figure 4.2 for some example trajectories. The trajectories for m_1 and m_2 diverge, slowly spiraling away from the origin. The trajectory of H_1 shown in the figure starts at $x = 0.1$ and



(a) Mode m_1 (b) Mode m_2 (c) Hybrid System H_1

Figure 4.2: Example Trajectories for The System From Example 4.1

$y = 0.3$ and converges to the equilibrium. In fact, this is the case for all trajectories of the system. This can be shown with help of the LMI methods described in Section 3.5 [Pettersson, 1999, p.115].

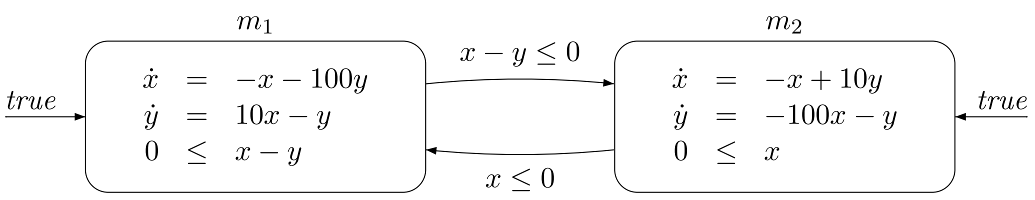
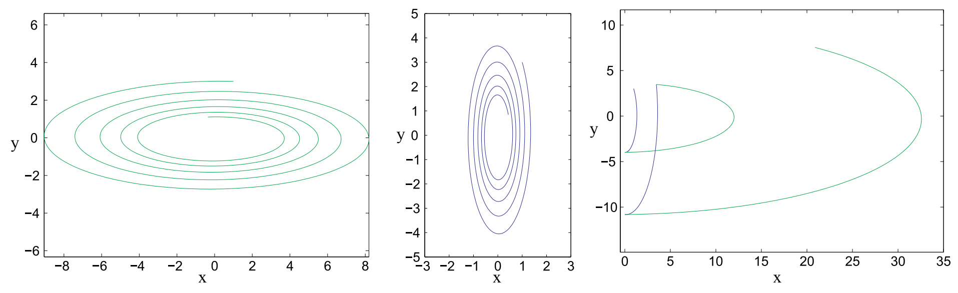


Figure 4.3: Unstable System with Two Stable Differential Equations



(a) Mode m_1 (b) Mode m_2 (c) Hybrid System H_2

Figure 4.4: Example Trajectories for The System From Example 4.2

Example 4.2 (Stable Differential Equations Resulting in an Unstable Hybrid System)**.** In contrast, the hybrid system H_2 given by the automaton in Figure 4.3 consists of two modes that are GAS. Trajectories of the differential equations of the individual modes and H_2 are given in Figure 4.4. In both cases, the trajectories converge, spiraling inward. However, H_2 is not stable, since the switching logic has been chosen such that for each time interval spent in m_2 the distance to the equilibrium increases significantly. The result is a divergent trajectory starting at $x = 1$ and $y = 3$.

Since neither stability nor instability are preserved under switching between two subsystems, composition/decomposition rules need to exploit knowledge on both the discrete structures (i.e., the underlying graph) and the continuous structures (i.e., the differential equations, invariants and guards) of the automaton. Therefore, the goal of this chapter is the derivation of local conditions on the automaton that ensure composability of the stability results for the sub-automata. This enables us to also do decomposition, by

1. splitting the automaton into suitable sub-automata,
2. proving stability properties on these sub-automata, plus some information to identify stability-preserving switching strategies between these sub-automata, and finally
3. combining these results into a stability proof for the entire system.

In line with the Lyapunov-based proof methods presented in Section 3.5, the decomposition rules are based on Lyapunov functions. However, Lyapunov function computations will only be done *locally* on the automaton, that is, only for as small as possible subsets of $\mathcal{M}$ at a time. We compute Lyapunov function for sub-automata, and prove theorems that ensure compatibility of these functions, such that existence of a *global* Lyapunov function for the entire automaton is guaranteed. This completes the stability proof. This global Lyapunov function will not need to be be explicitly computed, but a procedure to derive this function, if so desired, is presented. The decomposition results are applicable to both standard and probabilistic hybrid automata, and to a certain degree also to systems with stochastic differential equations. In the probabilistic setting, even stronger decompositions are possible under certain circumstances. This case will be discussed in Chapter 5.

4.2 Graph Structures

This section gives the graph theoretic definitions that are required for the decomposition. First, we need to define sub-automata of a hybrid automaton. Informally, sub-automata are simply automata that can be obtained by removing transitions and/or modes.

Definition 4.1 (Sub-Automaton)**.** A *sub-automaton* of a hybrid automaton H is a hybrid automaton H' with $\mathcal{M}_{H'} \subseteq \mathcal{M}_H$, $\mathcal{S}_{H'} = \mathcal{S}_H$, $\mathcal{V}_{H'} = \mathcal{V}_H$, $\mathcal{T}_{H'} \subseteq \mathcal{T}_H$, $\mathit{Flow}_{H'}(m) = \mathit{Flow}_H(m)$ and $\mathit{Inv}_{H'}(m) = \mathit{Inv}_H(m)$ for all $m \in \mathcal{M}_{H'}$, and

$$\begin{aligned}\mathit{Init}_{H'} \;=\; & (\mathit{Init}_H \cap (\mathcal{M}_{H'} \times \mathcal{S}_{H'})) \cup \\ & \{(m_2, x') \in \mathcal{M}_{H'} \times \mathcal{S}_{H'} \mid \exists (m_1, m_2, G, U) \in \mathcal{T}_H - \mathcal{T}_{H'} : \exists x \in G : x' = U(x))\}.\end{aligned}$$

Denote this relation as $H' \subseteq H$.

The initial states for the sub-automaton include all initial states of the original automaton which relate to modes in the sub-automaton, plus all states resulting from incoming edges into the sub-automaton.

Next, we move on to definitions of graphs and their sub-components. We will view hybrid automata as a special type of graph for the purpose of decomposition.

Definition 4.2 (Graph)**.** A *(directed) graph* G is a tuple (V_G, E_G, L_G^V, L_G^E), where

- V_G is a set of *nodes*,
- E_G is a multiset of *edges* $e \in V_G \times V_G$, and
- L_G^V and L_G^E are *labeling functions* mapping V_G and E_G, respectively, to some label set.

A *sub-graph* $G' = (V_{G'}, E_{G'}, L_{G'}^V, L_{G'}^E)$ of G is a graph such that

- $V_{G'} \subseteq V_G$,
- $E_{G'} \subseteq E_G$, and
- $L_{G'}^V$ and $L_{G'}^E$ are the restrictions of L_G^V and L_G^E onto $V_{G'}$ and $E_{G'}$, respectively.

The *degree* of a node $v \in V$ is defined as the cardinality of the set of incident edges, $|\{(v_1, v_2) \in E \mid v_1 = v \vee v_2 = v\}|$. Similarly, the *indegree* and *outdegree* are defined as $|\{(v_1, v_2) \in E \mid v_2 = v\}|$ and $|\{(v_1, v_2) \in E \mid v_1 = v\}|$, respectively.

Definition 4.3 (Graphs Associated with Hybrid Automata)**.** The graph associated with a hybrid automaton $H = (\mathcal{M}, \mathcal{S}, \mathcal{V}, \mathcal{T}, \mathit{Flow}, \mathit{Inv}, \mathit{Init})$ is $G(H) = (V, E, L^V, L^E)$ with

- $V = \mathcal{M}$,
- E is a multiset with one edge $e = (m_1, m_2)$ for each transition $(m_1, m_2, G, U) \in \mathcal{T}$,
- $L^V(v) = \mathit{pred}(\mathit{Flow}(v)) \wedge \mathit{pred}(\mathit{Inv}(v))$, and
- $L^E(e) = \mathit{pred}(G) \wedge \mathit{pred}(U)$, where $(m_1, m_2, G, U) \in \mathcal{T}$ is the transition corresponding to edge e.

with $\mathit{pred}(\mathit{Inv}(v))$ and $\mathit{pred}(G))$ being predicates describing the sets $\mathit{Inv}(v)$ and G, $\mathit{pred}(\mathit{Flow}(v))$ being the differential inclusion $\mathit{Flow}(v)$ in predicate notation, and $\mathit{pred}(U)$ being the update function U in predicate notation, as discussed in Remark 3.3.

Note that the graph associated with a hybrid automaton does not contain information on the *Init* predicate. Since initial states do not play any role in Lyapunov function constraint systems, we do not directly exploit this information, so that there is no need to include it in the graph-theoretic model. Initial states can, however, be helpful for a priori reachability analysis, allowing us to tighten invariants and guard of the automaton according to the reach set. For this analysis, the *Init* predicate should be kept separately.

The graph $G(H')$ of a sub-automaton $H' \subseteq H$ will therefore always be a subgraph of $G(H)$. We also need the notion of *paths* to define the decomposition. A path is simply a sequence of connected edges in a graph. A single node or edge of a graph can also be traversed multiple times by a path, unless the path is *simple*. Closed paths lead back to the same node where they started from.

Definition 4.4 (Path)**.** A *path* in a directed graph G is a finite sequence of edges $e_1, \dots, e_n, e_i = (v_i, v_i') \in E_G$, such that for all $1 < i \leq n$ $v_{i-1}' = v_i$ holds. A path is *closed* if $v_1 = v_n'$ and *simple* if all $v_i \neq v_j$ for all $i \neq j$.

Each directed graph can be decomposed into a number of sub-graphs called *strongly connected components*. They are defined as follows.

Definition 4.5 (Strongly Connected Component)**.** A *strongly connected component (SCC)* of a directed graph G is a maximal sub-graph G', such that for each pair of nodes $n_1 \neq n_2$ in G' there exists a path in G' from n_1 to n_2.

Here, maximal is taken to mean that no further nodes and edges can be added to the sub-graph such that the existence of such paths is preserved. An SCC G' will always contain all edges in G connecting nodes in G', since adding edges alone cannot destroy the existence of connecting paths.

Note that the family of SCCs for a graph is unique, since SCCs are always maximal (i.e., no edges or nodes can be added without destroying strong connectedness). If two SCCs are connected via an edge, there cannot be another connecting edge between the SCCs in the other direction. Therefore, a partial order on the SCCs of a graph can be defined as follows.

Definition 4.6. For a graph $G = (V, E, L^V, L^E)$ Define the successor relation "$\prec$" on strongly connected components $C_1 = (V_1, E_1, L_1^V, L_1^E)$ and $C_2 = (V_2, E_2, L_2^V, L_2^E)$ as follows:

$$C_1 \prec C_2 :\Longleftrightarrow \exists (v_1, v_2) \in E : v_1 \in V_1 \wedge v_2 \in V_2.$$

Each node in the graph belongs to exactly one SCC, while each edge belongs to at most one. Edges not belonging to any SCC are also called *bridges*.

Remark 4.1 (Computation of Strongly Connected Components)**.** The time complexity for computing the SCCs of an arbitrary graph is $O(|V| + |E|)$, for instance by using Tarjan's Algorithm [Tarjan, 1972].

See Figure 4.5 for an example of an SCC decomposition. Another level of decomposition within an SCC deals with cycles, which are subgraphs which can be covered by a single closed path.

Definition 4.7 (Cycle)**.** A *cycle* of graph G is a subgraph G' for which there exists a closed path $e_1, \dots, e_n$ with $e_i = (v_i, v_i')$ such that $E_{G'}$ is the set of all edges e_i on the path and $V_{G'}$ is the set of the nodes v_i of any such edges. A cycle is *simple* if there exists a simple closed path with this property.

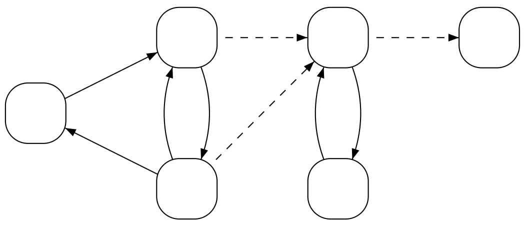
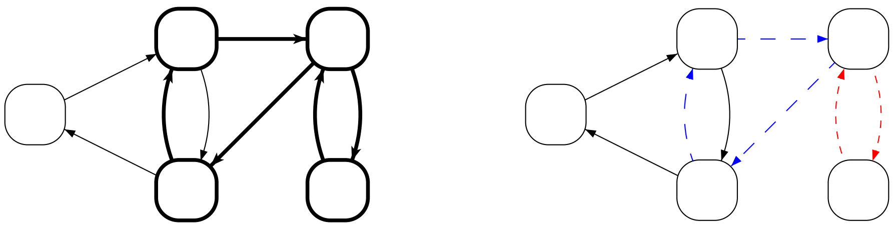


Figure 4.5: Decomposition of a Graph into SCCs (bridges dashed)

A cycle cover of a graph is a family of simple cycles covering all its nodes and edges.

Definition 4.8 (Cycle Cover)**.** A *cycle cover* of a graph G is a set of simple cycles $\{C_1, \dots, C_n\}$, $C_i = (V_i, E_i, L_i^V, L_i^E)$, such that $\bigcup_i V_i = V_G$ and $\bigcup_i E_i = E_G$.

See Figure 4.6(a) for a non-simple cycle in an SCC. Figure 4.6(b) then depicts a cycle cover of this SCC, consisting of three simple cycles.



(a) cycle of a graph (bold), consisting of two simple cycles

(b) simple cycle cover consisting of three cycles (solid/black, dashed/red, dashed/blue)

Figure 4.6: Cycles and Cycle Covers

Within an SCC, each node and each edge within an SCC lies on at least one simple cycle. Therefore, each SCC possesses at least one cycle cover. Bridges do not lie on any cycles of the graph, but do not belong to any SCC either.

Remark 4.2 (Computation of Cycle Covers)**.** While the problem of finding a cycle cover for an SCC with the minimal sum of cycle lengths is NP-hard, there exist numerous polynomial-time algorithms which can guarantee upper bounds on the cover size. See [Thomassen, 1997] and the references therein for a discussion. If the goal is just the computation of any cycle cover (without constraints in its size), then simple depth-first or breadth-first search can be used to successively identify cycles covering previously uncovered edges until the entire graph is covered. Since SCCs always permit a cycle cover, this simple greedy algorithm is guaranteed to terminate.

For each cycle of a cycle cover of an SCC, the nodes which also intersect with other cycles in the cover have a special role in the decomposition. They are called *border nodes*, since the cycle borders the "rest of the SCC" in these nodes.

Definition 4.9 (Border Node)**.** A *border node* of a cycle C_i in a simple cycle cover $\{C_1, \ldots, C_n\}$ is a node $b \in V_{C_i}$, such that there exists a $j \neq i$ with $b \in V_{C_j}$.

Note that the border nodes of a cycle are exactly all nodes of the cycle which have a degree larger than two in the full graph.

As a tool for visualizing the decomposition properties, we will now define so-called *constraint graphs*. Constraint graphs are graphs with one node corresponding to each mode and one edge corresponding to each non-loop transition of the automaton. Here, loop transitions are taken to be transitions from a mode back and back to the same mode. Nodes and edges are labeled with the *Lyapunov function constraints* corresponding to the mode or transition in question instead of differential inclusions, invariants, guards or updates. This means that each node label will represent a constraint on the LLF of that particular mode and each edge label will represent a constraint on the two LLFs of the modes corresponding to the two incident nodes in the constraint graph. Since loop transitions a represented as a constraint on just one LLF, they are not represented as edges in the constraint graph. Instead, the constraint is attached to the node itself. As Lyapunov functions constraints come in two classes, constraints on the dynamics of a single mode and constraints on two neighboring modes due to discrete transitions, there are no ternary or higher-degree constraints in terms of LLFs. Note that constraint graphs do not allow multiple edges in the same direction between two nodes. Instead, such multiple edges are collapsed into one edge and their constraints are conjoined. Therefore, the edges are given as a simple set and not as a multiset. Formally, constraint graphs are defined as follows.

Definition 4.10 (Constraint Graphs)**.** The *constraint graph* $C(H) = (V_C, E_C, L_C^V, L_C^E)$ of a hybrid automaton H with underlying graph $G(H) = (V_G, E_G, L_G^V, L_G^E)$ is a graph with:

1. $V_C = V_G$
2. $E_C = \{(m_1, m_2) \in E_G \mid m_1 \neq m_2\}$
3. $L_C^V(m)$ is the conjunction of Constraints (1) and (2) from Theorem 3.7 for the mode m and all Constraints (3) corresponding to loop transitions attached to m, that is,

$$L_C^V(m) :\Longleftrightarrow \left(\begin{array}{rl} & \exists f_1, f_2 \text{ in class } K^\infty : \\ & x \in \mathit{Inv}(m) \implies f_1(||x_{|\mathcal{V}_m}||) \leq V_m(x) \leq f_2(||x_{|\mathcal{V}_m}||) \\ \wedge & \exists f_3 \text{ in class } K^\infty : x \in \mathit{Inv}(m) \implies \dot{V}_m(x) \leq -f_3(||x_{|\mathcal{V}_m}||) \\ \wedge & \forall (m, m, G, U) \in \mathcal{T} : (x \in G \implies V_m(U(x)) \leq V_m(x)) \end{array} \right)$$

4. $L^E_C((m_1, m_2))$ is Constraint (3) from Theorem 3.7 for all transitions (m_1, m_2, G, U), $m_1 \neq m_2$, that is,

$$L^E_C((m_1, m_2)) :\Longleftrightarrow \bigwedge_{(m_1,m_2,G,U)\in\mathcal{T}_H} (x \in G \implies V_{m_2}(U(x)) \leq V_{m_1}(x))$$

Define *constr*(H) as the conjunction of all constraints in $C(H)$, that is:

$$\mathit{constr}(H) := \bigwedge_{m\in V_C} L^V_C(m) \wedge \bigwedge_{e\in E_C} L^E_C(e)$$

The predicate *constr*(H) subsumes all constraints on a GLF for the hybrid automaton H, and also allows us to analyze sub-problems generated by sub-automata of H.

We will usually express the constraints $L^V_C(m)$ and $L^E_C((m_1, m_2))$ as LMI problems, as detailed in Theorem 3.10. In this case, the nodes are labeled with instances of the inequalities (3.3), (3.5), (3.6), and (3.7), and the edges are labeled with instances of the inequalities (3.4) and (3.8). The constraints (3.1) and (3.2) do not talk about any individual LLFs, and therefore must be satisfied in addition to *constr*(H), in order to obtain a valid GLF for the system.

In informal discussions, for ease of writing, we will often identify a hybrid automaton with its graph. For instance, the "SCCs of hybrid automaton H" are actually the SCCs of $G(H)$, and also the SCCs of $C(H)$, since automaton and constraint graph share the same graph structure, apart from initial states and loops.

The following example illustrates the use of the constraint graph notion, and its visualization.

Example 4.3 (Velocity Controller with Wear and Tear)**.** Figure 4.7 shows a hybrid automaton modeling a very simple velocity control system. There are two types of modes: those modes modeling the engine behavior (labeled N_1, N_2, and N_3), and those modes modeling the brakes (labeled B_1, B_2, and B_3). The engine behavior is given as a simple univariate linear differential equation. The continuous variable v represents the velocity differential, the difference of current velocity and the desired set point. Initially, the system will be in mode N_1, if the velocity is close to the set point $v = 0$ or too low, or in mode B_1, if the velocity is significantly too high, switching between these two modes as appropriate. However, at any point in time, the system can take a transition from N_1 to N_2 or from B_1 to B_2, respectively. While N_1 and N_2 are identical in terms of behavior, B_2 can achieve less deceleration than B_1, modeling an aging process of the brake, representing wear and tear and resulting in reduced braking effect. Since this transition can take place at any time, the hybrid automaton actually models all possible timings of the aging. The same applies to N_3 and B_3, the latter of which again results in a further decreased braking power.

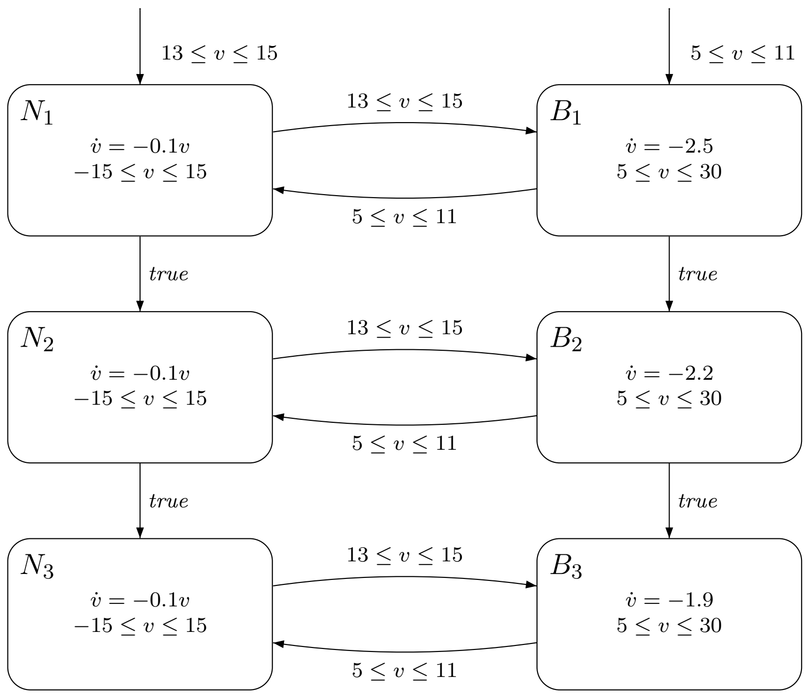


Figure 4.7: Cruise Controller with Wear and Tear

Figure 4.8 gives the constraint graph for this system with:

$$\begin{array}{rcl}
c_{N_i} & :\iff & \exists f_1, f_2 \text{ in class } K^{\infty} : -15 \le v \le 15 \implies f_1(||v||) \le V_{N_i}(v) \le f_2(||v||) \\
& & \wedge\ \exists f_3 \text{ in class } K^{\infty} : -15 \le v \le 15 \implies \dot{V}_{N_i}(v) \le -f_3(||v||), i \in \{1,2,3\} \\
c_{B_i} & :\iff & \exists f_1, f_2 \text{ in class } K^{\infty} : 5 \le v \le 30 \implies f_1(||v||) \le V_{B_i}(v) \le f_2(||v||) \\
& & \wedge\ \exists f_3 \text{ in class } K^{\infty} : 5 \le v \le 30 \implies \dot{V}_{B_i}(v) \le -f_3(||v||), i \in \{1,2,3\} \\
c_{N_i,B_i} & :\iff & 13 \le v \le 15 \implies V_{B_i}(v) \le V_{N_i}(v), i \in \{1,2,3\} \\
c_{B_i,N_i} & :\iff & 5 \le v \le 11 \implies V_{N_i}(v) \le V_{B_i}(v), i \in \{1,2,3\} \\
c_{N_i,N_{i+1}} & :\iff & V_{N_{i+1}}(v) \le V_{N_i}(v), i \in \{1,2\} \\
c_{B_i,B_{i+1}} & :\iff & V_{B_{i+1}}(v) \le V_{B_i}(v), i \in \{1,2\}
\end{array}$$

Formally, this graph is given as (V_G, E_G, L_G^V, L_G^E) with:

$$\begin{array}{rcl}
V_G & = & \{N_1, N_2, N_3, B_1, B_2, B_3\} \\
E_G & = & \{(N_1,N_2),(N_2,N_3),(B_1,B_2),(B_2,B_3),(N_1,B_1),(B_1,N_1),(N_2,B_2), \\
& & (B_2,N_2),(N_3,B_3),(B_3,N_3)\} \\
L_G^V(v) & = & c_v \\
L_G^E((v_1,v_2)) & = & c_{v_1,v_2}
\end{array}$$

$$\begin{tikzcd}
c_{N_1} \arrow[rr, bend left, "c_{N_1,B_1}"] \arrow[d, "c_{N_1,N_2}"] & & c_{B_1} \arrow[ll, bend left, "c_{B_1,N_1}"] \arrow[d, "c_{B_1,B_2}"] \\
c_{N_2} \arrow[rr, bend left, "c_{N_2,B_2}"] \arrow[d, "c_{N_2,N_3}"] & & c_{B_2} \arrow[ll, bend left, "c_{B_2,N_2}"] \arrow[d, "c_{B_2,B_3}"] \\
c_{N_3} \arrow[rr, bend left, "c_{N_3,B_3}"] & & c_{B_3} \arrow[ll, bend left, "c_{B_3,N_3}"]
\end{tikzcd}$$

Figure 4.8: Constraint Graph for the Cruise Controller

The predicate *constr*(H) is the conjunction of all these constraints, representing a sufficient condition for GAS of the system. When conducting a monolithic stability proof, one would search for a family of LLFs V_m fulfilling this constraint system.

4.3 Decomposition of Stability Proofs into Strongly Connected Components

The first step of decomposition of a stability proof for a hybrid automaton is based on strongly connected components. Intuitively, the argument is simple. Consider a hybrid automaton with a graph structure as in Figure 4.5 on page 84. Any mode sequence of an infinite trajectory of this automaton can either:

- stay inside the first (i.e., leftmost) SCC forever, or,
- take a transition to the second SCC, staying there forever, or,
- take a transition from the second to the third (i.e., rightmost) SCC and stay there forever.

In the first case, existence of a global Lyapunov function for the first SCC implies convergence of $x(t)$ to 0. Now, whenever $x(t) \to 0$, then this also holds for all suffixes of $x(t)$. Conversely, if any suffix of $x(t)$ converges to 0, then so does $x(t)$. Therefore, for the second case, it is sufficient to look at the suffix of $x(t)$ starting at the time the switch to the second SCC takes place. This suffix converges to 0, if there exists a GLF for the second SCC. This is equivalent to convergence of $x(t)$. The same applies to the third case and third SCC. Therefore, it is sufficient to have one *separate* GLF for each SCC. Most importantly, these Lyapunov functions need not be interrelated in any way. It is not necessary to have a decrease of the global Lyapunov function upon transitioning between SCCs to prove convergence. Therefore, the edges corresponding to these transitions are dashed in Figure 4.5. They are not mapped onto Lyapunov function constraints. The constraint graph for the hybrid automaton can therefore be decomposed into the constraint graphs pertaining to the SCCs. Then, for each SCC C_i, the constraint system $\mathit{constr}(C_i)$ can be solved independently to complete the attractivity proof for the entire system. The constraints implied by the bridges of the automaton are not needed here and can be dropped completely.

To also prove stability (that is, global *asymptotic stability* instead of just global *attractivity*), one additional condition needs to be satisfied: the update functions associated with the bridges need to bounded by linear functions. If this property holds, global asymptotic stability as a whole can be shown by this decomposition. This boundedness property can be checked in beforehand, before any GLF computations take place, as the argument is not related to the parameters for the Lyapunov functions at all. A theorem formalizing this decomposition, with a complete proof, is given next.

Theorem 4.1 (Decomposition into Strongly Connected Components)**.** Let H be a hybrid automaton. If all sub-automata pertaining to the SCCs of H are globally attractive then so is H. If all SCCs C_i are globally stable and all transitions (m_1, m_2, G, U) corresponding to bridges of $G(H)$ are *sub-linear*, that is,

$$\exists c > 0 : \forall x \in G : ||U(x)|| \leq c||x||,$$

then H is globally stable.

Proof.
Global Attractivity:
Let $x(t)$ be a fixed infinite trajectory of H, and (m_i) the associated mode sequence. Let (C_k) be the sequence of SCCs (m_i) enters, in order. Since no SCC can occur twice in (C_k), and the total number of SCCs is finite, (C_k) must be finite. Let t_0 be the point in time when $x(t)$ enters the final SCC of C_k and let $\tilde{x}(t) = x(t - t_0)$. Since SCC C_i is globally attractive, we have $\tilde{x}(t) \to 0$ for $t \to \infty$, which implies $x(t) \to 0$.

Global Stability:
Let P be the set of all possible SCC sequences $C_0 \prec \ldots \prec C_n$, connected by all possible bridge sequences (b_i) of H, with $b_i = (m_i, \tilde{m}_i, G_i, U_i)$. Let c_i be a sub-linearity factor of b_i. We will now prove global stability of the overall system by successively applying in an alternating manner: 1) the stability property of an SCC, and 2) the sub-linearity property of the bridge leading into the SCC. This is first done for an individual $p \in P$. Let $\epsilon > 0$. Select a $p \in P$. Let $x(t)$ be a trajectory corresponding to p (i.e., the associated mode sequence traverses the SCCs C_i in the order given by p), and let (t_i) be the sequence of switching times between the C_i. Let $x_i(t)$ be the trajectory segment corresponding to C_i. Beginning with C_n, we have

$$\exists\, \delta_n^p > 0\, \forall t \geq t_n : (||x_n(t_n)|| < \delta_n^p \implies ||x_n(t)|| < \epsilon)\,.$$

In particular, we also have $\delta_n^p \leq \epsilon$. Exploiting sub-linearity, we then obtain

$$||x_{n-1}(t_n)|| < \delta_n^p / c_{n-1} \implies ||U_n(x_{n-1}(t_n))|| < \delta_n^p$$

for the bridge between C_{n-1} and C_n. In the same fashion, for $0 < i < n$, we can conclude

$$\exists\, \delta_i^p > 0\, \forall t \in [t_i, t_{i+1}] : \left(||x_i(t_i)|| < \delta_i^p \implies ||x_i(t)|| < \delta_{i+1}^p / c_i \right),$$

which implies $\delta_i^p \leq \delta_{i+1}^p / c_i$. For the bridge between C_{i-1} and C_i we have

$$||x_{i-1}(t_i)|| < \delta_i^p / c_{i-1} \implies ||U_n(x_{i-1}(t_i))|| < \delta_i^p.$$

Finally, for C_0, we arrive at

$$\exists\, \delta_0^p > 0\, \forall t \in [0, t_0] : (||x_0(t_0)|| < \delta_0^p \implies ||x_0(t)|| < \delta_1^p / c_0)$$

and $\delta_0^p \leq \delta_1^p / c_0$. Set

$$\epsilon' := \frac{\epsilon}{\prod_i \min(1, c_i)}$$

Since $\epsilon \leq \epsilon'$ and for all $i > 0$

$$\delta_i^p / c_{i-1} \leq \delta_n^p / \prod_{j=i}^{n-1} c_j \leq \epsilon / \prod_{j=i}^{n-1} c_j \leq \epsilon',$$

together with the sequence of implications for the C_i and the bridges above, this implies

$$||x(0)|| \leq \delta_0^p \implies \forall t : ||x(t)|| \leq \epsilon'.$$

In other words, all trajectories corresponding to p and starting within a δ_0^p-ball around the origin will stay within an ϵ'-ball at all times, for any $\epsilon > 0$. Since ϵ and ϵ' are related by a fixed factor only, this implies the existence of a suitable δ_0^p for any ϵ'. We will now take all these δ_0^p-balls for the different $p \in P$ and a given ϵ' and combine them into one bound that guarantees the desired boundedness condition over all $p \in P$. For any ϵ', define $\delta^p(\epsilon')$ as the corresponding δ_0^p. Now, take the minimum over all p:

$$\delta(\epsilon') := \min\{\delta^p(\epsilon') \mid p \in P\}.$$

This minimum exists, as P is finite. This gives us

$$\forall \epsilon' > 0, t \geq 0 : ||x(0)|| < \delta(\epsilon') \implies ||x(t)|| < \epsilon',$$

which completes the proof. □

If we are concerned only with GAS with respect to only a subset of variables $\mathcal{V}' \subseteq \mathcal{V}$, then the sub-linearity constraint must be replaced by

$$\exists c > 0 : \forall x \in G : ||U(x)|| \leq c||x||_{|\mathcal{V}'}.$$

In that case, the proof can be conducted in the same manner.

Sub-linear transitions include cases like the identity function (i.e., no actual update) or linear update functions of the form $U(x) = Ax$, where x is the state vector and A is a matrix. Not all transitions with constant updates are sub-linear. For instance, the one-dimensional update function $U(x) = 5$ with guard $x = 2$ is sub-linear with factor $c = 2.5$. If the guard is changed to *true*, this update is not sub-linear with respect to any factor. Therefore, in order to conclude stability, it is paramount to have guards that are as strict as possible, including few or no states which are actually unreachable. Reachability-based tightening of guards can possibly be used to salvage sub-linearity and therefore stability.

Example 4.4. Figure 4.9 shows the velocity controller system of Example 4.3, decomposed into its SCCs. The system can be decomposed into three two-mode automata, each of which represents one wear and tear stage. Each sub-automaton can now be analyzed completely separately. Since the bridges of the original automaton had no discrete updates of the continuous variables, global attractivity and stability are preserved under this decomposition. The three sub-automata can be therefore analyzed completely independently for GAS, with completely separate GLF computations.

In case the update on a bridge is not sub-linear, the decomposition actually produces stronger results than solving the monolithic problem. Consider the following example.

Example 4.5 (Attractive, but Unstable System)**.** The system given by Figure 4.10 consists of two globally asymptotically stable modes (which can again be verified by eigenvalue computation). Therefore, per Theorem 4.1, the whole system is attractive. However, the update function $U(x) = 5$ is not sub-linear on $x = y$. Therefore it is not possible to conclude global asymptotic stability of the system. In fact, the system is not globally asymptotically stable. Select an $\epsilon < 5$. No matter how close to the origin we start, the trajectory following the dynamics of m_1 will eventually activate the guard $x = y$. This can, in turn, trigger the transition, with the update function setting variable x to 5. Therefore, there is no δ-ball around the origin, for which all trajectories starting there are guaranteed to stay within the ϵ-ball around the origin for all future times. See Figure 4.11 for some example trajectories. Note that x can eventually be set to 5, regardless of initial state, so, the stability condition is, for instance, violated for $\epsilon = 4$. Therefore, global stability does not hold for this system. This implies per contraposition of Theorem 3.5 that there can be no GLF for the entire automaton. Therefore, monolithic Lyapunov function computation to prove convergence must fail. However, if we use the decomposition proposed in Theorem 4.1, we are able to prove attractivity decompositionally. This leads to the following important observation.

In case a hybrid automaton is globally attractive, but not globally stable, there cannot exist a continuous or discontinuous Lyapunov function for the whole system. However, all of the individual sub-automata corresponding SCCs of the automaton might be attractive, if viewed in isolation. According to Theorem 4.1, this implies attractivity of the entire system. Therefore, in this case, the decomposition can actually lead to *stronger*

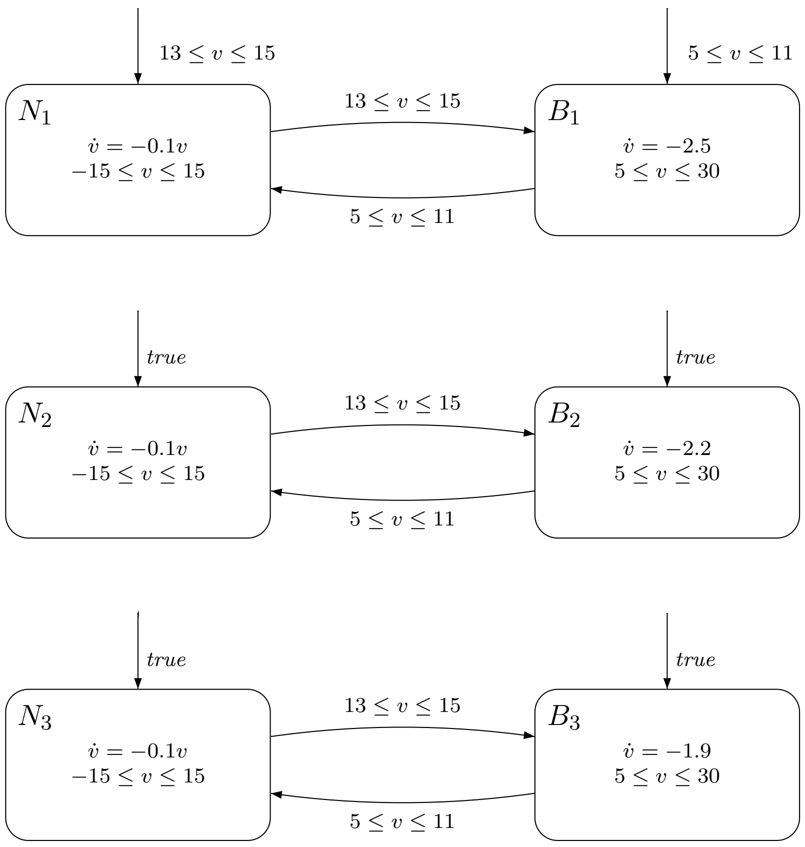


Figure 4.9: Velocity Controller with Wear and Tear Decomposed into three SCCs

results than the standard monolithic verification. The decomposition into SCCs can be employed to enlarge the set of possible Lyapunov function for proving attractivity only. For the system in Example 4.5, this turns an infeasible constraint system into two feasible constraint systems.

If one is only interested in proving global attractivity, then it suffices to consider SCCs in which trajectories can potentially stay infinitely long, that is, SCCs that can be the final SCC a trajectory moves into. This is a consequence of the proof of global attractivity of Theorem 4.1, which only requires the last SCC in (any possible) sequence to be GAS. For global stability, in contrast, knowledge of the stability of all reachable SCCs is required in the proof, even if they are only transient. Unreachable SCCs can trivially be ignored in any case. This is formalized in the following corollary.

Corollary 4.1. Let H be a hybrid automaton and let $C_1, \dots, C_n$ be all strongly connected components of H for which there exists a trajectory with a mode sequence entering but never leaving the component. If all C_i are globally attractive, then so is H.

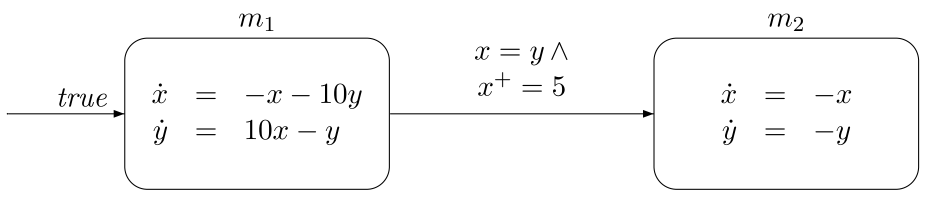
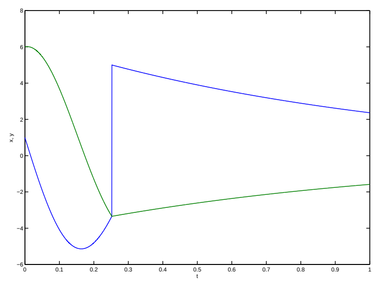


Figure 4.10: Attractive, Unstable System


(a) Initial state $x = 6$, $y = 1$

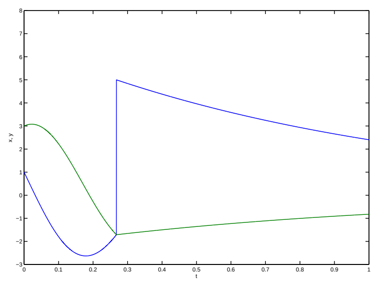

(b) Initial state $x = 3$, $y = 1$

Figure 4.11: Example Trajectories for the System Given in Figure 4.10, state versus time

The following theorem implies that this decomposition is not conservative with respect to Lyapunov functions. In other words, whenever it is possible to identify a GLF for the entire system, it is also possible to find individual GLFs for all SCCs.

Theorem 4.2. Let H be a hybrid automaton and let $C_1, \dots, C_n$ be the SCCs of hybrid automaton H. If there exists a GLF $V(x, m)$ for H, then there exists a GLF for each SCC C_i.

This property follows directly from the fact that a GLF can be split up into several Lyapunov functions on the sub-automata. The implication is that there exists no system for which the constraint system has a solution, but the corresponding decompositional constraint systems do not. Therefore, there is no loss in considering all SCCs individually. As discussed above, the class of systems for which attractivity can be shown decompositionally is even larger. On top of this, we obtain the additional benefit that the resulting constraint systems are also smaller than in the monolithic case, and are therefore more easily solved.

Note that there might be a different representation of the same system as a hybrid automaton that consists of smaller SCCs and therefore results in a less difficult Lyapunov function computation problem for the individual SCCs. To discover such representations, for instance by replacing one large SCC by two smaller ones exhibiting equivalent continuous behavior, reachable set computation can be employed. An example of this is given next.

Example 4.6 (Hybrid System with Hidden SCCs)**.** Figure 4.12 gives a very simple example of a hybrid automaton which has an equivalent representation with a higher number of SCCs. The original automaton consists of two SCCs, one containing only mode m_1, and one containing modes m_2 and m_3. The two dashed edges labeled $x = 0$ are not reachable, since the modes m_1 and m_2 are entered with $x \neq 0$, and the differential equations $\dot{x} = -2x$ and $\dot{x} = -3x$, respectively, cannot result in trajectories crossing the plane $x = 0$. Therefore, these two transitions can simply be removed from the automaton without changing its semantics. The result is a hybrid automaton with three single-mode SCCs.

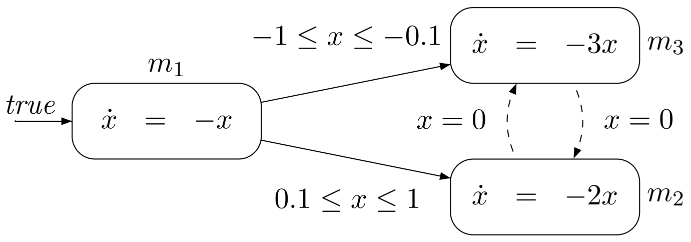


Figure 4.12: Hybrid Automaton with Hidden SCCs (Unreachable Edges Dashed)

This kind of system simplification also can be conducted with help of LLFs of the individual modes, as discussed in Section 3.4, since each LLFs can be used to generate suitable barrier certificates. The barrier certificate constraint $V_m \leq c$ can then be conjoined with both the mode invariant and the guards of all outgoing edges, without changing the semantics of the hybrid automaton. If any edge receives a guard *false* through this procedure, it can be removed. Therefore, it can be advantageous to conduct LLF computations for all modes before making GLF computations for the entire system or its SCCs. This first analysis can also help tightening the guards and invariants of the system and therefore increase the success rate of the GLF computation that follows.

For SCCs of hybrid automata that are transient, there is also another interpretation of their stability property. Lyapunov functions imply the absence of infinite trajectories that do not converge to the equilibrium point. If the equilibrium point is not reachable within the sub-automaton H corresponding to the SCC, then the existence of a GLF for H tells us that there can in fact be no infinite trajectories as per Definition 3.13 at all in H. In other words, all trajectories of H must be finite, as some invariant will always be violated at some point in time with no guards active within the SCC itself. In this case, the Lyapunov function takes the role of a *termination function* of the component, proving that, eventually, the invariant of the currently active mode is violated with no active guard to continue the run of the system. For all SCCs with this property, their corresponding GLFs actually prove the transience of the component.

In fact, the traditional definition of a discrete-time termination function is directly related to Lyapunov functions. Termination functions are required to decrease at least by some positive ϵ at every time step and to be bounded from below. While Lyapunov functions do not, in general, possess an absolute minimum decrease rate (i.e., a negative upper bound to $\dot{V}(x)$ over all x), they do so on any closed set I that does not contain the equilibrium x_e. This closed set can, for instance, be a mode invariant. Furthermore, Lyapunov functions are always bounded from below. This means that termination can also be shown with Lyapunov techniques. See Figure 4.13 for an illustration, with I denoting the closed set and the dashed line denoting a contour line of the Lyapunov function which is eventually crossed left to right. Since it is not possible to return to I after crossing the contour line, this implies the "termination" of I.

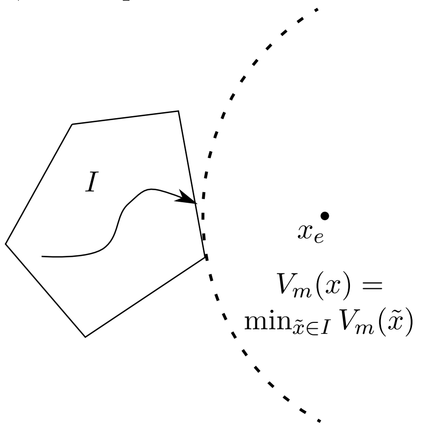


Figure 4.13: Lyapunov Function V_m as Termination Function for Set I

While it may seem at first sight that Lyapunov functions impose stronger constraints than termination functions, since they are used to show convergence to a state, this is actually not the case. Lyapunov conditions must only hold for states that are actually reachable within the mode in question. This means that, if the chosen equilibrium lies outside the invariant set, we actually do not prove convergence within this mode. This is because no constraints at all are imposed in a neighborhood around the equilibrium which does not intersect with the invariant set. Therefore, if we were to show only termination of a mode with help of a LLF, we could theoretically pick *any* state as "equilibrium" which possesses such a neighborhood. With this degree of freedom, a Lyapunov function becomes exactly a termination function for the mode in question.

The following corollary considers the case that the SCCs do not share the same equilibrium. Instead, each SCC is allowed to have a *separate* equilibrium state. In this case, Lyapunov functions can be used to prove convergence to *some* equilibrium for each trajectory.

Corollary 4.2 (Convergence to Multiple Equilibria)**.** If each SCC C_i of hybrid automaton H has a (possibly different) equilibrium x_{C_i}, then the existence of a GLF V_{C_i} for each SCC implies for each trajectory $x(t)$ of hybrid automaton H that there exists an i with $x(t) \to x_{C_i}$.

This is a direct consequence of Theorem 4.1.

In the same vein as for the termination discussion above, equilibria which are not covered by any invariant of the SCC they belong to can be excluded. This follows directly from the termination arguments: in this case, the GLF of the SCC serves as a proof that the SCC is transient, meaning that each trajectory entering it will leave it again at some future point in time. Since this unreachable equilibrium plays no role in the convergence result itself, the GLF of the transient component could, as discussed above, theoretically use *any* $x_e \in \mathcal{S}$ as "equilibrium point."

Remark 4.3. When using this decomposition technique, note that one does not obtain a GLF for the entire system, but only for the system's SCCs. Therefore, upon switching from one SCC to another, it is possible to observe an *increase* in Lyapunov function value. As shown in Theorem 4.1, this is still sufficient to prove attractivity, and if bridges are sub-linear, then also sufficient to prove global stability. However, one is sometimes interested not only in a stability proof, but also in a truly global Lyapunov function serving as a certificate of this property. For such cases, it is possible to construct a GLF for the entire system from the GLFs V_{C_i} of the SCCs C_i. The basic idea is that scalar multipliers are applied to the V_{C_i}, such that the conditions implied by the bridges (which were not used in any of the sub-LMIs for the SCCs) are also satisfied.

