AVACS – Automatic Verification and Analysis of Complex Systems

REPORTS

of SFB/TR 14 AVACS

Editors: Board of SFB/TR 14 AVACS

Component Based Design of Hybrid Systems: A Case Study on Concurrency and Coupling

by

Werner Damm, Willem Hagemann, Eike Möhlmann, Astrid Rakow

AVACS Technical Report No. 95
April 2014
ISSN: 1860-9821

Publisher: Sonderforschungsbereich/Transregio 14 AVACS
(Automatic Verification and Analysis of Complex Systems)
Editors: Bernd Becker, Werner Damm, Bernd Finkbeiner, Martin Fränzle,
Ernst-Rüdiger Olderog, Andreas Podelski
ATRs (AVACS Technical Reports) are freely downloadable from `www.avacs.org`

Author(s) contact: Werner Damm (`wd@avacs.org`).

Component Based Design of Hybrid Systems: A Case Study on Concurrency and Coupling

Extended Version

Werner Damm, Willem Hagemann, Eike Möhlmann, Astrid Rakow
Carl von Ossietzky University of Oldenburg
Department of Computer Science
D-26111 Oldenburg, Germany
{werner.damm,willem.hagemann,eike.moehlmann,astrid.rakow}@informatik.uni-oldenburg.de

April 11, 2014

In the search of design principles that allow compositional reasoning about safety and stability properties of hybrid controllers we examine a case study on a simplified driver assistance system for lane keeping and velocity control. We thereby target loosely coupled systems: the composed system has to accomplish a task that may depend on several of its subcomponents while little coordination between them is necessary. Our assistance system has to accomplish a comfortable centrifugal force, lane keeping and velocity control. This leads to an architecture composed of a velocity controller and a steering controller, where each controller has its local objectives and together they maintain a global objective. The steering controller makes time bounded promises about its steering, which the velocity controller uses for optimization. For this system, we deductively prove from the components' properties that the objectives of the composed system are accomplished.

Keywords:
Hybrid Systems; Automatic Verification; Stability; Safety; Composition; Interfaces; Specifications; Assume-Guarantee; Computer-Aided Design

1 Introduction

In [4] Damm et al. proposed a library based design methodology for constructing hybrid controllers from a component library. The suggested framework preserves safety and stability properties. Verification conditions were identified to alleviate the support by automatic verification tools.

While in [4], control components are sequentially active, our research aims to extend this approach by allowing sequential as well as concurrent control components. A first step towards this goal is this case study on an advanced driver assistance system (ADAS) as a loosely coupled concurrent control application. In the following presentation of our case study we highlight what we identified as fundamental characteristics of the application domain and resulting key insights for the extension of [4].

Basically, a component can be thought of as an encapsulation of an implementation that interacts with its environment via its interface and that can be (re-)used in certain deployment contexts, where it guarantees to provide its services. The importance of contract-based compositional design has long been recognized as an instrument to reduce the number of late integration errors [17] and to boost re-use of components [8]. Contract-based specifications have been integrated into the Core meta Model underlying the CESAR reference technology platform for critical system design, which is recognized by the Artemis Industrial Association as the unifying platform for building a suite of interoperable methods and tools covering the complete system life cycle for critical systems [1]. Projects like Speeds[1] have provided formal contract based component interface specifications for real-time and safety requirements (as in [6, 7]).

Determining an appropriate system decomposition into subcomponents is a challenging task though. There certainly is an area of conflict arising from the benefits of re-using components and the cost payed for decoupling the component from its environment in order to make it reusable. We therefore focus on *loosely coupled systems*, where one component needs to know little about its concurrent environment. In the search of design principles that allow compositional reasoning of loosely coupled components we examine a simplified ADAS in the setting of single lane country roads with no additional traffic. The ADAS has to

(o1) maintain a centrifugal force comfortable for a driver,

(o2) bring and then keep the car on the center of its lane,

(o3) control the speed also considering driver requests for a certain speed value.

Our ADAS consists of two concurrent controllers, a velocity controller, VC, and a steering controller, SC. The velocity controller is responsible for its *local objective* of controlling the speed, the steering controller is responsible for lane keeping and together they have to accomplish the *global objective* of guaranteeing a comfortable centrifugal force. The global objective implies a loose coupling between VC and SC, because maintaining a comfortable centrifugal force means that a car cannot go arbitrarily fast in a narrow turn and a fast car may not decide to change its steering angle too abruptly. But as steering angles usually are small, the coupling is quite loose.