A simple solution is as follows. Consider a system with two SCCs C_1 and C_2 connected by a single sub-linear bridge (m_1, m_2, G, U), where m_1 is a mode of SCC C_1 and m_2 is a mode of SCC C_2. One additional assumption is that the guard G is contained in the invariant $Inv(m_1)$. If this is not the case, then the hybrid system can be equivalently reformulated accordingly. Now assume that we have two GLFs V_{C_1} and V_{C_2} for the two SCCs, computed from the LMI problem in Theorem 3.10. We need to apply a positive multiplier λ to V_{C_1} in order to obtain a GLF for the entire hybrid automaton. From the LMI constraint (3.5) of Theorem 3.10, it is clear that $V_{C_1}(x, m_1) \geq ||x||^2$ for $x \in Inv(m_1)$. Furthermore, constraint (3.6) implies that there exists a $\beta > 0$ with $V_{C_2}(x, m_2) \leq \beta ||x||^2$ for all x. Assume that c is the sub-linearity factor for the bridge. Then,

$$(1/\beta)V_{C_2}(U(x), m_2) \leq ||U(x)|| \leq c||x|| \leq cV_{C_1}(x, m_1).$$

Therefore,

$$V_{C_2}(U(x), m_2) \leq \beta c V_{C_1}(x, m_1).$$

The scalar βc is a possible such multiplier, but not necessarily the optimal one. To obtain better multipliers for V_{C_1}, we can also formulate this as a univariate optimization problem: find the minimal multiplier such that the bridge constraint (3.8) of Theorem 3.10 is fulfilled. This is again an LMI problem where the $\mathcal{S}$-procedure can be used in the usual manner, and the multiplier is the only unknown.

For a system with several SCCs, one simply exploits the relative order defined by the "$\succ$" relation. All SCCs which are final (i.e., have no successors) do not receive a multiplier. We then first assign multipliers to their predecessors and then continue to work backwards through the system. If an SCC has several outgoing bridges, then the maximal multiplier resulting from these computations must be used.

Obviously, this simple decomposition into SCCs can only serve as a first step. While hybrid automata can consist of several SCCs, for instance modeling different layers of emergency behavior, the individual SCCs can still be very large. For instance, consider a system that 1) detects an emergency, 2) reacts accordingly and then 3) switches back to normal behavior, once the hazardous situation has been averted. The returning transition from emergency to standard control strategies makes a decomposition into SCCs impossible. Clearly, there is a need for a decomposition strategy within an SCC. Such a method is proposed in the next section.

4.4 Decomposition within Strongly Connected Components

The decomposition of a hybrid automaton into SCCs can be achieved in a lossless manner, as far as the existence of Lyapunov functions is concerned. This is guaranteed by Theorem 4.2. However, inside an SCC, this is generally no longer the case, as dependencies between sub-graphs within an SCC are *bidirectional*, whereas dependencies between SCCs are *unidirectional* because of the relative order of SCCs. As demonstrated in Examples 4.1 and 4.2, switching back and forth between two subsystems preserves neither stability nor instability. Therefore, stability proofs for each subsystem in isolation cannot be combined into a stability proof for a composed automaton without exploiting of additional information.

The first question to be answered is: "What kind of graphs should be used as a basis for decomposition inside SCCs?" There are many possibilities, but in the scope of this thesis, the decomposition is based on *cycles*. The justification is as follows, and the points are discussed in detail below.

- Each node (=mode) and each edge (=transition) within an SCC lies on at least one simple cycle.
- By solving LMI problems on cycles, one can conduct decomposition and avoid explicit fixed point iterations.
- For large classes of systems, the length of simple cycles in a SCC is relatively small, while the number of cycles can often be large.

Each node (=mode) and each edge (=transition) within an SCC lies on at least one simple cycle: It is possible to cover an SCC completely, using only cycles. Each cycle can be broken down into a number of simple cycles, that is, cycles which do not contain the same node twice. Moreover, cycle covers can be computed efficiently (see Remark 4.2), and these graph algorithms will generally not be the bottleneck of the computation. Computation times for the actual GLFs corresponding to the cycles can be expected to dominate these costs.

By solving LMI problems on cycles, one can conduct decomposition and still avoid explicit fixed point iterations: Imagine a decomposition approach where one 1) looks at each mode individually, 2) solves its local LMI problem and then 3) tries to fit together the individual solutions (i.e., LLFs) to obtain a stability proof. Within a cycle consisting of three modes $m_1, m_2 m$ and m_3, with transitions from m_1 to m_2, m_2 to m_3, and m_3 to m_1, one would start by solving the constraint system for m_1, obtaining a LLF V_{m_1}. In a second step, one would then try to compute a V_{m_2}, such that the constraints imposed by the transition from m_1 to m_2 are also satisfied. Therefore, this second computation would need the previously computed function V_{m_1} as input. If this succeeds, one would repeat the procedure for m_3, whose LLF V_{m_3} would implicitly depend on V_{m_1} and V_{m_2}. However, V_{m_3} would have to fulfill two constraints: the decreases upon transitioning from m_2 to m_3 and from m_3 to m_1. With the wrong choice for V_{m_1} in the first step or the wrong choice for V_{m_2} in the second step, this problem is likely to have no solution. This would force us to again re-compute a different V_{m_1}, then V_{m_2}, and then hope for a better situation. This iteration would have to be repeated until a fixed point is reached, that is all three V_{m_i} are compatible.

Contrarily, using an LMI solver for this cycle alleviates this problem, since the descent-like methods used in the non-linear optimization routines effectively consider all three modes at once. Therefore, all fixed point computations are left to the optimization software, resulting in a clean and more transparent composition that does not have to cope with this problem directly. This makes cycles a favorable abstraction point for both compositional design and decompositional analysis.

While it might seem that systems containing cycles of cycles would remain a problem, this is not the case. This problem can be alleviated by a node splitting operation that preserves the semantics of the hybrid automaton, but can be used to get rid of such circular dependencies among cycles. Therefore, the computational procedure for the decomposition proposed in this chapter is such that, even in this case, fixed point iteration can be avoided. However, the cost of such a decomposition naturally increases with the interconnectedness of the sub-graphs to be decomposed.

For large classes of systems, the length of simple cycles in a SCC is relatively small, while the number of cycles can often be large: Scenarios with cycle lengths of more than three are not all too common. For this to occur, the hybrid system needs to go through a fixed sequence of modes in a fixed order. In the control domain, this might occur when, after a switch, the new dynamics need a "warm up phase," until they reach full effect. However, long sequences of such transient states do not seem to occur in many applications. In fact, most of the time, transitions between two modes will be possible in both directions, resulting in a simple cycle of length two. In contrast, a hybrid model can often have many cycles. If a mode is used to model some temporarily different behavior, then this will usually be reflected by a cycle, often of length two: a mode with an incoming edge (with the activation condition of the differing behavior as guard) and an edge leading back to the rest of the system (with the exit condition as guard). The tendency to have many small cycles makes this decomposition particularly attractive, as the result is a strong decomposition resulting in many sub-components. While automata obtained as the parallel composition of various sub-automata sometimes exhibit larger cycles, methods for *parallel decomposition* can be employed to treat the sub-automata separately, alleviating this problem. However, methods for parallel decomposition are not within the scope of this thesis and therefore not discussed in detail.

For these reasons, *simple cycles* are a suitable choice for further decomposition. However, global asymptotic stability of two cyclic automata does not have any general implication with respect to the stability of a composition, unless they are composed in a manner that results in two separate SCCs. Therefore, additional information is needed to combine the results from the two cycles. This leads to the second question to be answered: "What kind of information do we need to store for the computations for the individual cycles?" This information needs to be such that we can always ensure *correctness*. However, it is in general impossible to avoid forfeiting *completeness*, compared to the monolithic computation. In other words, when decomposing individual SCCs, the class of systems for which a *monolithic* stability proof is theoretically possible is larger than the class of systems for which a *decompositional* stability proof is theoretically possible. This is in contrast to the decomposition into SCCs, which is lossless, and also permits attractivity proofs of more systems than before. However, in Section 4.7, we will introduce a refinement procedure that can be used to reduce this gap.

The goal is to choose the level of decomposition such that this additional incompleteness is outweighed by the gain of practical tractability of the problem. We want to be able to conduct decompositional stability proofs in cases were the computations become numerically difficult or simply too computationally complex for the monolithic case. Nevertheless, there is a tradeoff between tractability and completeness here. The more information we keep in between cycles, the closer we will be to the monolithic case, leading to frequent numerical instability, and, generally, strongly coupled computations for the individual cycles and a weak decomposition. On the other end of the scale, if we keep little information, we effectively reduce the number of systems for which stability can be shown, while keeping the individual computations simple and relatively decoupled. In addition, the incompleteness is counterweighed by new opportunities for composition and re-design. If a Lyapunov function computation for a sub-system fails, it is clear that something must be changed in this sub-system to avert the problem. A strong decomposition means that these sub-systems will be small, making the identification of the problematic, stability-destroying system parts easier. Furthermore, even during the design process, criteria for the choice of switching or control strategies for individual transitions or modes can be formulated based on the decomposition results. This increased transparency is a great advantage in the decompositional approach, very often outweighing its inherent incompleteness.

There will be systems that are more or less amenable to decomposition. They key property is the connectedness of the discrete structures. Clearly, tightly interacting modes will be difficult to separate without incurring major conservativeness in the computations. This is not avoidable, so the composition techniques are developed with systems with a "somewhat sparse" discrete structure in mind. Generally, tree-shaped structures are easiest to decompose (even losslessly, as was shown in Section 4.3), while systems represented by fully connected automata are hardest to decompose. Therefore, for practical application of the decomposition, an a priori decision process identifying the parts of the systems to be decomposed might be useful. If cliques (i.e., fully connected sub-graphs) can be identified in the graph, then they can be marked as non-decomposable, if so desired. While the decomposition techniques will still work for fully connected graphs, their efficiency will suffer, so that it might be valuable to exclude parts of the automaton from the decomposition.

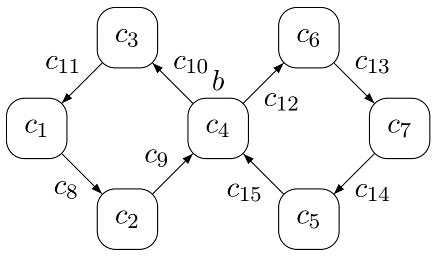
In order to conduct cyclic decomposition, a decision on how to represent the information to be kept “inbetween computations” needs to be made. Since the stability analysis in this thesis is based on the notion of Lyapunov functions, it is natural to also use Lyapunov functions in the decomposition. The property that is exploited is that the set of all Lyapunov functions for any system (continuous/discontinuous Lyapunov functions for continuous/discrete/hybrid systems) will always form a convex cone (see Theorem 3.6).

This has an important consequence: Whenever we have finitely many global Lyapunov functions V^i for a given hybrid system H, then their conic hull will only contain Lyapunov functions of H. This means that conic, polytopic infinite sets of Lyapunov functions can be represented efficiently by just keeping track of the GLF V^i. These GLF V^i form the “corner points” of the conic polytope. Moreover, conic polytopic sets can be used to under-approximate the actual sets of Lyapunov functions fulfilling the monolithic LMI constraints arbitrarily closely. Therefore, we will employ polytopic cones of Lyapunov functions to ensure that the stability proofs of the individual cycles lead to a stability proof for the entire system.

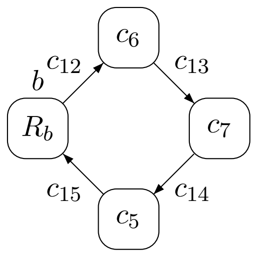
To decompose a system (i.e., its constraint graph) into simple cycles, we need a simple cycle cover (see Definition 4.8), which acts as a basis of the decomposition. Such a cyclic decomposition exists, since each node and edge inside an SCC lies on at least one cycle. The idea is to first identify a cycle to be examined. Then the constraint system for this cycle is solved not only once, but several times with different objective functions, in order to obtain the information that we need in order to ensure “compatibility with the following computations.” We only choose cycles that overlap with the rest of the graph in at most one border node. If no such cycle exists, a transformation procedure on the hybrid automaton is invoked to produce one. After the computations, this cycle (apart from the border node, which is potentially shared with other cycles) is collapsed and replaced by the computed compatibility information. This information comes in the form of a conic polytopic set of LLFs for the border node, and is attached to it as an extra constraint. Per Theorem 3.6, we can simply represent this set of LLFs by its corner points, representing an infinite set of functions by a finite set. The resultant constraint graph is smaller as all non-border nodes of the reduced cycle have been removed. Most importantly, the constraint system *does not talk about these nodes any more.* Moreover, since the newly added predicate only refers to LLFs for the border node, the “size” of the information that is kept is not dependent on the number of nodes that have been removed. Therefore, there is *no additional blowup* with respect to the size of the collapsed cycle, as the complexity of the predicates is not dependent on the number of nodes that have been abstracted away. This procedure is then repeated until the constraint graph is reduced to an empty graph, and the conic constraints for the border nodes are taken into account for any further constraint systems containing the node. Eventually, this procedure will result in a step-by-step computation of a cycle cover, interrupted by possible transformation steps to produce reducible cycles.

See Figure 4.14 for an illustration of an example. A system of two cycles, represented by its constraint graph, is reduced in this example. First, one of the cycles is picked (either one would be possible, as both of them have exactly one border node), in this case the left one. By obtaining multiple solutions to its constraint system (that is, the constraints c_1 to c_4 and c_8 to c_{11}), a conic predicate R_b on the free parameters of the LLF for the border node b is computed, and attached to b. Then, all non-border nodes of the left cycle are removed, and there are no constraints referring to their LLFs left in the constraint graph after this removal. Instead, the removed nodes and their corresponding constraints (as well as the constraint c_4) have been subsumed by predicate R_b, which only talks about the LLF V_b of the border node. After the reduction, only one cycle remains. Therefore, to prove GAS of the entire graph, one would then only have to solve the constraint system consisting of c_5 to c_7, c_{12} to c_{15}, and R_b.

Thus, two sub-graph-local computations have taken the place of one global computation. We call the nodes that have such predicates R_b attached *reduced nodes*.



(a) before reduction



(b) after reduction

Figure 4.14: Constraint Graphs before and after Cycle Reduction

Definition 4.11 (Reduced Nodes)**.** A *reduced node* b corresponding to cycle C in a constraint graph is a node that is labeled with a predicate R_b which has been obtained by collapsing cycle C. For a reduced node b, define *cycle*(b) as the corresponding cycle C.

Recall that border nodes (as defined in Definition 4.9) of cycle C with respect to some cycle cover are the nodes of C that are also part of other cycles in the cover. During a reduction step, all non-border nodes of a cycle are removed, and the predicate R_b is attached to the border node. Therefore, the reduction procedure turns border nodes into reduced nodes. One can simply view reduced nodes as border nodes to which the reduction has already been applied.

Assume that the SCC contains at least one cycle. The computational method is the summed up in Algorithm 1. The equivalent transformation in line 3 will be described in detail later in this section, as well as methods for arriving at the predicate R_b, as required in line 10.

In case no Lyapunov function is found in lines 7 or 10, this could have several causes:

```
repeat
    if there are no cycles with at most one border node then
        equivalently transform the graph to produce one;
    end
    select a cycle $C$ with at most one border node;
    if cycle $C$ has no border nodes then
        compute a GLF for $C$;
    end
    if cycle $C$ has one border node $b$ then
        compute a conic constraint $R_b$ on the free parameters of $V_b$ that implies
        the existence of a GLF;
        replace the constraint for $b$ in the constraint graph by $R_b$, thereby
        turning $b$ into a reduced node;
    end
    remove all non-border nodes of the cycle $C$ from the graph, together with all
    edges of the cycle $C$;
until graph is empty;
```

Algorithm 1: Reduction Algorithm Outline

- the cycle C does not have a GLF of the chosen parametrization: This can be checked by disregarding all constraints R_b from previously reduced cycles and replacing them by the original constraints on the reduced nodes before the reduction. In Figure 4.14(b), this would mean that also the constraint system consisting of c_4 to c_7 and c_{12} to c_{15} has no solution. If there is no solution even in this case, then on can attempt to use a different Lyapunov function parametrization for some of the nodes in C, or possibly the system is in fact unstable. In both cases, we now have the information that changes need to be made to C (either the hybrid automaton itself or the Lyapunov function parametrizations) in order to successfully conduct a stability proof.

- the cycle C, disregarding all constraints R_b from previously reduced cycles, possesses GLFs but the overall automaton does not: This means that the interplay of cycle C and other cycles in the automaton causes the instability. In this case, changes to LLF parametrizations or system behavior in any of the involved cycles can potentially result in a successful stability proof. The culprit is not a simple cycle, but the non-simple cycle which is the union of the involved cycles.

- the overall hybrid automaton possesses GLFs which are not found because the predicate R_b is too conservative: In this case, we can backtrack to *cycle*(b) and refine the R_b. In the context of Figure 4.14(b), we would try to identify a better predicate R_b by looking at the first cycle again, in the hope that this allows the GLF computation for the second cycle.

- the overall hybrid automaton possesses GLFs which are not found because a border node predicate computed earlier was too conservative: This can for instance occur if the system consists of three cycles in a row and the problem is caused by an inadequate predicate R_b for the first cycle. This means that further backtracking is needed.

These points are all discussed in detail in Section 4.7, where a refinement and backtracking procedure that can also detect the guaranteed non-existence of GLFs for two intersecting cycles is presented.

The following theorem forms the foundation of the computation of the predicates R_b in line 10 of Algorithm 1. Note that it works not only for decomposition into cycles, but also into arbitrary sub-graphs, as long as they only intersect in one node.

Theorem 4.3 (Decomposition within SCCs and Stability)**.** Let

$$H = (\mathcal{M}, \mathcal{S}, \mathcal{V}, \mathcal{T}, \mathit{Flow}, \mathit{Inv}, \mathit{Init})$$

be a hybrid automaton, with two sub-automata

$$C_1 = (\mathcal{M}_1, \mathcal{S}_1, \mathcal{V}_1, \mathcal{T}_1, \mathit{Flow}_1, \mathit{Inv}_1, \mathit{Init}_1)$$

and

$$C_2 = (\mathcal{M}_2, \mathcal{S}_2, \mathcal{V}_2, \mathcal{T}_2, \mathit{Flow}_2, \mathit{Inv}_2, \mathit{Init}_2),$$

and let $b \in \mathcal{M}$ be a mode of H such that:

- $\mathcal{M}_1 \cup \mathcal{M}_2 = \mathcal{M}$,
- $\mathcal{T}_1 \cup \mathcal{T}_2 = \mathcal{T}$,
- $\mathcal{M}_1 \cap \mathcal{M}_2 = \{b\}$, and
- $\mathcal{T}_1 \cap \mathcal{T}_2 = \emptyset$.

Let the $V^i_{C_1}, 1 \leq i \leq k$, be a family of GLF for sub-automaton C_1. If there exists a GLF $V_{C_2} : \mathcal{S} \times \mathcal{M}_2 \to \mathbb{R}$ for sub-automaton C_2 such that the constraint R_b defined as

$$R_b :\Longleftrightarrow V_{C_2}(\cdot, b) \in \mathit{cone}(\{V^i_{C_1}(\cdot, b) | 1 \leq i \leq k\})$$

holds, then the hybrid automaton H is GAS.

Proof. We prove that there exists a GLF for H. Per assumption, for each i, $V^i_{C_1}$ is a GLF for sub-automaton C_1. Per Theorem 3.6, this implies that all $V \in \mathit{cone}\left(\left\{V^i_{C_1} \;\middle|\; 1 \leq i \leq k\right\}\right)$ are also GLFs for sub-automaton C_1. Additionally, V_{C_2} is a GLF for sub-automaton C_2 and $V_{C_2}(\cdot, b) \in \mathit{cone}\left(\left\{V^i_{C_1}(\cdot, b) \;\middle|\; 1 \leq i \leq k\right\}\right)$. Let the λ_i be the multipliers of $V_{C_2}(\cdot, b)$ in this conic set. Then, define the function $V : \mathcal{S} \times \mathcal{M} \to \mathbb{R}$ with

$$V(\cdot, m) = \begin{cases} \sum_{i=1}^k \lambda_i V^i_{C_1}(\cdot, m) & \text{if } m \in \mathcal{M}_1 \setminus \{b\} \\ V_{C_2}(\cdot, m) & \text{if } m \in \mathcal{M}_2 \end{cases}$$

All constraints in $\mathit{constr}(C_2)$ are fulfilled by V, as they are already satisfied by V_{C_2}. The constraints in $\mathit{constr}(C_1)$ are fulfilled by each $V^i_{C_1}$ and therefore also by their weighted sum with respect to the λ_i. Since $\mathit{constr}(H) = \mathit{constr}(C_1) \wedge \mathit{constr}(C_2)$, this implies that V is a GLF for H. $\square$

Since the LLF constraints for mode b are the same for both sub-graphs, they can actually be dropped for C_2 and replaced by the R_b above. Every function in the conic hull of the $V^i_{C_1}(\cdot, b)$ will already be a LLF for b, also in the constraint system for C_2. Therefore, R_b implies that $V_{C_2}(\cdot, b)$ is a LLF and the original constraints for b become redundant for C_2.

The constraint R_b requires the LLF for mode b for the second sub-automaton to be inside the conic hull of the LLFs for mode b for the first sub-automaton. This means that we require the existence of non-negative multipliers λ_i (at least one of which is strictly positive), such that the weighted sum of the $V^i_{C_1}(\cdot, b)$ (which is always a LLF for b), is the LLF $V_{C_2}(\cdot, b)$ for C_2. For a scenario with two intersecting cycles, the implication of this theorem is as follows. For the first cycle C_1, we need to compute a number of different GLFs $V^i_{C_1}$, ideally such that their conic hull covers a large portion of the solution space. For the second cycle C_2, we then simply use a different parametrization for the LLF of the border node. This LLF is parametrized as

$$V_{C_2}(\cdot, b) = \lambda_i V^i_{C_1}(\cdot, b).$$

Here, the multipliers λ_i become the new free parameters for the LLF, with the additional restriction that they must be non-negative and at least one of them positive. The number of the GLFs $V^i_{C_1}$ and of the multipliers λ_i can be chosen freely. The more GLFs we compute for C_1, the closer we can approximate the actual solution set by the conic hull. On the other hand, each additional GLF introduces another free variable λ_i for the second cycle C_2. Therefore, the replacement of a border node constraint by a conic, polytopic constraint does only incur local, simple changes to the constraint system. If the GLFs $V^i_{C_1}$ are well chosen, not many corner points are needed and we can keep the constraint system for C_2 small.

Representing the under-approximation of the solution set by a predicate corresponding to a conic polytope has one additional advantage: it is straightforward to add this constraint to an LMI problem for a neighboring cycle. This is discussed in Section 4.5.

If we can arrange the cycles of an SCC in a partial order, then the reduction process can simply be applied repeatedly. Imagine a SCC which simply consists of a number of cycles C_i, where C_i and C_{i+1} always share a single border node and no other cycle intersection exists (as in Figure 4.15(a)). In this case, we can start by applying the reduction procedure to C_1. We compute a number of GLFs for C_1 and obtain the conic predicate R_{n_4} for the node n_4, which then takes the role of a reduced node. Then, we take the constraint system for C_2 (containing the predicate R_{n_4}), and obtain with a number of GLFs for C_2. Due to the additional constraint R_{n_4}, all of these GLFs have the property that there always exists a compatible GLF for C_1, and therefore a GLF for the graph $C_1 \cup C_2$. We then obtain a new predicate R_{n_7} for the reduced node n_7, which is taken into account for C_3. Once the constraint system for C_3 is solved, we terminate.

If, instead of a "list of cycles," the graph is a "tree of cycles" (see Figure 4.15(b)), the procedure works the same, except that one cycle can now contain more than one reduced node. We would start with one of the three cycles with just one border node (C_1, C_2, or C_4). In case of C_2 or C_4, we then obtain a list of cycles after one reduction step. If it is C_1, then we again have the choice between three cycles for the next reduction (C_2, C_3, or C_4).

However, at this point, this approach is not applicable in situations where

- two cycles overlap in more than one node, or
- the cycles are again connected in a cyclic pattern.

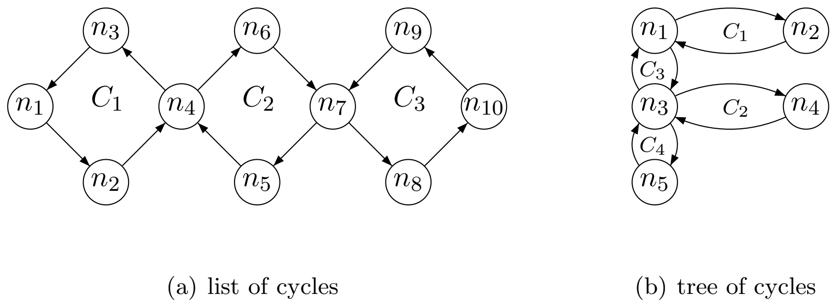
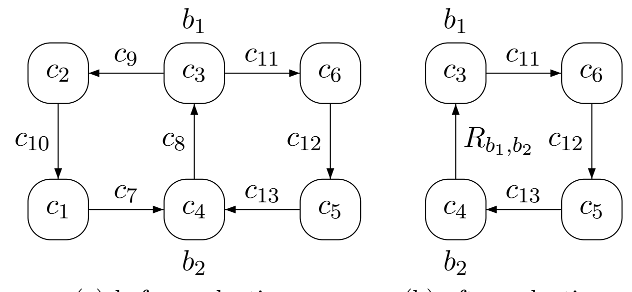


Figure 4.15: List and Tree of Cycles

First, consider the case where two cycles are intersecting in more than one node. For instance, assume that they intersect in border nodes b_1 and b_2. See Figure 4.16 for an illustration. Then, it is *not* in general sufficient to compute a P_{b_1} and a P_{b_2} and attach these to the nodes. The reason is that not every LLF computed in this manner for border node b_1 will be compatible with every LLF for border node b_2. Instead, we would need to under-approximate the Cartesian product of the two solution spaces for V_{b_1} and V_{b_2}. This is always correct, since again convexity can be exploited in the same way as above. However, in this case we have to under-approximate a solution set of double dimension, which will naturally result in greater losses and more computation steps. Therefore, the result would be a predicate R_{b_1,b_2} containing information about both modes at once. So we cannot simply attach this predicate to a node upon reduction, but have to attach it to an edge connecting b_1 and b_2 in the constraint graph (creating such an edge if it does not exist). If the two sub-graphs even overlapped in three nodes, the result would be an under-approximation in tripled dimension and a new hyperedge between the three nodes. While this approach is generally possible, there is another way of dealing with multiple overlappings: equivalent transformations of the hybrid automaton.

Here, the idea is to reduce the number of border nodes per cycle. This is achieved by choosing a border node and splitting it into several nodes, each having a degree of two, with only one incoming and one outgoing edge. In order to keep the semantics of the hybrid automaton (i.e., with respect to trajectories) unchanged, every combination of incoming and outgoing edge that is possible in the original automaton results in a



(a) before reduction (b) after reduction

Figure 4.16: Cycles with Two Border Nodes

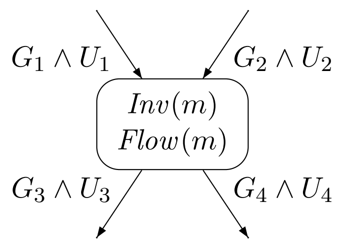
new mode with this particular combination of incoming and outgoing edge. Therefore, all trajectories of the original automaton will also remain possible in the transformed automaton. This means that applying such a transformation always correctly preserves global asymptotic stability.

See Figure 4.17 for an illustration of the splitting procedure for a single mode with indegree two and outdegree two. Figure 4.17(a) shows a mode with two incoming and two outgoing transitions and Figure 4.17(b) the result of the splitting. Each of the $2 \cdot 2 = 4$ new modes receives the same invariant and flow as the original mode. Furthermore, each new mode is connected by one incoming and outgoing edge, each of them corresponding to a transition of the original automaton. The source mode for the incoming transition and the target mode for the outgoing transition remain the same as before. Furthermore, each combination of incoming and outgoing transition is represented with exactly one of the new modes.

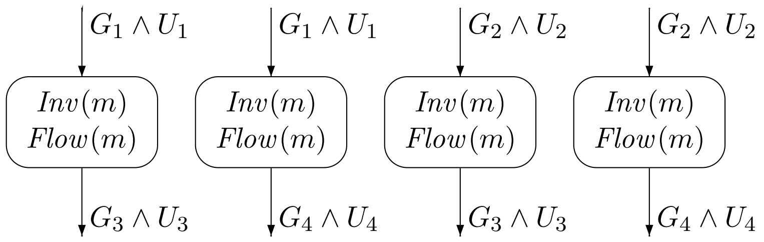
After splitting, it might also be possible to exclude some of the new modes, because their outgoing transition is not reachable from their incoming transition. This can for instance be done by computing a LLF for the new mode and using it to obtain a barrier certificate (see Section 3.4). It it is indeed unreachable, then the new mode can be removed immediately, provided that this does not happen to all modes with the same incoming transition. This is to make sure that the case where we enter the original mode through one of the incoming transitions without ever take an outgoing transition is still covered in the new automaton. We require at least one mode with every incoming edge to ensure this.

Apart from their role in reachability computations to tighten guards and invariants, initial sets should *not* be treated as an additional incoming edge. Instead, all new modes obtained through splitting should simply inherit the initial "edge" in the graphical depiction of the hybrid automaton from the original mode. This is because the knowledge of possible initial states has no use in Lyapunov-based stability verification beyond a priori reachability analysis. Remember that initial states are not represented in the Lyapunov constraints and therefore have no effect on a system's constraint graph.

Since the splitting operations take place on the hybrid automaton itself, they can also simply be mirrored on the constraint graph. The newly acquired nodes inherit the



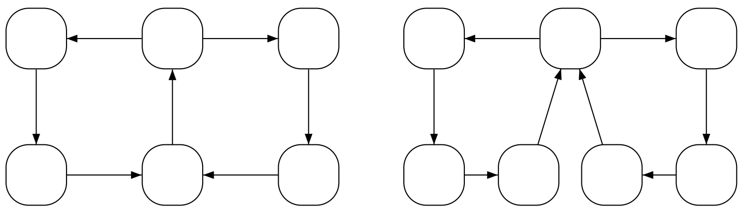
(a) before splitting



(b) after splitting

Figure 4.17: Splitting a Node

constraint from the original node (including predicates R_b). The same holds for the edges. Therefore, all new nodes obtained as a result of splitting a reduced node will be reduced nodes corresponding to the same cycle.



(a) before splitting (b) after splitting

Figure 4.18: Eliminating Border Nodes through Node Splitting

Figure 4.18 shows how a border node can be eliminated through this procedure. Figure 4.18(a) shows a graph with a simple cycle cover consisting of two cycles with length four. These cycles intersect in two nodes. By splitting one of these two border nodes in two (as its indegree is two and its outdegree one), we obtain two new modes, each inheriting invariants and guards from the old one, as depicted in Figure 4.18(b). Each of these new nodes has degree two, and there is only one border node remaining. Therefore, we can now apply the reduction procedure on one of the two cycles.

Theorem 4.4 (Splitting Modes)**.** Let $H = (\mathcal{M}, \mathcal{S}, \mathcal{V}, \mathcal{T}, \mathit{Flow}, \mathit{Inv})$ be a hybrid automaton, and let $\tilde{m} \in \mathcal{M}$ be a mode of H. Let

$$o(\tilde{m}) = \{o_0, \ldots, o_{k-1}\}$$

be the set of all outgoing transitions of $\tilde{m}$ and let

$$i(\tilde{m}) = \{i_0, \ldots, i_{l-1}\}$$

be the set of all incoming transitions. Let $q := k \cdot l$ be the product of the indegree and the outdegree of $\tilde{m}$. H is globally asymptotically stable if and only if the hybrid automaton

$$H' = (\mathcal{M}', \mathcal{S}', \mathcal{V}', \mathcal{T}', \mathit{Flow}', \mathit{Inv}')$$

with

1. $\mathcal{M}' = (M \;-\; \{\tilde{m}\}) \cup \{\tilde{m}_0, \ldots, \tilde{m}_{q-1}\}$,
2. $\mathcal{S}' = \mathcal{S}$,
3. $\mathcal{V}' = \mathcal{V}$,
4. $\mathcal{T}' = \mathcal{T} \;-\; \{(m, m', G, U) \in \mathcal{T} \,|\, m = \tilde{m} \;\vee\; m' = \tilde{m}\} \cup$
   $\{(\tilde{m}_n, m', G, U) \,|\, 0 \leq n \leq q-1 \wedge o_j = (\tilde{m}, m', G, U) \in o(\tilde{m}) \;\wedge\; n \,\text{modulo}\, k = j\}$
   $\cup\; \{(m, \tilde{m}_n, G, U) \,|\, 0 \leq n \leq q-1 \wedge i_j = (m, \tilde{m}, G, U) \in i(\tilde{m}) \;\wedge\; \lfloor n/k \rfloor = j\}$,
5. $\mathit{Flow}'(m) = \mathit{Flow}(m)$ for $m \notin \{\tilde{m}_1, \ldots, \tilde{m}_q\}$ and $\mathit{Flow}'(\tilde{m}_1) = \ldots = \mathit{Flow}'(\tilde{m}_q) = \mathit{Flow}(\tilde{m})$, and
6. $\mathit{Inv}'(m) = \mathit{Inv}(m)$ for $m \notin \{\tilde{m}_1, \ldots, \tilde{m}_q\}$ and $\mathit{Inv}'(\tilde{m}_1) = \ldots = \mathit{Inv}'(\tilde{m}_q) = \mathit{Inv}(\tilde{m})$

is globally asymptotically stable.

Proof. We will show that the hybrid automata H and H' permit exactly the same trajectories $x(t)$. Since the system remains unchanged outside mode $\tilde{m}$ and its incident edges, we only need to show that all trajectories passing through $\tilde{m}$ in the original system H have an equivalent trajectory passing through one of the modes $\tilde{m}_n$ in hybrid automaton H' and vice versa.

First, show that all trajectories of hybrid automaton H have an equivalent trajectory in hybrid automaton H'. Let $x(t)$ be a trajectory of hybrid automaton H. Without loss of generality, assume that the associated mode sequence (m_i) will eventually enter mode $\tilde{m}$. Let $T_1 = (m_1, \tilde{m}, G_1, U_1) \in \mathcal{T}$ be the transition through which mode $\tilde{m}$ is entered.

Then, by construction of $\mathcal{T}'$, there exist $k-1$ different modes $\tilde{m}_n$ with incoming transitions $(m_1, \tilde{m}_n, G, U)$. Furthermore, each of these modes $\tilde{m}_n$ has the same flow and invariant as mode $\tilde{m}$. Therefore, if (m_i) does not leave mode $\tilde{m}$, the property is shown. If (m_i) does leave mode $\tilde{m}$ through a transition $T_2 = (\tilde{m}, m_2, G_2, U_2) \in \mathcal{T}$, then, again by construction, there exists exactly one outgoing edge $(\tilde{m}_n, m_2, G_2, U_2) \in \mathcal{T}'$ from one of the modes $\tilde{m}_n$. Therefore, whenever (m_i) enters mode $\tilde{m}$, there exists an equivalent of the corresponding trajectory segment in hybrid automaton H'.

The argument for the existence of a corresponding trajectory in hybrid automaton H for each trajectory in hybrid automaton H' is similar. Let $x(t)$ be the trajectory of hybrid automaton H' with mode sequence (m_i). Whenever (m_i) enters one of the modes $\tilde{m}_n$, there is an equivalent to the transition in hybrid automaton H, leading into mode $\tilde{m}$. Since mode $\tilde{m}$ has the same flow and invariant as all the modes $\tilde{m}_n$, there is a corresponding trajectory segment in hybrid automaton H. If the mode $\tilde{m}_n$ is left again through some transition, there is a corresponding transition with the same guard, update, and target mode in hybrid automaton H.

□

The goal of this transformation is to reduce the number of intersections of cycles in the simple cycle cover. Splitting a node is a method of explicitly separating a system's mode sequences (which correspond to the paths of the underlying graph) into disjoint paths. This approach can be used to produce cycles that only overlap with the rest of the graph in exactly one node. The reduction procedure can then be applied to these cycles. If the splitting is done correctly (how this must be done is detailed later in this section), then this procedure will always terminate. A reduction step means that the cycle that was reduced can now be traversed any number of times within any path of the remaining graph — the predicate R_b ensures that such a traversal still leads to a monotonic decrease of the Lyapunov function value. The key observation is that we do not need to enumerate the (infinitely many) paths/mode sequences. If we intertwine the splitting and the reduction, it is sufficient to compute predicates R_b for *some cycles only*, regardless of the system. Any mode sequence/path of the original system can be constructed as a concatenation of these simple cycles.

There is also a different way of viewing this transformation. Since the new modes inherit all behavior from the original mode $\tilde{m}$, the two things that differ between the new modes $\tilde{m}_n$ are the incoming and outgoing transitions (only one of each) and, potentially, the Lyapunov functions. Whenever mode $\tilde{m}$ is entered through transition t_1, at the time of entering it is not clear which mode $\tilde{m}_n$ would be entered in the transformed system, since there might be several possible nodes that are all connected via an equivalent of transition t_1. Only when mode $\tilde{m}_n$ is left again via transition t_2, it is clear which mode $\tilde{m}_n$ was the right one to continue the mode sequence: the transition connected to the equivalent of t_2. All other choices lead to finite solutions, as the matching outgoing transition is missing. If mode $\tilde{m}$ is not left again, then it can be any $\tilde{m}_n$. Since all modes $\tilde{m}_n$ can have different Lyapunov functions (theoretically also with possibly different parametrizations), this choice of $\tilde{m}_n$ can be interpreted as a choice of Lyapunov function. Mapping this back on the original system, this is comparable to choosing a different LLF for mode $\tilde{m}$, *depending on how $\tilde{m}$ is entered and left.* Therefore, it is suddenly possible to have more than one LLF for a mode, and in contrast to other approaches with multiple

LLFs per mode (see, for instance, [Oehlerking *et al.*, 2007]), the Lyapunov function is not chosen in a static manner, based on some partitioning of the state space, but based on knowledge that is only available at runtime. This knowledge is the particular sequence of transitions that is used to enter and leave the node. Since the outgoing transition also plays a role here, when simulating a system run, it is not even clear which LLF should be used during simulation time, but only *afterwards*, when we know what the next transition is. Therefore, this node splitting approach is equivalent to the option of choosing a LLF based on the mode sequence of the particular system run, not only in the past, but also in the future. Quite possibly, this extra degree of freedom might allow for more Lyapunov function computations than without the splitting. However, this question is not approached in this thesis.

When using the splitting operation, we need to take care to apply it properly. When splitting without care, we can potentially do so in a cyclic manner. Imagine that we split mode m_1 which lies on a cycle C containing modes m_1 to m_n. After splitting m_1, m_2 will have a degree greater than two, so this mode can be split. In this manner, we can continue splitting along the cycle, until we again arrive at one of the new modes obtained by splitting m_1. It can again be split, creating an infinite sequence of splitting operations. We want to avoid this situation, since the goal is to produce cycles intersecting with the rest of the graph only in a single node. Repeated splitting along a cycle is equivalent to "unrolling" the cycle, creating unboundedly many internally disjoint cyclic paths (i.e., paths which may only overlap in the starting and end nodes), each corresponding to a number of traversals of the original cycle. These cyclic paths are of no use to us, since we are interested in creating simple cycles with a single border node and not in unrolling a cycle into its non-simple cycles.

The countermeasure is simple: At any point of the reduction process, we protect a number of modes from splitting. To avoid circular splitting, at least one mode on each simple cycle (and therefore at least one mode on each cycle overall) is protected in this manner. All other modes can be split freely. This strategy guarantees that, eventually, after some splittings and reductions, we can unprotect one of these modes. This happens because cycles with just one protected node are reduced one by one. Eventually, no such cycles will remain, allowing us to unprotect a single node and still maintain the existence of one protected node per cycle. This is repeated until only one protected mode is left. At that point, it is possible to reduce the graph to the empty graph.

A key observation is that this works for *all* SCCs, even if they contain cycles of cycles. Through the splitting process, they will be gradually unrolled, resulting in an equivalent hybrid automaton without cycles of cycles. The GLF then applies to this equivalent automaton, completing the stability proof.

Of course, in order for this to work, we have to assume that the GLF computations for the individual cycles are always successful. Section 4.7 deals with situations where this is not the case. The following theorem gives a computational procedure which is guaranteed to terminate.

Theorem 4.5. Let H be a hybrid automaton consisting of a single SCC with at least one cycle. Consider the following reduction procedure:

1. Pick a set $\mathcal{N} \subseteq \mathcal{M}_H$, such that for every cycle C of constraint graph $C(H)$ there exists a node $m \in \mathcal{N}$ with $m \in V_C$.
2. Find a conic hull of GLFs for all simple cycles in the graph which contain exactly one border node b. Reduce these cycles as per Theorem 4.3, replacing the constraint on the border node b by a conic constraint R_b.
3. If only one cycle is left in the graph, compute a GLF for it and terminate.
4. Check, whether any node $m \in \mathcal{N}$ lies only on cycles also containing other nodes in $\mathcal{N}$. If yes, pick such an m, remove it from $\mathcal{N}$, and return to Step 2.
5. If there are nodes in $\mathcal{M} - \mathcal{N}$ with degree higher than 2, then pick one, split it as per Theorem 4.4, and return to Step 2.

If all reduction steps are successful (i.e., GLFs for every reduced cycle in Steps 2 and 3 can be found), then the algorithm will always terminate, and hybrid automaton H is GAS.

Proof. After Step 1, there are no cycles in any $C(G)$ that do not contain nodes that are also in $\mathcal{N}$. Splitting operations on $\mathcal{M} - \mathcal{N}$ preserve this property, as do reductions. In Step 4, we only remove nodes from $\mathcal{N}$ when they are redundant. Therefore, we can conclude that this property always holds throughout the reduction procedure.

We will now show that, under this precondition, the check in Step 4 will eventually be successful, resulting in the removal of one node in $\mathcal{N}$. Until this is the case, Steps 2, 3 and 5 are executed in an alternating manner.

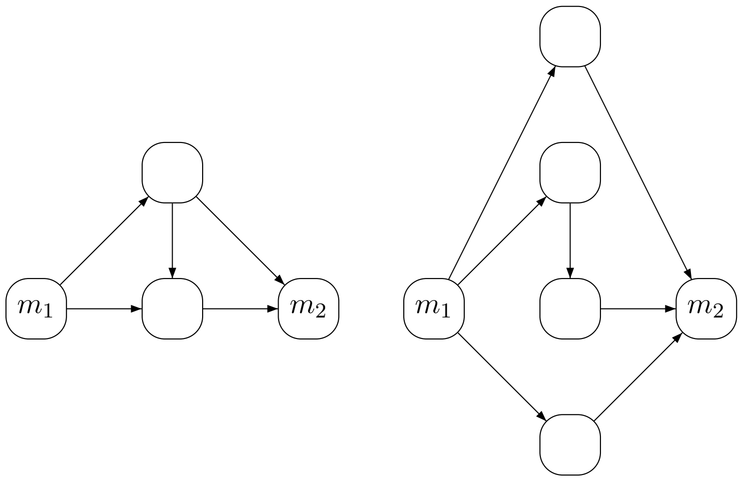
For any pair of nodes m_1 and m_2 in $\mathcal{N}$ ($m_1 = m_2$ included), consider the sub-graph G_{m_1,m_2} of G consisting exactly of all nodes and edges that lie on paths starting in node m_1, ending in node m_2 and not crossing any nodes in $\mathcal{N}$ inbetween.

Since all cycles contain at least one mode which is in $\mathcal{N}$ and since G_{m_1,m_2} cannot contain any other nodes in $\mathcal{N}$ besides the nodes m_1 and m_2, G_{m_1,m_2} cannot contain any cycles which do not contain either m_1 or m_2. This prohibits infinite application of the splitting operation in any G_{m_1,m_2}, as long as $\mathcal{N}$ remains unchanged. Splitting nodes inside a G_{m_1,m_2} as long as possible eventually results in one disjoint path from m_1 to m_2 for each (not necessarily disjoint) path from node m_1 to node m_2 in the original graph (see Figure 4.20). At that point, no further splittings are possible.

Now, consider the case $m_1 = m_2$. In this case, repeated application of the splitting as in Step 5 eventually results in a sub-graph G_{m_1,m_1} consisting only of simple cycles. Furthermore, due to the disjointness of the paths, all of these cycles will overlap with the rest of $C(G)$ in only one mode: m_1 (see Figure 4.19).

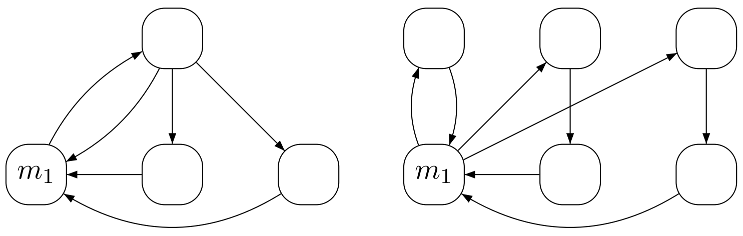
Therefore, all of these cycles are eventually reduced in Step 2, so that G_{m_1,m_1} only consists of m_1 itself. This results in a graph where all simple cycles containing node m_1 must contain at least another node from $\mathcal{N}$, as all cycles containing only node m_1 have been eliminated. Therefore, at this point we are able to remove node m_1 from $\mathcal{N}$ in Step 4.