In the sequel, we will develop hybrid interfaces for the two controllers and suggest a way of coordination between them. In particular, we suggest (a) to use an assume-guarantee approach and annotate assumptions and guarantees as part of the interface. A component's guarantee describes its services. The assumption specifies a component's deployment context. Any deployment context has to refine the

[1] IST Project 03347, see www.speeds.eu.com

assumption. That way the assumption specifies the most adverse environment, in which the component is guaranteed to deliver its services. We further suggest (b) the use of events encoding time bounded promises for communication between concurrent controllers. By this means, SC can tell VC that it will do a gentle turn, so that the VC may increase the velocity. Finally we will deductively prove that our ADAS accomplishes its objectives, (o1) to (o3), using automatic verification tools.
Within our research on component based specification, this case study was used to provide an exemplary interface specification for loosely coupled components and to illustrate the resulting proof obligations and tool support for dismissing them.

Outline In the next section we briefly introduce the basic concepts of the framework we suggest. In particular we motivate the interface ingredients based on our ADAS example. In Sect. 2, we present the ADAS specification in terms of its components, that is a steering controller SC, a velocity controller VC and the event based coordination, following our formalism. Before we show in Sect. 5 that both SC and VC satisfy their interface specification, we briefly introduce in Sect. 4 the tools Stabhyli and SoapBox applied to establish the interface implementation. From the interface specifications of SC and VC we derive an interface specification for the ADAS composed of SC and VC in Sect. 6.

2 Specifying the ADAS

Our driver assistance system has to perform concurrently steering and velocity control for a car on a road. With other words, we study a system of a plant, which is the controlled car, and a composite ADAS controller with its two subcomponents, velocity controller VC and steering controller SC. The ADAS controller perceives the plant via sensors and controls the plant via actuators. We specify all components of our system as hybrid automata with inputs and outputs. The modelled system has as inputs disturbances, that is variable values whose evolution is not captured by our model: a disturbance s on the actuator *acc*, the user's desired speed vel_{des} and the course of the road (cf. Fig. 1). The desired speed vel_{des} may only take integer values within the considered velocity range of $[10\frac{\mathsf{m}}{\mathsf{s}}, 45\frac{\mathsf{m}}{\mathsf{s}}]$. Actuators of the ADAS are the acceleration *acc* and steering, β_{steer}. The ADAS reads via sensors plant variables: the car's orientation β_{ori}, its position relative to the mid of the lane $dist_{\mathsf{cur}}$ (cf. Fig. 2), and the current velocity vel_{cur}. The plant defines the continuous evolution of plant variables and builds an open system with actuators (and possibly disturbances) as inputs. Our plant model is illustrated in Fig. 3 To rule out write conflicts, each actuator belongs to one concurrent controller only. SC controls β_{steer} and VC actuator *acc*. Consequently SC is in charge of lane keeping (o2), and VC of speed control (o3). These objectives canonically imply, that SC reads via sensors $\beta_{\mathsf{ori}}, dist_{\mathsf{cur}}$ and VC reads vel_{cur}. SC and VC together have to establish the global objective (o1), guaranteeing a comfortable centrifugal force.
In the remainder of this section we will discuss design decisions on the coordination between VC and SC.

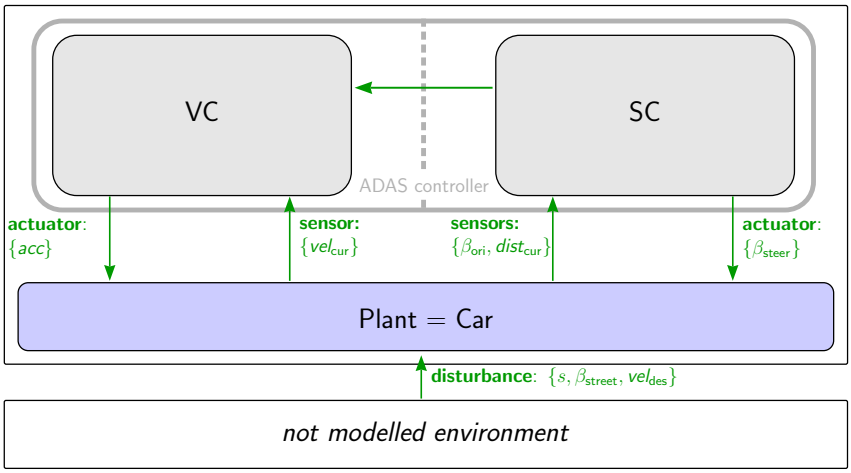


Figure 1: Overall architecture

2.1 Coordination

To summarize, we want to design a composite ADAS controller that accomplishes objectives (o1) to (o3). Lane keeping (o2) is a local objective of subcomponent SC and speed control (o3) is local to VC. Both subcomponents influence the (hence global) objective of guaranteeing comfortable centrifugal forces (o1).
We aim to extend the framework of [4], which targets to bridge the gap between specification and implementation models. Following [4] we aim to support distributed component based design of control systems taking realistic assumptions about reaction times into account. Thus we target a setting where instantaneous reactions are considered as inadequate, because of time penalties for task switching and delays in sensing and actuating and where coordination messages may get lost between controllers on different ECUs.