We can then repeat the procedure, and $|\mathcal{N}|$ decreases with every repetition, until it reaches 1. Eventually, only a single cycle is left, which can then be eliminated in Step 3, causing the termination of the algorithm. By Theorems 4.3, and 4.4, this proves GAS of hybrid automaton H. $\square$



(a) before repeated splitting (b) after repeated splitting

Figure 4.19: Repeated Splitting Resulting in Disjoint Paths, $m_1 \neq m_2$



(a) before repeated splitting (b) after repeated splitting

Figure 4.20: Repeated Splitting Resulting in Simple Cycles Overlapping in $m_1 = m_2$

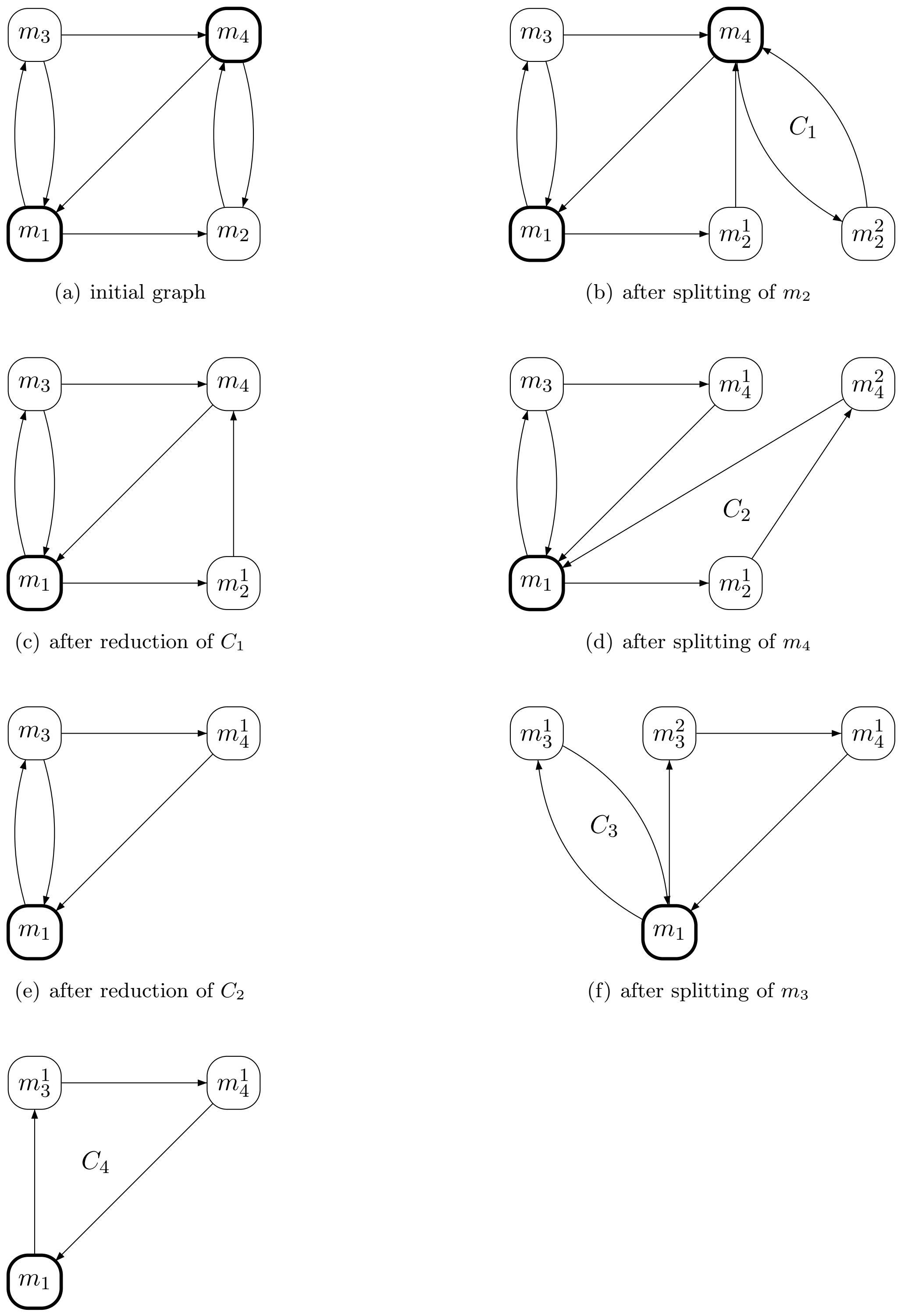
Example 4.7. Figure 4.21 shows an example for a reduction based on this computational procedure. The node labels are not the constraints, but just the names of the modes for easier reference. Initially, we set $\mathcal{N} = \{m_1, m_4\}$ (see Figure 4.21(a)). Since there are no reducible cycles and since no node in $\mathcal{N}$ is superfluous, we directly proceed to Step 5, picking a node not in $\mathcal{N}$ to be split. Both modes m_2 and m_3 can be split into two new nodes — here we picked mode m_2. The resulting graph is shown in Figure

4.21(b). The algorithm returns again to Step 2. Since cycle C_1 is now reducible, we conduct the GLF computation for this cycle. Assuming that this computation is successful, we obtain the graph in Figure 4.21(c). At this point, no further reducible cycles exist. However, we find that there is no single cycle in the graph containing mode m_4, but not mode m_1. Therefore, we can remove mode m_4 from $\mathcal{N}$ in Step 4. Since mode m_4 is now splittable, we can now proceed by splitting this node in two, resulting in the graph shown in Figure 4.21(d). Reduction of the cycle C_2 then gives us Figure 4.21(e). Then we can split m_3 (Figure 4.21(f)) and reduce cycle C_3 (Figure 4.21(g)). The result is a single cycle C_4 which can be reduced in a final step, causing the successful termination of the algorithm.

The complexity of this reduction procedure, measured by the number of GLF computations needed, relates to the size of the cycle cover that is computed implicitly. How this cycle cover looks like depends on the choices for $\mathcal{N}$ and the nodes that are split. If no splittings are needed at all, then the number of GLF computations will be in $O(|E|)$ in the worst case. If splittings are needed, then the complexity depends on the resultant graph after the splittings. Generally, the exponential growth in the number of cycles in a fully connected graph means that decomposition based on discrete states is not recommended in these cases. Instead, it is often desirable to use a common LLF for modes in such cliques in the graph, effectively treating a fully (or nearly fully) connected sub-graph as a single node. A criterion is, for instance, given in Section 4.5.

Since it is desirable to keep the constraint graph small after splitting, it is a reasonable heuristic to first split nodes with a low product of indegree and outdegree and to place nodes with a high product of indegree and outdegree in the set $\mathcal{N}$. To come up with a set $\mathcal{N}$ initially, one can, for instance, use a simple greedy approach: Initially, $\mathcal{N}$ is empty. We then look for a cycle in the graph and add its highest-degree node to $\mathcal{N}$. In the next step, we look for another cycle not containing any nodes in $\mathcal{N}$, again adding the node with the highest degree product. We continue this procedure until no further cycles of this type can be found. While this simple procedure is not guaranteed to give an optimal set $\mathcal{N}$, it is not costly and usually results in the highest-degree nodes being protected from splitting. Optimal strategies or sub-optimal strategies with guaranteed bounds on the number of reductions are beyond the scope of this thesis and can be seen as potential future work.

This decompositional proof process does not explicitly compute a GLF for the entire SCC. Instead, just the existence of this function is guaranteed, which is sufficient to conclude GAS. To obtain an actual GLF, a simple additional step is needed. For this, we need to keep track of the cycle reduction order. We do so with the help of a reduction graph, which has a node for each reduced cycle, and directed edges signifying the reduction dependencies between cycles. If a cycle C, at the time of its reduction, contains predicates R_b resulting from the reduction of other cycles C_i, an edge from C to C_i is introduced. An edge can be interpreted as representing the reduction order of two overlapping cycles. Later, in Section 4.7, we will also use this reduction graph to do backtracking-based refinement of the R_b predicate. Formally, we define this graph as follows.



(a) initial graph

(b) after splitting of m_2

(c) after reduction of C_1

(d) after splitting of m_4

(e) after reduction of C_2

(f) after splitting of m_3

(g) after reduction of C_3

Figure 4.21: Example Reduction Procedure (nodes in $\mathcal{N}$ denoted with thick lines)

Definition 4.12 (Reduction Graph)**.** A *reduction graph* is a graph which is constructed throughout the reduction process as follows.

- When reducing a cycle C_i, add a new reduction graph node representing this cycle. For ease of notation, we will identify a cycle C_i with its corresponding reduction graph node. This new cycle name must be unique.
- If, at the time of reduction, C_i contains reduced nodes, for each such node m add an edge in the reduction graph from the node labeled C_i to the node labeled with $\mathit{cycle}(m)$, and label the edge with m.
- When splitting a reduced node m in the constraint graph into j new nodes, take the sub-tree of the reduction graph rooted in the node labeled $\mathit{cycle}(m)$ and multiply it into j copies of itself.

Throughout the reduction process, the reduction graph will always be a forest, with the root nodes being the cycles corresponding to the reduced nodes that can currently be found in the constraint system. After the reduction has been completed, the graph will be a tree with the root node representing the cycle that was reduced last.

Example 4.8. Figure 4.22 shows how the reduction graph for the example of Figure 4.21 is constructed incrementally. After the reduction of cycle C_1, there is just one node in the reduction graph, labeled with C_1 (see Figure 4.22(a)). Since node m_4 is split in two new nodes in Figure 4.21(d) and since m_4 is the reduced node corresponding to cycle C_1, we duplicate the sub-tree rooted in this reduction graph node (i.e., only the node itself). Cycle C_2 contains one of these newly obtained reduced nodes, namely m_4^2. Therefore, when we reduce cycle C_2, we must add an edge from one of the reduction graph nodes labeled C_1 to the node for cycle C_2 and label it with m_4^2 (see Figure 4.22(b)). Cycle C_3 then contains reduced node m_1, with $\mathit{cycle}(m_1) = C_2$, therefore another edge must be added upon reduction of cycle C_3 (see Figure 4.22(c)). Note that the roots of the sub-trees of the reduction graph are always labeled with the cycles whose reduced nodes are currently present in the graph. At this point, we have one reduced node corresponding to cycle C_1 (the node m_4^1) and one reduced node corresponding to cycle C_3 (the node m_1). In a final step, we then reduce cycle C_4, which contains both of these nodes. Therefore, two edges must be added resulting in the final reduction graph given in Figure 4.22(d).

The reduction graph can be used to obtain a hybrid automaton H' which is equivalent to the original automaton H, but has a tree-like cycle structure. This is simply achieved by piecing together the cycles given by the nodes of the reduction graph, with each edge of the reduction graph specifying the overlapping node. In other words, two cycles overlap in node m if and only if there is an edge labeled m in the reduction graph. Note that cycles which appear multiple times in the reduction graph must also be multiplied. The time complexity of the reduction/splitting procedure in case splittings occur is in $O(|E'|)$ in the worst case, where E' is the edge set of $G(H')$.

Figure 4.23 gives the graph structure of this equivalent automaton for the reduction graph in Figure 4.22(d). The cycle C_1 appears twice, and each occurrence is allowed to

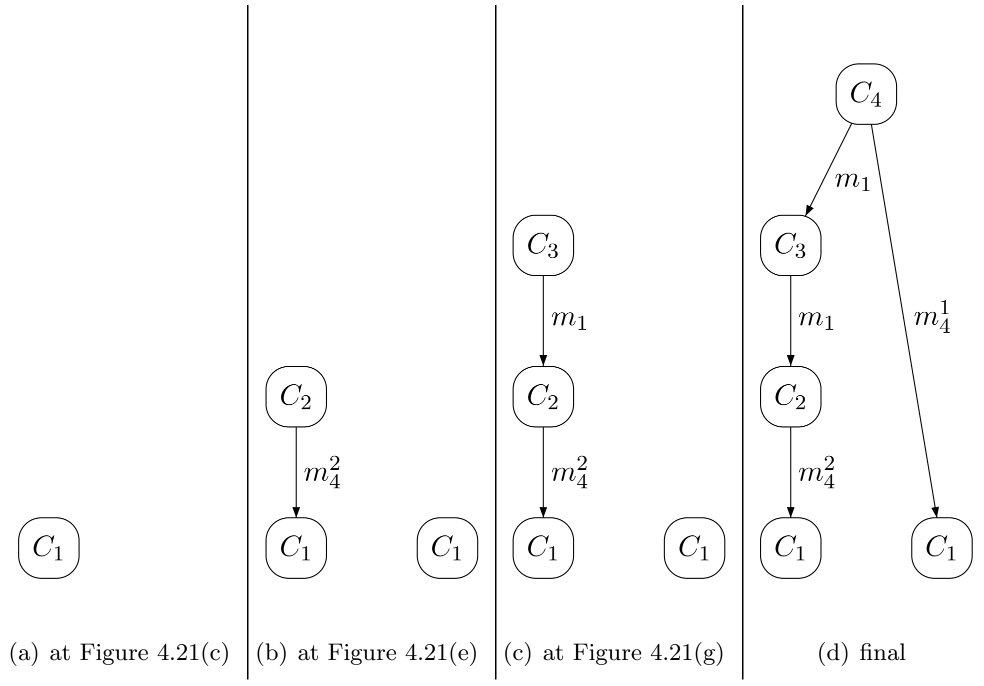


Figure 4.22: Reduction Graph Construction for the Example from Figure 4.21

have its own Lyapunov function. The GLF which we can compute for the SCC is for this equivalent hybrid automaton.

Remark 4.4 (GLF computation within an SCC)**.** In order to compute a GLF for an SCC, we also need to keep track of the following information during the reduction process:

- all the computed GLFs V_C^k for each cycle C (for the root cycle C_r, this is only one GLF $V_{C_r}^1$), and
- for each computed GLF V_C^k, the λ_i-parameters as per Theorem 4.3 for the LLF $V_C^k(\cdot, m)$ of each reduced node m in C, as $\lambda_i(C, m, k)$, such that

$$V_C^k(\cdot, m) = \sum_i \lambda_i(C, m, k) V^i_{cycle(m)}(\cdot, m).$$

These multipliers $\lambda_i(C, m, k)$ arise directly as unknowns in the constraint system and are therefore readily available. To compute a GLF, we now traverse the reduction tree top-down (i.e., starting with the last reduced cycle), and

- set the LLFs of all modes m lying on the root cycle C_r to the LLFs of the computed GLF for this cycle, that is $V(\cdot, m) = V_{C_r}^1(\cdot, m)$,
- for each outgoing reduction graph edge of C_r, labeled with m, define $\tilde{\lambda}_i(C_r, m) := \lambda_i(C_r, m, 1)$,

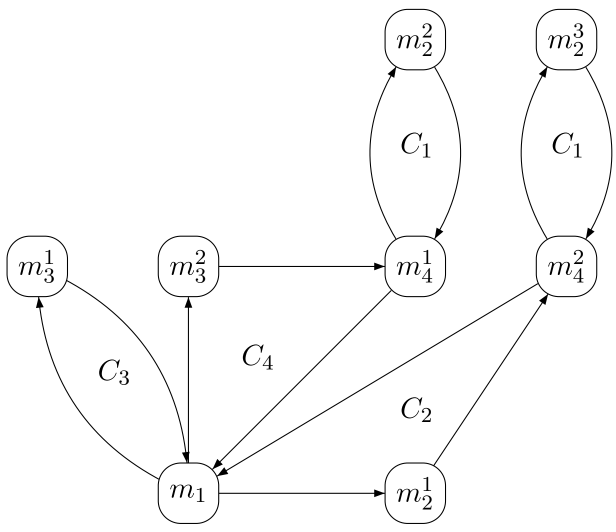


Figure 4.23: Graph Structure of a Hybrid Automaton Equivalent to Figure 4.21(a), Corresponding to Reduction Graph 4.22(d)

- for each non-root node of the reduction graph, labeled with C and connected via an edge labeled with m to a parent reduction graph node labeled with C_p:
  - for all modes m of cycle C which do not have a LLF assigned yet, set the LLF to $V(\cdot, m) = \sum_i \tilde{\lambda}_j(C_p, m) V^i_{C_p}(\cdot, m)$,
  - for all outgoing edges, labeled with m, of the node labeled with C in the reduction graph, define $\tilde{\lambda}_k(C, m) := \sum_i \tilde{\lambda}_i(C_p, m)\lambda_i(C, m, k)$

These LLFs form a GLF of the SCC. The intuition is as follows. Since the splitting procedure has resulted in a tree-shaped dependency structure between the GLFs of the cycles, we start with the GLF for the final cycle. We know that there exists a GLF for the rest of the automaton which is compatible with it. To compute this GLF, we use the same procedure that was already employed in the proof of Theorem 4.3: we generate the GLF of the neighboring cycles by using the λ_i-parameters. The actual GLF for a neighboring cycle is simply the sum of the GLFs computed during the decomposition, weighted by the λ_i induced from the parent cycle. The $\tilde{\lambda}_i(C, m)$ are used to represent the information on the GLF in the conic hull which was selected in the parent node. This information is then propagated "down the reduction graph," and the corresponding LLFs for the nodes in each cycle are selected.

For the reduction graph in Figure 4.22(d), we start with the nodes in cycle C_4 (m_1, m_3^2, and m_4^1) and assign to them LLFs from the computed GLF for C_4. The λ_i-variables used for the LLF of m_1 are then named the $\tilde{\lambda}_i(C_4, m_1)$. The LLF for the additional mode in cycle C_3 (this is only m_3^1) is then obtained by weighting the LLFs from the computed GLFs for cycle C_3 by these multipliers $\tilde{\lambda}_i(C_4, m_1)$. Since C_3 has another child cycle, we also need to compute the weighting parameters for the next level, the multipliers $\tilde{\lambda}_k(C_3, m_1)$, which are obtained as a linear combination from the multipliers $\tilde{\lambda}_i(C_4, m_1)$ and the variables $\lambda_i(C_3, m_1, k)$. Similarly, we assign a LLF to mode m_2^2 through the instance of cycle C_1 connected via mode m_4^1. Then, we can assign LLFs to modes m_2^1 and m_4^1 (though cycle C_2) and mode m_2^3 (through the second occurrence of cycle C_1), resulting in a GLF for the entire automaton in Figure 4.23.

As can be seen, no complex computations are required to obtain a LLF. In fact, if we compute multiple LLFs for the root cycle, we can even characterize an infinite set of GLFs for the SCC, by taking the conic hull of the individual GLFs computed in the above fashion (see Theorem 3.6). In general, it does not matter in which exact order the reduction graph is traversed, as long as the traversal is done top-down.

4.5 Lyapunov Functions for Single Cycles

This section describes the actual computation of Lyapunov functions and border predicates for the single cycles obtained through decomposition. We have chosen to employ the LMI-based methods described in Section 3.5, since the problem sizes are now much smaller. For systems with a moderately complex discrete transition relation, LMI methods in general behave quite well. Furthermore, LMI problems are just another representation of a particular type of convex optimization problems, and the solvers for this class of problems generally allow optimization of the solution in a particular direction. These optimal values directly correspond to the "corner points" of the border predicates. However, note that it is, of course, also possible to employ different Lyapunov function computation techniques (for instance based on linear programming or purely symbolic methods), as long as Lyapunov functions serving as extremal points of the conic under-approximation can be computed. The solution proposed in this section is to be taken as one particular choice of back-end that works well on a relatively large class of systems.

First, in Section 4.5.1, we discuss how the constraints generated by Theorem 4.3 can be expressed as LMI problems. In particular, we also describe how to identify "good" functions $V_{C_i}^i$ that are not redundant and result in acceptable coverage of the solution space. However, in this section only static strategies are discussed. Dynamic refinement of the under-approximation is then examined in detail in Section 4.7.

The second focus of this section is the avoidance of numerical problems by formulating the optimization problems in a well-posed manner. Most importantly, implicit equality constraints in the optimization problems should be avoided whenever possible, since problems containing such constraints are intrinsically numerically unstable. For instance, this occurs when the same hyperplane triggers a transition between two modes in both directions. In this case, the Lyapunov conditions from Section 3.5 implicitly require equality of the respective local Lyapunov functions. The solution to this problem is to make the equality constraint explicit, by already encoding it in the problem formulation. This problem is dealt with in Section 4.5.2.

Furthermore, there are cases where the usage of different LLFs for different modes of a sub-automaton does not make sense. Specifically, one can identify situations where

LLFs of a certain parametrization for the modes of a cycle must be equal. In this case, one should use a common Lyapunov function for the cyclic sub-automaton instead of a piecewise one. Section 4.5.3 will give a discussion of such situations.

Also, sometimes reachability analysis can be used to conclude that a cycle can in fact not be fully traversed. This can happen if the outgoing transition of a mode in the cycle is unreachable from the incoming transition. In this case, the GLF computation for the cycle becomes superfluous. This is discussed in Section 4.5.4.

4.5.1 Local Constraint Systems as LMIs

For cyclic constraint graphs not containing reduced nodes, the LMI problem can simply be formulated as in Theorem 3.10, with global Constraints (3.1) and (3.2), Constraints (3.3), (3.5), (3.6), and (3.7) per mode and Constraints (3.4) and (3.8) per edge. Suppose there is a reduced node in the constraint graph of cycle C, corresponding to hybrid system mode m. Furthermore suppose that the conic constraint R_m is of the form

$$V_C(\cdot, m) \in cone(\{V_i^m(\cdot) \mid 1 \le i \le k\}),$$

where each V_i^m is a quadratic function, such that

$$V_i^m(\tilde{x}) = \tilde{x}^T S_i^m \tilde{x}.$$

Then we can represent this constraint R_m in the LMI by adding scalar variables λ_i^m and the LMI constraints

$$\text{for all } i : \lambda_i^m - \epsilon \succeq 0$$

for some small $\epsilon > 0$. Furthermore, all occurrences of the matrix P_m in edge constraints must be replaced by the weighted sum

$$\sum_{i=1}^{k} \lambda_i^m S_i^m.$$

The matrix variable P_m, as well as all $\mathcal{S}$-procedure-variables pertaining to mode m can be dropped entirely, as can all Constraints (3.3), (3.5), (3.6), or (3.7) for mode m. This procedure is equivalent to using this weighted sum as the new parametrization for P_m, which is no longer a matrix variable with arbitrary entries, but a structured matrix which must lie in a prescribed conic set. Note that this representation is slightly conservative, as we required all multipliers λ_i^m to be strictly positive, whereas all but one multiplier could be exactly zero in the original constraint. If ϵ is small, this is however of little practical consequence. This ϵ should be chosen based on the numerical properties of the solver that is used. Typically, ϵ will lie in the area of 10^{-6} to 10^{-8}, for double precision floating point software like CSDP, as a guard against numerical inaccuracies inherent to the solvers. Since a good value of ϵ will also sometimes depend on the problem itself, it is also possible to try various ϵ-values, followed by a check of the solution obtained, until a satisfactory solution is found.

Another question that remains to be answered is how to compute the individual LLFs V_i^m during the decomposition process. Fortunately, SDP solvers allow not only for the formulation of constraints as LMIs, but also for *linear objective functions* on any of the free variables. In particular, this includes the free parameters of the Lyapunov functions, that is, the entries of the P-matrices. Therefore, it is possible to identify solutions (i.e., Lyapunov functions) which maximize/minimize the individual free parameters, yielding solutions that always lie on the boundary of the solution set in some direction. Since we are interested only in keeping the LLFs V_i^b of the border node b, we only need to chose optimization directions based on the free parameters of $V(\cdot, b)$. Their conic hull is then an under-approximation of all LLF for border node b which are part of at least one GLF for the entire cycle. Furthermore, if this computational approach is used, the corner points of the convex set will be extremal points of the exact set of admissible LLFs for b, yielding better under-approximations than if some internal points were used. However, one must take care when doing these computations, for several reasons.

As the solution set is not only convex, but also conic, it is natural that extremal points in some directions do not exist. In this case, the SDP solver will usually identify the optimization problem as infeasible, as there is no optimum in the specified direction. This can be alleviated by fixing the values of a single free LLF parameter relative to other parameters of the same LLF (e.g., by setting it to some positive number minus the sum of the other parameters), thereby just considering a “slice” of the conic solution set, or by artificially constraining the solution set with absolute bounds on the parameter values.

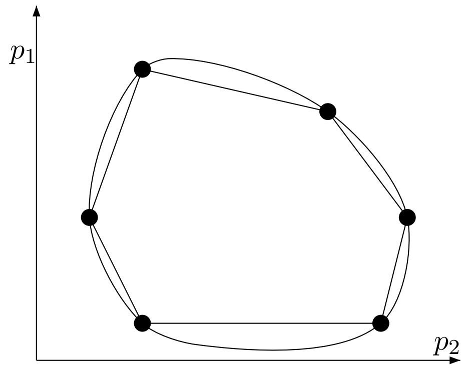


Figure 4.24: Normal-Boundary Intersection of a Two-dimensional Set with Two Variables p_1 and p_2

In mulitobjective optimization, this idea is also called *normal-boundary intersection* [Das & Dennis, 1998]. See Figure 4.24 for an illustration of an under-approximation of a convex set by the convex hull of some extremal points in various optimization directions. It is important that these optimization directions are spaced appropriately, as to avoid too conservative under-approximations. However, in general, “good” spacings are determined by the shape of the set to be under-approximated, so the choice of directions must be handled heuristically.

To avoid exponential blowup, a possible starting point for high-dimensional problems is the choice of two optimization directions per LLF parameter of the border node: one maximizing the variable and one minimizing it. This means that the representation of the under-approximation will grow linearly with the number of variables. A procedure for adding new directions while exploiting information on the automaton is then presented in Section 4.7. Since LLFs can have many free variables, especially if the sums-of-squares decomposition is employed, it is in general desirable to keep the magnitude of the number of optimization directions small, if at all possible. If the LLF V_b is parametrized as $\bar{x}^T P \bar{x}, P \in \mathbb{R}^{n \times n}$, then there are $n(n+1)/2$ free variables and $n(n+1)$ optimization directions to be covered. We can then simply solve the LMI problem $n(n+1)$ times for the different directions, yielding $n(n+1)$ extremal points spanning the convex cone for the LLFs of the border node. If this is not sufficient to prove GAS of the system, then the refinement procedures described in Section 4.7 can be used to add further optimization directions.

4.5.2 Continuous versus Discontinuous Lyapunov Functions

When two modes of a hybrid system are connected by transitions in both directions, and both of these transitions have the same guard set which can be represented by a hyperplane and no discrete update, then the two LLFs must be equal on this boundary. In this case, it is advisable to set the corresponding LLFs equal *a priori*, instead of letting the SDP software find a solution where the corresponding constraints are fulfilled. The reason is simple: the software *approximates* a solution, and if the actual solution is of lower dimension than the solution space, it usually returns something that is a marginal non-solution, slightly violating the constraints. This is the case in the scenario outlined above, as there is a reduced number of degrees of freedom: the LLF of the second mode can only take parameters that match the LLF of the first mode in the sense that equality on the boundary results.

The countermeasure is to encode this reduced dimensionality already in the LMI problem formulation. For linear guards and quadratic LLFs this works as follows: The Lyapunov function for one of the two modes is encoded normally in the LMI problem. However, we explicitly define the Lyapunov function for the other mode as the weighed sum of the Lyapunov function of the first mode and some matrix terms dependent on the guard. These matrix terms are selected such that the corresponding quadratic function will be zero on the guard set. The result is a LLF for the second mode which is equal to the LLF on the first mode on the guard set. Moreover, in the linear guards/quadratic LLFs case, these matrices are not only easy to to compute, but this approach is also lossless. This means that all LLFs for the second mode that can conceivably be part of the GLF for the cycle can be constructed in this manner.

Theorem 4.6 (Continuous Piecewise Lyapunov Functions)**.** Let G be a set of the form $G = \{x \in \mathbb{R}^n \mid c^T x + d = 0\}$ for some non-zero vector $c \in \mathbb{R}^n$ and some scalar $d \in \mathbb{R}$.

Let $e_i \in \mathbb{R}^{n+1}$ be the i-th canonical basis vector of $\mathbb{R}^{n+1}$. Define

$$Q_i(c,d) := e_i[c^T, d] + \begin{bmatrix} c \\ d \end{bmatrix} e_i^T$$

Let V_1 be a function of the form

$$V_1(x) = [x^T, 1] \; P_1 \begin{bmatrix} x \\ 1 \end{bmatrix}$$

for some symmetric $(n+1) \times (n+1)$-matrix P. Then, for all

$$V_2(x) = [x^T, 1] \; P_2 \begin{bmatrix} x \\ 1 \end{bmatrix}$$

such that $V_1(x) = V_2(x)$ for all $x \in G$, there exists a family of multipliers $\lambda_i \in \mathbb{R}$ such that

$$P_2 = P_1 + \sum_i \lambda_i Q_i(c,d).$$

Proof. Let V_1 be a function as above. Assume without loss of generality that $c_{(1)} \neq 0$. Note that $c^T x + d = 0$ is equivalent to $x_{(1)} = \sum_{j=2}^{n}(-c_{(j)}/c_{(1)}x_{(j)}) - d/c_{(1)}$. Define $C := [-c_{(2)}/c_{(1)}, \ldots - c_{(n)}/c_{(1)}, -d/c_{(1)}]$ and let I be the $n \times n$ identity matrix. Furthermore, define

$$\bar{P} := [C^T, I] P \begin{bmatrix} C \\ I \end{bmatrix}$$

for some symmetric matrix $P \in \mathbb{R}^{n+1 \times n+1}$. We will now treat P as a matrix of unknowns and consider the constraint system $\bar{P} = 0$. Since P is symmetric, it has $(n+1)(n+2)/2$ unknowns. Multiplying out $\bar{P}$ results in a matrix of dimension $n \times n$. $\bar{P} = 0$ is equivalent to all matrix entries of $\bar{P}$ being 0, and $\bar{P}$ has $n(n+1)/2$ distinct entries. Therefore, identifying values for the entries in P such that $\bar{P} = 0$ is a linear equality system with $(n+1)(n+2)/2$ parameters and $n(n+1)/2$ constraints. We then obtain

$$\bar{P}_{(i,j)} = c_{(i)}c_{(j)}P_{(1,1)} + c_{(i)}P_{(1,j+1)} + c_{(j)}P_{(i+1,1)} + P_{(i+1,j+1)}.$$

Due to the last summand this means that all constraints $\bar{P}_{(i,j)} = 0$ are linearly independent. Therefore, the solution space for $\bar{P} = 0$ has dimension $(n+1)(n+2)/2 - n(n+1)/2 = (n^2 + 3n + 2 - n^2 - n)/2 = n + 1$. We will now show that for all i, the matrix $-Q_i(c,d)$ is a P that solves $\bar{P} = 0$. Inserting $-Q_i(c,d)$ for P, we obtain:

$$\bar{P} := [C^T, I](-Q_i(c,d)) \begin{bmatrix} C \\ I \end{bmatrix} =$$

$$[C^T, I]\left(-e_i[c^T, d] - \begin{bmatrix} c \\ d \end{bmatrix} e_i^T\right) \begin{bmatrix} C \\ I \end{bmatrix} =$$

$$-[C^T, I]e_i[c^T, d] \begin{bmatrix} C \\ I \end{bmatrix} - [C^T, I] \begin{bmatrix} c \\ d \end{bmatrix} e_i^T \begin{bmatrix} C \\ I \end{bmatrix}$$

This term is equal to zero, since

$$[c^T, d] \begin{bmatrix} C \\ I \end{bmatrix} = 0.$$

Since the $n+1$ matrices $-Q_i(c,d)$ are also linearly independent, they form a basis for the solution space of $\bar{P} = 0$. Therefore, every solution for $\bar{P} = 0$ can be constructed as $P = \sum_i \lambda_i(-Q_i(c,d))$ for some family of $\lambda_i \in \mathbb{R}$. Now set $P = P_1 - P_2$. Observe that for all $x \in \mathbb{R}^n$, the vector

$$\bar{x} = \begin{bmatrix} C \\ I \end{bmatrix} x$$

will lie in G. Therefore, $x \in G \implies x^T(P_1 - P_2)x = 0$ implies that $\bar{P} = 0$. For all $P_1 - P_2$ which are equal on G (when interpreted as quadratic functions), the $-Q_i(c,d)$ then form a basis. This means that there exists a family of $\lambda_i \in \mathbb{R}$, such that $P_1 - P_2 = \sum_i \lambda_i(-Q_i(c,d))$, which concludes the proof. $\square$

Consider a hybrid system with just two modes m_1 and m_2 and two transitions, given as (m_1, m_2, G_1, I) and (m_2, m_1, G_1, I), where $G_1 = \{x \in \mathbb{R}^n \mid c_1^T x + d_1 = 0\}$ is a guard set, and I is the identity function. In this case, the LLF $V(x, m_1)$ can be parametrized as a quadratic function in the usual manner, as

$$V(x, m_1) = [x^T, 1]\; P_1 \begin{bmatrix} x \\ 1 \end{bmatrix}.$$

Then, it is losslessly possible to set

$$V(x, m_2) = [x^T, 1] \left(P_1 + \sum_i \lambda_i Q_i(c_1, d_1) \right) \begin{bmatrix} x \\ 1 \end{bmatrix}$$

with free parameters λ_i. With this parametrization, equality of the two Lyapunov functions for $c^T x + d = 0$ is automatically preserved. Furthermore, $V(x, m_2)$ requires only $n+1$ free parameters, instead of $(n+2)(n+1)/2$. If there is a third mode m_3, connected with mode m_2 by two transitions (m_2, m_3, G_2, I) and (m_3, m_2, G_2, I) with $G_2 = \{x \in \mathbb{R}^n | c_2^T x + d_2 = 0\}$, then another sum of matrices and parameters can be added:

$$V(x, m_3) = [x^T, 1] \left(P_1 + \sum_i \lambda_i Q_i(c_1, d_1) + \sum_i \mu_i Q_i(c_2, d_2) \right) \begin{bmatrix} x \\ 1 \end{bmatrix}$$

The LMI constraints introduced by Johansson & Rantzer [1998] exhibit some similarities to this approach, as equality of LLF on the switching surfaces is also enforced a priori. In their work, this was achieved by setting the LLFs in a *fixed* relation depending on the switching surfaces. This was achieved by parametrizing the LLFs as $x^T F_m^T T F_m x$, where F describes the switch surface, and T is a matrix of free parameters shared by all modes. In contrast to this, in Theorem 4.6, the LLFs of different modes are not in a fixed relation. Instead, the relation is governed by the choice of the multipliers λ_i, yielding extra degrees of freedom.

A particular class of systems where this approach is useful are regular grids. Suppose that some complex non-linear differential equation has been approximated by a gridding approach: put a Cartesian grid over the state space and turn every box into a mode of a hybrid automaton, each of them with relatively simple (e.g., linear/affine) dynamics. In this case, we have precisely the scenario as described above. Two neighboring boxes always form a cycle of length two, as they are connected by bidirectional edges, with a guard set corresponding to the boundary of the boxes (see Fig. 4.25).

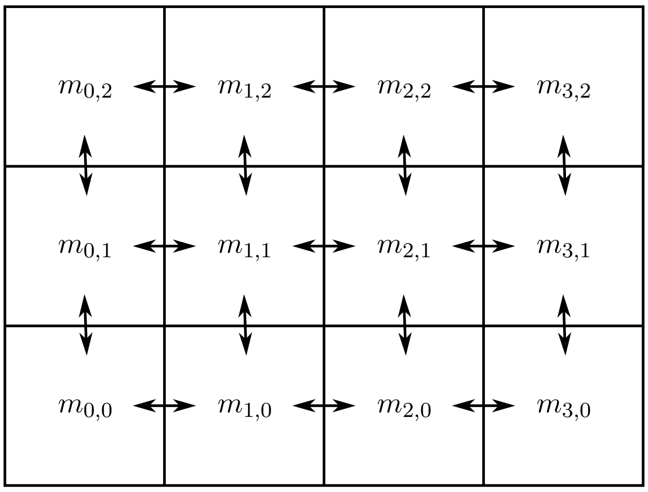


Figure 4.25: Regular Grid with $4 \cdot 3$ Modes

In this case, we can assign a LLF parametrization to a single box, and then use the result of Theorem 4.6 to obtain an LMI system with relatively few free variables. For each column of the grid, free variables λ_i as described above are introduced. For each row, introduce free variables μ_i.

Assume that the $m_{i,j}$ are the modes of the system, where i is the column and j is the row. If $m_{0,0}$ is the single box whose LLF receives the full parametrization. The LLFs of other boxes of the 0-th column are then obtained from the function $V(\cdot, m_{0,0})$ and the weighted sums of the multipliers λ_i and the Q_i-matrices for the transitions as outlined above.

The same is done for the j-th row with the multipliers μ_j and the Q_j-matrices. For the entry in the i-th column and j-th row, these weighted sums are simply summed up, so the LLF parametrization for mode $m_{2,1}$ is

$$(x, m_{2,1}) = [x^T, 1] \left(P_{0,0} + \sum_i \lambda_i^1 Q_i(c_1, d_1) + \sum_i \lambda_i^2 Q_i(c_2, d_2)) + \sum_i \mu_i^1 Q_i(e_1, f_1) \right) \begin{bmatrix} x \\ 1 \end{bmatrix},$$

where $P_{0,0}$ is the LLF for mode $m_{0,0}$, λ_i^k and μ_i^l are the newly introduced parameters for the k-th column and l-th row, and $c_k^T x + d_k = 0$ and $e_l^T x + f_l = 0$ are the hyperplanes describing the transitions between the columns and rows.

Using a standard LMI approach, the number of free variables would grow linearly with the number of cells. On the contrary, with this approach, the growth is just linear in the number of columns and rows. The result is an LMI problem of greatly reduced complexity. As shown in Theorem 4.6, this problem is equivalent to the original one.

We can also conduct this kind of analysis a priori, identifying sub-graphs which require a continuous GLF. This includes cycles of length two and grids, which can be seen as collections of such cycles. Since, only some λ-multipliers need to be added for each mode, and not fully parametrized LLF, one can decide not to decompose such sub-graphs at all. This can be reflected in the constraint graph by merging all the nodes on the cycles/the grid into a single node with the conjunction of all the constraints. This transformation would effectively protect the sub-automaton from being decomposed.

4.5.3 Common versus Piecewise Lyapunov Functions

As discussed above, in some situations, the constraints on a global Lyapunov function implicitly impose equality of two local Lyapunov functions of a hybrid system. Specifically, this occurs in the following situation.

Theorem 4.7 (Equality of two LLFs)**.** Let m_1 and m_2 be two neighboring modes of a hybrid automaton H, with two transitions (m_1, m_2, G_1, I) and (m_2, m_1, G_2, I), where G_1 and G_2 are guard sets and I is the identity function on $\mathbb{R}^n$ (i.e., no discrete update occurs). Suppose there is a global Lyapunov function $V(x, m)$ for H as per Theorem 3.5, such that for each mode m, $V(\cdot, m)$ is a polynomial. If the interior of $G_1 \cap G_2$ is non-empty, then we have $V(x, m_1) = V(x, m_2)$.

Proof. Show that $\forall x : \bar{V}(x) := V(x, m_1) - V(x, m_2) = 0$. Since the interior of $G_1 \cap G_2$ is non-empty, there exists an $x_0 \in \mathbb{R}^n$ and an $\epsilon > 0$, such that $B(x_0, \epsilon) \in G_1 \cap G_2$. Constraint (3) of Theorem 3.5 gives us $x \in G_1 \implies \bar{V}(x) \geq 0$ and $x \in G_2 \implies \bar{V}(x) \leq 0$. Therefore, for all $x \in B(x_0, \epsilon)$, we have $\bar{V}(x) = 0$. Since $\bar{V}$ is a polynomial, this implies $\forall x : \bar{V}(x) = 0$ and $V(x, m_1) = V(x, m_2)$. $\square$

This theorem can be interpreted as follows. If, on a cycle of two nodes, the guards of both edges overlap on a set whose interior is not empty, and there are no discrete updates, then it is not useful to designate different polynomial LLFs for the two nodes. Instead, it is sufficient to employ a common LLF for the nodes in question. This is achieved by using common parameters in the two LLFs, effectively reducing the search space of the SDP problem. This result can also be generalized to larger cycles as follows.

Corollary 4.3 (Equality of LLFs on a cycle)**.** Let H be a cyclic hybrid automaton with modes $m_0, \ldots, m_r$. Let the transitions be given as $(m_0, m_1, G_0, I), \ldots, (m_r, m_0, G_r, I)$, where the $G_i, 0 \leq i \leq r$, are guard sets and I is the identity function on $\mathbb{R}^n$. Suppose there is a global Lyapunov function $V(x, m)$ for H as per Theorem 3.5, such that for each mode m_i, $V(x, m_i)$ is a polynomial. If the interior of $\bigcap_i G_i$ is non-empty, then we have $V(x, m_i) = V(x, m_j)$ for all i, j.

Proof. Each transition gives us a constraint

$$x \in G_i \implies V(x, m_i) \geq V(x, m_{i+1 \,\text{modulo}\, (r+1)}).$$

Since there exists an $x_0 \in \mathbb{R}^n$ and an $\epsilon > 0$, such that $B(x_0, \epsilon) \in \bigcap G_i$, we have for all $x \in B(x_0, \epsilon)$:

$$V(x, m_0) \geq V(x, m_1) \geq \ldots \geq V(x, m_r) \geq V(x, m_0)$$

and therefore $\forall i \neq j, x \in B(x_0, \epsilon) : V(x, m_i) = V(x, m_j)$. With the same arguments as in the previous proof, this gives us $V(x, m_0) = \ldots = V(x, m_r)$ for all $x \in \mathbb{R}^n$. □

Whether the interior of the intersection of the guards in a cycle is empty is often easy to check. For instance, if the guard sets are of the form $g_i = \{x \in \mathbb{R}^n \mid p_i(x) \geq 0\}$, and p is continuous, then it is sufficient to find an x_0 with $\forall i : p_i(x) > 0$. If the p_i are linear, then this is a linear optimization problem, and if they are convex, this is a convex optimization problem. If the interior can be shown to be non-empty, the entire cycle can use the same polynomial Lyapunov function template, and this does not restrict the set of computable GLFs. This kind of analysis can also be conducted as a preprocessing step, before starting the decomposition. If such cycles can be detected in the graph, then we know that a common LLF should be used. Therefore, we can as well merge the nodes into one node in the constraint graph, simplifying the subsequent decomposition.

4.5.4 Reachability Analysis within Cycles

When a reducible cycle has been identified, it can be beneficial to perform another check before conducting the Lyapunov function computation. Within the cycle, there might be edges which are actually unreachable, effectively interrupting the cycle. Once this happens, the cycle decomposes into a number of one-mode SCCs, and (if all edges have sub-linear updates) this means that it is sufficient to only provide LLFs for each mode. Since many of these modes will also be part of other cycles for which the GLF computation has already been completed, we will often already have such LLFs, rendering any further computations unnecessary for the cycle. This reachability check can, for instance, be carried out with the help of such LLFs for the modes, employing them as barrier certificates that prove the unreachability of the guard of the outgoing edge (as discussed in Section 3.4).

One important point to note is that initial states of the SCC can be disregarded for such a reachability check. Instead, we only need to determine whether each guard of an outgoing edge of a mode of the cycle is reachable from within the mode, under the assumption that the mode was entered through its incoming edge. We can disregard the case when the mode was initial. The reasoning is as follows. Consider a cycle with a single initial mode, as depicted in Figure 4.26(a). We want to verify that the edge labeled with a sub-linear guard/update g_2/U_2 can be removed from the cycle.

Consider the hybrid automaton given in Figure 4.26(b), which was obtained by splitting the mode m_1 into two modes m_1^1 and m_1^2. Mode m_1^2 is reachable only if it is the initial state of the mode sequence, while mode m_1^1 can only be entered after traversing

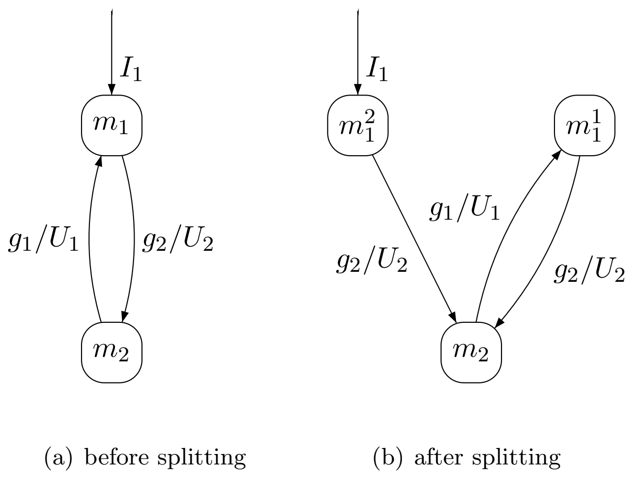


Figure 4.26: Reachability Analysis within a Cycle

the cycle at least once. Clearly, these two hybrid automata are equivalent. However, mode m_1^2 has been split off into a separate SCC and can therefore be treated completely separately. The reason this transformation is possible lies in the fact that an initial "edge" can only be taken once, and for this reason separated from the rest of the cycle. Therefore, we just need to show that guard g_2 is unreachable within mode m_1^1 only when starting from the transition g_1/U_1 in order to break the cycle. If this is the case, then the entire cycle decomposes into one-mode SCCs for which only a separate LLF needs to be provided. If such a one-mode SCC consists of a reduced node, then a GLF for the automaton it was obtained from has already been computed, making any further computations unnecessary.

Additionally, the cycle could also be entered through incoming edges which are not part of the cycle. It is also possible to conduct a reachability analysis for all such incoming transitions separately. If the cycle can be broken for all such incoming transitions, then it can be unrolled into a sequence of SCCs, rendering the GLF computation unnecessary.

While reachability analysis can be used a priori, before any splitting or reduction, there are also some benefits from doing this separately for each cycle. In order to remove an edge in a general hybrid automaton, we need to make sure that it is unreachable, no matter how its source mode was entered. This may or may not be possible — in fact, it may be the case that the edge proved to be unreachable only for some ways of entering the mode. In this case, we cannot simplify the automaton. When conducting reachability analysis per cycle, however, we have isolated one way of entering the mode. So it is possible to remove a duplicate of this edge for some cycles (making their analysis superfluous), but not for others.

In theory, one could also continue to unroll a cycle. By repeated reachability analysis, iterating around the cycle, it may be possible to prove that the cycle can only be traversed finitely often. With a similar argument as for the initial states, a prefix can then be split off into a separate SCC, and the unreachable edge removed, again resulting in a graph consisting of one-mode SCCs. However, this is of no use if there is no upper bound on the number of traversals for the cycle, and very costly if the number of traversals is large, so Lyapunov-based analysis will often be preferable.

We will now give a detailed example demonstrating the cycle based decomposition as well as the Lyapunov function computations for the individual cycles. The example contains also some instances of cycles which can be broken via reachability analysis.

4.6 Cruise Control Example

This section describes in detail the application of the decompositional stability verification approach to a relatively complex hybrid system modeling a cruise controller.

The hybrid automaton defining the system is given in Figure 4.27. The goal is to drive the velocity of a vehicle toward a set point defining a target velocity. The differential between current and target velocity (in m/s) is modeled by the continuous variable v, such that $v > 0$ implies that the current velocity is too high and $v < 0$ implies that it is too low, compared to the set point. There are two auxiliary continuous variables, x and t, which are not required to converge. Therefore, we are interested in proving GAS with respect to the set of variables $\{v\}$.

To avoid clutter, the initial states are not drawn in Figure 4.27. We assume that each discrete mode m may be initial, as long as initially $x \in \mathit{Inv}(m)$ holds.

The discrete behavior of the system is defined through six modes of operation. The mode N models a PI controller which is active around $v = 0$, and drives the velocity differential v towards 0 asymptotically. Here, the auxiliary variable x takes the role of the integral of v (i.e., the integral component of the PI-controller). Both a positive velocity differential and a positive integral value result in a negative acceleration and negative values in a positive acceleration. This mode may be active for velocity differentials with absolute value of at most 15 m/s and integral values with absolute value of at most 500. The bounds on x have been chosen such that they will not be violated if the PI-controller is activated with $-15\ m/s \leq v \leq 15\ m/s$ and $x = 0$. This additional information is helpful for the computation, as a simpler $\mathcal{S}$-procedure representation of the invariants and guards is possible. In general, it is advisable to always exploit the knowledge of such bounds for Lyapunov function computation. The variable x is also reset to 0 every time mode N is entered to model the re-initialization of the PI-controller upon every activation.

The mode A constantly accelerates the vehicle at 1.5 m/s^2, modeling a saturation cutoff for the acceleration. A transition from mode N to mode A may take place as soon as v reaches $-14\ m/s$ and, due to the invariant of N, it must take place when v reaches $-15\ m/s$. The system will then remain in A until v has again reached $-6\ m/s$. At this point, a switch back to N may take place, and at $v = -5\ m/s$, it must. Such a gap between the switching points from N to A and vice versa is called a *hysteresis* and prevents chattering or Zeno behavior when switching, as the transition guards do not overlap. Note that the auxiliary variable x, which plays no role in A and therefore

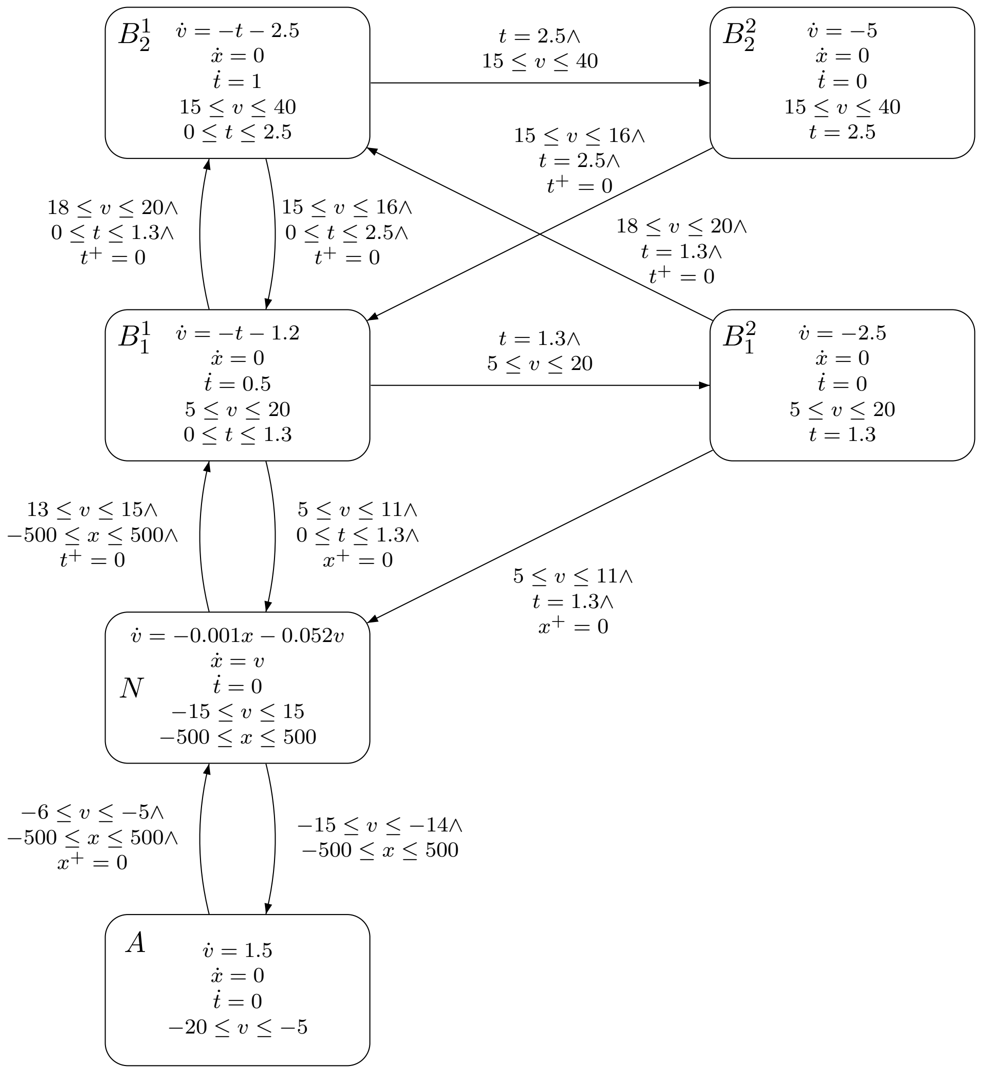


Figure 4.27: Cruise Control Automaton

remains constant, is reset to 0 upon the return to N.

The cruise controller is equipped with two levels of brakes, which can be considered *service* and *emergency* brakes. In addition, the mode N can of course also decelerate the system, modeling the braking behavior of the engine itself. The first level of brakes is defined through the modes B_1^1 and B_1^2. We assume that the brakes do not immediately start to brake at full effect upon activation. Instead, there is a “warmup” phase, modelled by B_1^1, which takes a set amount of time, in this case 2.6 seconds. Initially, the brake will decelerate the vehicle at $-1.2\ m/s^2$. As long as B_1^1 is active, the deceleration will increase linearly up to $-2.5\ m/s^2$ after 2.6 seconds. The auxiliary variable t is used to model a timer increasing at the rate of 0.5 per second. When t reaches 1.3 after 2.6 seconds, the transition to B_1^2 is taken. This mode keeps the deceleration constant at $-2.5\ m/s^2$. The activation and deactivation of the brake are again governed by a hysteresis. The first level brake may be activated already at $v = 13\ m/s$ and must be activated once v exceeds $15\ m/s^2$, leaving a range of possible switching points. A proof of GAS for this system will mean that any choice of switching point within this interval leads to the desired stable behavior. Likewise, a transition from both B_1^1 and B_1^2 back to N takes place on the interval $5\ m/s \leq v \leq 11\ m/s$, again with a reset of the integrator variable to 0.

The second level of brakes, modeled by modes B_2^1 and B_2^2 works in the same fashion, but with a higher peak deceleration of $-4\ m/s^2$. This brake can be triggered if we detect a very high velocity differential in either mode B_1^1 or mode B_1^2.

The constraint graph for this automaton is given in Figure 4.28. The constraints on the nodes and edges are defined as in Definition 4.10.

$$\begin{tikzcd}
C_{B_2^1} \arrow[rr, "C_{B_2^1,B_2^2}"] \arrow[d, bend left, "C_{B_2^1,B_1^1}"] & & C_{B_2^2} \arrow[dll, "C_{B_2^2,B_1^1}"'] \\
C_{B_1^1} \arrow[u, bend left, "C_{B_1^1,B_2^1}"] \arrow[rr, "C_{B_1^1,B_1^2}"] \arrow[d, bend left, "C_{B_1^1,N}"] & & C_{B_1^2} \arrow[ull, "C_{B_1^2,B_2^1}"'] \arrow[dll, "C_{B_1^2,N}"] \\
C_N \arrow[u, bend left, "C_{N,B_1^1}"] \arrow[d, bend left, "C_{N,A}"] \arrow[d, phantom, "C_1"] & & \\
C_A \arrow[u, bend left, "C_{A,N}"] & &
\end{tikzcd}$$

Figure 4.28: Constraint Graph of the Cruise Control Automaton

Now, we start the reduction and splitting procedure for this hybrid automaton. We can see that there is exactly one cycle with just one border node: the cycle C_1 consisting of the modes A and N and the two transitions between them. Therefore, we can reduce this cycle in the graph if we succeed in computing a predicate on the LLF of mode N as required by Theorem 4.3. As discussed in Section 4.5.1, this can be achieved by repeated solution of the LMI for cycle C_1 with different optimization directions: the predicate is then represented as the conic hull of the LLFs for N that have been computed in this manner. We therefore formulate the LMI problem for cycle C_1 as per Theorem 3.10. The LLF for N, $V(\cdot, N) = \tilde{x}P^N\tilde{x}$ cannot have a constant or a linear part, since the mode invariant contains the equilibrium. Also, both $V(\cdot, N)$ and $V(\cdot, A)$ cannot depend on t, since this variable does not change in the mode dynamics. Therefore, we can set the corresponding entries in the matrices to zero in beforehand, resulting in only three unknowns for P^N and six unknowns for P^A. We also add an additional constraint to the LMI, bounding β from above by 100. This is done to bound the solution space, guaranteeing the existence of a solution in all optimization directions. Without this addition, the conic, unbounded solution space would mean that we could only optimize in some directions. The resulting LMI problem looks as follows:

Find $P_A, P_N \in \mathbb{R}^{3\times 3}$ and $\alpha, \beta, \mu^A, \mu^N, \nu^A, \nu^N, \eta^A, \eta^N, \vartheta^{A,N}, \vartheta^{N,A} \in \mathbb{R}$, such that

$$\begin{aligned}
\alpha - \epsilon &\succeq 0\\
\beta - \epsilon &\succeq 0\\
100 - \beta &\succeq 0\\
\mu^A, \mu^N, \nu^A, \nu^N, \eta^A, \eta^N &\succeq 0\\
\vartheta^{A,N}, \vartheta^{N,A} &\succeq 0\\
P^A - \mu^A Q^A - J &\succeq 0\\
P^N - \mu^N Q^N - \tilde{I} &\succeq 0\\
P^A + \nu^A Q^A - \beta J &\preceq 0\\
P^N + \nu^N Q^N - \beta \tilde{I} &\preceq 0\\
\begin{bmatrix} (A^A)^T & 0 \\ (b^A)^T & 0 \end{bmatrix} P^A + P^A \begin{bmatrix} A^A & b^A \\ 0 & 0 \end{bmatrix} + \eta^A Q^A + \alpha J &\preceq 0\\
\begin{bmatrix} (A^N)^T & 0 \\ (b^N)^T & 0 \end{bmatrix} P^N + P^N \begin{bmatrix} A^N & b^N \\ 0 & 0 \end{bmatrix} + \eta^N Q^N + \alpha \tilde{I} &\preceq 0\\
P_A - \begin{bmatrix} (A^{A,N})^T & 0 \\ (b^{A,N})^T & 1 \end{bmatrix} P^N \begin{bmatrix} A^{A,N} & b^{A,N} \\ 0 & 1 \end{bmatrix} - \vartheta^{A,N} R^{A,N} &\succeq 0\\
P_N - \begin{bmatrix} (A^{N,A})^T & 0 \\ (b^{N,A})^T & 1 \end{bmatrix} P^A \begin{bmatrix} A^{N,A} & b^{N,A} \\ 0 & 1 \end{bmatrix} - \vartheta^{N,A} R^{N,A} &\succeq 0
\end{aligned}$$

with

$$\begin{aligned}
\tilde{I} &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, J = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix},\\
A^A &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, b^A = \begin{bmatrix} 1.5 \\ 0 \\ 0 \end{bmatrix},\\
A^N &= \begin{bmatrix} -0.001 & -0.052 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, b^N = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix},\\
P^N &= \begin{bmatrix} p^N_{1,1} & p^N_{1,2} & 0 & 0 \\ p^N_{1,2} & p^N_{2,2} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, p^N_{1,1}, p^N_{1,2}, p^N_{2,2} \in \mathbb{R},
\end{aligned}$$

$$P^A = \begin{bmatrix} p^A_{1,1} & p^A_{1,2} & 0 & p^A_{1,4} \\ p^A_{1,2} & p^A_{2,2} & 0 & p^A_{2,4} \\ 0 & 0 & 0 & 0 \\ p^A_{1,4} & p^A_{2,4} & 0 & p^A_{4,4} \end{bmatrix}, p^A_{1,1}, p^A_{1,2}, p^A_{1,4}, p^A_{2,2}, p^A_{2,4}, p^A_{4,4} \in \mathbb{R},$$

$$Q^A = \begin{bmatrix} -0.0089 & 0 & 0 & -0.1111 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -0.1111 & 0 & 0 & -0.3889 \end{bmatrix}, Q^N = \begin{bmatrix} -0.0022 & 0 & 0 & 0 \\ 0 & -0.000002 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$A^{A,N} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, b^{A,N} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, A^{N,A} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, b^{N,A} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix},$$

$$R^{A,N} = \begin{bmatrix} -2 & 0 & 0 & -11 \\ 0 & -0.000002 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -11 & 0 & 0 & -59.5 \end{bmatrix},$$

$$R^{N,A} = \begin{bmatrix} -2 & 0 & 0 & -29 \\ 0 & -0.000002 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -29 & 0 & 0 & -419.5 \end{bmatrix}.$$

The $\mathcal{S}$-procedure matrices $Q^A, Q^N, R^{A,N}$, and $R^{N,A}$ were obtained through the ellipsoid based procedure described in [Pettersson, 1999, p. 105]. As optimization directions, we simply minimize or maximize each single free variable of matrix P^N, resulting in a total of six optimization directions. For each of the six solutions returned by the SDP solver (in this case CSDP), we need to keep track of the computed LLFs for mode N. They are:

$$\begin{aligned} V_N^1(v,x,t) &= 93.1487v^2 + 6.3888vx + 0.4043x^2 \\ V_N^2(v,x,t) &= 10.0034v^2 + 0.01vx + 0.01x^2 \\ V_N^3(v,x,t) &= 79.5725v^2 + 7.7816vx + 0.3439x^2 \\ V_N^4(v,x,t) &= 56.6641v^2 + 0.01vx + 0.0569x^2 \\ V_N^5(v,x,t) &= 100v^2 + 0.01vx + 0.1005x^2 \\ V_N^6(v,x,t) &= 3.0618v^2 + 0.1618vx + 0.0121x^2 \end{aligned}$$

The parameter values of these quadratic LLFs are rounded to four significant digits. The actual computed values, which are re-used in the next steps are double precision numbers computed up to machine precision.

Since the GLF computation was successful for cycle C_1, we can now reduce the cycle in the constraint graph. Figure 4.29 shows the constraint graph after this reduction step. The predicate R_N, which can be defined as

$$R_N :\Longleftrightarrow V(\cdot, N) \in \mathit{cone}(\{V_N^1(\cdot), \ldots, V_N^6(\cdot)\})$$

has been attached to the node N. This means that, for the next LMI computation containing node N, the conic combination of the computed LLFs has to be used as the new parametrization for node N.

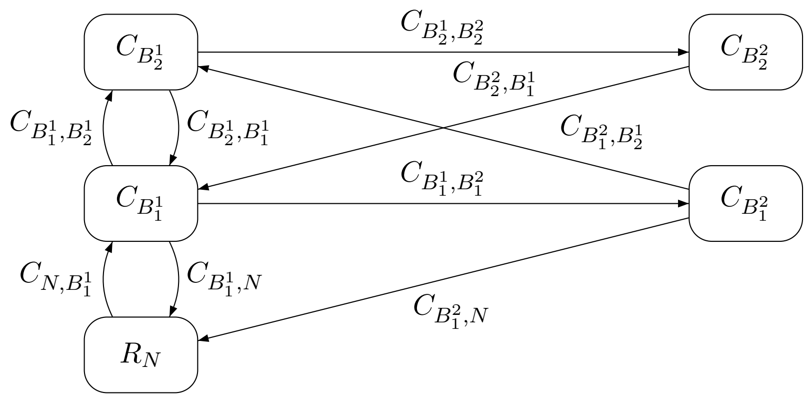


Figure 4.29: Constraint Graph after Reduction of Cycle C_1

Now we can continue with the reduction, but the constraint graph in Figure 4.29 does not contain any cycles with only a single border node. Therefore, we have to apply the node splitting procedure. As pointed out before, a reasonable heuristic is to start splitting the node with the smallest product of indegree and outdegree, which will lead to the smallest number of new nodes. Both nodes B_1^2 and N can be split into two nodes, and therefore splitting any of these two is reasonable. Figure 4.30 shows the constraint graph after the splitting of B_1^2. As we can see, there is still no cycle with a single border node, so we proceed by also splitting N.

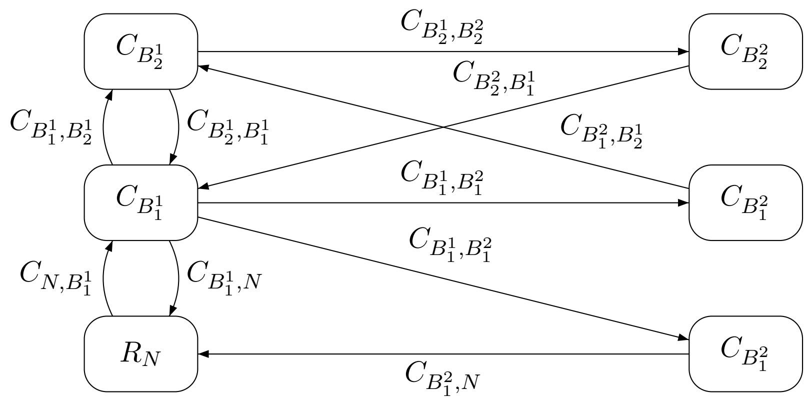


Figure 4.30: Constraint Graph after Splitting of Node B_1^2

The resulting constraint graph is depicted in Figure 4.31. Note that the predicate R_N remains attached to both of the newly added nodes. Now there are two cycles with just one border node, the cycle C_2 consisting of N and B_1^1, and the cycle C_3 consisting of N,

B_1^1, and B_1^2. We can therefore now conduct Lyapunov function computations to reduce these two cycles.

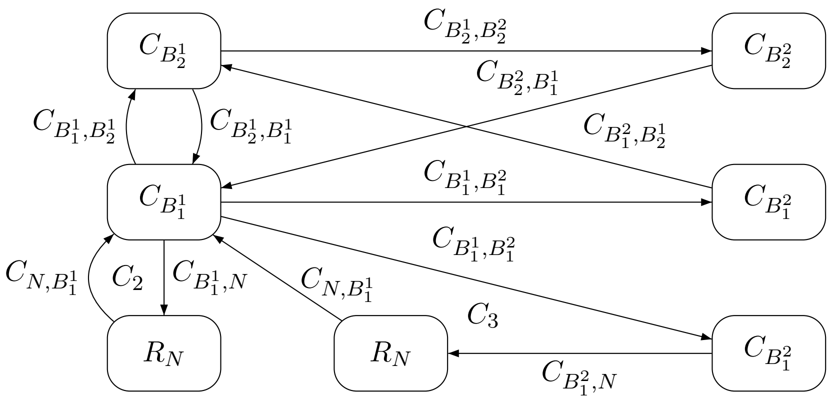


Figure 4.31: Constraint Graph after Splitting of Node N

Next, we reduce cycle C_2 in the lower left, containing N and B_1^1. The LMI computation works as usual, with the difference that the LLF parametrization for node N is given by $\sum_{i=1}^{6} \lambda_i V_N^i(\cdot)$. These multipliers λ_i must be non-negative and one of them strictly positive. However, it is simpler to encode in the LMI and not considerably conservative if we require all of them to be strictly positive. To bound the solution space in all directions, we also require that each multiplier λ_i is less than 100.

Since the LLFs for mode B_1^1, the border node of the cycle, do not need to refer to the variable x (which plays no role in the mode), we decide to maximize and minimize the free parameters corresponding to v^2, vt, and t^2, again resulting in six optimization directions. The six extremal LLFs are the following:

$$\begin{aligned}
V_{B_1^1}^1(v,x,t) &= 460.9169v^2 + 3666.3589tv - 7818.4308v + 14938.2327t^2 - 49420.3989t + 54170.5562\\
V_{B_1^1}^2(v,x,t) &= 79.3041v^2 + 6.359tv + 2493.9413v - 4365.2903t^2 + 5603.0088t - 6811.669\\
V_{B_1^1}^3(v,x,t) &= 837.1111v^2 + 5279.9837tv - 18227.0739v + 8988.1138t^2 - 6934897t + 126287.5833\\
V_{B_1^1}^4(v,x,t) &= -158.9733v^2 - 1456.1625tv + 6035.4231v - 483.4607t^2 + 17906.6609t - 18743.4165\\
V_{B_1^1}^5(v,x,t) &= 860.0469v^2 + 5189.2642tv - 18617.1601v + 7077.4376t^2 - 66864.1442t + 128990.3238\\
V_{B_1^1}^6(v,x,t) &= -195.2294v^2 - 1165.7724tv + 6622.0794v - 2418.8871t^2 + 15993.5875t - 18865.6355
\end{aligned}$$

Again, these LLFs are subsumed into a conic predicate $R_{B_1^1}$ which is attached to node B_1^1, resulting in the constraint graph given in Figure 4.32. Next, the cycle C_3 can be reduced in the same manner. Note that there are two reduced nodes on this cycle, so that two LLFs receive a parametrization corresponding to a conic predicate, namely the LLFs for nodes N and B_1^1. The same six optimization directions can be used, resulting in the following LLFs for B_1^1.

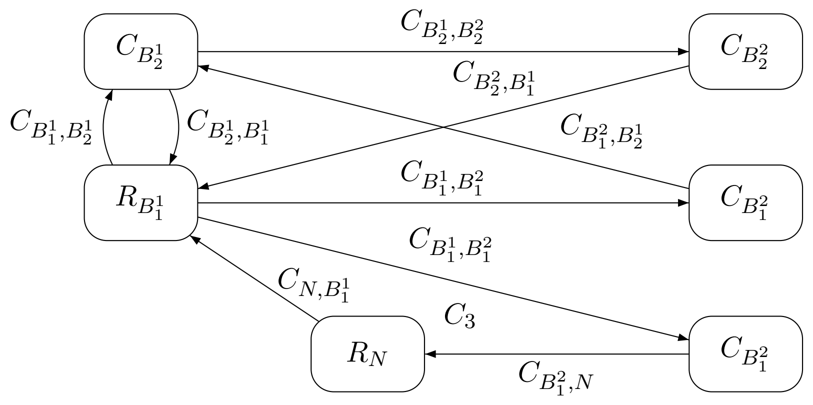


Figure 4.32: Constraint Graph after Reduction of Cycle C_2

$$\begin{aligned}
V^7_{B_1^1}(v,x,t) &= 934.5849v^2 + 6653.9553tv - 18131.8823v + 20023.9711t^2 - 88656.645t + 125628.6472\\
V^8_{B_1^1}(v,x,t) &= 102.7078v^2 + 235.5093tv + 1203.1979v - 3238.0142t^2 + 1591.5665t + 182.2822\\
V^9_{B_1^1}(v,x,t) &= 1184.2884v^2 + 7629.1137tv - 25121.9157v + 15047.3507t^2 - 100237.1764t + 174059.5928\\
V^{10}_{B_1^1}(v,x,t) &= -83.8847v^2 - 929.5328tv + 5085.5704v + 818.1647t^2 + 11628.5909t - 11992.3360\\
V^{11}_{B_1^1}(v,x,t) &= V^9_{B_1^1}(v,x,t)\\
V^{12}_{B_1^1}(v,x,t) &= -97.251v^2 - 394.4151tv + 4891.9132v + 675.6803t^2 + 5590.4664t - 7434.0458
\end{aligned}$$

The LLFs $V^{11}_{B_1^1}$ and $V^9_{B_1^1}$ are equal because two optimization directions resulted in the same extremal point in the solution space. Note that the conic hull of these six functions forms a sub-set of the conic hull of the functions computed for node B_1^1 in the previous step: the result is a set of LLFs that is both compatible with cycles C_2 and C_3.

The resulting constraint graph is given in Figure 4.33. Again, we require node splittings in order to obtain reducible cycles. We can either choose to split node B_1^1 or node B_2^1, since these are the only nodes with degree larger than two. We choose to split node B_2^1, resulting in four new nodes.

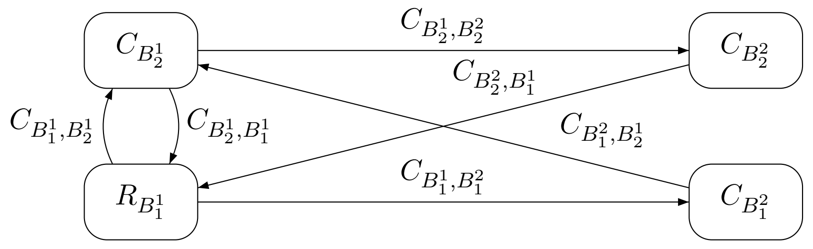


Figure 4.33: Constraint Graph after Reduction of Cycle C_3

Figure 4.34 gives the result. We can observe that there now is another reducible cycle,

C_4, consisting of the nodes B_1^1 and B_2^1 on the bottom left. Again, the six optimization directions result in six LLFs for B_1^1.

$$\begin{aligned}
V_{B_1^1}^{13}(v,x,t) &= 3122.0271v^2 + 20588.2371tv - 58398.218v + 51612.4851t^2 - 271911.9246t + 454321.4531\\
V_{B_1^1}^{14}(v,x,t) &= 102.7391v^2 + 235.7153tv + 1202.6139v - 3237.4980t^2 + 1588.8474t + 186.8254\\
V_{B_1^1}^{15}(v,x,t) &= 3405.8678v^2 + 22147.6787tv - 67172.416v + 46880.6732t^2 - 287539.2591t + 473929.9210\\
V_{B_1^1}^{16}(v,x,t) &= -181.1016v^2 - 1323.7263tv + 9976.8119v + 1494.3138t^2 + 17216.1819t - 19421.6426\\
V_{B_1^1}^{17}(v,x,t) &= V_{B_1^1}^{15}(v,x,t)\\
V_{B_1^1}^{18}(v,x,t) &= V_{B_1^1}^{16}(v,x,t)
\end{aligned}$$

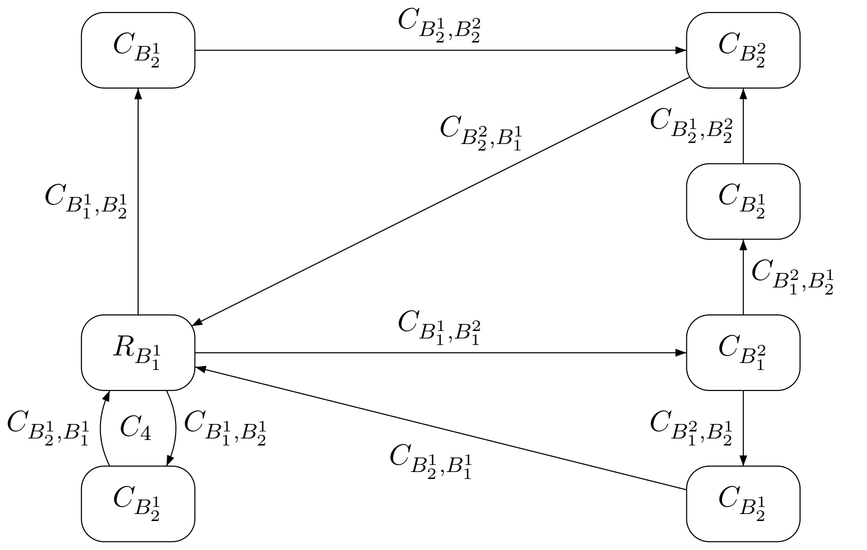


Figure 4.34: Constraint Graph after Splitting of Node B_2^1

Therefore, we can reduce cycle C_4, resulting in the constraint graph given in Figure 4.35. To continue the reduction, we require another splitting, this time either of mode B_1^2 in the center right or of mode B_2^2 at the top right.

We choose to split B_1^2, resulting in the constraint graph depicted in Figure 4.36. Therefore, we can now reduce the cycle in the lower part of the graph, containing the nodes B_1^1, B_1^2, and B_2^2, again with B_1^1 as the border node. We call this cycle C_5. For this cycle, we only require one single Lyapunov function, for reasons that will be explained in the following. One possible LLF within C_5 for B_1^1 is given next.

$$V_{B_1^1}^{19}(m,v,x,t) = 8687.5308v^2 + 56245.0238tv - 156163.583v + 130845.202t^2 - 731410.246t + 1227213.693$$

Figure 4.37 shows the constraint graph after this reduction.

Without the help of reachability checks, we would have to continue the reduction process from this point, but analysis of the remaining cycles will show that they cannot

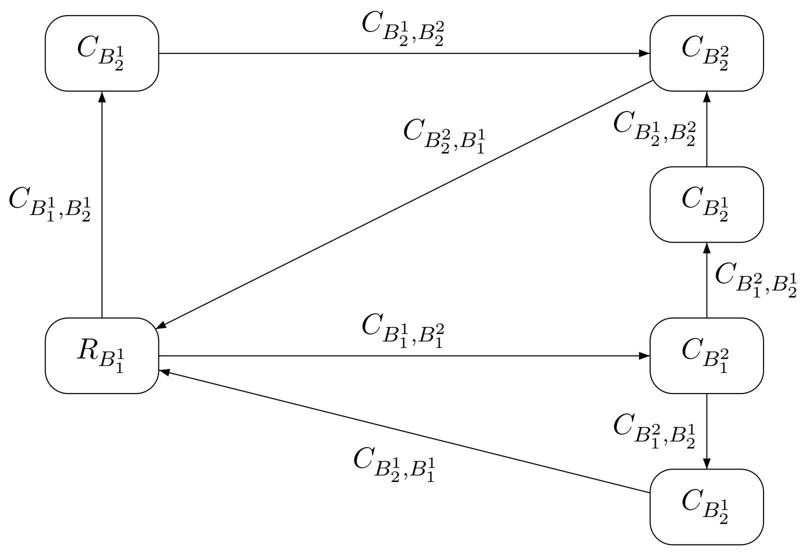


Figure 4.35: Constraint Graph after Reduction of Cycle C_4

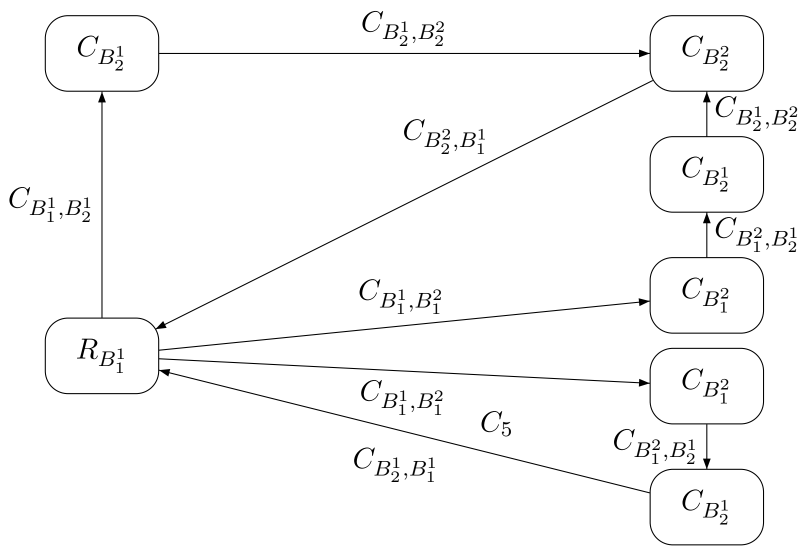


Figure 4.36: Constraint Graph after Splitting of Node B_1^2

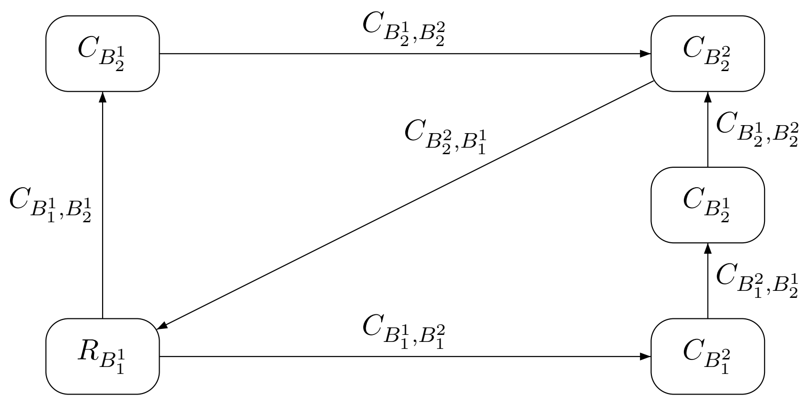

$$\begin{tikzcd}
C_{B_2^1} \arrow[rr, "C_{B_2^1,B_2^2}"] & & C_{B_2^2} \arrow[ddll, "C_{B_2^2,B_1^1}"'] \\
& & C_{B_2^1} \arrow[u, "C_{B_2^1,B_2^2}"'] \\
R_{B_1^1} \arrow[uu, "C_{B_1^1,B_2^1}"] \arrow[rr, "C_{B_1^1,B_1^2}"] & & C_{B_1^2} \arrow[u, "C_{B_1^2,B_2^1}"']
\end{tikzcd}$$

Figure 4.37: Constraint Graph after Reduction of Cycle C_5

be fully traversed by runs of the hybrid automaton. Figure 4.38 depicts the hybrid automaton corresponding to the constraint graph. With the help of LLFs for mode B_2^1 as barrier certificates (as described in Section 3.4), we can conclude that both transitions from the duplicates of B_2^1 (top left corner and second on the right hand side) to B_2^2 (top right corner) cannot be taken. For instance, consider the function $V_{B_2^1}(v, x, t) = v^2+45t^2$, which is a LLF for B_2^1. When entering either of the duplicates of B_2^1, we set $t^+ = 0$ and v lies in the range from 18 to 20. Therefore, the maximum value of $V_{B_2^1}$ when entering each of these modes is 400. When leaving either of the modes, t is 2.5 and v is at least 15. Therefore, the minimum value of $V_{B_2^1}$ when taking a transition to the top right mode is $15^2 + 45 * 2.5^2 = 506.25$. Since $V_{B_2^1}$ is non-increasing within mode B_2^1, we can conclude that both of the transitions leading to the top right mode are unreachable. This means that these edges can be left out.

Figure 4.39 shows the resulting decomposed hybrid automaton without the unreachable edges. Since all edges have sub-linear updates, and since each node in this automaton is actually an SCC of its own, this means that we only need to compute LLFs for each of the remaining nodes. However, for the modes B_1^1, B_1^2, and B_2^1, we already computed some LLFs, as these modes were part of previously reduced cycles. This is not the case for mode B_2^2. Therefore, we only need to find a LLF for B_2^2 to finish the proof of GAS for the entire automaton, for instance by using $V_{B_2^2}(v, x, t) = v^2$. This decompositional proof now implies GAS for H.

The reduction procedure for this example was immediately successful, because all conic predicates for the border nodes were sufficiently tight under-approximations. Next, we discuss how to deal with situations where this is not the case. The solution we propose is refinement of the border node predicates.

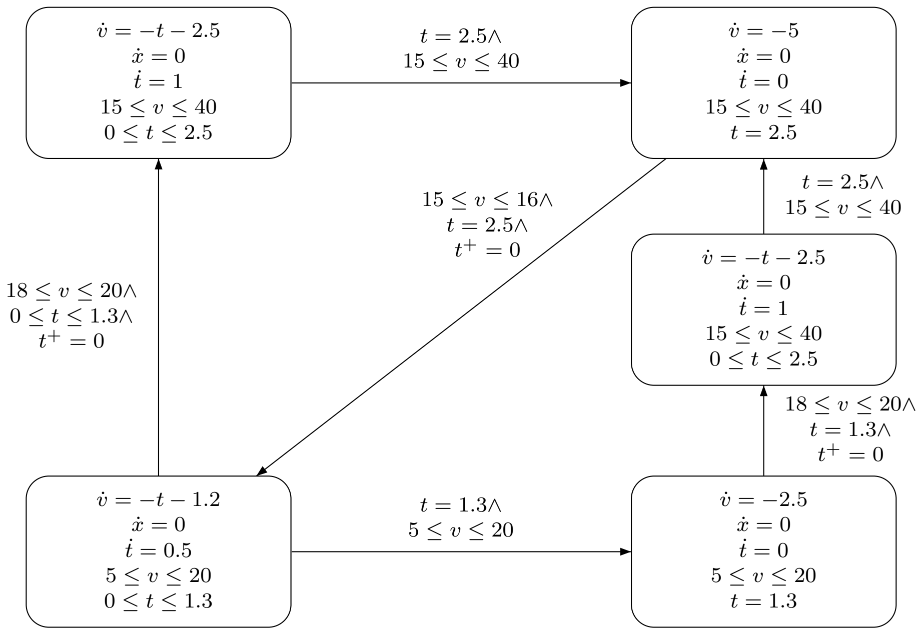


Figure 4.38: Hybrid Automaton after the Reductions and Splittings

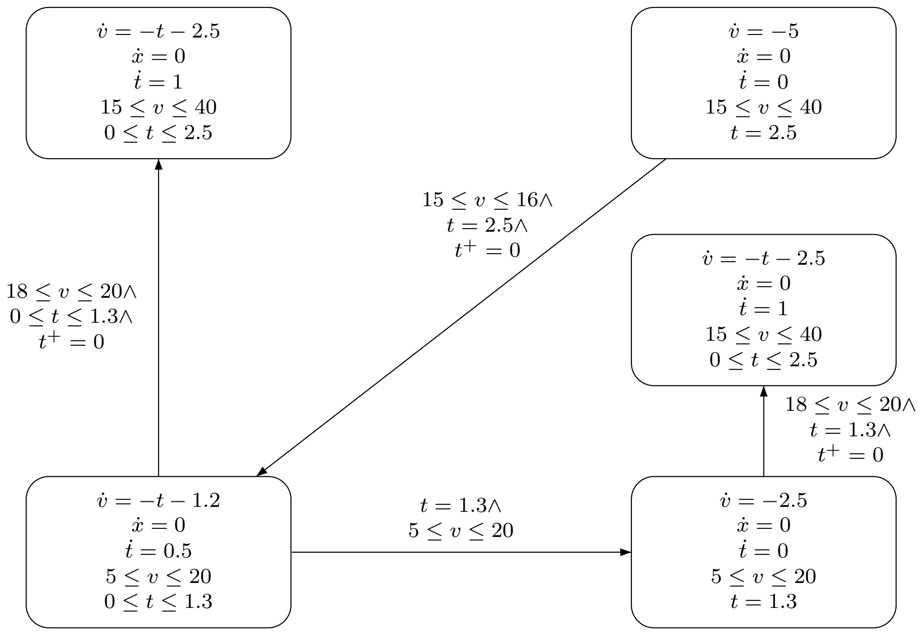


Figure 4.39: Hybrid Automaton after Reachability Analysis

4.7 Refining the Border Node Predicates

Since the under-approximations of the Lyapunov function sets by predicates R_b as computed in Section 4.5.1 are usually conservative in the sense that some valid Lyapunov functions are excluded, it is sometimes necessary to refine the predicates in order to provide a stability proof. This refinement takes place by adding new extremal points spanning the conic set defined by the predicate. In the following, for the ease of writing, we informally identify a predicate R_b with the set of parameters satisfying it.

In particular, situations where the under-approximations are not tight enough occur when two overlapping cycles show significantly different behavior, resulting in a small intersection between the two LLF sets stemming from the two cycles for the border node. See Figure 4.40(a) for an illustration of two overlapping ellipsoid sets for which the under-approximations consisting of the extremal points in the cardinal directions do not overlap. When reducing the first cycle and computing the predicate R_b, by default we optimize in a number of fixed directions, for example the cardinal directions. However, the intersection of the two solution sets might lie in a diagonal direction, or this under-approximation might be otherwise disadvantageous. For example, see Figure 4.40(b) for a set that is poorly approximated by only the extremal points in the cardinal directions (the solid polytope). Note that, in this case, the poor under-approximation is a result of the choice of optimization directions, as diagonal directions would have given a far better result (the dashed polytope). Refinement can simply be conducted by adding new corner points to the inner polytope, such that the gap between actual solution set and under-approximation is narrowed sufficiently. This section deals with these refinement strategies.

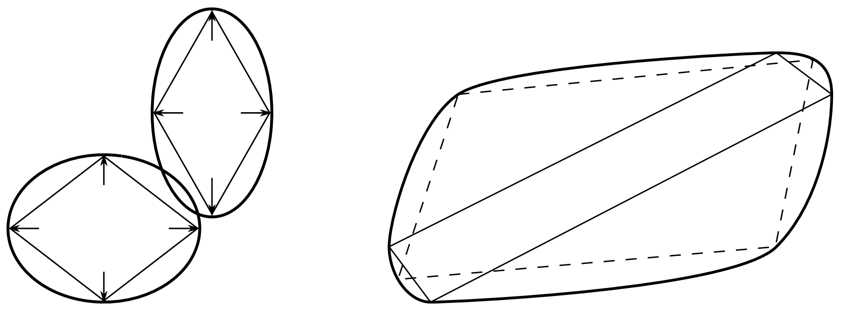

(a) two marginally intersecting solution sets
(b) poorly under-approximated solution set

Figure 4.40: Conservative Under-Approximations of Lyapunov Function Sets

First, we will focus on such a refinement between two overlapping cycles of a hybrid automaton. Note that we can obtain and use some information on where intersections between the two cycles are likely to lie. This can be achieved by conducting GLF computations for the second cycle *without taking into account the first cycle.* If we now examine the sets of admissible LLFs for the border node within this second cycle, this can help us determine advantageous optimization directions for the border node predicate

R_b for the first cycle. It generally advisable to prefer directions which generally "point toward the solution set of the second cycle." However, depending on the shape of the solution sets to be approximated, it is also possible that we can only find an intersection in another direction, especially if the dimension of the solution space is high. To ensure that an existing intersection is eventually detected, it is therefore necessary to also cover these seemingly less promising optimization directions.

First, we will give an algorithm that will eventually find an existing intersection if its interior is non-empty. It is in general not possible to give an upper bound on the number of refinement steps needed, since the actual solution set can lie arbitrarily close to the border of the two sets. For instance, Figure 4.40(a) shows a small intersection, which will only be detected once either an extremal point lying directly inside the intersection has been found, or two extremal points lie sufficiently close to it such that their connecting line crosses the intersection. Therefore, the efficiency of this refinement procedure heavily depends on the robustness of the stability property, much like the computation time for convex optimization depends on the size, shape, and position of the solution set. To complement this analysis, we also employ over-approximations of the solution set, which come at a low extra cost, so that also the infeasibility of the entire problem can eventually be detected. While the under-approximation is known through the extremal points spanning the convex set, this over-approximation is represented as a number of linear constraints, each forming a tangential plane to one extremal point. This is important for the refinement algorithm, since conversions between these two polytope formats are costly and therefore should be avoided. Such an exhaustive approach can also be combined with heuristics providing priorities to different directions. As long as these heuristics are fair in the sense that refinement directions do not ignore parts of the state space, all intersections with non-empty interior will still be detected, while improving termination times.

As a more efficient alternative to the exhaustive algorithm, we also propose a greedy approach for detecting the intersection between the two LLF sets. Here, the optimization direction can determined heuristically by various measures. One example is the minimization of the "violation" of the polytopic constraints of the under-approximation for the set we want to intersect with. Basically, optimization directions are selected such that some easy-to-compute approximation of the "distance" to the second cycle's LLF set is minimized. These heuristics are not guaranteed to detect all intersections, but if they do, then termination will be much faster than for the exhaustive case. The exhaustive and the greedy approach can also be combined. A straightforward approach is to try the greedy approach first. If stagnation is detected there, then other optimization directions can be added based on the exhaustive approach, in order to get the greedy approach "unstuck." This type of alternation can guarantee fast results in case the greedy approach works and still guarantee successful termination if the interior of the solution set is non-empty.

Secondly, we will focus on how to implement these refinement strategies in the presence of *multiple* intersecting cycles. If a refinement between two cycles terminates unsuccessfully (either because infeasibility has been detected or a threshold in the number of corner points has been reached), then a backtracking algorithm for refining the computations for the previously collapsed cycles is given, resulting in a complete refinement procedure for a hybrid automaton. However, we begin with the two-cycle case.

4.7.1 Approximation Refinement for Two Intersecting Cycles

We will first give a general outline of a refinement algorithm which does not specify the actual selection of new optimization directions to be added. This algorithm outline can then be cast into several actual algorithms, including a version enumerating all possible optimization directions and greedy methods which are not guaranteed to detect all intersections but can find most intersections faster.

The refinement algorithm outline is given in Algorithm 2. Assume that we have two cycles, C_1 and C_2, intersecting in exactly one mode b, with cycle C_1 being reduced first. Cycle C_1 is collapsed into a conic, polytopic predicate $R_b(C_1)$ based on a number of fixed optimization directions (line 1). Since we must be careful to match the predicates R_b to the constraint systems that were used to compute them, we add the cycle they are based on in parentheses. Also, define $constr_b(C_i)$ as the projection of $constr(C_i)$ onto the free parameters of V_b, existentially quantifying over all other parameters of $constr(C_i)$. Therefore, $constr_b(C_i)$ represents the exact set of parameter values for V_b, which allow for a GLF for cycle C_i. Now, as required by Theorem 4.3, the predicate $R_b(C_1)$ is conjoined with the constraint system $constr(C_2)$ for the second cycle, and if there is a solution, then the two-cycle system is GAS (lines 2 and 3). If we find no solution, then it might be the case that the approximation by $R_b(C_1)$ is too coarse, or a GLF of the chosen parametrization might not exist at all. In this case, we first check the constraint system $constr(C_2)$ for solutions — if it does not have any, then there cannot be any solutions and refinement of $R_b(C_1)$ is of no use (line 9). If we find a solution, then we compute another conic, polytopic predicate R_b, but this time stemming from the constraint system $constr(C_2)$, and denote it $R_b(C_2)$ (line 6). Our goal is to refine $R_b(C_1)$ such that $constr(C_2) \wedge R_b(C_1)$ finally has a solution. The predicate $R_b(C_2)$ can be used to guide this refinement, since it gives us some information of the position of $constr_b(C_2)$ in the parameter space, and we should therefore try to choose optimization directions for the refinement of $R_b(C_1)$ that lead us towards the set of states satisfying $R_b(C_2)$.