Events As we aim to support the design of safety critical systems, critical objectives always have to be guaranteed, even if all messages are lost. Hence we suggest to use coordination messages, so called *events*, that are optional for the receiving component. By sending an event, the sender promises a certain future behavior. Promises are tagged with a time stamp and a validity duration. While being aware of signal latencies, the receiving component can make use of this information for optimization, but has to be able to fulfill its guarantees even without any events.

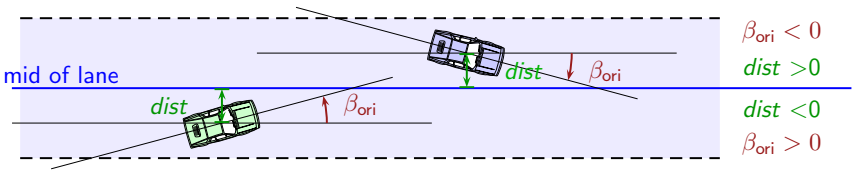


Figure 2: Car orientation and distance

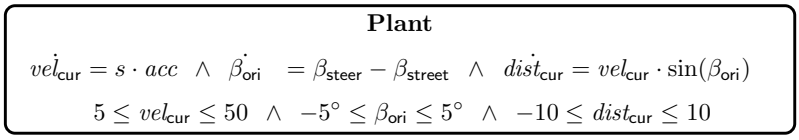


Figure 3: Hybrid automaton of the plant.

Assumptions To specify a component's deployment context, the assumptions φ^{assm} of the component are specified as part of its interface. A component with assumption φ^{assm} has to be able to deal with any behavior that satisfies φ^{assm}.

Control Hierarchies For coordination between coupled components, we derive a hierarchy on the controller's subcomponents from the priorities of controller's objectives. We give top priority to the driver's well-being, that is maintaining a comfortable centrifugal force F_{cntrif}. As $F_{\mathsf{cntrif}} = mass \cdot \omega^2 \cdot r$ [2], both VC and SC influence whether F_{cntrif} is comfortable. Hence the two controllers are coupled: VC may not choose an arbitrary speed, when SC is currently on a narrow curve and vice versa. VC becomes subordinate to SC, as lane keeping is considered as more important than reaching the user desired speed. This means that VC has to be able to deal with what ever steering the SC chooses, whereas SC assumes a cooperative VC, that never jeopardizes a comfortable F_{cntrif}. As VC cannot simultaneously be ready to deal with arbitrary steering and reach the user requested speed, we have to adopt the objective (o3): vel_{des} has to be reached if compatible with steering.

2.2 Interface Ingredients

In this section we briefly elaborate on the main ingredients for interfaces of concurrent control components in our framework.

Assumptions and Guarantees Assumptions express constraints on the component's environment. Our components specify *base assumptions* and *event assumptions*. A base assumption φ^{assm} has to hold unless the component is bound to fulfill an event promise. When it is bound to a promise, it may be able to relax some of its assumptions, hence it declares at its interface event assumptions. Assumptions may refer to inputs of the composite of plant and controller, Var_A. A control component specifies *guarantees* φ^{guara}, which may refer to Var_G, the outputs of the composite of plant and controller. Basically, as long as its base assumption holds, the component has to accomplish its guarantees. To cope with sensing delays, the component has to even accomplish its guarantees for a limited future of $\Delta^{time}_{\mathsf{lat}}$ after the base assumption expires. At the moment we think of φ^{assm} and φ^{guara} as simple predicates of the form $\Box \bigwedge_{v \in Var_A \cup \dot{Var}_A} (v \in I_a(v))$ and $\Box \bigwedge_{v \in Var_G \cup \dot{Var}_G} (v \in I_g(v))$ with intervals I_a, I_g, but plan to extend their expressiveness.