At this point, the main refinement loop starts. We also compute over-approximation predicates $U_b(C_1)$ and $U_b(C_2)$ as duals to the under-approximation predicates (line 12). The reasoning is as follows. We can argue that the following constraint holds for all parameter vectors p for V_b whose corresponding LLF fulfills $constr_b(C_i), i \in \{1, 2\}$:

$$\bigwedge_{p_b^j(C_i)} \left\langle (p - p_b^j(C_i))^T \;\middle|\; d_b^j(C_i) \right\rangle \leq 0,$$

where $d_b^j(C_i)$ is the optimization direction vector belonging to the extremal parameter vector $p_b^j(C_i)$ spanning $R_b(C_i)$. Since $p_b^j(C_i)$ is the extremal point in the $d_b^j(C_i)$-direction, no p can lie beyond the tangential line at $p_b^j(C_i)$, and therefore the scalar product of the difference of p to $p_b^j(C_i)$ and the normal vector given by the optimization direction

```
calculate $R_b(C_1)$ based on a fixed set of directions;
if $constr(C_2) \wedge R_b(C_1)$ has a solution then
    return true;
end
if $constr(C_2)$ has a solution then
    calculate $R_b(C_2)$ based on a fixed set of directions;
end
else
    return false;
end
repeat
    compute over-approximations $U_b(C_1)$ and $U_b(C_2)$ from $R_b(C_1)$ and $R_b(C_2)$;
    if $U_b(C_1) \wedge U_b(C_2)$ has no solution then
        return false;
    end
    choose an optimization direction $d$;
    add extremal point of $constr_b(C_1)$ in direction $d$ to list of extremal points
    for $R_b(C_1)$;
    add extremal point of $constr_b(C_2)$ in direction $(-d)$ to list of extremal
    points for $R_b(C_2)$;
until $constr(C_2) \wedge R_b(C_1)$ has a solution;
return true;
```

Algorithm 2: Refinement Algorithm Outline

must be negative. See Figure 4.41 for an illustration. The solid line running from top left to bottom right is the tangential line at $p_b^j(C_i)$, with the angle between $p - p_b^j(C_i))$ and $d_b^j(C_i)$ being larger than 90 degrees.

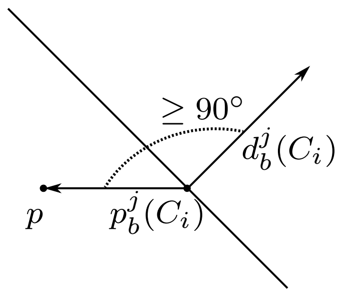


Figure 4.41: Over-Approximation Constraints

If the $d_b^j(C_i)$ are the cardinal directions, then the resulting overapproximation is simply a box, otherwise it is a convex polytope. (see Figure 4.42).

If the two over-approximations $U_b(C_1)$ and $U_b(C_2)$ of $constr_b(C_1)$ and $constr_b(C_2)$ do not overlap, then $constr_b(C_1) \wedge constr_b(C_2) = false$ and we terminate unsuccessfully,

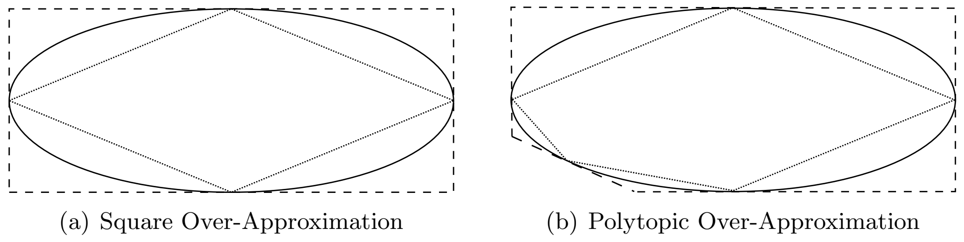

(a) Square Over-Approximation (b) Polytopic Over-Approximation

Figure 4.42: Over-Approximation (dashed) of Solution Sets (solid) for a Given Under-Approximation (dotted)

knowing that there is no GLF of the chosen parametrization. If we cannot disprove the existence of a GLF at this point, the actual refinement takes place. First, an optimization direction for adding a new extremal point to $R_b(C_1)$ is determined (line 16). This choice of optimization direction can be based on the relative positions of the extremal points of $R_b(C_1)$ and $R_b(C_2)$. Then, the corresponding LMI problem is solved to obtain a new extremal point of $constr_b(C_1)$ (line 17). Finally, we also refine $R_b(C_2)$, adding the extremal point of $constr_b(C_1)$ in the opposite direction (line 18). Since $R_b(C_1)$ now contains an extra vertex, we can now retry to solve $constr(C_2) \wedge R_b(C_1)$ with this new information. If this is still unsuccessful, then we repeat the refinement process. Note that each refinement of the under-approximation also leads to a refinement of the corresponding over-approximation, as can be seen in Figure 4.42. Therefore, a refinement step might also help us detect that $constr_b(C_1)$ and $constr_b(C_2)$ do not have an intersection, and then the same holds for $constr(C_1) \wedge constr(C_2)$ leading to an unsuccessful stability proof with this particular LLF parametrization.

In the presence of numerical errors, it is important to note that for the under-approximation we need to round inward (i.e., toward the interior of $constr_b(C_i)$) and for the over-approximation we need to round outward. In other words, the $p_b^j(C_i)$ used for the over-approximation should ideally slightly violate $constr_b(C_i)$ in order to be on the safe side. On the other hand, the extremal points used for the under-approximation should fulfill $constr_b(C_i)$ by some safe margin.

The key point in Algorithm 2 is the choice of refinement directions. While it is in general possible to exclude some optimization directions for $constr_b(C_1)$ altogether, as they cannot lead to an intersection with $constr_b(C_2)$, this is computationally costly. Especially in high dimensions, it is difficult to exclude even those directions pointing away from the under-approximation $R_b(C_2)$, since an unfortunate initial approximation might still lead to an intersection in this direction.

The brute force approach is to enumerate optimization directions in an evenly spaced manner, regardless of the relative positions of the two under- and over-approximations. This enumeration needs to be done such that eventually every facet of the under-approximation is refined, in order to ensure that $constr_b(C_1)$ is eventually approximated arbitrarily close. Heuristics can still be used to order these optimization directions in a favorable manner.

If the number of extremal points truly becomes to large to handle, disregarding individual points for a solution attempt of $constr(C_2) \wedge R_b(C_1)$ can however be a good idea. This is not harmful if the disregarded points are chosen carefully. Since merely one extremal point e will have been added since the last attempt at solving $constr(C_2) \wedge R_b(C_1)$, any other extremal points which cannot form a new facet with e are technically not needed for the check.

One basic strategy for the selection of directions is as follows. We divide the enumeration into different steps, starting with the (positive and negative) cardinal directions in step 1. The step $i+1$ then consists of the sums of all directions of the i-th step with the cardinal directions. If a coordinate of such a direction is positive or negative, we do not need to add the cardinal direction pointing in the other direction (e.g., direction $[1,0]^T$ does not need to be combined with $[-1,0]^T$, since this would result in the zero vector). In other words, the i-th step consists of all direction vectors with a 1-norm of i.

For instance, assume that we start with the three cardinal vectors $[1,0,0]^T, [0,1,0]^T$ and $[0,0,1]^T$ and their three negatives $[-1,0,0]^T, [0,-1,0]^T$ and $[0,0,-1]^T$ in three dimensions. The next step would then result in a total of 18 directions. For instance, $[1,0,0]^T$ would be combined with five cardinal vectors, resulting in five new directions $[2,0,0]^T, [1,1,0]^T$, $[1,-1,0]^T$, $[1,0,1]^T$ and $[1,0,-1]^T$. Intuitively, we simply enumerate all possible integer direction vectors, which eventually leads to an arbitrarily close coverage of all possible optimization angles. The duplicate angles that occur in this enumeration (for instance $[1,0,0]^T$ and $[2,0,0]^T$, or $[1,1,0]^T$ and $[2,2,0]^T$) can be detected by filtering out all vectors for which the greatest common divisor of the norms of the non-zero coordinates, denoted as $gcd(\cdot)$, is greater than one. The full algorithm for this case is given in Algorithm 3.

Since we conduct a full enumeration of all rational optimization directions, and since the constraints of $constr(C_1)$ and $constr(C_2)$ are always conjuncts of polynomial constraints if they can be formulated as LMIs, this procedure leads to the following result.

Remark 4.5. If $constr(C_1)$ and $constr(C_2)$ can be formulated as LMIs and the intersection between the interiors of $constr_b(C_1)$ and $constr_b(C_2)$ is non-empty, then Algorithm 3 will eventually terminate with *true*. If the intersection between the closures of $constr_b(C_1)$ and $constr_b(C_2)$ is empty, then Algorithm 3 will eventually terminate with *false*. This is a simple consequence of the fact that both $R_b(C_1)$ and $U_b(C_1)$ will converge toward $constr_b(C_1)$ due to the even spacing of the optimization directions. Therefore, every point in the interior of $constr_b(C_1)$ will eventually lie in $R_b(C_1)$, and every point in the complement of the closure of $constr_b(C_1)$ will eventually lie outside $U_b(C_1)$. The same applies to $constr_b(C_2)$, $R_b(C_2)$ and $U_b(C_2)$.

This exhaustive enumeration can be inefficient, since the number of optimization directions grows exponentially with the number of steps needed. Therefore, as an alternative, heuristics for determining a promising optimization direction can be used.

One approach is to simply use the difference between the center points of $R_b(C_1)$ and $R_b(C_2)$ as the optimization direction. This heuristic has the advantage of not requiring any advanced computations. However, in general, there is the possibility that an existing intersection is not found using this method. In particular, this occurs if the two center

```
$D$ := list of cardinal directions in the parameter space;
calculate $R_b(C_1)$ based on $D$;
$D_1 := D$;
$D_2 := \{(d_1 + d_2) \mid d_1 \in D \wedge d_2 \in D \wedge d_1 \neq -d_2\}$;
if $constr(C_2) \wedge R_b(C_1)$ has a solution then
    return true;
end
if $constr(C_2)$ has a solution then
    calculate $R_b(C_2)$ based on $D$;
end
else
    return false;
end
repeat
    compute over-approximations $U_b(C_1)$ and $U_b(C_2)$ from $R_b(C_1)$ and $R_b(C_2)$;
    if $U_b(C_1) \wedge U_b(C_2)$ has no solution then
        return false;
    end
    if $D_2 = \emptyset$ then
        $D_2 := \big\{(d + d_1) \,\big|\, d \in D \wedge d_1 \in D_1 \wedge d^T d_1 \geq 0\big\}$;
        $D_1 := \emptyset$;
    end
    select a $d \in D_2$;
    $D_2 := D_2 - \{d\}$;
    $D_1 := D_1 \cup \{d\}$;
    if $gcd(d) = 1$ then
        add extremal point of $constr_b(C_1)$ in direction $d$ to list of extremal
        points for $R_b(C_1)$;
        add extremal point of $constr_b(C_2)$ in direction $(-d)$ to list of extremal
        points for $R_b(C_2)$;
    end
until $constr(C_2) \wedge R_b(C_1)$ has a solution;
return true;
```

Algorithm 3: Exhaustive Refinement Algorithm

points of $R_b(C_1)$ and $R_b(C_2)$ are not close enough to the center points of $\mathit{constr}_b(C_1)$ and $\mathit{constr}_b(C_2)$. This center point heuristic can however be combined with the full enumeration given in Algorithm 3. The sets of directions D_2 computed in each algorithm step can be sorted by angle to the difference of the two center points. This requires a scalar product computation for each possible (normed) optimization direction. The direction with the smallest angle can be assumed to be more likely to lead to a success and should therefore be selected first. After generating a new set D_2 in line 20, it can simply be sorted by this angle. The advantage of this approach lies in the fact that the result from Remark 4.5 is preserved.

A second heuristic uses extra variables in the LMI which have to be minimized. The general idea is – instead of picking a fixed optimization direction – to add the extremal point of $\mathit{constr}_b(C_1)$ that lies closest to $R_b(C_2)$. See Figure 4.43 for an illustration of this idea. This approach requires some changes to the LMI for a cycle, with the introduction of a *slack variable* serving as the distance measure. This auxiliary variable is required to be positive and then minimized. Assume that the extremal local Lyapunov functions of under-approximation $R_b(C_2)$ are given as $V_1(\cdot, b), \ldots, V_n(\cdot, b)$ for border node b.

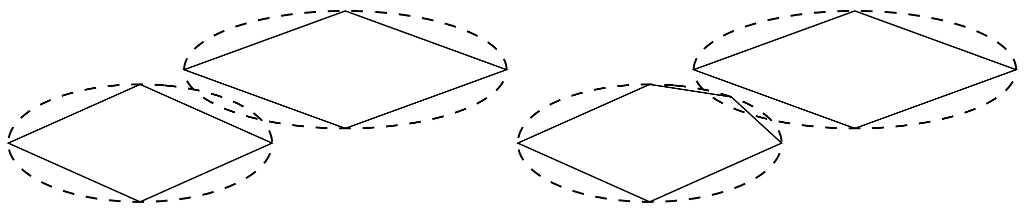

(a) before refinement (b) after refinement

Figure 4.43: Greedy Refinement Step

Then, the procedure is as follows. We solve the LMI problem for the sub-graph C_1, but with a modified LLF parametrization for the border node. This parametrization is $\sum_i \lambda_i V_i$, but we do not require the λ_i to be non-negative. Instead, we enforce the constraint that $\lambda_i \geq -\delta$, with $\delta \geq 0$. The optimization direction for the LMI is then set such that δ is minimized. If we obtain a solution with $\delta = 0$, then this implies that the newly discovered extremal point of $\mathit{constr}_b(C_1)$ intersects with $R_b(C_2)$. Therefore the newly updated $R_b(C_1)$ will do the same. If we obtain a solution with $\delta > 0$, then we add the solution to $R_b(C_1)$ anyway, as it represents the closest solution to $R_b(C_2)$ with respect to the distance metric defined by this slack variable. The process can then be repeated in an alternating manner for the two cycles, adding extremal points to both underapproximation until we obtain $\delta = 0$.

Note that this procedure is not complete, as progress is not necessarily guaranteed. In particular, the newly obtained extremal point might already be in $R_b(C_1)$, even though there is an intersection between $\mathit{constr}_b(C_1)$ and $\mathit{constr}_b(C_2)$. This can occur whenever the initial under-approximations are "bad" in the sense that they leave large parts of the solutions space uncovered, in an asymmetric manner. See Figure 4.44 for an illustration of such a situation, where the under-approximations $R_b(C_1)$ and $R_b(C_2)$ (both dashed) do not approximate $constr_b(C_1)$ and $constr_b(C_2)$ (both solid) well. The corner point of $R_b(C_1)$ is the closest point (wrt. the ∞-norm) to $R_b(C_2)$ in $constr_b(C_1)$ and vice versa.

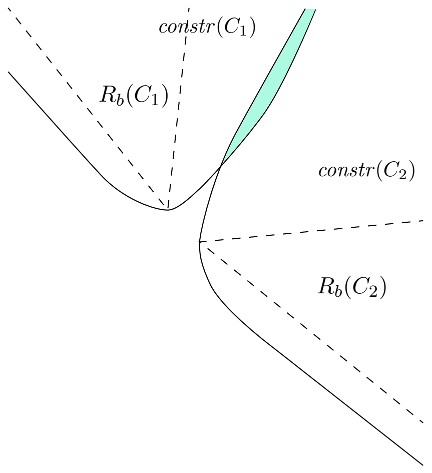


Figure 4.44: Deadlock Situation for Greedy Refinement

However, in many cases, this heuristic leads to fast termination. Unlike with the full enumeration approach, only extremal points in directions which seem promising are added. In case this heuristic shows no progress, the exhaustive enumeration approach can still be used as a fallback solution. It can also be useful to run the facet enumeration for a limited time, and then restart the distance heuristic approach with this additional information, in the hope that the tighter under-approximations will now result in convergence.

Instead of an ∞-norm based distance measure, other measures can also be used. The center point heuristic is one example, and also the 1-norm can be expressed as a optimization direction for an LMI. In the exhaustive method as an immediate fallback solution, switching to another distance measure first can also help enforcing progress in the refinement.

We now given an example for the two-cycle refinement procedure.

Example 4.9 (Refinement)**.** This example demonstrates the three different refinement approaches for two intersecting cycles. Consider the hybrid system given in Figure 4.45, which consists of two cycles with a total of three modes. Since all transition guards are *true*, per Theorem 4.7, each cycle requires a common Lyapunov function for its two modes.

For the left-hand cycle C_1 consisting of modes m_1 and m_2, we use a quadratic Lyapunov function template without linear or constant part of the form

$$V_{m_1} = V_{m_2} = [x, y]P \begin{bmatrix} x \\ y \end{bmatrix}, P = \begin{bmatrix} p_{1,1} & p_{1,2} \\ p_{1,2} & p_{2,2} \end{bmatrix}.$$

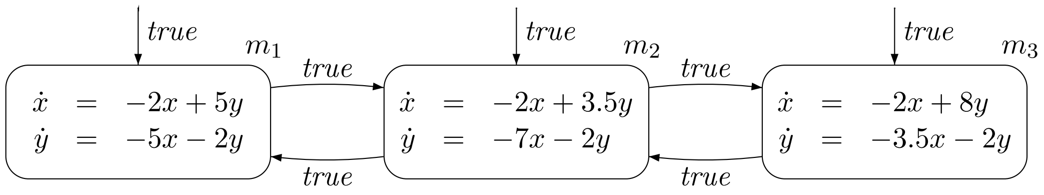


Figure 4.45: Two-Loop System

We start by formulating the LMI problem for this cycle and optimizing in the cardinal directions for the three free parameters, resulting in six Lyapunov functions (after normalization)

$$\begin{aligned}
V^1_{m_2,C_1} &= x^2 + 0.7388y^2\\
V^2_{m_2,C_1} &= 0.9618x^2 + y^2\\
V^3_{m_2,C_1} &= 0.8514x^2 + 0.4834xy + 0.6070y^2\\
V^4_{m_2,C_1} &= 0.8514x^2 - 0.4834xy + 0.6070y^2\\
V^5_{m_2,C_1} &= 0.9691x^2 + y^2\\
V^6_{m_2,C_1} &= x^2 + 0.5894y^2
\end{aligned}$$

with their conic hull forming the predicate $R_{m_2}(C_1)$. If we now attempt to compute a GLF for the second cycle C_2, using a parametrization $\sum_i \lambda_i V^i_{m_2}$ for the LLF of m_2, we find that the computation is not successful. Since, for the center point and minimal distance heuristics, we also require a predicate $R_{m_2}(C_2)$, we now formulate the LMI for the second cycle (without taking predicate $R_{m_2}(C_1)$ into account), again optimizing in the same directions and arrive at the following six Lyapunov functions:

$$\begin{aligned}
V^1_{m_2,C_2} &= 0.9161x^2 + y^2\\
V^2_{m_2,C_2} &= 0.9129x^2 + y^2\\
V^3_{m_2,C_2} &= 0.9026x^2 + 0.0662xy + 0.9887y^2\\
V^4_{m_2,C_2} &= 0.9026x^2 - 0.0662xy + 0.9887y^2\\
V^5_{m_2,C_2} &= 0.9129x^2 + y^2\\
V^6_{m_2,C_2} &= 0.9159x^2 + y^2
\end{aligned}$$

For the center point heuristic, we obtain an optimization direction given by the function $0.0285p_{1,1} - 0.2392p_{2,2}$ which is to be minimized. The extremal function of $constr_b(C_1)$ in this direction is

$$V^7_{m_2,C_1} \quad = \quad 0.9099x^2 + y^2.$$

If we add the extremal point $V^7_{m_2,C_1}$ to $R_{m_2}(C_1)$, we find that the Lyapunov function computation of cycle C_2 is now successful, completing the stability proof. Using the minimal distance heuristic, we obtain a slightly different extremal function, namely

$$V^7_{m_2,C_1} \quad = \quad 0.9141x^2 + y^2,$$

which again leads to a successful computation for C_2. The full enumeration approach succeeds as soon as the diagonal optimization direction minimizing $p_{1,1} - p_{2,2}$ is added, resulting in the same extremal function as for the center point approach.

4.7.2 Approximation Refinement for Strongly Connected Components

In the previous section, approximation refinement for two intersecting cycles has been discussed. We now need to extend these methods to approximation refinement across entire SCCs. Fortunately, this is rather straightforward. As the reduction graph as given in Definition 4.12 already gives us a tree structure to work with, we can use it to do backtracking, if the immediate refinement of two intersecting cycles does not lead to a positive result. This is the case if the two over-approximations in Algorithms 2 or 3 do not overlap or if we reach a previously defined upper bound on the number of refinement steps. In this case, we have to go back to another previously reduced cycle, refining its predicate R_b. It is desirable to do this backtracking on a breadth-first basis, since refinement of cycles which overlap directly is less costly.

Nevertheless, for the sake of completeness, we also give an approach to do refinement across several depths in the refinement graph, for instance refining a cycle on depth 3 to produce a solution for the root cycle. Just as in the previous section with two neighboring cycles, the refinement is based on a pair of cycles: the top-level cycle C_n for which we want to find a solution and the bottom-level cycle C_1, whose predicate R_b is to be refined. The cycles $C_2, \ldots, C_{n-1}$ complete the path in the refinement graph connecting C_1 and C_n. See Figure 4.46 for an example with $n = 4$.

Again, we employ guided refinement. To achieve this, we have to propagate some information "down the tree." This is done with the help of predicates $R^d_b(C)$, which are again under-approximation predicates. In contrast to this, the $R^u_b(C)$ predicates (which are equal to the original predicates $R_b(C)$ that were used for the reduction), propagate information up in the tree. Define the constraint systems

$$\begin{aligned} S(C_j) \quad := \quad & \mathit{constr}(C_j) \wedge \\ & \bigwedge \{R_{b_C}(C) \mid C \text{ is a child of } C_j \text{ connected by edge } b_C \wedge C \neq C_{j-1}\} \end{aligned}$$

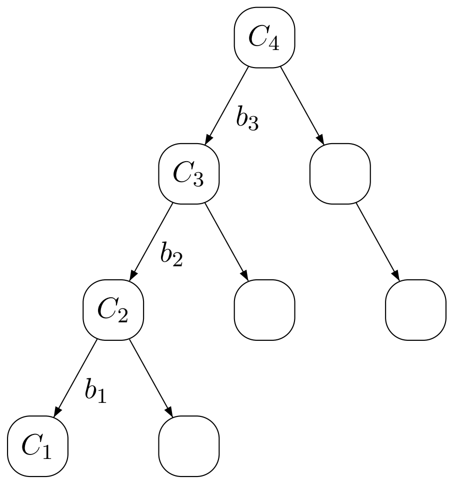


Figure 4.46: Reduction Graph with Connecting Path between Top-Level Cycle C_4 and Cycle to be Refined C_1

for the cycles C_2 to C_n, and

$$S(C_1) \quad := \quad \mathit{constr}(C_1) \wedge \bigwedge \{R_{b_C}(C) \,|\, C \text{ is a child of } C_1 \text{ connected by edge } b_C\}$$

for the cycle C_1 whose predicate $R_{b_1}(C_1)$ is to be refined. We assume that, at the time of refinement, we still know the under-approximation predicates $R_{b_C}(C)$ computed for each reduced cycle C, denoted as $R^u_{b_C}(C)$.

A refinement algorithm is then given in Algorithm 4. This algorithm is supposed to be executed if Algorithm 2 returns *false* or the loop in lines $11 - 20$ of Algorithm 2 has reached a fixed number of iterations, which is interpreted as a timeout. In this case, Algorithm 2 can be run to refine the predicate $R_{b_{C_1}}(C_1)$ of another previously reduced cycle C_1.

There are some differences to the two-cycle case. Most notably, one cycle can now intersect with several other cycles instead of just one. Also, if we want to guide the refinement of predicate $R^u_{b_{n-1}}(C_n)$, we somehow have to propagate information "down the tree." The idea here is similar to the two-cycle case. First, an under-approximation predicate $R^d_{b_{n-1}}(C_n)$ is computed in line 1. In contrast to the two-cycle case, we also need to take into account all the $R_{b_C}(C)$-predicates of other child cycles of C_n (apart from C_{n-1}), which is reflected in the definition of constraint system $S(C_1)$. This $R^d_{b_{n-1}}(C_n)$ is then used to compute a $R^d_{b_{n-2}}(C_{n-1})$ for C_{n-1}. Here, we need to take into account the $R_{b_C}(C)$-predicates from other children of C_{n-1}, as well as the $R^d_{b_{n-1}}(C_n)$. In this manner, recursively, we compute $R^d_{b_1}(C_2)$, which can then be used to guide the refinement of $R^u_{b_1}(C_1)$ in the same manner as for the two-cycle case. This propagation happens in lines 3-6. The loop in lines 7-15 then essentially conducts the same refinement as for just two intersecting cycles. If the refinement terminates with *true*, this means that $R^u_{b_1}(C_1)$

```
1  calculate $R^d_{b_{n-1}}(C_n)$ as an under-approximation predicate of $S(C_n)$ for the LLF
   of $b_{n-1}$ based on a fixed set of directions;
2  $j := n - 1$;
3  while $j \geq 2$ do
4      calculate $R^d_{b_{j-1}}(C_j)$ as an under-approximation predicate of
       $S(C_j) \wedge R^d_{b_j}(C_{j+1})$ for the LLF of $b_{j-1}$, based on a fixed set of directions;
5      $j{:=}j - 1$;
6  end
7  repeat
8      compute over-approximations $U^d_{b_1}(C_2)$ and $U^u_{b_1}(C_1)$ from $R^d_{b_1}(C_2)$ and
       $R^u_{b_1}(C_1)$;
9      if $U^d_{b_1}(C_2) \wedge U^u_{b_1}(C_1)$ has no solution then
10         return false;
11     end
12     choose an optimization direction $d$;
13     add extremal point of $S(C_1)$ in direction $d$ to list of extremal points for
       $R^u_{b_1}(C_1)$;
14     add extremal point of $S(C_2) \wedge R^d_{b_2}(C_3)$ in direction $(-d)$ to list of extremal
       points for $R^d_{b_1}(C_2)$;
15 until $S(C_1) \wedge R^d_{b_1}(C_2)$ has a solution;
16 return true;
```

Algorithm 4: Refinement Algorithm Outline

is now shaped such that the existence of a solution for the top cycle is guaranteed. However, it is now still necessary to re-compute $R^u_{b_2}(C_2)$, then $R^u_{b_3}(C_3)$, et cetera, until we can compute a conic predicate for C_n and continue with the decomposition. Since, for instance, the predicate $R^u_{b_2}(C_2)$ could theoretically again be too coarse, further refinement loops may be needed. The selection of cycles to which the refinement is applied should ideally be done by breadth-first search, since cycles with direct intersection points are much more likely to result improvement. Therefore, one should first try to refine the cycles at depth 2 of the tree, then at depth 3 and so on. This has the additional advantage that some computation results in Algorithm 4 can be cached and re-used. In particular, this applies to the computation of the predicates $R^d_{b_{j-1}}(C_j)$ which are used to guide the refinement. If a cycle is chosen for refinement, all of the $R^d_{b_{j-1}}(C_j)$-predicates for all cycles on the path to the root cycle have already been computed in previous steps, since refinement has been attempted unsuccessfully on these cycles before. This reduces the loop in lines 3-6 to a single computation: the last iteration only.

As soon as refinement across several cycles is necessary, this will of course greatly increase the computation time. This primarily occurs if solution sets from neighboring cycles overlap only marginally. Usually, just refinement of direct neighbors is sufficient, and often no refinement at all is needed (as, for instance, in Example 4.9). The need for multilevel refinement can also be alleviated by a suitable choice of initial optimization directions in the first place. Algorithm 4 is provided here for the sake of completeness, but it is usually preferable to prevent situations where this is required to begin with by choosing a larger set of initial optimization directions if necessary.

4.8 Summary

In this chapter, one of the main contributions of this thesis, a decompositional framework for Lyapunov-function-based stability analysis for hybrid systems, was presented. The first level of decomposition consisted of the identification of a hybrid automaton's SCCs, which can be treated completely independently, as far as Lyapunov function computation is concerned. If each SCC is globally attractive, then so is the entire system. Moreover, if each strongly connected component is globally stable and all bridge transitions are sub-linear, then the entire hybrid automaton will also be globally stable. Since Lyapunov functions that prove attractivity also always prove stability, this decomposition allows attractivity proofs also for some systems for which a monolithic Lyapunov-Function-based proof is impossible. In particular, this occurs when two GAS SCCs are connected via a not sub-linear transition into an automaton which is attractive, but not stable. Furthermore, this decomposition is lossless as far as the existence of Lyapunov functions is concerned. Therefore, whenever we can find a Lyapunov function proving GAS for the entire hybrid automaton, we can also do so individually for all SCCs. The result of this decompositional computation is a family of independent GLFs, one for each SCC. However, if so desired, these SCC-local GLFs can be combined into a true GLF for the entire system, whenever all bridge transitions are sub-linear. This computation is straightforward and based on the sub-linearity factors of the bridges.

The second level of decomposition involves the identification of cycle covers within an SCC. This is possible since each mode and transition within an SCC must lie on at least one simple cycle. We gave a reduction procedure that identifies simple cycles in the graph that have only one single intersection node with the rest of the automaton. For such a cycle, a series of GLF computations is conducted, yielding a number of Lyapunov functions proving the stability of the cycle itself. From these GLF computations, we deduced a conic, polytopic predicate on the free parameters of the single border node. The cycle is then removed from the graph and replaced by this predicate, which gives a sufficient condition on the free parameters of the border node for the existence of a Lyapunov function for the reduced cycle. If, by repeated application of this step, all cycles of the SCC have been removed, then it is GAS, and a GLF certifying this can be computed. If we have cycles intersecting with the rest of the automaton in more than one node, we used an equivalent transformation of the automaton to reduce the number of border nodes. This procedure takes nodes of degree larger than two and splits them into a number of nodes corresponding to the product of its indegree and outdegree. We also gave an algorithm conducting this splitting and the reduction in a manner such that termination is guaranteed. We then described how these conic, polytopic predicates can be computed using LMI methods and how to treat special cases of cycles which require common or continuous Lyapunov functions. Furthermore, we discussed how LLFs within a cycle can be employed to provide barrier certificates, proving that the cycle is not traversable and can therefore be ignored for the analysis.

The reduction procedure was then demonstrated on a cruise control application with six modes of operation, modeling maximum acceleration, standard acceleration and deceleration and two levels of brakes, each represented by a pair of modes. For this example, reduction, splitting and removal of non-traversable cycles were employed to arrive at a decompositional stability proof.

For the case that the conic, polytopic underapproximation of the LLF sets for the border nodes are too coarse, we also provided several algorithms for the refinement of these predicates. Two heuristics for directed refinement were given, choosing new optimization directions for the extremal point of the polytopic set based on the relative position of the solution sets for two intersecting cycles. Furthermore, we gave a refinement procedure which enumerated optimization angles in an evenly spaced manner, guaranteeing that solution set intersections which lie in the interior of the two solution sets are eventually detected. The refinement procedure can not only be used for neighboring cycles but also for pairs of cycles that do not intersect in the graph, if so desired.

In summary, a complete decompositional proof of GAS could be conducted as follows:

1. optionally, conduct reachability analysis, tightening the guards and invariants and removing superfluous transitions,
2. decompose the system into SCCs as in Section 4.3,
3. check whether all bridges have sub-linear updates: if yes, we show GAS, otherwise we show GA,
4. optionally, within each SCC, identify subgraphs that should receive a common or continuous GLF, as per Sections 4.5.2 and 4.5.3, and merge them into a single constraint graph node with all constraints conjoined,
5. apply the reduction and splitting procedure from Section 4.4 to every SCC, and for each cycle:
   a) check whether it is traversable, as described in Section 4.5.4,
   b) if it is, determine whether to use a common or continuous Lyapunov function as discussed in Sections 4.5.2 and 4.5.3,
   c) solve the corresponding LMI problem as per Section 4.5.1, and if no solution can be found, apply the reduction procedure described in Section 4.7 as needed, and
6. optionally, compute a GLF for the entire systems as per Remarks 4.3 and 4.4.

The following chapter will now carry these results into the domain of probabilistic hybrid automata: hybrid automata with probabilistic transitions. For this class of systems, probability-1-stability can also be shown by a variant of a GLF, which can again be computed decompositionally.

5 Stability Analysis of Stochastic Hybrid Systems

In this chapter, the general stability results of Section 3.3 and the decomposition approach of Chapter 4 are transferred to the domain of *probabilistic hybrid automata*. Probabilistic hybrid automata are a combination of hybrid automata and *discrete-time Markov processes*. In contrast to standard hybrid automata, where transitions lead to a single target mode, probabilistic hybrid automata allow for a Markovian probability distribution over several possible successor modes. Such a transition is still triggered by a guard set, but can now cause a mode change that is governed by a probability distribution, leading to different successor modes with different probabilities. Each successor mode can also have its own update function for the continuous variables. With this formalism, one can, for instance, model probabilistic plant behavior, with effects like the decay of system components or randomly varying environmental conditions. To discuss the stability of such systems, we define *global stability in probability* and *almost sure global attractivity*. Both of these properties are defined in a stochastic manner, allowing us, for instance, to reason about the probability of convergence. As it turns out, the application of Lyapunov functions to this class of systems is surprisingly straightforward: we only have to require that a Lyapunov function is always *expected to decrease* at any state and at any point in time. If we can prove the existence of such a relaxed Lyapunov function, then the system will always converge toward the equilibrium with probability 1. For probabilistic hybrid automata, we can again cast the problem of identifying such a Lyapunov function into an LMI problem. These parallels to non-probabilistic systems open up the possibilities of applying the very same decomposition techniques as presented in Chapter 4. In addition, some even stronger decomposition results can be obtained for probabilistic systems. These results are obtained by abstracting the system into the underlying Markov decision process and examining its steady state distributions.

First, in Section 5.1, we introduce probabilistic hybrid automata. We then move on to the definition of the relevant stability properties for this class of systems. The core Lyapunov theorem is then given in Section 5.2, together with a discussion of the corresponding LMI problems. Finally, in Section 5.3, we discuss the various decomposition techniques that can be applied to this class of systems.

5.1 Probabilistic Hybrid Automata and Stochastic Stability

First we define probabilistic hybrid automata, which form the base model for the analysis in this chapter. For the most part, the syntactic definition remains unchanged from the hybrid automata as defined in Section 3.2.2. The one exception are the transitions, which now take a probability distribution of possible successor modes instead of a single mode.

Definition 5.1 (Probabilistic Hybrid Automaton)**.** A *probabilistic hybrid automaton H* is a tuple

$$(\mathcal{M}, \mathcal{S}, \mathcal{V}, \mathcal{T}, \mathit{Flow}, \mathit{Inv}, \mathit{Init}),$$

where

- $\mathcal{M}$ is a finite set of *modes*,
- $\mathcal{S} = \mathbb{R}^{|\mathcal{V}|}$ is the *continuous state space*,
- $\mathcal{V}$ is the set of *continuous variables*, with each variable corresponding to a coefficient of $x \in \mathcal{S}$, in some fixed order,
- $\mathcal{T}$ is a set of *transitions* given as tuples (m, T, G, U), where
  - $m_1 \in \mathcal{M}$ is the *source mode*,
  - $T : \mathcal{M} \to [0, 1]$ with $\sum_{m \in \mathcal{M}} T(m) \leq 1$ is the *target mode distribution*,
  - $G \subseteq \mathcal{S}$ is the closed *guard set*,
  - $U : \mathcal{M} \to [\mathcal{S} \to \mathcal{S}]$ is the *update function map* for the continuous state variables,
- $\mathit{Flow} : \mathcal{M} \to [\mathcal{S} \to \mathcal{P}(\mathcal{S})]$ is the *flow function*, mapping each mode onto a set-valued function which in turn maps each $x \in \mathcal{S}$ onto a closed sub-set of $\mathcal{S}$, which is taken as the right-hand side of a differential inclusion $\dot{x} \in \mathit{Flow}(m)(x)$,
- $\mathit{Inv} : \mathcal{M} \to \mathcal{P}(\mathcal{S})$ is the *invariant function*, mapping each mode onto a closed sub-set of the continuous state space, and
- $\mathit{Init} \subseteq \mathcal{M} \times \mathcal{S}$ is the closed set of combinations of *initial discrete and continuous states.*

The function T inside each transition models the probabilities with which a target mode is selected. We do not require these probabilities to sum up to one. The remaining probability mass is interpreted as the chance to "end with a finite trajectory." We allow this behavior, since it is helpful for the decomposition of probabilistic hybrid automata. Some target modes of a transition in a sub-automaton might lie in a different sub-automaton. This definition ensures that sub-automata are indeed valid automata, by interpreting the probability of entering such a mode as the probability of a non-extendable finite trajectory.

Probabilistic hybrid automata are visually represented as hypergraphs, much in the same manner as standard hybrid automata are represented as graphs (see Remark 3.3). Each hyperedge connects the source mode to all modes m with $T(m) > 0$. The outgoing part of the hyperedge is labeled with the guard set, and the several incoming parts are labeled with the transition probabilities given by function T and the updates given by function U (see Figure 5.1 for an example). An example of a probabilistic hybrid automaton is given next.

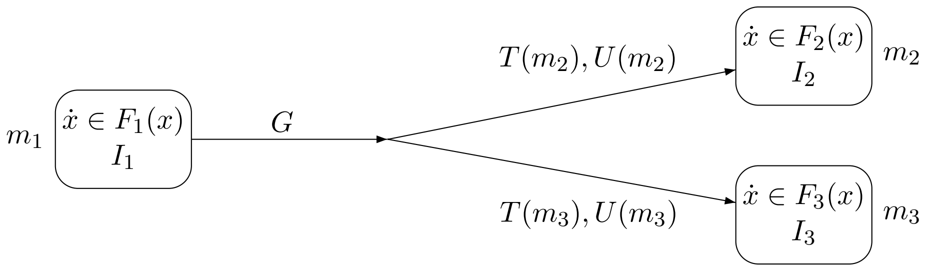


Figure 5.1: Graphical Representation of a Probabilistic Transition (m_1, T, G, U) with Two Target Modes

Example 5.1 (Probabilistic Cruise Controller)**.** An example automaton is given in Figure 5.2. The system is a probabilistic variant of the cruise controller from Section 4.6. The modes A and N remain unchanged, modeling maximal acceleration and behavior near the set point, respectively. In this example, there is only one level of brakes, which is modeled by the two modes B_1 and B_2, which decelerate the vehicle at a constant rate. However, upon activation of the brakes, there is a probabilistic transition, leading to each braking mode with a probability of 0.499. With the remaining probability of 0.002, the service brakes fail altogether, resulting in a transition to mode F, which models a backup emergency brake. In addition, mode E can be entered at any time from modes A, N, B_1, and B_2, modeling an external signal to stop the vehicle (for instance, effectuated by a human operator once the destination is about to be reached).

Probabilistic hybrid automata require a modified solution concept, since it no longer is useful to quantify over all individual trajectories. We are interested in stability probabilities for this class of systems. Therefore, we must be able to capture the probability mass corresponding to sets of trajectories. We will deal with *trajectory bundles*, which are represented as continuous-time stochastic processes. Recall that a trajectory of a standard hybrid automaton is essentially a resolution of its non-determinism. One of all the possible initial states is chosen, one possible solution of a differential inclusion is chosen, one of multiple possible successor modes is chosen and so on. If we do the same for probabilistic hybrid automata (resolving the non-determinism only and not the probabilism), we obtain a tree of trajectories where each branch carries a probability mass. This bundle of trajectories can therefore be viewed as a stochastic process, describing a "tree" of trajectories with associated probabilities. In contrast to trajectories of non-probabilistic systems, the distinction between a finite and an infinite time horizon, which was required for the definition of the stability property (see Definition 3.15), cannot be made on entire bundles of trajectories, since some branches may be finite and others infinite. Therefore, we encode the termination of a trajectory by the additional symbol "$\perp$". If the total probability mass of a target distribution $T(m)$ is less than one, then the remaining probability mass is also interpreted as leading to termination of the trajectory, encoded by "$\perp$."

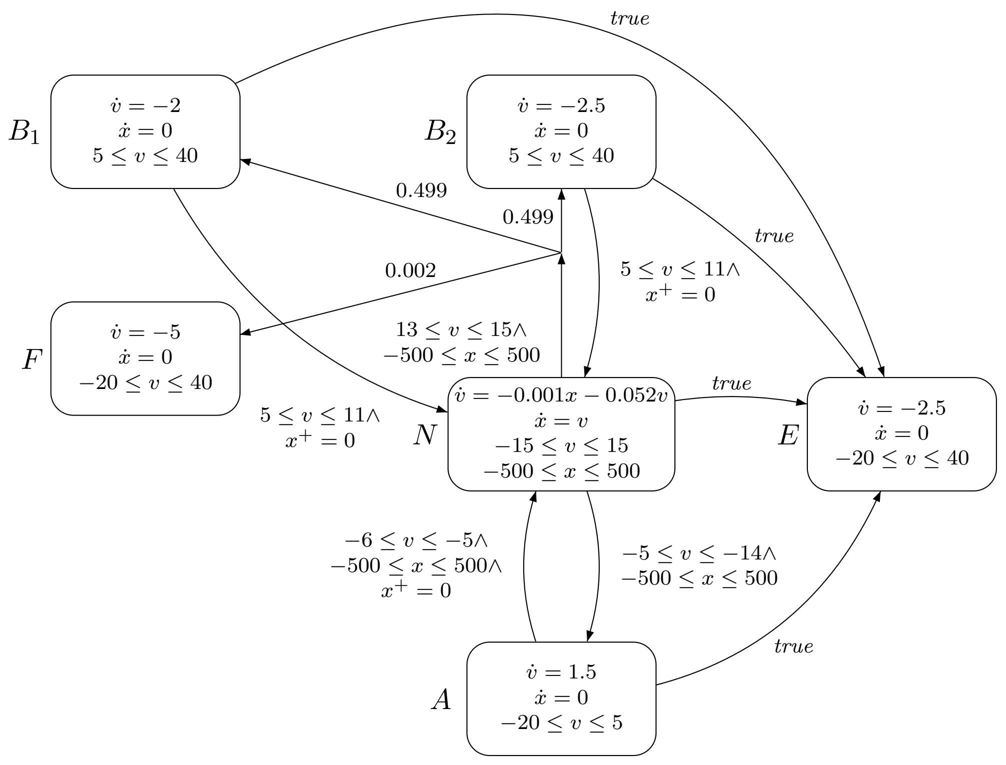


Figure 5.2: Example of a Probabilistic Hybrid Automaton

Definition 5.2 (Trajectory Bundle)**.** A *trajectory bundle* of a probabilistic hybrid automaton H is an absolutely continuous stochastic process $X(t)$ on $\mathcal{S}_H \cup \{\perp\}$ obtained by resolving the non-determinism of H in any manner, by choosing any strategy,

- selecting a fixed initial state x_0,
- choosing a trajectory segment for every differential inclusion in every mode, every time the mode can be reached,
- deciding whether to switch or not if guards and invariants overlap, and
- selecting an outgoing transition if several guards overlap.

Furthermore,

- $\text{prob}\{X(t) = x\}$ is the probability that a trajectory is in state x at time t, and
- $\text{prob}\{X(t) = \perp\}$ is the probability that a trajectory has terminated at time t.

Define $\mathcal{R}(H)$ as the set of all such stochastic processes obtained by all possible resolutions of non-determinism. Also, with each trajectory bundle $X(t)$ associate the stochastic process $M(t)$ describing the probability distribution of the discrete mode over time. Note that the stochastic processes $X(t)$ and $M(t)$ are not independent.

The distributions $X(0)$ and $M(0)$ are Dirac distributions, that is, $X(0)$ and $M(0)$ have a probability of 1 for one single initial state and mode, respectively. This is because resolution of non-determinism includes the selection of a unique initial state. Therefore, we will sometimes use $X(0)$ and $M(0)$ to denote the initial state and mode instead of the Dirac distributions.

In contrast to GAS as defined in Section 3.3.1, stochastic stability properties do not necessarily guarantee a desired behavior under all circumstances, but only with a certain probability. First, we will define *global stability in probability* and *almost sure global attractivity.*

Definition 5.3 (Global Stability in Probability and Almost Sure Global Attractivity). A probabilistic hybrid automaton H is *globally stable in probability (GS-P)* if

$$\forall p \in [0,1) : \forall \epsilon > 0 : \exists \delta > 0 : \forall X \in \mathcal{R}(H) : ||X(0)|| < \delta \implies \\ \operatorname{prob}\{\forall t : (X(t) \neq \bot \implies ||X(t)|| < \epsilon)\} > p$$

and *almost surely globally attractive (AS-GA)* if for all trajectory bundles $X \in \mathcal{R}(H)$

$$\operatorname{prob}\left\{(\forall t : X(t) \neq \bot) \implies \lim_{t\to\infty} X(t) = 0\right\} = 1.$$

A GS-P system guarantees that, for each bound ϵ on the distance to the equilibrium and for each probability $p < 1$, there exists a bound δ on the initial state, such that all trajectories starting in the δ-ball will remain in the ϵ-ball forever with probability p. This is a straightforward modification of the standard global stability property as defined in Definition 3.15 on page 39. Note that we do not require the existence of a δ-ball for $p = 1$, and therefore allow the probability of eventually leaving any ϵ-ball to be larger than zero in all cases. For global stability, we considered all finite and infinite trajectories and required the states $x(t)$ to be within the ϵ-ball for all times for which $x(t)$ was defined. We made no assertions after the termination of a finite non-extendable trajectory. For GS-P, we handle this similarly: Trajectories are counted as if they are inside the ϵ-ball once they have terminated. Clearly, every globally stable system is also GS-P.

The definition of AS-GA is similarly straightforward. We require that the trajectories of the system converge to the equilibrium with probability 1. In line with the definition of global attractivity, where we only considered infinite trajectories, we count every finite trajectory (reaching "$\bot$" at some point in time) as convergent. The difference in treatment of GS-P and AS-GA is rooted in the fact that for stability, we consider all trajecories *until their termination*, whereas for attractivity it only makes sense to consider *non-terminating trajectories.* Note that AS-GA is weaker than standard global attractivity. Probabilistic hybrid automata usually permit infinitely many possible system trajectories. Therefore, for an AS-GA system, it may still be possible that a sub-set of trajectories does not converge, as long as this sub-set has probability mass 0.

Next, we outline how GS-P and AS-GA can be shown via relaxed Lyapunov functions and how such functions can be computed numerically.

5.2 Lyapunov Functions for Probabilistic Systems

The Lyapunov functions for GS-P and AS-GA defined next are similar to Lyapunov functions for GAS, in the sense that we again allow a LLF of a given parametrization for each mode. As in the non-probabilistic case, these LLF are interrelated by a non-increasingness condition such that a GLF for the entire system results. In contrast to the non-probabilistic case, we can however relax the constraints somewhat. Instead of requiring a decrease of the function value at all times, it is sufficient to require a decreasing *expected value*. In case of probabilistic hybrid automata, this entails weakened non-increasingness conditions that only require the expected value after a switch to be smaller than the value prior to the switch. Therefore, we explicitly allow case where an increase in Lyapunov function value takes place, as long as this is not the expected behavior. In the long run, this “counterproductive” behavior will be compensated by the expected, convergent behavior.

For the proof, we first need to define *supermartingales*. Supermartingales are stochastic processes which capture exactly this behavior: expected non-increasingness. Moreover, powerful existing theorems on the convergence of supermartingales will allow us to conduct a proof of GS-P and AS-GA.

Definition 5.4 (Supermartingale [Shiryaev, 1996, p. 474]). A (continuous-time) *supermartingale* is a continuous-time stochastic process $X(t)$ such that for all $s \geq 0$ and $t \geq s$:

$$E(X(t) \mid \{X(\tau), \tau \leq s\}) \leq X(s)$$

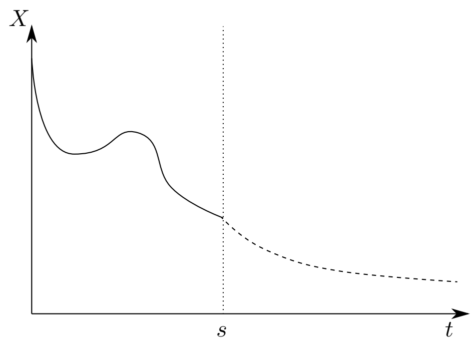


Figure 5.3: Sketch of the Behavior of a Martingale: Observed Values of X up to Time s (solid) and Expected Future Values (dashed)

Here, $E(X(t) \mid \{X(\tau), \tau \leq s\})$ denotes the conditional expected value of random variable X at time t, under the assumption that the behavior up to time s is given as the $X(\tau), \tau \leq s$. Informally, a supermartingale is a stochastic process where, for any evolution up to time s, the expected value of $X(t)$ for all future times t is lower than the value at time s. One might picture this as a stochastic process whose value is never expected to increase under any circumstances. See Figure 5.3 for an illustration. The graph shows the evolution of a stochastic process which is a supermartingale. Up to time s, the process behaved according to the solid line. Even though the process was never expected to increase, this does of course not mean that an increase is impossible. However, it is improbable in the long run. Nevertheless, the process in Figure 5.3 shows a temporary increase for some time. Now, the dashed line shows the *expected* values for times $t > s$. Since the process is a supermartingale, none of these values can be higher than $X(s)$. The expected values could stay constant at $X(s)$, however.

Doob's theorem, as given next, is a fundamental result on supermartingales. It can be seen as the stochastic equivalent to the *monotone convergence theorem* from classical calculus, which states that a monotonic, bounded function on $\mathbb{R}$ must always converge. In contrast, Doob's theorem talks about supermartingales, which can be seen as a stochastic equivalent of a monotonic function, and states that supermartingales with bounded expectation converge almost surely.

Theorem 5.1 (Doob's Theorem [Shiryaev, 1996, p. 505])**.** Let $X(t)$ be a supermartingale with $\sup_t E(|X(t)|) < \infty$. Then, with probability 1, the limit $\lim_{t\to\infty} X(t)$ exists, and $E(\lim_{t\to\infty} X(t)) < \infty$.

Doob's theorem can be exploited to prove a Lyapunov function theorem for the properties GS-P and AS-GA of probabilistic hybrid automata. The basic idea is to require only the expected value of the Lyapunov function to decrease. Now, for the hybrid automaton, look at any trajectory bundle $X(t)$ with associated mode sequence bundle $M(t)$. Clearly, the evolution of the Lyapunov function value over time for $X(t)$ and $M(t)$ results in a stochastic process. Moreover, since the Lyapunov function values are always expected to decrease, the stochastic process describing them is a supermartingale. The application of Doob's theorem then gives the desired property. This Lyapunov theorem for probabilistic hybrid automata is given next. Its proof is based on a proof sketch by Loparo & Feng [1996] for stochastic differential equations, which in turn is inspired by ideas of Kushner [1967]. In contrast, the following theorem focuses on probabilistic hybrid automata.

Theorem 5.2 (Discontinuous Lyapunov Functions for Probabilistic Hybrid Systems)**.** Let H be a probabilistic hybrid automaton. If for each $m \in \mathcal{M}$ there exists a continuously differentiable function $V_m : \mathcal{S} \to \mathbb{R}$ such that

(1) for each $m \in \mathcal{M}$ there exist class K^∞ functions f_1 and f_2 such that for all $x \in Inv(m)$: $f_1(||x||) \leq V_m(x) \leq f_2(||x||)$,

(2) for each $m \in \mathcal{M}$ there exists a class K^∞ function f_3 such that for all $x \in Inv(m)$: $\dot{V}_m(x) \leq -f_3(||x||)$, where $\dot{V}_m(x) := \sup\left\{\left\langle \frac{dV}{dx} \mid y\right\rangle \middle| y \in Flow(m)(x)\right\}$,

(3) for each mode transition $(m_1, T, G, U) \in \mathcal{T} : x \in G \implies$
$\sum_{m\in\mathcal{M}} (T(m) \cdot V_m(U(m)(x))) \leq V_{m_1}(x) \cdot \sum_{m\in\mathcal{M}} T(m)$,

then H is GS-P and AS-GA. The family of LLFs $V_m, m \in \mathcal{M}$ forms a function $V(x,m) : \mathbb{R}^n \times \mathcal{M} \to \mathbb{R}$, where $V(x,m) = V_m(x)$. This function $V(x,m)$ is called the *probabilistic global Lyapunov function (P-GLF)* of H.

Proof. Attractivity: Let $X(t) \in \mathcal{R}(H)$ be a trajectory bundle of H and let $M(t)$ be the associated stochastic process describing the evolution of the discrete mode over time. Define $\bar{X}(t)$ as the stochastic process over $\mathcal{S}$ obtained from $X(t)$ by removing all occurrences of $\perp$, that is for all t,

$$\text{prob}\{\bar{X}(t) = x\} := \text{prob}\{X(t) = x \mid X(t) \neq \perp\}$$

and define $\bar{M}(t)$ analogously:

$$\text{prob}\{\bar{M}(t) = m\} := \text{prob}\{M(t) = m \mid M(t) \neq \perp\}.$$

Define the stochastic process $W(t)$ with

$$W(t) = V(\bar{X}(t), \bar{M}(t))$$

giving the probability distribution of the global Lyapunov function value at time t. Also define $\dot{W}(t)$ with

$$\dot{W}(t) = \dot{V}(\bar{X}(t), \bar{M}(t))$$

describing the distribution of the time derivative of the Lyapunov function value over time, if W is differentiable at t.

Define $\mathcal{I}$ as the set of all time intervals (t_I^L, t_I^U) such that for no trajectory of the bundle $\bar{X}(t)$ a transition in the hybrid automaton is taken within the interval. Define $\mathcal{D}$ as the set of mode switch events (including loop transitions) along any trajectory of $\bar{X}(t)$. For a $D \in \mathcal{D}$, define Δ_D as the difference of Lyapunov function values before and after the switch and p_D as the probability mass of the switch event within the trajectory bundle. By continuous differentiability of V, $W(t)$ is absolutely continuous on each interval $[t_I^L, t_I^U]$. Therefore, there exist sub-sets $\mathcal{I}' \in \mathcal{I}$ and $\mathcal{D}' \in \mathcal{D}$ such that

$$E(W(t)) = W(0) + E\left(\sum_{I\in\mathcal{I}'} \int_{t_I^L}^{t_I^U} \dot{W}(\tau) d\tau\right) + E\left(\sum_{D\in\mathcal{D}'} p_D \Delta_D\right).$$

Per Condition (2), $\dot{V}$ is non-positive, and therefore also $E(\dot{W}(\tau))$ for all τ. This means that the second summand is non-positive. Per Condition (3), $E(\Delta_D)$ is non-positive and therefore also the third summand. We obtain

$$E(W(t)) \leq W(0),$$

and since $W(t)$ is time-invariant,

$$E\left(W(t) \,|\, \{W(\tau), \tau \leq s\}\right) \leq W(s).$$

Therefore, $W(t)$ is a supermartingale. Furthermore,

$$0 \leq E(W(t)) \leq W(0) = V(\bar{X}(0), \bar{M}(0)) < \infty.$$

Therefore, $E(|W(t)|) = E(W(t)) < \infty$ for all $t \geq 0$ and Doob's theorem can be applied, giving us

$$\text{prob}\left\{\exists x_0 : \lim_{t\to\infty} W(t) = x_0\right\} = 1.$$

Condition (2) implies that $x_0 = 0$, since there must be at least one $m \in \mathcal{M}$ with $\dot{V}(x_0, m) = 0$. Therefore

$$\text{prob}\left\{\lim_{t\to\infty} W(t) = 0\right\} = 1,$$

and per the definition of $W(t)$ and condition (1),

$$\text{prob}\left\{\lim_{t\to\infty} \bar{X}(t) = 0\right\} = 1.$$

This directly implies

$$\text{prob}\left\{(\forall t : X(t) \neq \bot) \implies \lim_{t\to\infty} X(t) = 0\right\} = 1.$$

Stability: Since $W(t)$ is a supermartingale, the following inequality [Shiryaev, 1996, p. 493] holds for all $\tilde{\epsilon} > 0$:

$$\text{prob}\left\{\exists t : W(t) \geq \tilde{\epsilon}\right\} \leq W(0)/\tilde{\epsilon}.$$

Let f be a class K^∞ function which is pointwise smaller than all class K^∞ functions f_1 for all modes $\mathcal{M}$ in Condition (1). Let $\epsilon > 0, 0 \leq p < 1$, and pick some $\tilde{\epsilon} < f(\epsilon)$. Then,

$$\text{prob}\{\exists t : W(t) > f(\epsilon)\} \leq \text{prob}\{\exists t : W(t) \geq \tilde{\epsilon}\} \leq \frac{W(0)}{\tilde{\epsilon}(1-p)}(1-p).$$

Set

$$\delta := \inf\{||x|| \,|\exists m \in \mathcal{M}_H : V_m(x) \geq \tilde{\epsilon}(1-p)\} > 0,$$

then $||X(0)|| < \delta$ implies that

$$\text{prob}\left\{\exists t : W(t) > f(\epsilon)\right\} < 1 - p,$$

and per the definition of $W(t)$ and condition (1),

$$\text{prob}\left\{\exists t : ||\bar{X}(t)|| > \epsilon\right\} < 1 - p,$$

and

$$\text{prob}\left\{\forall t : (X(t) \neq \bot \implies ||X(t)|| < \epsilon)\right\} \geq \text{prob}\left\{\forall t : ||\bar{X}(t)|| < \epsilon\right\} > p.$$

$\square$

Note that the Constraints (1)-(3) in Theorem 5.2 are very similar to those in Theorem 3.5. The only difference lies in Constraint (3): Instead of requiring non-increasingness of the GLF value upon switching, we average the LLF values of different possible target modes, weighting them by their transition probabilities. The result is that we just require the GLF to decrease on average. However, this averaging constraint can again be expressed as an LMI constraint in the very same manner as before. Since all other constraints remained unchanged altogether, the search for a P-GLF can again be conducted with the help of LMI solvers.

If the mode dynamics are given by Itō-type *stochastic differential equations* [Øksendal, 2003] of the form

$$dx = f_m(x)dt + g_m(x)dB,$$

where B is a standard Wiener process, then a P-GLF V proving GS-P and AS-GA can be defined in a similar manner. By Itō's stochastic calculus, we would obtain on each interval $I \in \mathcal{I}'$

$$\begin{aligned} dW(t) \;&=\; \left(\left\langle \frac{dV}{dx}(x)\middle| f_m(x)\right\rangle + \frac{1}{2} g_m(x)^T \frac{d^2V}{dx^2}(x) g_m(x)\right) dt \\ &\quad + \left(\left\langle \frac{dV}{dx}(x)\middle| g_m(x)\right\rangle\right) dB, \end{aligned}$$

for each mode m that can be active on I. Therefore, Condition (2) can be replaced by

(2) for each $m \in \mathcal{M}$ there exists a class K^∞ function f_3 such that for all $x \in Inv(m)$: $\dot{V}_m(x) \leq -f_3(||x||)$, where $\dot{V}_m(x) := \left\langle \frac{dV}{dx}(x)\middle| f_m(x)\right\rangle + \frac{1}{2} g_m(x)^T \frac{d^2V}{dx^2}(x) g_m(x)$

in this case, implying that $E(\dot{W}(t))$ is still non-positive for all t. This still lets us conclude that $W(t)$ is a supermartingale, with the rest of the proof remaining unchanged.

Due to the similarity to the non-stochastic case, such P-GLFs can be computed via LMI solvers in an analogous manner. This is true both for probabilistic hybrid automata with standard differential inclusions and also for stochastic differential equations.

Theorem 5.3 (LMI for GS-P and AS-GA of Probabilistic Hybrid Automata with Convex, Affine Differential Inclusions)**.** Let $\epsilon > 0$ and let H be a probabilistic hybrid automaton. Let the $Q_j^m \in \mathbb{R}^{n+1} \times \mathbb{R}^{n+1}$ be matrices, such that for every mode $m \in \mathcal{M}$

$$x \in Inv(m) \implies [x, 1]\, Q_j^m \begin{bmatrix} x \\ 1 \end{bmatrix} \geq 0,$$

and let the $R_j^e \in \mathbb{R}^{(n+1)\times(n+1)}$ be matrices, such that for every transition $e \in \mathcal{T}$ with guard set G

$$x \in G \implies [x, 1]\, R_j^e \begin{bmatrix} x \\ 1 \end{bmatrix} \geq 0.$$

Assume that *Flow*(m) is a convex, affine differential inclusion for each m, given by a set of functions $f_i^m(x) = A_i^m x + b_i^m$. Furthermore, assume that, for each transition e

and each target mode m with $T(m) > 0$, the associated update function is affine, i.e., $U(m)(x) = A^e_m x + b^e_m$. Define

$$\tilde{I} = \begin{bmatrix} I & 0 \\ 0 & 0 \end{bmatrix},$$

where I is the $n \times n$ identity matrix. If the LMI problem

Find $P_m \in \mathbb{R}^{n+1} \times \mathbb{R}^{n+1}$ and $\alpha, \mu^m_j, \nu^m_j, \eta^m_j, \vartheta^e_j \in \mathbb{R}$, such that

$$\begin{aligned}
\alpha - \epsilon &\succeq 0 && (5.1)\\
\beta - \epsilon &\succeq 0 && (5.2)\\
\text{for all } m, j : \mu^m_j, \nu^m_j, \eta^m_j &\succeq 0 && (5.3)\\
\text{for all } e, j : \vartheta^e_j &\succeq 0 && (5.4)\\
\text{for all } m : P_m - \sum_j \mu^m_j Q^m_j - \tilde{I} &\succeq 0 && (5.5)\\
\text{for all } m : P_m + \sum_j \nu^m_j Q^m_j - \beta\tilde{I} &\preceq 0 && (5.6)\\
\text{for all } m, i : \begin{bmatrix} (A^m_i)^T & 0 \\ (b^m_i)^T & 0 \end{bmatrix} P_m + P_m \begin{bmatrix} A^m_i & b^m_i \\ 0 & 0 \end{bmatrix} + \sum_j \eta^m_j Q^m_j + \alpha I &\preceq 0 && (5.7)\\
\text{for all } e = (m_1, T, G, U) : \sum_{m \in \mathcal{M}} T(m) P_{m_1} - &&&\\
\sum_{m \in \mathcal{M}} \left(T(m) \begin{bmatrix} (A^e_m)^T & 0 \\ (b^e_m)^T & 1 \end{bmatrix} P_m \begin{bmatrix} A^e_m & b^e_m \\ 0 & 1 \end{bmatrix} \right) - \sum_j \vartheta^e_j R^e_j &\succeq 0 && (5.8)
\end{aligned}$$

has a solution, then H is GS-P and AS-GA.
In case of stochastic differential equations, with

$$dx = (A_m x + b_m)dt + (C_m x + d_m)dB,$$

Constraint (5.7) needs to be replaced by (see [Korenevskii, 1987]):

$$\begin{aligned}
&\text{for all } m : \begin{bmatrix} (A^m)^T & 0 \\ (b^m)^T & 0 \end{bmatrix} P_m + P_m \begin{bmatrix} A^m & b^m \\ 0 & 0 \end{bmatrix} + \\
&\begin{bmatrix} (C^m)^T & 0 \\ (d^m)^T & 0 \end{bmatrix} P_m \begin{bmatrix} C^m & d^m \\ 0 & 0 \end{bmatrix} + \sum_j \eta^m_j Q^m_j + \alpha I \preceq 0
\end{aligned}$$

In the differential inclusion case, the only difference to the LMI system of Theorem 3.10 lies in Constraint (5.8), which has been changed to match Constraint (3) of Theorem 5.2. The weighted sum of matrices represents the expected Lyapunov function value after the probabilistic switch and update.

Now we have established means for stability verification for probabilistic hybrid systems and systems with stochastic differential equations. Due to the similarity to the non-probabilistic case, all the arguments for decomposition, as given in Section 4.1, also apply here. In the following, we will examine the applicability of decomposition techniques to the stochastic case in detail.

5.3 Decomposition Techniques

In order to conduct decomposition, we need to formulate an equivalent to the constraint graph formalism of Chapter 4. Since probabilistic hybrid automata allow for several target modes of a transition, we will now define *constraint hypergraphs* which are simply hypergraphs with Lyapunov constraints. The nodes are labeled in the same manner as for constraint graphs. The edges also are labeled with transitions constraints, but with one constraint for each target vertex. In addition to this constraint hypergraph, we also need to keep track of the target distributions T_e for each edge e. The hypergraphs are defined as follows.

Definition 5.5 (Hypergraph)**.** A *(directed) hypergraph* G is a tuple (V_G, E_G, L_G^V, L_G^E), where V is a set of *nodes* and $E_G \subseteq \mathcal{P}(V_G \times \mathcal{P}(V_G))$ is a multiset of *hyperedges*, and L_G^V and L_G^E are labeling functions defined on V_G and $E_G \times V_G$, respectively, and mapping to some label set. Here, $L_G^E(e, v'), e = (v, V)$ is a partial function that is defined if and only if $v' \in V$.

Paths in hypergraphs can be defined analogously to graphs, and therefore also SCCs and cycles. Note that simple cycles of hypergraphs are actually graphs, since each edge inside a simple cycle can only have one single successor node. The underlying hypergraph $G(H)$ of a probabilistic hybrid automaton H can then also be defined analogously, dropping the information on transition probabilities (i.e., the node and edge labels are the flows, invariants, guards and updates only, but target modes with $T(m) = 0$ can be dropped). We can then define constraint hypergraphs as follows.

Definition 5.6 (Constraint Hypergraph)**.** The *constraint hypergraph*

$$C(H) = (V_C, E_C, L_C^V, L_C^E)$$

of a probabilistic hybrid automaton H with underlying hypergraph

$$G(H) = (V_G, E_G, L_G^V, L_G^E)$$

is a graph with:

1. $V_C = V_G$
2. $E_C = E_G$

3. $L_C^V(m)$ is the conjunction of the constraints (1) and (2) from Theorem 5.2 for the mode m, that is,

$$L_C^V(m) :\iff \left(\begin{array}{cl} & \exists f_1, f_2 \text{ in class } K^\infty : \\ & x \in \mathit{Inv}(m) \implies f_1(||x_{|\mathcal{V}_m}||) \leq V_m(x) \leq f_2(||x_{|\mathcal{V}_m}||) \\ \wedge & \exists f_3 \text{ in class } K^\infty : x \in \mathit{Inv}(m) \implies \dot{V}_m(x) \leq -f_3(||x_{|\mathcal{V}_m}||) \end{array} \right)$$

4. $L_C^E((m_1, M), m_2), m_2 \in M$ is Constraint (3) from Theorem 3.7 for all transitions (m_1, T, G, U), that is,

$$L_C^E((m_1, M), m_2) :\iff x \in G \implies (V_{m_2}(U(m_2)(x)) \leq V_{m_1}(x))$$

For all $e \in E_C$, let $T_e(e)(\cdot)$ be the target distribution of the associated transition $(m_1, T, G, U) \in \mathcal{T}_H$. Define *constr*$(H)$ as:

$$\mathit{constr}(H) := \bigwedge_{m \in V_C} L_C^V(m) \wedge \bigwedge_{(m_1, M) \in E_C}$$
$$\left(x \in G \implies \sum_{m \in M} T_{(m_1,M)}(m) V_m(U(m)(x)) \leq \sum_{m \in M} T_{(m_1,M)}(m) V_{m_1}(x) \right)$$

Here, each node is labeled with a constraint in the same manner as for constraint graphs for non-probabilistic hybrid automata. Each edge has one label per target node: the non-decreasingness condition for the unique source mode and the particular target. The global constraint *constr*(H) then averages all constraints belonging to a transition according to the transition probabilities, yielding the overall GLF constraint.

The constraint hypergraph itself contains the non-increasingness constraints for a normal GLF and not the Constraints (3) from Theorem 5.2. This is because, during the decomposition, individual target modes of transitions might be removed, leaving us to re-compute the weighted sum in that constraint. Therefore, this linear combination according to the transition probabilities is done in *constr*(H) instead of in the constraint hypergraph itself. In order to obtain *constr*(H), we therefore require the distributions $T(e)$ in addition to the constraint graph.

Note that, for constraint hypergraphs, we cannot contract loops into nodes as we did for constraint graphs. The reason for this is that each loop transition has an associated transition probability which we must keep track of.

One key observation is that probabilistic hybrid automata can be seen as a special case of *discrete-time Markov decision processes (MDPs)*, enriched with continuous behavior which is attributed to the nodes and transitions of the MDP. The MDP belonging to a probabilistic hybrid automaton is obtained by taking its hypergraph and labeling the transitions with the transition probabilities. Markov decision processes are, in our case, defined as follows.

Definition 5.7 (Markov Decision Process)**.** A *(discrete-time) Markov decision process (MDP)* is a tuple (V, E), where V is a set of *nodes* and E is a set of *transitions.* Each transition $e \in E$ is given as a tuple of the form (v, T), where $v \in V$ is the *source node* and $T : V \to \mathbb{R}$ with $\sum_i T_i \leq 1$, gives the *transition probability* from v to any other node. Also define $\mathit{trans}(M) \subseteq V$ as the set of *transient states* of M, that is, the set of all states with supremal steady-state probability of 0 (i.e., all states with steady-state probability of zero regardless of resolution of non-determinism).

Essentially, a MDP is a *Markov chain* enhanced by possible non-determinism. In contrast to Markov chains, a state of an MDP can have several outgoing transitions, each with its own probability distribution. One of these outgoing transitions can be chosen non-deterministically. Hence, steady-state distributions must either be associated with one particular resolution of this non-determinism, or given by upper and lower bounds over all such resolutions.

This definition of MDPs is simplified, as we do not attach *actions* or *rewards* to the transitions, as it is sometimes done in the literature [Puterman, 1994]. In general, these actions and rewards can be used to formulate strategies to resolve the non-determinism inside an MDP. We assume that each transition corresponds to an unique action, which we do not need to refer to directly, as we will always quantify over all possible resolution strategies. Therefore, it is not necessary to explicitly include actions in the model. Also, we do not need to associate with transitions, since the Lyapunov functions essentially take this role. Furthermore, we allow that the transition probabilities of a transition do not sum up to 1, but to a smaller number. We interpret the remaining probability mass as resulting in a transition to an implicit “sink state.” This “sink state” is used to model probabilistic termination of a trajectory.

A probabilistic hybrid automaton can simply be abstracted into an MDP by disregarding all invariants, flows, guards, updates, and initial states.

Definition 5.8 (MDP Corresponding to a Probabilistic Hybrid Automaton)**.** The MDP (V, E) corresponding to a probabilistic hybrid automaton

$$H = (\mathcal{M}, \mathcal{S}, \mathcal{V}, \mathcal{T}, \mathit{Flow}, \mathit{Inv}, \mathit{Init})$$

is given by

1. $V = \mathcal{M}$,
2. $E = \{(m_1, T) \mid (m_1, T, \cdot, \cdot) \in \mathcal{T}\}$.

Denote the MDP corresponding to probabilistic hybrid automaton H as $M(H)$.

In the following, three kinds of decomposition for probabilistic hybrid automata are discussed. First, in Section 5.3.1, we will discuss SCC-based decomposition, which also allows for quantitative analysis of probabilistic stabilization. Then, in Section 5.3.2, a cycle-based decomposition approach is outlined. In contrast to standard hybrid automata, the probabilistic case does not necessarily require a GLF for each cycle. Instead, deficiencies in one cycle can potentially be compensated by the stability of adjacent cycles. Finally, in Section 5.3.3, we describe a third type of decomposition, which maintains AS-GA only. However, this decomposition is again lossless with respect to Lyapunov functions, and it can be applied *inside* SCCs. The method is based on analyzing the steady-state behavior of underlying MDPs.

5.3.1 Decomposition of Probabilistic Hybrid Automata into Strongly Connected Components

The first type of decomposition is simply a straightforward application of the results for non-probabilistic systems given in Section 4.3. Quite simply, the results from Theorem 4.1 apply also for this case, replacing global stability by global stability in probability and global attractivity by almost sure global stability. Therefore, if we can decompose a probabilistic hybrid automaton H into several SCCs C_i, and if all the C_i are AS-GA, then so is H. Furthermore, if all bridge transitions are sub-linear and each SCC C_i is GS-P, then so is H. This is formalized in the following theorem.

Theorem 5.4 (Decomposition of Probabilistic Hybrid Automata into Strongly Connected Components)**.** Let H be a probabilistic hybrid automaton. If all sub-automata pertaining to the SCCs of H are AS-GA then H is AS-GA. If all C_i are GS-P and all transitions (m_1, m_2, G, U) corresponding to bridges of $M(H)$ are sub-linear, then H is GS-P.

The proof of Theorem 4.1 can be adapted in a straightforward manner. To prove GS-P, include the stochastic relaxation in the chain of implications, that is:

$$\exists\, \delta_i^p > 0 : ||X_i(t_i)|| < \delta_i^p \implies \operatorname{prob}\left\{\forall t \in [t_i, t_{i+1}] : ||X_i(t)|| < \delta_{i+1}^p / c_i\right\} > \sqrt[s]{q}.$$

Here, $0 \le q < 1$ is some probability and s is the total number of SCCs. Then, with the same arguments as in the proof of Theorem 4.1, this implies that for each ϵ there exists a $\delta(\epsilon)$ with

$$\forall \epsilon > 0 : ||X(0)|| < \delta(\epsilon) \implies \operatorname{prob}\{\forall t \ge 0 : ||X(t)|| < \epsilon\} > q,$$

by multiplying the probabilities. The followup results and discussions to Theorem 4.1 are also readily adapted to the probabilistic case.

Example 5.2. The example system from Figure 5.2 can be broken down into three separate SCCs which can be treated separately, as shown in Figure 5.4. The bridges connecting the SCCs have been removed from the hybrid automaton, yielding three separate SCCs.

However, we can take these results one step further. Suppose that we are unable to prove GAS for one SCC C_i. This could be because SCC C_i is in fact unstable or because our Lyapunov function computation fails for other reasons. In the non-probabilistic case, we would not be able to deduce anything about the stability of the system, as we would have to assume the worst-case: instability of the system. However, in the probabilistic case, we can possibly still derive a lower bound on the probability of convergence for any given trajectory. This is because it might be possible to quantify the probability p_i that C_i is actually entered. Clearly, all trajectories not entering C_i will converge with probability 1. Therefore, even assuming a convergence probability of 0 within C_i (which might be too pessimistic), we can guarantee an overall convergence probability of p_i. Define global attractivity for probabilities below 1 as follows.

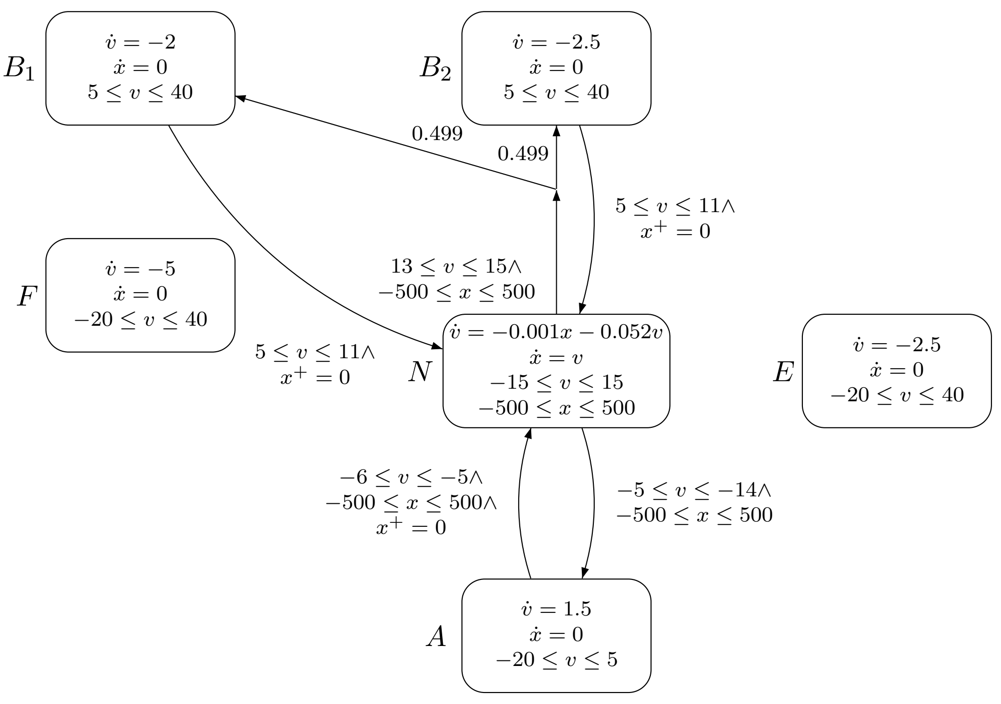


Figure 5.4: SCC Decomposition of Example Automaton

Definition 5.9 (Global Attractivity with Probability $p \le 1$)**.** A probabilistic hybrid automaton H is *globally attractive with probability* p *(GA(p))*, if for all trajectory bundles $X \in \mathcal{R}(H)$

$$\text{prob}\left\{(\forall t : X(t) \neq \bot) \implies \lim_{t\to\infty} X(t) = 0\right\} \ge p.$$

Figure 5.5 shows a schematic view of a system consisting of several SCCs which are GA(p) with different probabilities. Of course even every unstable system will always be GA(0), therefore the individual SCCs could also be unstable. The probabilities $\underline{q_i}$ are lower bounds on transition probabilities from one SCC to another, over all trajectory bundles. Now the probability of convergence for the entire system can easily be derived easily as follows.

Theorem 5.5 (Quantitative Analysis)**.** Let H be a probabilistic hybrid automaton, consisting of an SCC C which is GA(p) and a number of SCC $C_1, \dots, C_m$ that are successors of C. Assume that *Init* only contains hybrid states with nodes of C as their discrete state. Define $\underline{q_i} := \inf\{X \in \mathcal{R}(H) \,|\, \text{prob}\{M(t) \text{ enters } C_i\}\}$ and $\overline{q_i} := \sup\{X \in \mathcal{R}(H) \,|\, \text{prob}\{M(t) \text{ enters } C_i\}\}$. If each C_i is known to be GA(p_i) for some $0 \le p_i \le 1$,

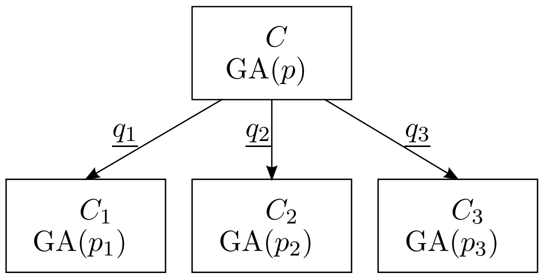


Figure 5.5: Schematic View of System with GA(p) SCCs

then H is GA($\bar{p}$) with

$$\bar{p} = \sum_i \underline{q_i} p_i + p \left(1 - \min \left(\sum_i \overline{q_i}, 0 \right) \right).$$

Proof. Let $X(t)$ be a trajectory bundle of H with discrete state distribution $M(t)$. If $M(t)$ enters C_i, then $\operatorname{prob}\{(\forall t : X(t) \neq \bot) \implies \lim_{t\to\infty} X(t) = 0\} \geq p_i$. If $M(t)$ does not enter any C_i, then $\operatorname{prob}\{(\forall t : X(t) \neq \bot) \implies \lim_{t\to\infty} X(t) = 0\} \geq p$. Since a lower bound on probability that no C_i is entered is given as $1 - \min(\sum_i \overline{q_i}, 0)$, we directly obtain the equation for $\bar{p}$ as the weighted sum of the individual p_i and p. $\square$

The computation of the lower and upper bounds on the transition probabilities between the SCC is essentially a probabilistic reachability problem. Therefore, tools from that domain can be used to compute such bounds, see, for example, the results in [Zhang *et al.*, 2010; Abate, 2007].

However, to obtain tight bounds on the stability probability, it is necessary to have an automaton model of the system where all "branching points" are visible as transitions between SCC in the graph structure. At this point, methods for reachable set computation of hybrid systems can be employed to discover semantically equivalent automata (with respect to the continuous behavior) that have a "finer" SCC structure. Here, the goal is to re-formulate and split only the SCCs for which stability could not be shown (i.e., where $p_i = 0$). If such an SCC C can be replaced by two SCC C_1 and C_2, and if stability can be shown for C_1, this leads to an increase in the overall chance of convergence, as long as C_1 is entered with a positive probability.

5.3.2 Decomposition of Probabilistic Hybrid Automata into Simple Cycles

Since the decomposition results given in Theorem 4.3 are only formulated on constraint graphs for Lyapunov function computation, the results can also be applied to P-GLFs.

A cycle inside a probabilistic hybrid automaton cannot have multiple target nodes for one transition, and also not more than one outgoing transition per node. Therefore, when formulating *constr*(C) for a cycle C, we can disregard the transition probabilities. A cycle inside a probabilistic hybrid automaton is in fact not probabilistic any more:

any transitions inside a cycle can only have one target mode, as outgoing transitions of the cycle are not part of the actual cycle.

Consider a system consisting of two overlapping cycles, as given in Figure 5.6. Here, the cycles overlap in a single node, but share a probabilistic transition. When the system is in mode m_1 and the transition is triggered, there is a probability of p_1 for traversing cycle C_1 and a probability of p_2 of traversing cycle C_2. Now, one could simply ignore the probability information and treat the hyperedge as two separate non-probabilistic transitions, leaving us with the same type of decomposition as in Section 4.4. This would lead to a proof of GAS, which is stronger than GS-P and AS-GA. However, this might of course not always be possible. As opposed to the non-probabilistic case, individual cycles of the automaton might actually be unstable, and yet the overall system is GS-P and AS-GA. Therefore, we need to cater for cases where this occurs.

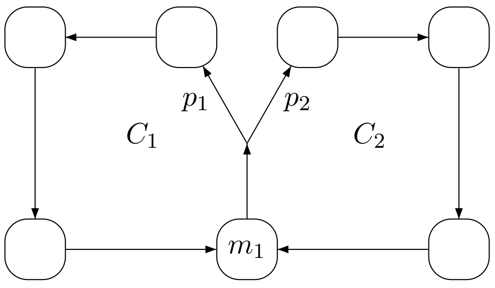


Figure 5.6: Two Cycles Sharing a Probabilistic Transition

Look at an automaton that is equal to C_1, with the only difference that the transition into cycle C_2 is eliminated. Now, assume that this system is unstable. In this case, we cannot find a GLF for C_1. Nevertheless, the entire system consisting of both C_1 and C_2 might be stable. This can occur if the "non-stability margin" of C_1 is outweighed by the "stability margin" of C_2. In other words, C_2 needs to be a) "stable enough" and b) entered with high enough probability, such that the behavior of C_1 is compensated.

To prove GS-P and AS-GA of these systems with unstable cycles, we need to be able to measure these "stability/non-stability margins" of individual cycles and weight them by the probabilities with which they are entered. If the average difference between the Lyapunov function values before and after the transitions is negative, then, per Condition (3) in Theorem 5.2, this does still imply GS-P and AS-GA.

Assume that C_1 has only a single border node b. Then, this can be achieved by relaxing the Condition (3) for the sub-automaton given by cycle C_1 as follows:

(3') for b's single outgoing transition $(b, m_2, G, U) \in \mathcal{T}$ inside cycle C_1 : $x \in G \implies V_{m_2}(U(m_2)(x)) \leq V_{m_1}(x) + \alpha_b ||x||^2$

for some $\alpha_b \in \mathbb{R}$. The α_b can be minimized in the LMI problem. If $\alpha_b \leq 0$, then cycle C_1 is GAS, and the value of α_b acts as a stability margin on the transition. If $\alpha_b > 0$, then cycle C_1 has not been shown to be GAS, and this behavior needs to be compensated by cycle C_2. In any case, we also need to modify Constraint (3) for cycle C_2. Replace it by

(3') for each mode transition $(r, m_2, G, U) \in \mathcal{T}$ originating in any reduced node r: $x \in G \implies V_{m_2}(U(m_2)(x)) \leq V_{m_1}(x) - \frac{\alpha_r p_1}{p_2}||x||^2$,

where p_1 and p_2 are the transition probabilities into cycles C_1 and C_2, respectively.

Here, the variable α_b is used to measure the "degree of satisfaction or violation" of the non-increasingness condition for the outgoing edge of b. This is the only edge inside cycle C_1 which can be part of a hyperedge with more than one possible target in H. This is caused by the fact that b is the only border node of cycle C_1.

In place of $\alpha_b||x||^2$, also any other suitable class-$\mathcal{K}^\infty$ function on $||x||$ could be used, which may depend on the special structure of the Lyapunov functions. For instance, for polynomial LLFs of degree 4, one might use a quartic instead of a quadratic function.

This approach allows for cycle-based decomposition of SCCs, much the same as for the non-probabilistic case. For the computation of the border predicates R_b, one can again compute extremal points of the solution set. If the cycle still contains probabilistic edges at the time of reduction, then we need to keep track of the value of variable α_b for each computed GLF of the cycle to be reduced. Note that the splitting procedure of Section 4.4 can be used to reduce the number of border nodes per cycle to one. This also implies that there can only be one such edge in the cycle, and it must originate in the border node. After reducing a cycle, we again obtain a border predicate R_b that can be used to define the LLF parametrization for the reduced node for the subsequent cycle. In addition, we obtain a family of values α_b, which can be linearly combined in the same manner as the individual corner LLFs of the conic polytopic predicate R_b. Therefore, the very same decomposition framework presented in Section 4.4 can be applied to probabilistic systems.

5.3.3 MDP-based Decomposition of Probabilistic Hybrid Automata within Strongly Connected Components

We described how automaton decomposition methods for non-probabilistic system can be lifted to the probabilistic case. In addition to these decomposition techniques, probabilistic hybrid automata also offer *additional* possibilities of decomposition. To see this, remember that LLFs for an individual mode can act as a termination function for that mode. To be exact, the existence of an LLF proves that for all trajectories entering the mode:

- either the system stays in this mode forever, in which case convergence of any trajectory to the equilibrium is shown,
- or the system eventually leaves the mode again.

In the first case, the trajectory is convergent. This holds even if the hybrid automaton contains probabilistic transitions. This case is the simple one: If this was true for all trajectories of the system, then the existence of LLFs for each mode would be enough to prove AS-GA. We could conduct an ideal decomposition by just looking at all modes individually. Of course, in general, this is not possible. We can, however, treat modes separately which are transient in the MDP. Now asume that the trajectory's mode sequence does not stay in a single mode forever. Then, the probability of being in any transient mode converges to 0 as time approaches infinity.

To identify nodes with steady state probability of 0, graph-based reasoning can also be employed. If all cycles a node lies on can only be traversed finitely often, then the node must be transient. We can, for instance deduce that a cycle has this property if a transition within the cycle has, with positive probability, another possible successor mode which lies in another SCC. In this case, this transition to another SCC is eventually taken with probability 1, rendering the cycle unreachable from that point on (see Figure 5.7).

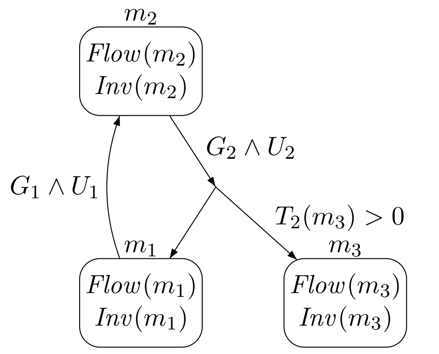


Figure 5.7: Transient Cycle Consisting of m_1 and m_2

We now give a theorem which allows for this separate treatment of transient modes.

Theorem 5.6 (MDP-based Decomposition)**.** Let H be an SCC of a probabilistic hybrid automaton. Let H' be the sub-automaton obtained by removing the modes of $\mathit{trans}(M(H))$ from H, together with their outgoing transitions and their probability mass in target distributions $T(m)$ in all other transitions. If the following two conditions both hold, then H is AS-GA:

1. for each $m \in \mathit{trans}(M(H))$ there exists a LLF V_m, and
2. there exists a P-GLF for H'

Proof. All trajectories with finite mode sequences are convergent, as there exists a LLF for the final mode of the sequence, either per Condition 1 or as part of a P-GLF per Condition 2. Therefore, it is sufficient to examine only trajectories with infinite mode sequences. Let $X(t)$ be a trajectory bundle of H, and define $\bar{X}(t)$ by removing all finite mode sequences:

$$\text{prob}\{\bar{X}(t) = x\} := \text{prob}\left\{X(t) = x | M(\cdot) \text{ changes infinitely often}\right\}.$$

Note that this also removes all terminating trajectories, for which eventually $M(t) = \perp$ holds. With the argument above, it is sufficient to show that $\bar{X}(t)$ converges to 0 almost surely. We know that

$$\text{prob}\{\exists t' > 0 : \forall t > t' : M(t) \notin \mathit{trans}(M(H))\} = 1,$$

since all nodes in $\mathit{trans}(M(H))$ have a steady-state probability of 0 and therefore the probability of visiting them infinitely often is zero. Since, per Condition 2, there exists a P-GLF for the sub-automaton consisting of the nodes in H', we obtain that $\bar{X}(t) \to 0$ with probability 1. □

Example 5.3. Consider again the example system from Figure 5.2, with its decomposition into SCCs given in Figure 5.4 The cycles formed by N and B_1 and N and B_2 are only traversed infinitely often with probability 0, and the modes B_1 and B_2 are transient in the MDP. This can easily be seen in Figure 5.2, since there is a positive probability of entering mode F every time a transition from N with B_1 or B_2 as possible target is triggered. Therefore, we can analyze B_1 and B_2 completely separately from the rest of the SCC.

This leads to the decomposition shown in Figure 5.8, with only one non-probabilistic cycle remaining. In order to show that this system is AS-GA, we would need to find a GLF for this cycle (e.g., via an LMI problem) and a LLF each for the modes E, F, B_1, and B_2.

5.4 Summary

In this section, the results for Lyapunov-based stability analysis from Chapter 3 and the decomposition results from Chapter 4 have been transferred to the stochastic domain. As our main model we used probabilistic hybrid automata, but we also discussed how stochastic differential inclusions can be analyzed in the modes. Instead of guaranteed convergence, we then obtain convergence with probability 1, implying that a only set of probability mass zero containing non-convergent trajectories might exist. The decomposition into SCCs as well as the cyclic decomposition can be conducted on probabilistic hybrid automata as well. The SCC-based analysis can also be employed for quantitative stability analysis, together with probabilistic reachability analysis. The cyclic decomposition can now tolerate unstable cycles, if their "degree of instability" is compensated by other cycles, as measured by a Lyapunov function. In addition, a further level of decomposition is possible by interpreting the hybrid automaton as a Markov decision process. If this Markov decision process contains discrete states with steady-state probabilities of zero, this implies that the probability of returning to the corresponding mode of the hybrid automaton infinitely often is also zero. This allows us to completely split the analysis of such modes from the rest of the automaton, since they are only relevant when they are the final mode in a mode sequence.

We now move on to the composition of stable hybrid systems, employing the decomposition results established so far in this thesis.

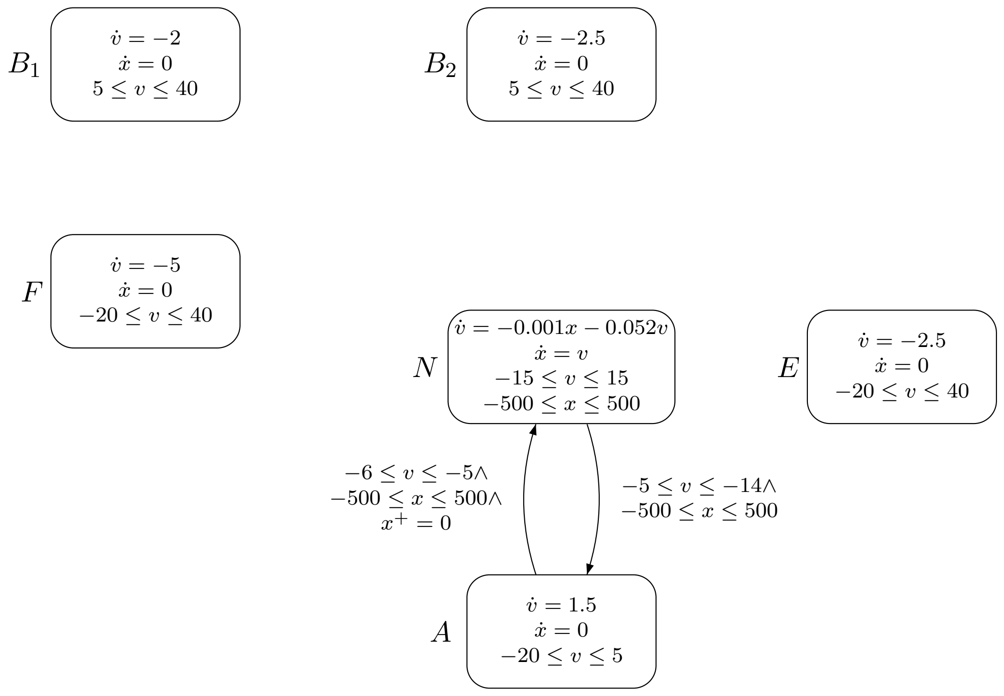


Figure 5.8: MDP-based Decomposition of Example Automaton

6 Structured Design of Stable Systems

The previous three chapters dealt with the problem of proving the stability and attractivity of a given hybrid automaton. However, it is not necessarily clear how to arrive at a GAS hybrid automaton, since the verification procedure is still used only *a posteriori*. Clearly, the results on decompositional verification can also be applied in reverse: for incremental design of stable hybrid automata. By exploiting the decomposition results, it is possible to exploit verification information to build a stable hybrid system model by subsequently adding new modes of operation in an admissible manner. This information can, for example, come in the form of Lyapunov functions or the SCC structure of the automaton. Another point of interest is the generalization of a stable hybrid automaton. The Lyapunov function information and the $\mathcal{S}$-procedure terms can be exploited to "robustify" a hybrid automaton by widening its differential inclusions, invariants, and guards. This information can be used to prove that a hybrid system is GAS even if perturbed by bounded modeling inaccuracies or permanent external disturbances.