[2] ω is the angular rate, r is the radius of the curve the car is on, and *mass* is the mass of the passenger

Entry Condition As in [4] a component specifies activation constraints. For now we assume that φ^{entry} has the same form as φ^{assm}, but only has to hold initially. In the following we hence treat the entry condition as a special kind of base assumption.

Events An event $\mathcal{E}$ is defined as $(\varphi^{prom}, dueTime, duration)$ with φ^{prom} a predicate over a set of variables the event refers to, $dueTime$ a due time, and $duration$ a validity duration. At its interface a component declares the set of outgoing events Ev^{out}. Similar to guarantees, φ^{prom} encodes intervals for variables, i.e. φ^{prom} has the form $\bigwedge_{v \in Var_G}(v \in I_o(v))$. But unlike guarantees, promises are time-limited: The sending component has to guarantee that variable values are within I_o after the due time elapses and at least for the promised validity duration. A component is bound to fulfill its promises unless the base assumptions holds: if there are several promises for the same point in time, the most restrictive promise has to be realized. A component may not send contradictory promises. As there also is a latency for sending events between distributed components, we further demand a separation of at least $\Delta^{time}_{\mathsf{sep}}$. An interface specifies for each $\mathcal{E} \in Ev^{out}$ an event assumption $\varphi^{assm}_{\mathcal{E}}$ which may relax the base assumption φ^{assm}. Analogously an interface specifies for each $\mathcal{E} \in Ev^{in}$ an event guarantee which may relax the base guarantee φ^{guara}. That way communication allows more behavior: if a controller sends an event it accepts extra behavior in its environment while a controller receiving an event might be allowed to exhibit extra behavior.
Incoming events Ev^{in} at the interface, declare what kinds of events the component can use for optimization. Note, that in our framework events are only used to optimize a controller's behavior. The controller guarantees have to be accomplished given its assumptions hold, with or without events.

Signal Latencies and Sending Events We assume that there is a global maximal latency $\Delta^{time}_{\mathsf{mlat}}$ between sending an event and receiving the event. In order to take this latency conservatively into account, the event's time gets adjusted: The time-limited promise of event $\mathcal{E} = (\varphi^{prom}, dueTime, duration)$ sent at time t, implies that the receiving component can rely on the promise encoded by $\mathcal{E}' = (\varphi^{prom}, dueTime, \max(0, duration - \Delta^{time}_{\mathsf{mlat}}))$ received at time t' with $t < t' \leq t + \Delta^{time}_{\mathsf{mlat}}$. Event $\mathcal{E}'$ has the same due time as $\mathcal{E}$, as the event may be received with (nearly) no latency. The validity duration in $\mathcal{E}'$ is shortened by the maximal latency, because the event may be received after $\Delta^{time}_{\mathsf{mlat}}$ the latest. Waiting for $dueTime$, then amounts to missing the actually promised start by $\Delta^{time}_{\mathsf{mlat}}$. We assume that events may get lost. Hence a component has to make sure that it is always prepared for the worst case, i.e. that no further events are received.

3 Components of the ADAS

In this section we describe the basic building blocks of ADAS system, that is the plant and the two control components the ADAS is made of, that is the VC and SC components. A component is an implementation accessed via an interface. The implementations of VC and SC in terms of hybrid automata are presented in Sect. 3.1. Their interface specifications are introduced in Sect. 3.2. The ADAS interface itself

is introduced in Sect. 6. There we also compositionally derive that objectives (o1) to (o3) hold for the composed ADAS.

3.1 Implementation

3.1.1 Plant

In Fig. 3 of Sect. 2 we have already presented the hybrid automaton modelling the plant. For convenience of the reader this is reproduced in Fig. 4.

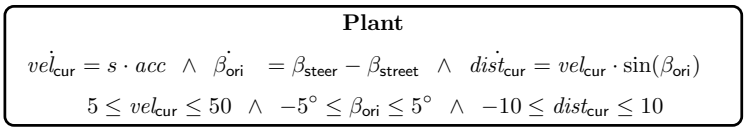


Figure 4: Hybrid automaton of the plant.

For now we consider only straight lanes (i.e. $\beta_{\mathsf{street}} = 0$), that is the mid of lane is fixed and we set the disturbance s to 1. Whereas the latter restriction can be easily dropped, the first is conceptual: In the following we will use that controllers stabilize to their certain equilibrium point. Having a dynamic road course, implies that there is no fixed equilibrium point. In control theory this is called reference tracking and a clever embedding in our interfaces needs further investigation.