This chapter is divided into two parts. First, rules for the composition and transformation of hybrid automata are presented, such that stability properties are preserved. The goal is to provide a rule set for the construction of stable hybrid automata, such that the stability property can easily be verified or is already guaranteed by construction. For instance, if a new mode of operation covering some emergency situation is to be added to a hybrid automaton, it is useful to know under what circumstances this addition does not destroy the stability of the system. Ideally, one wants to obtain this information without having to re-prove stability of the entire system. Section 6.1 deals with this issue on the basis of hybrid automata and provides such a rule set.

Then, in Section 6.2, a truly component-based design approach is presented. Here, the premise is that a component, which is associated with a hybrid automaton, is viewed as a "grey box." Neither the system dynamics nor the discrete switching inside a component are allowed to be visible at its interface. Also, some continuous variables of the hybrid automaton may be hidden from view. This information is replaced by Lyapunov function information on the remaining visible continuous variables which the component promises to control. The emphasis is to make only the minimum level of information visible that is necessary for a proof of stability, in order to achieve maximal information hiding within a component.

6.1 Stability-preserving Transformations of Hybrid Automata

This section builds on the results of Chapters 4 and 5 and gives a set of transformation rules for hybrid automata guaranteeing the preservation of GAS, GS-P, or AS-GA. These rules can be divided into four main groups.

- *Transformation rules that do not add new trajectories to the system:* This group is the most straightforward. If a system is GAS and we apply a transformation which is either equivalent in terms of trajectories, or which removes some trajectories from the system, then the stability property is trivially preserved. This group includes the splitting operation from Theorem 4.4, any removal of modes and edges, concretization of differential inclusions, and some other basic operations.
- *Transformation rules that add new trajectories but always maintain stability:* This group contains transformations which, while potentially adding new trajectories of the system, still maintain stability, regardless of the methods that were used for the proof on the original system. This includes equilibrium shifts, re-scaling of differential equations, compositions of stable automata as separate SCCs, and convex hull operations. For these transformations, no new verification conditions arise in the process.
- *Transformation rules which require Lyapunov-function-based reasoning:* In this group, additional verification conditions need to be satisfied in order to conduct the transformations. One example is the cyclic composition in the fashion of Section 4.4. Other transformations include changing a differential inclusion of a mode into another "wider" or non-intersecting differential inclusion, or $\mathcal{S}$-procedure-based changes to invariants and guards. As soon as edges between SCC are added which close a cycle in the automaton, Lyapunov arguments are also needed.
- *Transformation rules for stability in probability:* This group contains transformations that turn a GAS hybrid automaton into a GS-P and AS-GA probabilistic hybrid automaton, as well as transformation rules between probabilistic hybrid automata. This category also includes stochastic differential equations, probabilistic transitions, and transformations that decrease the probability of convergence but still maintain AS-GA with a certain probability.

In the following we give a collection of transformation rules, starting with the simple ones not requiring additional Lyapunov arguments.

Tightening of Sets, and Removal of Edges and Nodes. Clearly, any removal of transitions or modes from the original hybrid automaton without changes to the rest of the system can only remove possible trajectories. The same applies to the tightening of any initial, invariant, or guard sets or the right hand sides of any differential inclusions.

Note that, due to the $\mathcal{S}$-procedure over-approximations of invariant and guard sets, tighter invariants or guards can turn a system for which no Lyapunov function can be found into one for which stability can be shown. Therefore, it is of paramount importance to keep invariant and guard sets as small as possible, without impacting the set of possible trajectories of the system. This is achieved by making sure that as few unreachable states as possible are covered by these sets. For instance, it is generally unnecessary to include states in a guard set that are not also part of the invariant set of the transition's source mode. In this case, these enlarged guard sets can only add extra instantaneous transitions (i.e., transitions which fire immediately upon entering the mode). However, since we are only interested in the evolution of the continuous states, such transitions can be replaced by transitions bypassing the mode in question altogether.

If the guard in question can represented by a hyperplane, then sometimes even simpler reasoning is possible. It is sufficient to examine the vector field on this hyperplane, and if it points in the direction opposite to the guard set, then the transition is not possible.

Invariants can be tightened without removing any trajectories from the system by reachable set computation. This entails a computation (usually an over-approximation) of the set of states in the invariant which are actually reachable in the hybrid automaton. Since we apply Lyapunov-based methods, each Lyapunov function can simply also function as a barrier certificate, giving us such an over-approximation, as discussed in Section 3.4. The process of obtaining tight invariants is important, as the invariant is the set on which the corresponding Lyapunov function is defined and must adhere to the Lyapunov conditions. If this set is unnecessarily large, Lyapunov function computation may become impossible even for a stable system.

Splitting Modes. The splitting of modes as discussed in Section 4.4 is another transformation that preserves the trajectories of a hybrid automaton, as long as only continuous trajectories are considered.

There are also other uses for mode splitting besides enabling a cyclic decomposition. For instance, a mode m with invariant $\mathit{Inv}(m)$ can be split into two new modes m_1 and m_2 with invariants such that $\mathit{Inv}(m_1) \cup \mathit{Inv}(m_2) = \mathit{Inv}(m)$, and the same flow as mode m. Additionally, two transitions need to be added, one from m_1 to m_2 and one in the reverse direction. Each of these transitions is labeled with the guard set $\mathit{Inv}(m_2) \cap \mathit{Inv}(m_2)$ and no discrete update. Both modes m_1 and m_2 also inherit all incoming and outgoing edges to/from mode m.

Clearly, this transformation does not impact the set of possible trajectories of the automaton. However, when piecewise continuous Lyapunov functions with one LLF per mode are used, then this transformation allows the use of a *different* LLF for m_1 and m_2, resulting in extra degrees of freedom. See [Pettersson & Lennartson, 1996] for a detailed example of a system where this type of mode splitting enables stability verification that would otherwise be impossible. Moreover, if the partitioning of $\mathit{Inv}(m)$ is chosen in a smart manner, some of the edges might be superfluous. For instance, if the guard of the transition connecting m_1 and m_2 can be described by a hyperplane, then simple vector field analysis on this plane might indicate that transitions from m_1 to m_2 are indeed impossible. This is also a potential method to uncover new SCCs, and therefore reduce the complexity of the overall problem. However, identifying such smart refinements is in general a hard problem, requiring the use of heuristic algorithms [Oehlerking *et al.*, 2007].

Shifting the Equilibrium. One simple point that is widely exploited in stability analysis is *translation invariance*. In other words, shifting the coordinate axes for the continuous states does not impact stability. It is this property that allow us to assume, without loss of generality, that the equilibrium we are interested in is the origin of the continuous state space. This also works the other way: If we apply a constant shift to the entire automaton, then stability is maintained with respect to an equilibrium that is shifted in the same manner.

Moreover, if the system consists of several strongly connected components, then we can shift them separately. Shifting an equilibrium of only one SCC does not preserve global stability, but still guarantees global attractivity with respect to the *set of different local (i.e., per SCC) equilibria* in the system. Therefore, SCC can be treated independently if one is satisfied with the weakened property of convergence to a set of states instead of global asymptotic stability.

Re-scaling Differential Inclusions. Going one step further, global asymptotic stability is also invariant to re-scalings/multiplications in the state space. The important observation here is that this re-scaling need not be global, but *each mode can receive its own scaling factor.* Here, re-scaling is taken to mean the multiplication of the right hand side of the differential inclusion of a mode by a positive scalar. While it is easy to see that this works for one global scaling factor, the fact that scaling factors can be different for each mode needs some explanation.

Consider a two differential equations $\dot{x} = f(x)$ and $\dot{x} = 2f(x)$. They differ in the sense that the continuous variables of the latter change twice as fast as for the former. However, the phase plots with respect to the same initial state will look identical. The second system merely traverses it twice as fast. Since GAS in its basic form does not talk about convergence time, these two systems can be considered identical from a purely GAS standpoint. Naturally, the latter system will converge twice as fast, which will be mirrored by the class $\mathcal{K}^\infty$ function multipliers.

Another way to see this is through Lyapunov functions. We know that the set of Lyapunov functions form a conic set, and we also know that a single Lyapunov function for a number of dynamics always also is a Lyapunov function for their conic hull. The possible Lyapunov functions for $f(x)$ and $2f(x)$ with respect to any parameterization will always lie in exactly the same cones, so any Lyapunov-function-based stability proof for the former will also work for the latter and vice versa.

Composing Stable SCCs. As shown in Section 4.3, multiple hybrid automata can be composed into one new automaton if the connecting edges are such that each sub-automaton becomes a separate SCC. If the update functions are sub-linear, then the resulting system will be GAS, otherwise it will be globally attractive. Note that, except for the sub-linearity constraint, the guards and updates on the connecting edges can be *arbitrary.* In case only global attractivity is of interest, the guard can simply be set to *true* and the update to any function.

Note that this does *not* include connection structures that result in any newly introduced cycles. Alternatively, existing guards and updates on bridges of the system can also be modified, maintaining stability/attractivity, with the only constraint that sub-linearity must be guaranteed if global asymptotic stability is to be preserved.

Adding/Closing Cycles. As soon as new cycles are created inside a hybrid automaton, stability properties are no longer automatically preserved. Lyapunov functions can be used to derive conditions under which this is still the case. Most notably, such transformations result in new *local* conditions on parts of the hybrid automaton that need to be satisfiable in conjunction with the constraints implied by the "rest of the automaton." Suppose that we already have a family of GLFs V_i for the hybrid automaton (not including the new edge) and that a new edge (m_1, m_2, G, U) is to be added to the automaton. Then, we need to find a family of multipliers λ_i fulfilling the constraints

$$x \in G \implies \sum_i \lambda_i V_i(U(x), m_2) \leq \sum_i \lambda_i V_i(x, m_1),$$

which is a very simple LMI problem. Each family of λ_i satisfying this constraint then gives a possible GLF for the hybrid automaton *including* the new edge, given as $\sum_i \lambda_i V_i$. This approach is also used for the component-based design methodology described in the following Section 6.2.

Widening Guards and Invariants. In addition to changes to guards and invariants, a complete Lyapunov-function-based proof of GAS for a system allows for relaxations that widen the set of trajectories, but still maintain GAS. One source of relaxations is the $\mathcal{S}$-procedure. When solving an LMI problem (be it monolithically or cycle-by-cycle), one also obtains values for the $\mathcal{S}$-procedure multipliers $\mu_j^m, \nu_j^m, \eta_j^m$, and ϑ_j^e of Theorem 3.10. With the help of these multipliers, the actual $\mathcal{S}$-procedure-approximation of the guard and invariant in question can be deduced, as a set of the form

$$S = \left\{ x \;\middle|\; x^T \left(\sum_j \mu^m Q_j^m \right) x \geq 0 \right\}$$

(and analogously for ν_j^m, η_j^m, and ϑ_j^e). The proof of GAS is then valid also if we widen the invariant to this set S.

After identifying a LLF for a mode, one can – as a second step – also identify $\mathcal{S}$-procedure terms for which the Lyapunov conditions hold. This is again an LMI, but this time the dynamics and Lyapunov function are considered constant, and the additive $\mathcal{S}$-procedure term is the unknown. Most notably, if the $\mathcal{S}$-procedure term can also be a positive semidefinite matrix, then the Lyapunov conditions hold *globally* for this Lyapunov function, meaning that the invariant can even cover the entire state space.

Modifying Differential Inclusions. In a similar manner, it is also possible to generalize the differential inclusions of the individual modes. For a given quadratic LLF, we can compute an under-approximation of the set of all affine dynamics for which this function exhibits the Lyapunov properties. This can again be achieved by an LMI problem, by simply exchanging the roles of the P and the A matrices. The known LLF is assigned to

P, which turns into a constant expression, and the matrix A is turned into an unknown, whose matrix entries can again be optimized in various directions with the normal-boundary intersection approach. In this manner, a more general differential inclusion for the very same LLF can be computed. Attaching this differential inclusion to the mode will preserve GAS.

There are also ways of generalizing a system by merging modes. Assume that, within an SCC, there exists a sub-graph for which a common Lyapunov function has been identified and within which no discrete updates occur. Then, a transformation which adds new trajectories to the system can be applied, still maintaining stability. Since the set of dynamics for which a function has the Lyapunov property always forms a convex set it is possible to merge all the nodes in this sub-graph into one node (inheriting all incoming and outgoing edges). Suppose the modes in the set $\tilde{\mathcal{M}} = \{m_1, \dots, m_n\}$ have a common Lyapunov function. Then the new mode m will have the invariant

$$Inv(m) = \bigcup_i Inv(m_i)$$

and the flow

$$Flow(m)(x) = convex \left(\bigcup_{\tilde{m} \in \tilde{\mathcal{M}} \text{ with } x \in Inv(\tilde{m})} Flow(\tilde{m})(x) \right).$$

Informally speaking, the flow of the new "aggregate mode" will allow any behavior covered by the flow of the constituent modes m_i, as long as the corresponding invariant is true. This means that we have a) collapsed the modes m_i into one node, effectively allowing switching between them at any time, and b), exploited the convex properties of Lyapunov functions by allowing also any behaviors lying in the convex hull of the behaviors of all m_i that could theoretically be active. The result is a system that, while allowing considerably more dynamics, still is guaranteed to be GAS. This is due to the fact that the common Lyapunov function for the modes m_i will also fulfill the Lyapunov properties for mode m, which is easily verified.

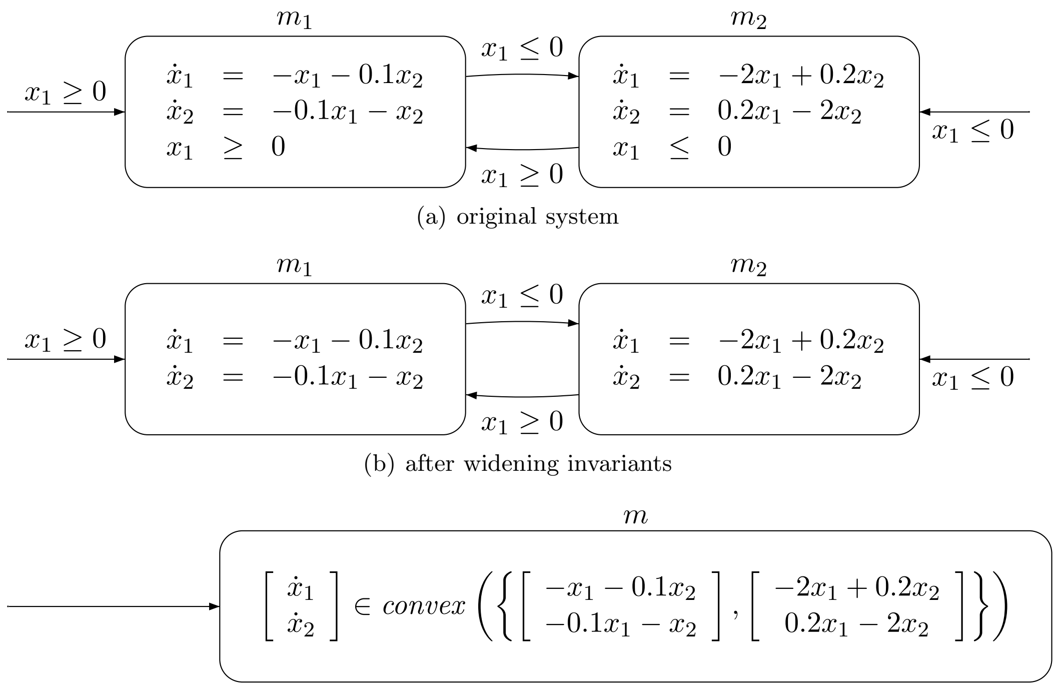
Example 6.1. The hybrid system given in Figure 6.1(a) is GAS, which can be shown by a single Lyapunov function. For example, $V(x_1, x_2) = x_1^2 + x_2^2$ acts as a LLF for both modes m_1 and m_2.

Because this is the case, the guards on the transitions are inconsequential. For *any* arbitrary switching strategy between the modes m_1 and m_2, the resulting system will be GAS. This is the case because the non-increasingness condition on each of the transitions will be true for any guard G, as

$$x \in G \implies V(x_1, x_2) \leq V(x_1, x_2)$$

trivially holds.

Observe that V is a LLF for both modes not only when the corresponding invariant holds, but for *any* state (x_1, x_2). This means, as discussed above, that both invariant sets can be widened to $\mathbb{R}^2$, still maintaining GAS with the same GLF (see Figure 6.1(b)). Also, V is a LLF for all modes whose dynamics are a convex combination of the dynamics of modes m_1 and m_2. This implies that we can subsume both modes into a new mode m with invariant set $\mathbb{R}^2$ and a differential inclusion that allows a non-deterministic choice of the time derivative for both variables (see Figure 6.1(c)). At each point in time, the time derivative (if existent) may lie in the convex hull of the corresponding values for the original modes. Despite allowing for more trajectories than the two individual modes m_1 and m_2, the resulting single mode system is GAS.


m_1
m_2
$x_1 \geq 0$
$x_1 \leq 0$
$\dot{x}_1 = -x_1 - 0.1x_2$
$\dot{x}_2 = -0.1x_1 - x_2$
$x_1 \geq 0$
$\dot{x}_1 = -2x_1 + 0.2x_2$
$\dot{x}_2 = 0.2x_1 - 2x_2$
$x_1 \leq 0$
m
$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} \in convex\left(\left\{\begin{bmatrix} -x_1 - 0.1x_2 \\ -0.1x_1 - x_2 \end{bmatrix}, \begin{bmatrix} -2x_1 + 0.2x_2 \\ 0.2x_1 - 2x_2 \end{bmatrix}\right\}\right)$

(a) original system

(b) after widening invariants

(c) after joining modes

Figure 6.1: Joining Modes

Stochastic Systems. In general, the same transformation rules as for standard hybrid automata can also be applied to probabilistic hybrid automata. This includes the addition of stable SCCs, changes to the invariants and guards, and widening of the differential inclusions, as well as mode splitting. When adding a new hyperedge, the compatibility of this edge with the LLFs of the adjacent modes must be checked, in the same manner as outlined above. In addition, also unstable SCCs can be added to an automaton, if the transition probability can be bounded from above, turning a AS-GA system into one that is only GA-(p) for some $p < 1$.

Another possibility is the replacement of an ordinary differential equation $\dot{x} = f(x)$ on a mode m by a stochastic differential equation $dx = f(x)dt + g(x)dB$ [Korenevskii, 1987]. For simplicity, suppose that $Inv(m) = \mathbb{R}^n$. In this case, the standard Lyapunov constraint

(2) there exists a class K^∞ function f_3 such that for all x: $\dot{V}(x) \leq -f_3||x||$, where $\dot{V}(x) := \left\langle \frac{dV}{dx}(x) \middle| f(x) \right\rangle$

must be replaced by

(2') there exists a class K^∞ function f_3 such that for all x: $\dot{V}(x) \leq -f_3||x||$, where $\dot{V}(x) := \left\langle \frac{dV}{dx}(x) \middle| f(x) \right\rangle + \frac{1}{2}g(x)^T \frac{d^2V}{dx^2}(x)g(x)$.

Now suppose that $g(x) = \delta x$ for some $\delta > 0$. This means that the influence of the Wiener process on the derivative of x depends linearly on x. In this case,

$$\dot{V}(x) = \left\langle \frac{dV}{dx}(x) \middle| f(x) \right\rangle + \frac{1}{2}\delta^2 \frac{d^2V}{dx^2}(x).$$

Assume that $f(x) = Ax$ is linear and that $V(x) = x^T Px$ is quadratic, and that P is a Lyapunov function for the ordinary differential equation, solving

$$A^T P + PA + \alpha I \preceq 0.$$

Then

$$\frac{d^2V}{dx^2}(x) = 2P,$$

and therefore, for the stochastic differential equation, we obtain an LMI constraint of the form

$$A^T P + PA + \delta^2 P + \alpha I \preceq 0.$$

Since we also have

$$P - \beta I \preceq 0,$$

this gives us

$$A^T P + PA + \delta^2 P + \frac{\alpha}{\beta} P \preceq 0,$$

a constraint which may or may not be satisfied due to the additional term. However, if $\delta^2 < \frac{\alpha}{\beta}$, then

$$A^T P + PA + \delta^2 P + \left(\frac{\alpha}{\beta} - \delta^2\right) P \preceq 0$$

is solved by the very same matrix P as for the ordinary differential equation, and the stochastic differential equation is GS-P and AS-GA with the same LLF. Therefore, such a linearly weighted Wiener process can be added as a disturbance to any differential equation as long as $\delta^2 < \frac{\alpha}{\beta}$, still maintaining GS-P and AS-GA. In this case, the same LLF remains valid for this node in the constraint graph, and therefore also all edge constraints of incoming and outgoing transitions remain satisfied. Therefore, this addition of a Wiener process disturbance can be conducted on a per-mode basis (with possibly different magnitudes given by different values of δ), and the newly obtained system will be GS-P and AS-GA without necessitating the computation of a new GLF.

Next, we extend the composition approach to a setting that a) separates plant and controller and b) allows for hierarchical composition of sub-automata, which are represented as *components* of the controller.

6.2 Component Based Design of Stable Hybrid Automata

The transformation rules presented in Section 6.1 can be used to incrementally construct stable hybrid automata. However, this construction procedure is not of hierarchical nature. Instead, we assumed that the entire hybrid automaton is visible at any time, with all modes and their associated invariants, dynamics, and transitions. In order to arrive at a truly compositional design procedure, it is however highly desirable to organize a hybrid automaton in a hierarchical manner. For a given plant model (which can be a differential inclusion, therefore actually representing an entire class of physical plants), this enables the use of re-usable *components* in order to satisfy safety and stability requirements. From this standpoint, a hybrid controller (i.e., a controller modeled as a hybrid system) consists of various such components, which can either be hybrid automata or again *transition compositions* of components, similar to hierarchical structures in model based design tools.

Ideally, we want to have an abstract outside view of a component's characteristics, without a full description of the component semantics. Instead, only the component's interface to the outside should be visible, together with enough information to conduct a composition such that the desired system property is verifiable. Of particular interest for hybrid systems are safety and stability. The focus of this section is a composition framework which is sufficient to prove GAS of such a system consisting of interconnected components.

This section summarizes the author's contributions to [Damm *et al.*, 2010], which in addition also deals with

- safety properties,
- an alarm system with alarms that are only picked up with a time delay,
- delayed mode switches,
- a detailed communication protocol between components, and
- a formal definition of the component semantics.

In the following we will only concentrate on the stability aspect, by utilizing the decompositional results presented in this thesis and exploiting them for composition.

The method introduced here is based on interconnected components which set off alarm signals whenever they detect that a situation they are not designed to deal with is imminent. When a sub-component C_i of another component C raises an alarm, there are two possible outcomes. The first case is that the alarm is picked up by another sub-component C_j on the same hierarchy level, which then takes over the task of controlling the plant after some time needed to execute the switch. This means that an alarm emitted by a sub-component C_i can be resolved within C itself, without necessitating calls for external help. The other case occurs when such an alarm cannot be resolved internally. The alarm from a sub-component C_i is then relayed to the outside of C. Another component on the higher hierarchical level will then take over control from C, resulting in C being deactivated as a whole.

Since the stability proof will again be based on Lyapunov functions, the outside view of a component is characterized by sets of permissible Lyapunov functions. The results from Chapter 4 are extended to cope with entire components in the place of single modes, such that stable hybrid systems can be constructed hierarchically. As we show, even in this case, the resulting constraint system can be rendered as an LMI problem, and the decomposition results apply.

6.2.1 Plants and Components

We assume that there is a single given plant, modeling the continuous-time process to be controlled. This plant model is represented by a single hybrid automaton P. Since P can contain general differential inclusions, we can use this formalism to model an entire *class of plants* by this single P, as long as all entities within the class adhere to the constraints on the dynamics formulated by P. Furthermore, we assume that some variables of P act as *input variables*. Usually, these variables will appear on some right hand sides of differential inclusions or on some guards, but remain unrestricted by the differential inclusions. In other words, an input variable x_i will be associated with the most general differential inclusion $x_i \in \mathbb{R}$, and therefore not be influenced by the plant in any way. On the other hand, some variables of P will also work as *output variables.* Output variables are influenced by P, and can be read by the controller and used to formulate an adequate control strategy.

For the following discussion, we assume that the plant P only consists of a single discrete mode m_p. This is not a principal limitation of the approach, but purely to avoid some technicalities with parallel compositions of hybrid automata. We also assume that each variable of P is either an input or output variable. The result of this assumption is that all variables of the plant are known to the controller, since they act as controller inputs or outputs. Of course, the controller is not required to make use of all these variables, so this is not a true limitation. Furthermore, we assume that the plant does not restrict the initial state of its variables.

Definition 6.1 (Plant)**.** A *plant* P is a hybrid automaton

$$P = (\{m_P\}, \mathcal{S}_P, \mathcal{V}_P, \mathcal{T}_P, \mathit{Flow}_P, \mathit{Inv}_P, \mathit{Init}_P)$$

with a sub-set of *input variables* $\mathcal{V}_P^I \subseteq \mathcal{V}_P$ and a set of *output variables* $\mathcal{V}_P^O \subseteq \mathcal{V}_P$, such that

- $\mathcal{V}_P^I \cup \mathcal{V}_P^O = \mathcal{V}_P$ and $\mathcal{V}_P^I \cap \mathcal{V}_P^O = \emptyset$,
- $\mathit{Init}_P = \{m_p\} \times \mathcal{S}_P$,
- Flow_P does not restrict variables in $\mathcal{V}_P^I$, and
- updates belonging to any transitions in $\mathcal{T}_P$ do not change variables in $\mathcal{V}_P^I$.

A *basic component* represents a possible controller for the plant P and is again given as a hybrid automaton C. Among the variables of C must be all the output and input variables (and therefore all variables) of P. The output variables of P act as inputs to C and must therefore remain unrestricted within the differential inclusions of C. On the other hand, for all the input variables of P, C defines the control strategy.

In order to be able to define the transition composition of basic components, we need to associate with each component C sets of *outgoing ports* $O(C)$. Each of these ports can be interpreted as a channel that can be used to broadcast an alarm signal. This happens whenever the system trajectory is in danger of leaving the sub-set of the state space for which a component can guarantee the desired behavior, in our case GAS. For instance, a braking component should activate an alarm before the velocity of a vehicle is about to sink below the desired set point, as braking is then no longer adequate. This outport is connected to other components, one of which will then take over from the brake component. In this case, this will usually be a component which is capable of gradually reducing the braking effort in order to steer the system toward the set point without a great overshoot.

Definition 6.2 (Basic Component)**.** A *basic component* C for a plant P is a hybrid automaton $(\mathcal{M}_C, \mathcal{S}_C, \mathcal{V}_C, \mathcal{T}_C, \mathit{Flow}_C, \mathit{Inv}_C, \mathit{Init}_C)$ where

- $\mathcal{V}_P \subseteq \mathcal{V}_C$,
- $\mathcal{S}_C = \mathbb{R}^{|\mathcal{V}_C|}$,
- Init_C does not restrict variables in $\mathcal{V}_P^O \cup \mathcal{V}_P^I$,
- Flow_C does not restrict variables in $\mathcal{V}_P^O$, and
- updates belonging to any transitions in $\mathcal{T}_C$ do not change variables in $\mathcal{V}_P^O$.

Let $O(C)$ be the set of outports of C and associate with each $o \in O(C)$ a set $\mathit{Alarm}(o) \subseteq \mathcal{S}_C$.

The set $\mathit{Alarm}(o)$ specifies the condition under which an alarm is sent on port o. As soon as the current system state lies inside $\mathit{Alarm}(o)$, an alarm signal is sent, resulting in a transfer of control to another component which is connected to this port, if such a component exists. An alarm on an unconnected outport cannot be taken, and if the invariants of the currently active basic components are violated and there is no connected outport with triggered alarm which can be used to transfer control, then the system run will terminate. This can be interpreted as a failure of the component, which either a) did not set off an alarm on time or b) whose alarm was not picked up. The set $\mathit{Alarm}(o)$ should be picked such that it must always be passed before a trajectory terminates inside component C, in order to avert a). The second problem b) is prevented as soon as all outports are connected to other components. Convergence can of course only be guaranteed for infinite runs, while stability, per definition, also applies to finite trajectories (see Definition 3.15).

The set Init_C is used to re-initialize only the local variables of the component C every time it takes over control of the plant. Therefore, Init_C cannot talk about any variables which are used to communicate with the plant. Resets on these variables can be modeled as part of a transition composition of several sub-components.

The closed loop $C||P$ consisting of a controller C and a plant P is essentially their parallel composition, which in this case can be defined as follows.

Definition 6.3 (Closed Loop)**.** The *closed loop* $C||P$ of plant P and basic component C is the hybrid automaton $(\mathcal{M}_H, \mathcal{S}_H, \mathcal{V}_H, \mathcal{T}_H, \mathit{Flow}_H, \mathit{Inv}_H, \mathit{Init}_H)$ with

- $\mathcal{M}_H = \mathcal{M}_C$,
- $\mathcal{S}_H = \mathcal{S}_C$,
- $\mathcal{V}_H = \mathcal{V}_C$,
- $\mathcal{T}_H = \mathcal{T}_C$,
- $\mathit{Flow}_H(m)(x) = \{y \in \mathcal{S}_C \mid y \in \mathit{Flow}_C(m)(x) \wedge y_{|P} \in \mathit{Flow}_P(m_P)(x)\}$, where $y_{|P}$ is the projection of y onto $\mathcal{S}_P$,
- $\mathit{Inv}_H(m) = \{y \in \mathcal{S}_C | y \in \mathit{Inv}_C(m) \wedge y_{|P} \in \mathit{Inv}_P(m_P)\}$, and
- $\mathit{Init}_H = \mathit{Init}_C$.

If we allow hybrid plants P with more than one mode, then the closed loop is the parallel composition of the two hybrid automata. The following method also works for this case. Plants were intentionally kept single-mode for the discussion, in order to keep the formalism simple.

By taking the parallel composition $H = P||C$, we thereby define the closed loop behavior obtained by letting component C interact with plant P. The question whether this automaton H is GAS can be answered with the help of Lyapunov functions, and these Lyapunov functions can be computed decompostionally, as described in Chapter 4. Apart from basic components, we also want to have a *transition composition of components*. Such a transition composition again results in a possible controller for P and is defined through a set of transitions connecting these ports to target components.

Definition 6.4 (Transition Composition)**.** A *non-basic component* C with respect to a plant P with a set of *outports* $O(C)$ is a tuple $(\{C_1, \ldots, C_n\}, I, K)$, where

- the C_i are (basic or non-basic) *sub-components*,
- $I \in \{C_1, \ldots, C_n\}$ is an *initial component*, and
- the *port connection* K consists of tuples of the form

  $$k = (s, \{(t_1, G_1, U_1), \ldots, (t_{r_k}, G_{r_k}, U_{r_k}\}),$$

  where

  1. $s \in \bigcup_i O(C_i)$,

2. for each outport $o \in O(C_i)$ for any component C_i, there is exactly one $(o, \cdot) \in K$,
3. for each outport $o \in O(C)$, there is exactly one $(s, T) \in K$, with exactly one tuple $(o, \cdot, \cdot) \in T$,
4. for all j: $t_j \in O(C) \cup \{C_1, \ldots, C_n\}$,
5. $\bigcup_j G_j = \mathcal{S}_C$,
6. for all j: $U_j : \mathcal{S}_P \to \mathcal{S}_P$, and
7. if $t_j \in O(C)$, then U_j is the identity function.

Condition 2 stipulates that each outport inside a transition composition must be connected. Conditions 3 and 8 require that each outport of the newly composed component C must be fed by exactly one port connection with the identity function as the update. Furthermore, for each transition, the guards must cover the entire state space, so that always a transition to some new component can be taken.

A component C resulting from a transition composition has the following semantics: Whenever component C is active, at any point in time, exactly one sub-component C_i is deemed active. If this C_i is a basic component, then it is simply represented by a hybrid automaton, which is used to control the plant P. If component C_i is again the result of a transition composition, then again one of its sub-components is active and so on, until we arrive at a basic component. Therefore, exactly one hybrid automaton corresponding to such a basic component is used to drive the plant at any time. This basic component has a number of outports o, each of which is associated with an alarm set $Alarm(o)$. As soon as the system state enters this alarm set, a transition to another component takes place. To decide which component must take over, the guards G_j^k of the unique transition

$$k = (s, \{(t_1, G_1, U_1), \ldots, (t_{r_k}, G_{r_k}, U_{r_k})\}),$$

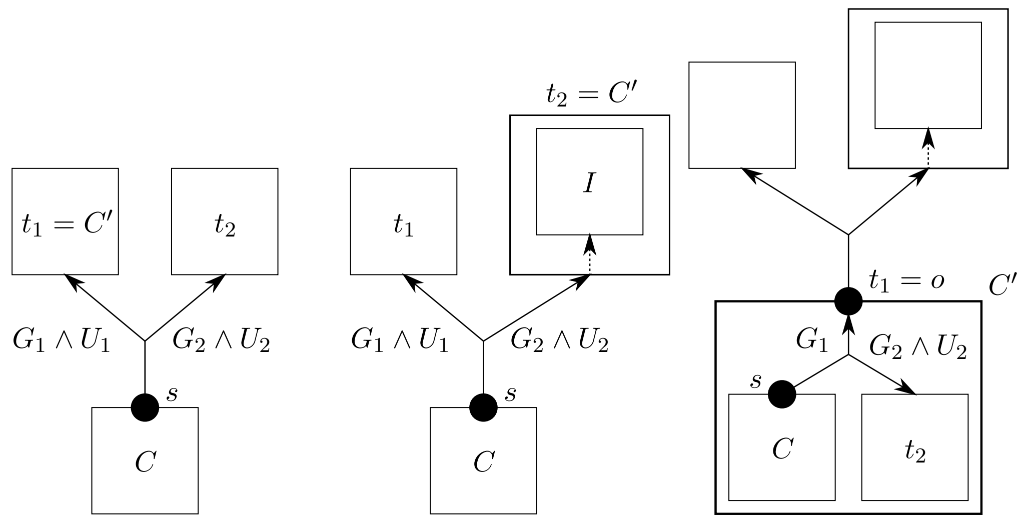
are evaluated. Whenever the current state lies in G_j, a transition to t_j may take place.

Assume that a basic component C is active and such an alarm occurs. Then, there are three possible cases:

- $C' = t_j$ is a basic component on the same hierarchy level as basic component C (see Figure 6.2(a)): In this case, control is passed to component C', all variables which are in $\mathcal{V}_P$ (and therefore visible globally) are updated according to U_j, and all variables which are not in $\mathcal{V}_P$ (and therefore local to C') are re-initialized according to $Init_{C'}$.

- $C' = t_j$ is a non-basic component on the same hierarchy level as basic component C (see Figure 6.2(b)): component C' is activated and basic component C deactivated, the update U_j is applied, and component C' then activates its initial sub-component I. This is recursively repeated until control has been passed to a basic component as above.

- $o = t_j$ is an outport of the component C' on the next highest hierarchy level (see Figure 6.2(c)): In this case, component C' relays the alarm signal to the outside via its outport o. If component C' is already the top-level component then this indicates a failure of C'. Otherwise, outport o must be connected to another component via a transition in the next highest hierarchy level. Again, the guards of this transition are evaluated to identify a component to which control is passed. For the component identified in this manner, the procedure above is applied to activate a basic component and re-initialize variables. A single transition between to basic components can pass through an arbitrary number of hierarchy levels in this manner, both up and down in the hierarchy.



(a) to a basic component (b) to a non-basic component (c) to an outport

Figure 6.2: Transfer of Control between Components

We need to re-initialize the internal component variables (i.e., all variables which do not appear in the plant as inputs or outputs) whenever we pass over control to a new basic component, since each basic component might define *different* internal variables. Furthermore, at composition level, we would like to hide these internal variables from sight, with only the interface to the plant being visible, on which update functions can be applied. A complete formal definition of trajectories of such composed systems is given in [Damm *et al.*, 2010].

This component-based model can also be flattened into a standard hybrid automaton by taking the hybrid automata of all basic components and connecting them appropriately according to the port connections. Such a flattened automaton would be defined over variable and mode sets which are the union of the variable and mode sets of all constituent basic components. For any mode, variables which do not occur in its basic component can simply be left unrestricted, since the stability property we are interested in only relates to the variables of the plant. By definition, the variables of the plant are common to all basic components.

6.2.2 Stability Proofs via Lyapunov Function Projections

Adhering to the principle of hierarchical controller design, we want to conclude GAS *without* explicitly looking into the individual components that have been composed. Instead, we associate with each component (be it basic or not) only the necessary information to conduct a proof of GAS for such compositions, and hide all other information inside the component. This information can be seen as an external *component interface*. As long as the component itself corresponds to an interface (taking the role of an *implementation* of the interface), the internal logic and the control strategies implemented within the component can be chosen freely, since only the interface information is required to prove GAS. We assume that each individual component within a transition composition has already been shown to be GAS and that it provides this interface. The interface needs to consist of the following:

- for each outport $o \in O(C)$, an *exit set* $\mathit{Exit}(o) \subseteq \mathcal{S}_P$, describing all plant states that are possible when a transition takes place through the outport o.
- a family of *Lyapunov function projections* $V_C^i : \mathcal{S}_P \to \mathbb{R}$ such that there exists a GLF V^i for component C which is pointwise lower for all initial states of C upon activation, and
- in the same manner, a family of Lyapunov function projections $V_o^i : \mathcal{S}_P \to \mathbb{R}$ for each outport $o \in O(C)$ such that the same GLF V^i as above is pointwise higher for all possible states upon exit of C through o.

The Lyapunov function projections are needed because a GLF for a component C may (and in many cases must) also talk about internal variables within components C which are not plant variables. Since these variables are assumed hidden to other components, they cannot appear in the interface. For the same reason, we cannot in general use the $\mathit{Alarm}(o)$ sets of a basic component as the $\mathit{Exit}(o)$ sets. Alarms might be triggered by internal variables, which may not appear in the exit set.

First, assume that C is a basic component. For any $x \in \mathcal{S}_H$, let $x_P \in \mathcal{S}_P$ the subvector containing only the values of variables in $\mathcal{V}_P$. In this case, $\mathit{Exit}(o)$ can simply be any set satisfying

$$x \in \mathit{Alarm}(o) \implies x_P \in \mathit{Exit}(o),$$

for instance by dropping all restrictions on internal variables from $\mathit{Alarm}(o)$. If states within $\mathit{Alarm}(o)$ are known to be unreachable (for instance, shown through reachability analysis within C), $\mathit{Exit}(o)$ can also be tightened, not including these unreachable states. The Lyapunov function projections are formally defined as follows.

Definition 6.5 (Lyapunov Function Projections of Basic Components)**.** Let C be a basic component, let

$$P = (\mathcal{M}_P, \mathcal{S}_P, \mathcal{V}_P, \mathcal{T}_P, \mathit{Flow}_P, \mathit{Inv}_P, \mathit{Init}_P)$$

be the plant and let the closed loop $C||P$ be given by the hybrid automaton

$$H = (\mathcal{M}_H, \mathcal{S}_H, \mathcal{V}_H, \mathcal{T}_H, \mathit{Flow}_H, \mathit{Inv}_H, \mathit{Init}_H).$$

For some $n > 0$, and each $1 \leq i \leq n$, let V^i be a GLF for H, and define the *Lyapunov function projections* for basic component C as:

- for each $o \in O(C)$, a function $V^i_o : \mathcal{S}_P \to \mathbb{R}$, and
- a function $V^i_C : \mathcal{S}_P \to \mathbb{R}$,

such that:

1) for each $o \in O(C)$, all $m \in \mathcal{M}_H$ and all $x \in \mathit{Alarm}(C) : V^i_O(x_P) \leq V^i(m, x)$, and
2) for all $(m, x) \in \mathit{Init}_H : V^i_C(x_P) \geq V^i(m, x)$.

These projections are simply under- and over-approximations of V^i, dropping all internal variables. Since the existence of such projections implies the existence of a GLF for H, this means that $C||P$ is GAS. In other words, as long as no outgoing alarm of C is triggered, we have guaranteed convergence to the equilibrium for the closed loop. Even though the internal variables have to be dropped for the projections, they can of course occur in the "internal" GLFs V^i, capturing their influence on the dynamics within the component.

For a component obtained through transition composition, the GAS proof is slightly more complicated. Again, as during the cyclic decomposition in Section 4.4, we utilize conic hulls of Lyapunov functions. However, the GLFs for a component are no longer given explicitly, since we do not know anything about the modes inside a basic component or the inner structure of the sub-components. Instead, we exploit that each family of Lyapunov function projections for a given index i, consisting of V^i_C and the V^i_o for each outport, implies the existence of a corresponding GLF for the sub-component. The same applies for each conic combination of the functions V^i_C and V^i_o. Therefore, if we find conic multipliers for each sub-component such that the Lyapunov function projections are non-increasing along every transition, then this implies the existence of a GLF for the flattened hybrid automaton corresponding to C. This implies GAS.

The set $\mathit{Exit}(o)$ for an outport of a component obtained through transition composition is simply $\mathit{Exit}(s) \cap G$ where (s, T) with $(o, G, U) \in T$ is the unique transition connecting to outport o. The update U has already been required to be the identity function in this case (see Definition 6.4). The Lyapunov function projections for the transition composition can be formalized as follows.

Definition 6.6 (Lyapunov Function Projections for Transition Compositions)**.** Let $C = (\{C_1, \ldots, C_n\}, I, K)$ be a transition composition of (basic or non-basic) components C_i. For an outport o, let $C(o)$ be the component it belongs to (i.e., the unique component C such that $o \in O(C)$). Assume that for each C_i there also exist m_{C_i} Lyapunov function projections $V^q_{C_i}$, $1 \le q \le m_{C_i}$ and for each $o \in O(C_i)$ there exist m_{C_i} Lyapunov function projections V^q_o. For some $n_C > 0$, and each $1 \le p \le n_C$, define the Lyapunov function projections for component C as:

- for each $o \in O(C)$, a function $V^p_o : \mathcal{S}_P \to \mathbb{R}$, and
- a function $V^p_C : \mathcal{S}_P \to \mathbb{R}$,

such that for each C_i and each $1 \le q \le m_{C_i}$ there exists a positive scalar $\lambda^p_{C_i,q}$ with:

1) for each $k = (s, \{(t_1, G_1, U_1), \ldots, (t_{r_k}, G_{r_k}, U_{r_k})\}) \in K$,
   for each $1 \le j \le r_k$ with $t_j \in \{C_1, \ldots, C_n\}$, and
   for all $x_P \in G_j \cap \mathit{Exit}(s)$:

$$\sum_q \lambda^p_{C(s),q} V^q_s(x_P) \ge \sum_q \lambda^p_{t_j,q} V^q_{t_j}(U_j(x_P)),$$

2) for each $k = (s, \{(t_1, G_1, U_1), \ldots, (t_{r_k}, G_{r_k}, U_{r_k})\}) \in K$,
   for each $1 \le j \le r_k$ with $t_j \in O(C)$, and
   for all $x_P \in G_j \cap \mathit{Exit}(s)$:

$$\sum_q \lambda^p_{C(s),q} V^q_s(x_P) \ge V^p_{t_j}(x_P),$$

3) for all $x_P \in \mathcal{S}_P$:

$$V^p_C(x_P) \ge \sum_q \lambda^p_{I,q} V^q_I(x_P).$$

The existence of multipliers $\lambda^p_{C_i,q}$ implies the existence of a GLF for the flattened hybrid automaton corresponding to C. Therefore, this computation of Lyapunov function projections again yields a GAS component which can be successively re-combined in another transition composition. Furthermore, Conditions 1) to 3) of Definition 6.6 are again in a form that allows the use of LMI methods for the computation of the $\lambda^p_{C_i,q}$. In order to produce multiple families of Lyapunov function projections, we can only repeatedly solve it with different optimization directions, using the normal-boundary intersection approach.

In order to be able to derive convergence rates for such transition compositions, some additional information needs to be stored in addition to the Lyapunov function projections. In particular, we need to associate a convergence rate with the Lyapunov function projections V^p_o and V^p_C. For basic components, this is simply the convergence rate α' for the GLF V^p, that is α/β, as discussed in Section 3.5.4. However, we need to keep track of the α-values and the β-values separately, as $\alpha_{C_i,p}$ and $\beta_{C_i,p}$, respectively, Now, assume that C is the result of a transition composition of the components C_i and that the conditions in Definition 6.6 hold. Furthermore assume that we have such values $\alpha_{C_i,q}$ and $\beta_{C_i,q}$ for each sub-component C_i. In particular, we know the multipliers $\lambda^p_{C_i,q}$, which, for each i, result in the existence of a Lyapunov function for C_i. This guarantees a convergence rate of

$$\frac{\sum_q \lambda^p_{C_i,q}\alpha^q_{C_i}}{\sum_q \lambda^p_{C_i,q}\beta^q_{C_i}}$$

for C_i. We are now interested in a lower bound on the convergence rate for C. A simple way of obtaining this information is by taking the slowest convergence rate among the sub-components, that is

$$\min_i \left(\frac{\sum_q \lambda^p_{C_i,q}\alpha^q_{C_i}}{\sum_q \lambda^p_{C_i,q}\beta^q_{C_i}} \right)$$

for any p. For component C, furthermore define

$$\alpha^p_C := \min_i \left(\sum_q \lambda^p_{C_i,q}\alpha^q_{C_i} \right)$$

and

$$\beta^p_C := \max_i \left(\sum_q \lambda^p_{C_i,q}\beta^q_{C_i} \right).$$

These values can then be used to estimate the convergence rate, should C again be used in a composition. A detailed discussion on the computation of convergence rates and convergence times to target sets in the compositional setting, including proofs, can be found in [Damm *et al.*, 2010].

As far as decomposition is concerned, a port connection can again be treated in the same manner as a hybrid automaton. A port connection can simply be viewed as a graph with components as its nodes and component transitions as its edges. This means that the decomposition results of Chapter 4 can be transferred also to this level for the computation of the per-component GLFs. If a port connection results in several SCCs, they can be treated independently for the constraint system in Definition 6.6, as long as the bridges are sub-linear and an actual GLF is computed according to the procedure described in Remark 4.3 of Section 4.3. It is important that an actual GLF is needed for each component and not just one GLF per SCC, as transitions outside the component could close a cycle that collapses SCCs inside the component. The cyclic decomposition results can also directly be applied inside a component. However, this only makes sense for individual components which are the result of very complex transition compositions, since the composition process in itself already works as a mechanism to keep the individual constraint systems relatively simple.

So far, we assumed that transitions between components are instantaneous. However, it is also possible to model different types of time delays. For instance, if the determination of a successor component takes some time (e.g., because a potential successor must first confirm its willingness to take over through some protocol), then a component will remain active for some time after triggering an alarm. Therefore, its bottom-level basic components should guarantee their invariants for this amount of time (guaranteeing a decrease of the GLF value while waiting for a response), and the exit sets should be enlarged, containing all states which are reachable from the original exit sets within the delay. If the transitions themselves take time, then this can also be achieved by adding a reach set computation to the constraint generated by the transition. However, in this case, the source component needs to communicate some over-approximation of its behavior to the outside, which can then be used to compute the time-bounded reach set from the component's exit set on the next higher level. A formal treatment of delayed switches can also be found in [Damm *et al.*, 2010].

6.3 Summary

In this section, we exploited the decomposition results of Chapter 4 to provide a number of composition rules which can be used for the incremental construction of a stable hybrid system. First, we focused on hybrid automata themselves, presenting transformation which can be applied while still maintaining GAS of the automaton. We then described a more realistic component-based approach which

- separates the controller and the plant model,
- includes information hiding, only supplying the necessary information for composability,
- relies on conic hulls of Lyapunov function projections, building on the decomposition results of Chapter 4.4, and
- still supports verification via LMI solvers.

These results can potentially be exploited for a hybrid controller design tool consisting of a library of re-usable components with pre-verified interfaces. For a composition to be GAS, only properties based on the interfaces need to be verified, avoiding complexity blowup. Of course, the other side of the coin is the conservativeness of the approach, as the conic under-approximations of Lyapunov function sets are usually conservative. Nevertheless, the compositional design method is promising for bridging the gap between system design methodologies and system verification, providing a structured design process yielding system models for which stability is verifiable by design.

7 Conclusion and Future Work

This chapter concludes the thesis with a review of the main results and an outlook on future work.

7.1 Conclusion

This thesis addressed the problem of decomposing *global asymptotic stability (GAS)* proofs for hybrid systems modeled as hybrid automata. GAS subsumes 1) attractivity, guaranteeing convergence to some equilibrium and 2) stability, guaranteeing a boundedness condition on the distance to the equilibrium for all trajectories. In contrast to safety properties, stability does not allow for straightforward decomposition into local arguments guaranteeing an automaton-global stability property. In order to maintain safety, it is sufficient that all discrete modes or all sub-automata forming a partitioning guarantee safety by themselves, and that all transitions between these modes an sub-automata always lead from safe states to safe states. However, stability is a property which can only guaranteed by the interplay if the discrete modes, which must work together in order to ensure convergence to an equilibrium state. Whether the interaction between the discrete modes is suitable for maintaining stability must therefore be checked when conducting decompositional verification.

As a vehicle for such proof we used a standard concept from control theory: *Lyapunov functions*, that is, functions measuring an abstract "energy level" of the system. This energy level must monotonically decrease towards a single global minimum at the equilibrium. As has been known in the control theory domain for some time, such functions can be automatically computed in parametrized form via convex optimization. To be exact, the Lyapunov function constraints can be expressed as *linear matrix inequalities (LMIs)*, that is, linear inequalities with matrix values together with an inequality operator which checks for *positive semidefiniteness* of a matrix. Since positive semidefiniteness corresponds to global non-negativity of the quadratic function represented by the matrix, Lyapunov constraints for affine differential inclusions and quadratic Lyapunov functions can be expressed naturally in this format. Also, it has been known that more complex cases can be handled by a substitution technique called *sums-of-squares decomposition.* In the scope of hybrid systems, separate *local Lyapunov functions (LLFs)* can be computed for a system's discrete modes, together forming a *global Lyapunov function (GLF).* We provided an in-depth discussion of these computations techniques in Chapter 3.

When applying these methods from the literature, LLFs are not computed independently, but rather as parts of a large LMI system. In order to ensure a suitable interplay between the modes, constraints for all transitions are also included. These constraints ensure that there is no “energy increase” as a transition is taken. However, these constraint systems can grow large and intractable in practice for the numerical solvers. Moreover, using these methods, it is difficult to design hybrid controllers with many discrete modes, since the interplay between the differential inclusions for the modes and the transitions between them is difficult to interpret even for the experienced engineer. Therefore, compositional design of hybrid systems is very desirable.

These drawbacks served as the motivation for the contributions of this thesis. Our decompositional approach, as presented in Chapter 4, uses a discrete structure of the hybrid automaton as a basis of decomposition, interpreting it as a labeled graph. After giving a general motivation for decomposition and some basic graph definitions in Sections 4.1 and 4.2, we introduced two levels of decomposition: 1) decomposition into *strongly connected components (SCCs)* and 2) decomposition within SCCs.

On the first level, the decomposition into SCCs as described in Section 4.3, we exploited the fact that no cycles between SCCs of a graph exist. This means that the interplay between discrete modes in two different SCCs is unidirectional, since a return is impossible after leaving a component. This fact can be exploited for a decomposition where SCCs can be analyzed independently of one another. One Lyapunov function per component (without any constraints relating the functions) already implies convergence to the equilibrium. The second requirement for a GAS system, stability, is fulfilled if the bridges connecting the components fulfill a sub-linearity condition. In that case, a GLF for the entire system can also be computed easily. An interesting observation on this decomposition is that it can actually be used to prove attractivity for systems which are not stable, which would be impossible without decomposition, since no GLF for the entire automaton can exist for non-stable systems. The decomposition into SCCs also allows for the extension of the convergence proof to multiple equilibria, since each component may have an equilibrium of its own.

Since there is no guarantee that a given hybrid automaton consists of more than one SCC and since SCCs can be large, we employed a second level of decomposition based on *simple cycles*. This level of decomposition was presented in Section 4.4. Since every node within a SCC lies on at least one simple cycle, simple cycles are a good basis for further reasoning, representing bidirectional dependencies between nodes. In this case, the compatibility of the Lyapunov functions for sub-automata needs to be explicitly checked, as stability of two cycles does have no direct implication with respect to the system consisting of their union. For the system to be stable, the individually computed Lyapunov functions for the cycles need to be compatible in the sense that they can be used to form a global Lyapunov function for the entire SCC. As an auxiliary structure, we introduced *constraint graphs*, which translate the Lyapunov constraint system for a given hybrid automaton into graph notion, making the locality of the constraints explicitly visible. The decomposition takes place on these constraint graphs. The property we exploited for this compatibility check is the conic shape of Lyapunov function sets. For any two sub-automata intersecting in just one node, we gave a theorem which allows decomposition based on conic, polytopic under-approximations of the LLF set for the intersection node. Such an infinite set of Lyapunov functions can be represented by finitely many functions: namely those ones forming the corner points of the polytope.

The computation of such a conic, polytopic set can again be conducted through LMIs, by optimizing in various parameter directions. The core result is however independent of the actual Lyapunov function computation method that is employed.

This decomposition result forms the basis of a reduction procedure on the constraint graph, considering one cycle of the SCC after the other. If a cycle intersects with the rest of the graph in only one node, then we can conduct the computation of such a set, remove the cycle from the graph except the border node, and attach a constraint corresponding to the newly computed under-approximation set to this border node. This constraint only refers to Lyapunov functions for the border node, subsuming the relevant characteristics of all other unknowns from the reduced cycles. Therefore, the complexity of the computations for the individual cycles does not grow with the number of cycles already reduced or their size. If no cycle with only one border node exists, then a splitting procedure is triggered. Essentially, we split a mode of the hybrid automaton, with all new nodes inheriting the same continuous behavior. However, each new node will have only one incoming and one outgoing transition, so that repeated application of the reduction will always result in reducible cycles. We also gave an algorithm interweaving reduction and splitting in a manner such that termination is guaranteed for all graphs, even in the presence of "cycles of cycles."

During this second type of decomposition, a GLF for the SCC is not explicitly computed, but guaranteed to exist. In order to actually compute this function, another step is required. To this end, a *reduction graph* which takes note of the reductions and splittings is built and then traversed top-down to compute a Lyapunov function. This Lyapunov function is, however, not for the original hybrid automaton, but for the equivalent automaton obtained after applying all the splittings. In this automaton, some modes of the original system might appear multiple times with different LLFs. Looking again at the original system, this is equivalent to having *context dependent* Lyapunov functions for the modes that were split. Such a mode has not only one LLF (or conic set thereof), but several functions which are chosen based on the discrete behavior directly before and after the mode is entered. In other words, depending on the discrete mode sequence associated with a trajectory, a different Lyapunov function may be used, but a suitable function guaranteeing stability is always known to exist. This is in contrast to existing approaches for multiple Lyapunov functions per mode in the literature, which use purely state-based reasoning without a temporal component.

Section 4.5 then gave a detailed description of the necessary per-cycle computations and some special cases where these computations can be simplified.

To demonstrate the decomposition, we gave a large case study modeling a velocity controller with several levels of brakes, which required numerous reductions of cycles, as well as the numerous splittings of nodes, to arrive at a decompositional stability proof. This example was presented in Section 4.6.

For the cycle-based decomposition procedure we also provided an algorithm outline for guided refinement of the conic under-approximation predicates in Section 4.7. This algorithm outline was then instantiated into several algorithm variants. One algorithm guarantees that all "robust" Lyapunov functions which do not lie on the boundary of the solution set are eventually found, using an evenly spaced enumeration of new optimization directions for the LMI of a cycle. Two other variants choose new optimization directions heuristically, exploiting knowledge about the other cycles the under-approximation is required to intersect with. One approach simply takes the center points of the two under-approximations while the other uses a distance measure which can be encoded as an LMI. While these "greedy" approaches are not guaranteed to find all existing intersections, they will in most cases provide a suitably refined predicate faster than the full enumeration method. We also outlined how refinement can take place across more than two cycles, using backtracking on reduction graphs.

The decomposition results based on graph theory and Lyapunov function computation were also transferred to the stochastic domain in Chapter 5. For stochastic systems, stability in probability and almost sure attractivity can be shown with very similar means as for standard hybrid automata. The difference here lies in the fact that Lyapunov functions only need to be *expected to decrease*, with the evolution of their value over time forming a *supermartingale* process. For probabilistic hybrid automata, the decreasingness of the expected value yields constraints on the Lyapunov function parameters which are very similar to the non-stochastic case. These constraints can again be encoded as an LMI problem and solved via convex optimization. The decomposition results based on SCCs can be directly transferred to probabilistic hybrid automata. Additionally, it is now possible to have SCCs which are by themselves unstable, if the probability of entering them can be bounded from above. This enables quantitative stability analysis with convergence probabilities below 1. The cycle-based decomposition is also applicable to this class of systems. Individual cycles are now also allowed to be unstable, if their instability is "compensated" by a stable cycle which is entered with high enough probability. Since a probabilistic hybrid automaton can be abstracted into a *Markov decision process*, we can moreover exploit steady state information of the discrete model to conduct even further decomposition. As it turns out, modes with steady state probability of zero in the Markov decision process (i.e., modes that are entered infinitely often with probability zero) can be considered completely separately for the attractivity analysis, much like SCCs. Moreover, the proof methods can be extended to systems with *stochastic differential equations*, as the expected decreasingness of the Lyapunov function can be rendered as an LMI problem even in this case.

We also tackled the problem of designing stable hybrid automata. This problem is difficult, since global asymptotic stability is generally not preserved under transition composition of hybrid automata. To this end, we provided a number of transformation rules for stable hybrid automata in Section 6.1. By following these rules, it is possible to devise provably stable hybrid automata by successively adding new modes or transitions or by modifying existing one. After an individual modification, it is then not necessary to re-prove stability for the entire automaton, but only for some parts of it. Most of these rules are a direct consequence of the decomposition results and allow for local transformations which either do not impact the Lyapunov function set at all or enlarge it. The rules include node splitting, node merging, widening of differential inclusions, guards, and invariants, as well as the addition of new subgraphs or edges. Again, conic hulls of Lyapunov functions can be used to determine whether a newly added transition within an SCC is compatible with the system in the sense that stability is maintained.

Moreover, Wiener process disturbances can be added to modes locally under certain circumstances, still maintaining stability in probability and almost sure attractivity.

Taking composition one step further, in Section 6.2, we then introduced a component based design framework for hybrid controllers for a given plant model. This framework allows the encapsulation of hybrid automata in so-called components, which hide all information of the automaton that is not needed for the stability proof. Such stable components can then be connected via transition compositions, still maintaining stability. The plant model may be defined in terms of differential inclusions and therefore represent a whole class of physical plants. This approach paves the way for library-based controller design, with re-usable components, whose properties in the context of a class of plants have already been shown in beforehand and do not have to be re-proved when such a component is used in a composition. Basic components are hybrid automata, together with a number of outgoing ports with associated alarms. These ports can then be connected to other components (be they basic or not), allowing the incremental construction of large controllers. Each component is also associated with an exit predicate and both incoming and outgoing Lyapunov function projections. Since a component's internal structure as well as its internal continuous variables are deemed hidden from the outside world, the exit predicates and Lyapunov function projections can only refer to variables visible on the component's external interface. The Lyapunov function projections are structured such that they guarantee the existence of a Lyapunov function for the (hidden) interior of the component. Moreover, if we compose several controllers, non-increasingness of the Lyapunov function projections along every port connection (=transition) implies the existence of a GLF for the entire system, including the hidden parts. For these projections, we again employ convex cones of Lyapunov functions. We attach several possible projections to each component and each outgoing port which can be conically combined to identify a suitable function. If we manage to identify projections in the convex cones of all connected components such that non-increasingness always holds, then we can guarantee GAS of the new component. Essentially, we know that a GLF corresponding to the "flattened" hybrid automaton exists without explicitly computing it. This enables us to compute another set of Lyapunov function projections, should this component in turn be used in another composition. The entire analysis can again be done based on LMIs, with relatively small constraint systems per component, since verification is done component by component. This compositional approach can also be combined with safety proofs via barrier certificates and with delayed switching between components. It is also possible to successively derive the convergence rates associated with each component.

Altogether, this thesis advanced the state of the art by systematically exploiting the discrete structures of a hybrid automaton for stability verification. The results can be used to:

- alleviate problems of complexity and numerical instability during stability proofs, by splitting large constraint systems into smaller, more manageable systems,
- allow for re-design suggestions when a proof fails,

- help the engineer's understanding of a stable hybrid automaton by viewing stability proofs as constraint graphs,
- develop software tools that allow for verification during the design, supporting a structured design process, and
- devise a library-based controller design approach which directly ties in with stability verification.

In the following, we give some possible enhancements and future extensions of the decomposition approach and its associated algorithms.

7.2 Future Work

Possible future work based on the results of this thesis extends in various directions. Some points are discussed in the following.

Tool development. Based on the decomposition results, it is now possible to provide tools which can support a structured design process for stable hybrid automata. An automaton can be designed incrementally without having to be completely re-verified once new modes or transitions are added. If the newly added parts are verified at regular intervals during the design process, then this means that the user can now pinpoint which changed caused the loss of GAS, in case he/she makes a mistake. This information allows for guided re-design, as the software could also suggest valid alternatives. Such a tool could also support probabilistic hybrid systems, stochastic differential equations, separation of plant and controller, and component based design with information hiding. At the time of writing, the verification tool *Stabhyli*, based on decompositional reasoning with Lyapunov functions, is being developed in the AVACS Transregional Research Center. Within a tool prototype, the two levels of decomposition (i.e., based on SCCs and cycles) have been implemented and validated, yielding a number of successful and fully automatic stability proofs. Also, the tool supports the hierarchical component-based design of stable hybrid controllers by automatically computing Lyapunov function projections, validating the concept of the compositional design framework.

System Generalization. For a given GLF of a hybrid automaton, it is possible to derive the set of dynamics, guards and invariants for which it will also be the function retains its GLF property, proving GAS. Since this problem can again be expressed as an LMI, with reversed roles of dynamics and Lyapunov functions, it is possible to automatically generalize a stable hybrid system model. This generalization could come in the form of widened differential inclusions or widened guards of invariants. Such a generalization can then be used to measure the robustness of the model with respect to external disturbances or modeling errors. Informally speaking, the "size" of the generalization along a particular dimension (for instance a differential inclusion parameter or the strictness of a guard constraint) indicates additional system behaviors which are also tolerable and therefore acts as an indicator of system robustness. With decompositional reasoning, it is possible to automatically conduct such a generalization locally on modes of the hybrid automaton.

Alternative Hybrid Automaton Models for a Hybrid System. The decompositional analysis uses the graph structure of a hybrid automaton, but, of course, the very same system can be represented by different automata with different graph structures. One such case is the splitting operation, which results in such a different but equivalent model which is easier to analyze. However, also for the discovery of SCCs, it is very useful to have an optimal representation (i.e., a representation resulting in the maximum number of SCCs). Therefore, potential future work could include the analysis of different models for the same system. With the help of reachability analysis, it might be possible to discover hidden "bifurcation points" of trajectories where small perturbations within a single mode lead to permanently different behavior. If such bifurcation points can be discovered, they can be made explicit in the discrete structure and exploited for decompositional analysis. In particular, this is interesting for quantitative analysis of stability in probability. If an SCC cannot be shown AS-GA, it might still be possible to identify hidden SCCs inside for which this might be the case and whose probability mass could then be added to the probability of convergence.

Clique Discovery. The decomposition procedure works well with sparse graphs, but is not very useful for dense graphs which are almost fully connected. Since hybrid systems with such a graph structure do not lend themselves to discrete decomposition at all, it is desirable to detect such sub-automata and mark them as exempt from the decomposition. Furthermore, such sub-graphs should use common or continuous Lyapunov functions in many cases, due to the strong interrelations between their LLFs. While we gave some criteria for identifying such sub-automata, there is also the need for an algorithm that discovers these cases (possibly heuristically) and treats them accordingly in the reduction procedure. Such fully or almost fully connected sub-graphs could be merged into a single "super-node" for the purpose of decomposition after their detection, to prevent their decomposition.

Heuristics for Lyapunov Function Computation, Reduction, Splitting and Refinement. As of now, the choice of a suitable LLF parametrization is still left to the user. Heuristic identification of suitable Lyapunov function templates is therefore also useful for a fully automated verification process. This analysis could be conducted by analyzing the differential inclusion in question and by successively using "more advanced" templates (e.g., templates containing higher maximum degree polynomials). A related idea is the use of discontinuous LLF templates, which can be modeled by artificially splitting a mode in two. Determining such splittings is a hard problem but it can in principle be conducted heuristically [Oehlerking *et al.*, 2007]. Also, the use of different verification methods for different parts of the automaton is possible in principle. For instance, for some modes, it might be more suitable to search for (piecewise) linear LLFs using linear programming methods. In this case, a decision procedure picking the most suitable tool for each sub-automaton is required. Also, the initial choice of optimization directions could be picked based on an a priori analysis of the mode dynamics. The result obtained from the reduction/refinement procedure also depends on the order in which nodes are split and cycles reduced. If there are various reducible cycles, then it may make sense to first reduce the cycle expected to result in less conservativeness. There may also be several nodes available for splitting at any one time. While the heuristic of looking at their in- and outdegrees is reasonable, it is not guaranteed to be optimal. Therefore, it is beneficial to look at optimality conditions for the splitting procedure. However, this is a difficult problem since it is already hard to define should be considered "optimal" here. The number of reduction plays a role, as well as the size of the individual cycles (and therefore the resulting LMI problems) and the conservativeness of each single reduction. The solution provided in this thesis is guaranteed to work, but does not make optimality assertions, which could be a potential area of future work.

As has been demonstrated by their inclusion in the prototype tool *Stabhyli*, the results presented in this thesis are already very useful for stability verification of hybrid automata with complex structure, Large systems do not result in intractably large constraint systems, and the stepwise reduction of graphs gives useful feedback in case verification fails. Furthermore, this thesis can be seen as a step of bridging the gap between the design of realistic complex systems and the design of systems for which stability is actually verifiable. By following a structured design procedure based on the decomposition and composition results in this thesis, it is possible to design complex hybrid automata for which stability can be guaranteed. The improvements above can still increase the usefulness of such a tool by improving its efficiency, broadening the class of systems for which verification is successful and by providing additional feedback to the user.

Bibliography

Abate, Alessandro. 2007. *Probabilistic Reachability for Stochastic Hybrid Systems: Theory, Computations, and Applications*. Ph.D. thesis, Univerity of California, Berkeley, USA.

Ahmadi, Amir Ali, & Parrilo, Pablo. 2008. Non-monotonic Lyapunov Functions for Stability of Discrete Time Nonlinear and Switched systems. *Pages 614–621 of: Proceedings of the 47th IEEE Conference on Decision and Control (CDC'08)*. IEEE.

Alur, Rajeev, Courcoubetis, Costas, Henzinger, Thomas, & Ho, Pei-Hsin. 1993. Hybrid Automata: An Algorithmic Approach to the Specification and Verification of Hybrid Systems. *Pages 209–229 of: Hybrid Systems*. Lecture Notes in Computer Science, vol. 736. Springer.

Bayen, Alexandre, Raffard, Robin, & Tomlin, Claire. 2004. Network Congestion Alleviation Using Adjoint Hybrid Control: Application to Highways. *Pages 95–110 of: Proceedings of the 7th International Workshop on Hybrid Systems: Computation and Control (HSCC'04)*. Lecture Notes in Computer Science, vol. 2993. Springer.

Bemporad, Alberto, Borrelli, Francesco, & Morari, Manfred. 2002. On the Optimal Control Law for Linear Discrete Time Hybrid Systems. *Pages 105–119 of: Proceedings of the 5th International Workshop on Hybrid Systems: Computation and Control (HSCC'02)*. Lecture Notes in Computer Science, vol. 2289. Springer.

Benson, Steven, Ye, Yinyu, & Zhang, Xiong. 2000. Solving Large-Scale Sparse Semidefinite Programs for Combinatorial Optimization. *SIAM Journal on Optimization*, **10**(2), 443–461.

Bhatia, Rajendra. 2007. *Positive Definite Matrices*. Princeton Series in Applied Mathematics. Princeton University Press, ISBN: 978-1400827787.

Blondel, Vincent, & Tsitsiklis, John. 1999. Complexity of Stability and Controllability of Elementary Hybrid Systems. *Automatica*, **35**(3), 479–489.

Blondel, Vincent, & Tsitsiklis, John. 2000. The Boundedness of all Products of a Pair of Matrices is Undecidable. *Systems and Control Letters*, **41**(2), 135–140.

Blondel, Vincent, Bournez, Olivier, Koiran, Pascal, Papadimitriou, Christos, & Tsitsiklis, John. 2001a. Deciding Stability and Mortality of Piecewise Affine Dynamical Systems. *Theoretical Computer Science*, **255**(1-2), 687–696.

Blondel, Vincent, Bournez, Olivier, Koiran, Pascal, & Tsitsiklis, John. 2001b. The Stability of Saturated Linear Dynamical Systems is Undecidable. *Journal of Computer and System Sciences*, **62**, 442–462.

Borchers, Brian. 1999. CSDP, a C Library for Semidefinite Programming. *Optimization Methods and Software*, **10**(1), 613–623.

Boyd, Stephen, & Vandenberghe, Lieven. 2004. *Convex Optimization.* Cambridge University Press, ISBN: 978-0521833783.

Boyd, Stephen, El Ghaoui, Laurent, Feron, Eric, & Balakrishnan, Venkataramanan. 1994. *Linear Matrix Inequalities in System and Control Theory.* Society for Industrial and Applied Mathematics, ISBN 0-89871-334-X.

Branicky, Michael. 1994. Stability of Switched and Hybrid Systems. *In: Proceedings of the 33rd Conference on Decision and Control (CDC'94).* IEEE.

Branicky, Michael. 1998. Multiple Lyapunov Functions and other Analysis Tools for Switched and Hybrid Systems. *IEEE Transactions on Automatic Control*, **43**(4), 475–482.

Cai, Chaohong, & Teel, Andrew R. 2005. Results on Input-to-state Stability for Hybrid Systems. *Pages 5403–5408 of: Proceedings of the 44th IEEE Conference on Decision and Control (CDC'05).* IEEE.

Cai, Chaohong, Teel, Andrew, & Goebel, Rafal. 2007. Smooth Lyapunov Functions for Hybrid Systems – Part I: Existence is Equivalent to Robustness. *IEEE Transactions on Automatic Control*, **52**(7), 1264–1277.

Cai, Chaohong, Teel, Andrew, & Goebel, Rafal. 2008. Smooth Lyapunov Functions for Hybrid Systems – Part II: (Pre)Asymptotically Stable Compact Sets. *IEEE Transactions on Automatic Control*, **53**(3), 734–748.

Casagrande, Alberto, Corvaja, Pietro, Piazza, Carla, & Mishra, Bud. 2008. Decidable Compositions of o-minimal Automata. *Pages 274–288 of: Proceedings of the 6th International Symposium on Automated Technology for Verification and Analysis (ATVA'08).* Lecture Notes in Computer Science, vol. 5311. Springer.

Cassandras, Christos, & Lygeros, John. 2007. *Stochastic Hybrid Systems.* Control Engineering Series. Taylor and Francis, ISBN: 978-0849390838.

Chatterjee, Debasish, & Liberzon, Daniel. 2006. Stability Analysis of Deterministic and Stochastic Switched Systems via a Comparison Principle and multiple Lyapunov functions. *SIAM Journal on Control and Optimization*, **45**(1), 174–206.

Chatterjee, Debasish, & Liberzon, Daniel. 2007. On Stability of Randomly Switched Nonlinear Systems. *IEEE Transactions on Automatic Cotnrol*, **52**(12), 2390–2394.

Chesi, Graziano. 2004. On the Estimation of the Domain of Attraction for Uncertain Polynomial Systems via LMIs. *In: Proceedings of the 43rd IEEE Conference on Decision and Control (CDC'04).* IEEE.

Cortés, Jorge. 2008. Discontinuous Dynamical Systems. *IEEE Control Systems Magazine*, **28**(3), 36–73.

Daafouz, Jamal, Riedinger, Pierre, & Iung, Claude. 2002. Stability Analysis and Control Synthesis for Switched Systems: A Switched Lyapunov Function Approach. *IEEE Transactions on Automatic Control*, **47**(11), 1883–1887.

Damm, Werner, Mikschl, Alfred, Oehlerking, Jens, Olderog, Ernst-Rüdiger, Pang, Jun, Platzer, André, Segelken, Marc, & Wirtz, Boris. 2007. Automating Verification of Cooperation, Control, and Design in Traffic Applications. *Pages 115–169 of:* Jones, Cliff, Liu, Zhiming, & Woodcock, Jim (eds), *Formal Methods and Hybrid Real-Time Systems.* Lecture Notes in Computer Science, vol. 4700. Springer.

Damm, Werner, Dierks, Henning, Oehlerking, Jens, & Pnueli, Amir. 2010. Towards Component Based Design of Hybrid Systems: Safety and Stability. *Pages 96–143 of:* Manna, Zohar, & Peled, Doron (eds), *Time for Verification - Essays in Memory of Amir Pnueli.* Lecture Notes in Computer Science, vol. 6200. Springer.

Das, Indraneel, & Dennis, John. 1998. Normal-Boundary Intersection: a New Method for Generating the Pareto Surface in Multicriteria Optimization Problems. *SIAM Journal on Optimization*, **8**, 631–657.

Dashkovskiy, Sergey, Rüffer, Björn, & Wirth, Fabian. 2008. Stability of Interconnections of ISS Systems. *In: Proceedings of the 8th SICE Annual Conference on Control Systems (electronic).*

Dimarogonas, Dimos, & Kyriakopoulos, Kostas. 2004. Lyapunov-like Stability of Switched Stochastic Systems. *Pages 1868–1872 of: Proceedings of the 2004 American Control Conference (ACC'04).*

Donkers, Tijs, Hetel, Laurentiu, & Heemels, Maurice. 2009. Stability Analysis of Networked Control Systems Using a Switched Linear System Approach. *Pages 150–164 of: Proceedings of the 12th International Conference on Hybrid Systems: Computation and Control (HSCC'09).* Lecture Notes in Computer Science, vol. 5469.

Feng, Gang. 2002. Stability Analysis of Piecewise Discrete-Time Linear Systems. *IEEE Transactions on Automatic Control*, **47**(7), 1108–1112.

Ferrari-Trecate, Giancarlo, Cuzzola, Francesco Alessandro, Mignone, Domenico, & Morari, Manfred. 2002. Analysis of Discrete-time Piecewise Affine and Hybrid systems. *Automatica*, **38**(12), 2139–2146.

FRANKOWSKA, HÉLÈNE, & AUBIN, JEAN-PIERRE. 2009. *Set-Valued Analysis.* Birkhäuser, ISBN: 978-0817648473.

FRÄNZLE, MARTIN, HERDE, CHRISTIAN, TEIGE, TINO, RATSCHAN, STEFAN, & SCHUBERT, TOBIAS. 2007. Efficient Solving of Large Non-linear Arithmetic Constraint Systems with Complex Boolean Structure. *Journal on Satisfiability, Boolean Modeling and Computation*, **1**, 209–236.

FREHSE, GORAN. 2008. PHAVer: Algorithmic Verification of Hybrid Systems past HyTech. *International Journal on Software Tools for Technology Transfer*, **10**(3), 263–279.

FREHSE, GORAN, HAN, ZHI, & KROGH, BRUCE. 2004. Assume-Guarantee Reasoning for Hybrid I/O-Automata by Over-Approximation of Continuous Interaction. *Pages 479 – 484 of: Proceedings of the 43rd IEEE Conference on Decision and Control (CDC'04).* IEEE.

GIRARD, ANTOINE. 2003. Computation and Stability Analysis of Limit Cycles in Piecewise Linear Hybrid Systems. *In: Proceedings of the 2003 IFAC Conference on Analysis and Design of Hybrid Systems (ADHS'03).* Elsevier.

GONÇALVES, JORGE. 2005. Regions of Stability for Limit Cycle Oscillations in Piecewise Linear Systems. *IEEE Transactions on Automatic Control*, **50**(11), 1877–1882.

GROSU, RADU, & STAUNER, THOMAS. 2002. Modular and Visual Specification of Hybrid Systems: An Introduction to HyCharts. *Formal Methods in System Design*, **21**, 5–38.

GULWANI, SUMIT, & TIWARI, ASHISH. 2008. Constraint-based Approach for Analysis of Hybrid Systems. *Pages 190 – 203 of: Proceedings of the 20th International Conference on Computer Aided Verification (CAV'08).* Lecture Notes in Computer Science, vol. 5123. Springer.

HABETS, LUC, COLLINS, PIETER, & VAN SCHUPPEN, JAN H. 2006. Reachability and Control Synthesis for Piecewise-affine Hybrid Systems on Simplices. *IEEE Transactions on Automatic Control*, **51**(6), 938–948.

HAFSTEIN, SIGURDUR FREYR. 2004. A Constructive Converse Lyapunov Theorem on Exponential Stability. *Discrete and Continuous Dynamical Systems*, **10**(3), 667–678.

HEEMELS, MAURICE, & WEILAND, SIEP. 2008. Input-to-state Stability and Interconnections of Discontinuous Dynamical Systems. *Automatica*, **44**(12), 3079–3086.

HEEMELS, MAURICE, WEILAND, SIEP, & JULOSKI, ALEKSANDAR. 2007. Input-to-State Stability of Discontinuous Dynamical Systems with an Observer-Based Control Application. *Pages 259–272 of: Proceedings of the 10th International Workshop on Hybrid Systems: Computation and Control (HSCC'07).* Lecture Notes in Computer Science, vol. 4416. Springer.

Henzinger, Thomas, & Rusu, Vlad. 1998. Reachability Verification for Hybrid Automata. *Pages 190–204 of: Proceedings of the 1st International Workshop on Hybrid Systems: Computation and Control (HSCC'98).* Lecture Notes in Computer Science, vol. 1386.

Hespanha, João, Liberzon, Daniel, & Teel, Andrew. 2008. Lyapunov Conditions for Input-to-state Stability for Impulsive Systems. *Automatica*, **44**(11), 2735–2744.

Hiskens, Ian. 2001. Stability of Limit Cycles in Hybrid Systems. *In: Proceedings of the 34th Hawaii International Conference on System Sciences (HICSS'01)*, vol. 2. IEEE.

Jansson, Christian. 2006. VSDP: A MATLAB Software Package for Verified Semidefinite Programming. *Pages 327–330 of: Proceedings of the 2006 International Symposium on Nonlinear Theory and its Applications (NOLTA'06).* IEICE.

Johansson, Mikael, & Rantzer, Anders. 1998. Computation of Piecewise Quadratic Lyapunov Functions for Hybrid Systems. *IEEE Transactions on Automatic Control*, **43**(4), 555–559.

Kalman, Rudolf, & Bertram, John. 1960. Control System Analysis and Design via the "Second Method" of Lyapunov. *Transactions of the ASME, Journal of Basic Engineering*, **82**, 371–400.

Khalil, Hassan. 1996. *Nonlinear Systems.* 2nd edn. Prentice-Hall, ISBN: 978-0130673893.

Korenevskii, D. G. 1987. Stability with Probability 1 of Solutions of Systems of Linear Ito Stochastic Differential-Difference Equations. *Ukrainian Mathematical Journal*, **39**(1), 26–30.

Kočvara, Michal, & Stingl, Michael. 2003. PENNON - A Generalized Augmented Lagrangian Method for Semidefinite Programming. *Pages 297–315 of: High Performance Algorithms and Software for Nonlinear Optimization.* Kluwer.

Kozin, F. 1972. Stability of the Linear Stochastic System. *Pages 186–229 of: Lecture Notes in Mathematics*, vol. 294. Springer.

Kushner, Harold. 1967. *Stochastic Stability and Control.* Mathetatics in Science and Engineering, vol. 33. Academic Press, ISBN: 978-0124301504.

Laila, Dina Shona, & Nešić, Dragan. 2003. Discrete-Time Lyapunov-Based Small-Gain Theorem for Parameterized Interconnected ISS Systems. *IEEE Transactions on Automatic Control*, **48**(10), 1783–1788.

Langerak, Rom, & Poldermans, Jan Willem. 2005. Tools for Stability of Switching Linear Systems: Gain Automata and Delay Compensation. *Pages 4867–4872 of: Proceedings of the 44th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC'05).*

Langerak, Rom, Polderman, Jan Willem, & Krilavičius, Tomas. 2003. Stability Analysis for Hybrid Automata Using Conservative Gains. *Pages 377–382 of: Proceedings of the 2003 IFAC Conference on Analysis and Design of Hybrid Systems (ADHS'03).*

Lazar, Mircea, & Jokić, Andrej. 2010. On Infinity Norms as Lyapunov Functions for Piecewise Affine Systems. *Pages 131–140 of: Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control (HSCC'10).* ACM.

Liberzon, Daniel. 2003. *Switching in Systems and Control.* Birkhäuser, ISBN: 9780817642976.

Liberzon, Daniel, & Nešić, Dragan. 2006. Stability Analysis of Hybrid Systems Via Small-Gain Theorems. *Pages 421–435 of: Proceedings of the 9th International Workshop on Hybrid Systems: Computation and Control (HSCC'06).* Lecture Notes in Computer Science, vol. 3927. Springer.

Löfberg, Johan. 2004. YALMIP: A Toolbox for Modeling and Optimization in MATLAB. *Pages 284–289 of: 2004 IEEE International Symposium on Computer Aided Control System Design (CACSD'04).* IEEE.

Loparo, Kenneth, & Feng, Xiangho. 1996. Stability of Stochastic Systems. *Pages 1105–1126 of: The Control Handbook.* CRC Press, ISBN: 978-0849385704.

Lyapunov, Aleksandr. 1907. Problème général de la stabilité du movement. *Ann. Fac. Sci. Toulouse*, **9**, 203–474. (Translation of a paper published in Comm. Soc. math. Kharkow, 1893, reprinted in Ann. math. Studies No. 17, Princeton University Press, 1949).

Lynch, Nancy, Segala, Roberto, & Vaandrager, Frits. 2003. Hybrid I/O Automata. *Information and Computation*, **185**(1), 105–157.

Manwell, James, & McGowan, Jon. 1993. Lead Acid Battery Storage Model for Hybrid Energy Systems. *Solar Energy*, **50**(5), 399–405.

Mitchell, Ian, & Susuki, Yoshihiko. 2008. Level Set Methods for Computing Reachable Sets of Hybrid Systems with Differential Algebraic Equation Dynamics. *Pages 630–633 of: 11th International Workshop on Hybrid Systems: Computation and Control (HSCC'08).* Lecture Notes in Computer Science, vol. 4981. Springer.

Nesterov, Yurii, & Nemirovskii, Arkadii. 1994. *Interior Point Polynomial Algorithms in Convex Programming.* Society for Industrial and Applied Mathematics, ISBN: 9780898715156.

Nešić, Dragan, & Liberzon, Daniel. 2005. A Small-Gain Approach to Stability Analysis of Hybrid Systems. *Pages 5409–5414 of: 44th IEEE Conference on Decision and Control and European Control Conference 2005 (CDC-ECC'05).* IEEE.

Oehlerking, Jens, & Theel, Oliver. 2009a. Decompositional Construction of Lyapunov Functions for Hybrid Systems. *Pages 276–290 of: Proceedings of the 12th International Conference on Hybrid Systems: Computation and Control (HSCC'09).* Leture Notes in Computer Science, vol. 5469. Springer.

Oehlerking, Jens, & Theel, Oliver. 2009b. A Decompositional Proof Scheme for Automated Convergence Proofs of Stochastic Hybrid Systems. *Pages 151–165 of: 7th International Symposium on Automated Technology for Verification and Analysis (ATVA'09).* Leture Notes in Computer Science, vol. 5799. Springer.

Oehlerking, Jens, Burchardt, Henning, & Theel, Oliver. 2007. Fully Automated Stability Verification for Piecewise Affine Systems. *Pages 741–745 of: Proceedings of the 10th International Workshop on Hybrid Systems: Computation and Control (HSCC'07).* Leture Notes in Computer Science, vol. 4416. Springer.

Øksendal, Bernt. 2003. *Stochastic Differential Equations: An Introduction with Applications.* Springer, ISBN: 9783540047582.

Papachristodoulou, Antonis. 2004. Analysis of Nonlinear Time-Delay Systems Using the Sum of Squares Decomposition. *Pages 4153–4158 of: Proceedings of the 2004 American Control Conference (ACC'04).* IEEE.

Papachristodoulou, Antonis, & Prajna, Stephen. 2002. On the Construction of Lyapunov Functions using the Sums of Squares Decomposition. *Pages 3482 – 3487 of: Proceedings of the 41st IEEE Conference on Decision and Control (CDC'02).* IEEE.

Papachristodoulou, Antonis, & Prajna, Stephen. 2005. Analysis of Non-polynomial Systems using the Sums of Squares Decomposition. *Pages 23–43 of: Positive Polynomials in Control.* Lecture Notes in Control and Information Sciences, vol. 312. Springer.

Parrilo, Pablo. 2003. Semidefinite Programming Relaxations for Semialgebraic Problems. *Mathematical Programming Ser. B*, **96**, 293–320.

Parrilo, Pablo, & Jadbabaie, Ali. 2007. Approximations of the Joint Spectral Radius of a Set of Matrices Using Sums of Squares. *Pages 444–458 of: Proceedings of the 10th International Workshop on Hybrid Systems: Computation and Control (HSCC'07).* Leture Notes in Computer Science, vol. 4416. Springer.

Parrilo, Pablo, & Lall, Sanjay. 2003. Semidefinite Programming Relaxations and Algebraic Optimization in Control. *European Journal of Control*, **9**(2-3), 307–321.

Peet, Matthew, Papachristodoulou, Antonis, & Lall, Sanjay. 2006. Positive Forms and Stability of Linear Time-Delay Systems. *SIAM Journal on Control and Optimization*, **47**(6), 3237–3258.

Pettersson, Stefan. 1999. *Analysis and Design of Hybrid Systems.* Ph.D. thesis, Chalmers University of Technology, Gothenburg, Sweden.

Pettersson, Stefan, & Lennartson, Bengt. 1996. Stability and Robustness for Hybrid Systems. *Pages 1202 – 1207 of: 35th IEEE Conference on Decision and Control (CDC'96).* IEEE.

Pettersson, Stefan, & Lennartson, Bengt. 1997. *A Converse Theorem for Exponential Stability using Piecewise Quadratic Lyapunov Functions.* Tech. rept. Control Engineering Lab, Chalmers University of Technology, Gothenburg.

Platzer, André, & Clarke, Edmund. 2008. Computing Differential Invariants of Hybrid Systems as Fixedpoints. *Pages 176–189 of: 20th International Conference on Computer Aided Verification (CAV'08).* Lecture Notes in Computer Science, vol. 5123. Springer.

Platzer, André, & Quesel, Jan-David. 2008. KeYmaera: A Hybrid Theorem Prover for Hybrid Systems. *Pages 171–178 of: International Joint Conference Automated Reasoning (IJCAR'08).* Lecture Notes in Computer Science, vol. 5195. Springer.

Podelski, Andreas, & Wagner, Silke. 2006. Model Checking of Hybrid Systems: From Reachability towards Stability. *Pages 507–521 of: Proceedings of the 9th International Workshop on Hybrid Systems: Computation and Control (HSCC'06).* Lecture Notes in Computer Science, vol. 3927. Springer.

Prajna, Stephen, & Jadbabaie, Ali. 2004. Safety Verification of Hybrid Systems Using Barrier Certificates. *Pages 477–492 of: Proceedings of the 7th International Workshop on Hybrid Systems: Computation and Control (HSCC'04)*, vol. 2993. Springer.

Prajna, Stephen, & Papachristodoulou, Antonis. 2003. Analysis of Switched and Hybrid Systems – Beyond Piecewise Quadratic Models. *Pages 2799–2784 of: Proceedings of the 2003 American Control Conference (ACC'03).* IEEE.

Prajna, Stephen, & Rantzer, Anders. 2005. On the Necessity of Barrier Certificates. *In: Proceedings of the 16th IFAC World Congress.* IFAC.

Puterman, Martin. 1994. *Markov Decision Processes.* Wiley, ISBN: 0-471-72782-2.

Ratschan, Stefan, & She, Zhikun. 2010. Providing a Basin of Attraction to a Target Region of Polynomial Systems by Computation of Lyapunov-like Functions. *SIAM Journal on Control and Optimization*, **48**(7), 4377–4394.

Reznick, Bruce. 2000. Some Concrete Aspects of Hilbert's 17th Problem. *Contemporary Mathematics*, **253**, 251–272.

Romanko, Oleksandr, Pólik, Imre, & Sturm, Jos F. 1999. *Using SeDuMi 1.02, a MATLAB Toolbox for Optimization over Symmetric Cones.* Manual. Published at `http://sedumi.ie.lehigh.edu`.

Rubensson, Marcus, & Lennartsson, Bengt. 2000. Stability of Limit Cycles in Hybrid Systems using Discrete-time Lyapunov Techniques. *Pages 1397–1402 of: Proceedings of the 39th IEEE Conference on Decision and Control (CDC'00).* IEEE.

SHIRYAEV, ALBERT. 1996. *Probability.* Second edn. Springer, ISBN: 978-0387945491.

SIMIĆ, SLOBODAN. 2002. Hybrid Limit Cycles and Hybrid Poincaré-Bendixson. *Pages 22–26 of: Proceedings of the 2002 IFAC World Congress.* IFAC.

SONTAG, EDUARDO. 1998. *Mathematical Control Theory: Deterministic Finite Dimensional Systems.* Second edn. Textbooks in Applied Mathematics, no. 6. Springer, ISBN: 9780387984896.

SPROSTON, JEREMY. 2000. Decidable Model Checking of Probabilistic Hybrid Automata. *Pages 501–514 of: Proceedings of the 6th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'00).* Lecture Notes in Computer Science, vol. 1926. Springer.

TAN, WEEHONG, & PACKARD, ANDREW. 2008. Stability Region Analysis Using Polynomial Lyapunov Functions and Sums-of-Squares Programming. *IEEE Transactions on Automatic Control*, **53**(2), 565–570.

TARJAN, ROBERT. 1972. Depth-first Search and Linear Graph Algorithms. *SIAM Journal on Computing*, **1**(2), 146–160.

THOMASSEN, CARSTEN. 1997. On the Complexity of Finding a Minimum Cycle Cover of a Graph. *SIAM Journal on Computing*, **26**(3), 675–677.

TOH, KIM-CHUAN, TODD, MICHAEL, & TÜTÜNCÜ, REHA. 1999. SDPT3 – a Matlab Software Package for Semidefinite Programming. *Optimization Methods and Software*, **11**, 545–581.

VU, LINH, & LIBERZON, DANIEL. 2005. Common Lyapunov Functions for Families of Commuting Linear Systems. *Systems and Control Letters*, **54**, 405–416.

WITSENHAUSEN, HANS. 1966. A Class of Hybrid-State Continuous Dynamical Systems. *IEEE Transactions on Automatic Control*, **11**(2), 161–167.

YAKUBOVICH, VLADIMIR. 1977. S-procedure in Nonlinear Control Theory. *Vestnik Leningrad Univ.*, **4**(1), 73–93.

YAMASHITA, MAKOTO, FUJISAWA, KATSUKI, & KOJIMA, MASAKAZU. 2003. Implementation and Evaluation of SDPA 6.0 (SemiDefinite Programming Algorithm 6.0). *Optimization Methods and Software*, **18**, 491–505.

YE, HUI, MICHEL, ANTHONY, & HOU, LING. 1998. Stability Theory for Hybrid Dynamical Systems. *IEEE Transactions on Automatic Control*, **43**(4), 461–474.

ZHAI, GUISHENG, LIN, HAI, & ANTSAKLIS, PANOS. 2003. Quadratic Stabilizability of Switched Linear Systems with Polytopic Uncertainties. *International Journal of Control*, **76**(7), 747–753.

Zhang, Lijun, She, Zhikun, Ratschan, Stefan, Hermanns, Holger, & Hahn, Ernst Moritz. 2010. Safety Verification for Probabilistic Hybrid Systems. *Pages 196–211 of: Proceedings of the 22nd International Conference on Computer Aided Verification (CAV'10).* Lecture Notes in Computer Science, vol. 6174. Springer.

Zolezzi, Tullio. 2002. Differential Inclusions and Sliding Mode Control. *Chap. 2, pages 29–52 of:* Perruquetti, Wilfrid, & Barbot, Jean Pierre (eds), *Sliding Mode Control in Engineering.* CRC Press, ISBN: 978-0824706715.

List of Figures