3.1.2 Velocity Controller

The VC is composed of two subcomponents: one component regulates the velocity, the other handles received events. In the sequel we refer to the regulating component as VC_PI (PI velocity controller) and the other component is called VC_ER (event receiver VC).
In Sect. 2.1, we motivated that VC has to be subordinate to SC. In particular, VC has to aim for the user desired velocity vel_{des}, or if this is too fast, VC has to aim for vel_{comf} that is the maximal velocity still comfortable at the current steering, $\dot{\beta}_{\mathsf{ori}}$. Within VC, VC_ER will use the events to determine a set point vel_{goal} for VC_PI basically setting vel_{goal} to $\min(vel_{\mathsf{des}}, vel_{\mathsf{comf}})$. So vel_{goal} is the maximal velocity the car is allowed to drive when it respects comfort constraints and driver speed requests. Additionally VC_ER takes into account that the VC_PI may overshoot the new vel_{goal} on its way of stabilizing. Later on in this section, we will give more details on the VC_ER implementation. Next we derive a way to determine vel_{comf}.
For a given change of orientation $\dot{\beta}_{\mathsf{ori}}$ and a centrifugal force F_{comf} that a passenger still experiences as comfortable, we can derive the according velocity vel_{comf}

$$vel_{\mathsf{comf}} = \frac{F_{\mathsf{comf}}}{mass \cdot \dot{\beta}_{\mathsf{ori}}} \tag{1}$$

from $F_{\mathsf{cntrif}} = mass \cdot \omega^2 \cdot r$, $\omega = \dot{\beta}_{\mathsf{ori}}$ and $vel_{\mathsf{cur}} = \omega \cdot r$. That is, vel_{comf} is the maximal velocity still experienced as comfortable at steering $\dot{\beta}_{\mathsf{ori}}$. The implied relation between vel_{comf} and steering for a fixed maximal centrifugal force is illustrated in Fig. 5.

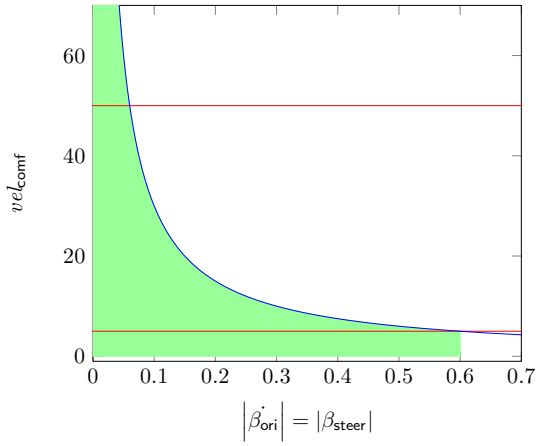


Figure 5: F_{comf} implies a relation for β_{steer} and vel_{comf} (blue line); global bounds on the velocity (red lines), allowed change of the orientation (green shape), $\frac{F_{\mathsf{comf}}}{mass} = 3\frac{\mathsf{m}}{\mathsf{s}^2}$.

A PI Velocity Controller The VC_PI (cf. Fig. 6) has three modes: **Norm**, **Accl**, **Decl**. In modes **Accl** and **Decl**, the, respectively, maximal tolerated acceleration and deceleration are applied.

Event Receiver The event receiver, VC_ER (cf. Fig. 7), runs in parallel to VC_PI. It receives events and sets the velocity setpoint vel_{goal} for VC_PI appropriately. VC is prepared to use events that promise $\dot{\beta}_{\mathsf{ori}} \in [-0.3, 0.3]$. So VC_ER receives events $\mathcal{E} = ([min, max], dueTime, duration)$ with $-0.3 \leq \mathcal{E}.min \leq \mathcal{E}.max \leq 0.3$.

Exploiting An Event. If no events are received, the VC has to guarantee that its velocity is comfortable at all assumed steering. It therefore makes the car drive at most with velocity vel_{safe}, where vel_{safe} is a velocity value small enough to guarantee comfortable centrifugal forces at all assumed steering activities. To guarantee, that VC is ready again for arbitrary steering when a promise $\mathcal{E}$ ends, vel_{goal} gets increased for a duration of $\mathcal{E}.duration - maxConvTime(vel_{\mathsf{cur}}, vel_{\mathsf{safe}})$ where $maxConvTime(vel_{\mathsf{cur}}, vel_{\mathsf{safe}})$ is a time sufficient for VC_PI to stably stay below vel_{safe}. Likewise, VC_ER ignores events with a duration $duration \leq maxConvTime(vel_{\mathsf{cur}}, vel_{\mathsf{safe}})$.

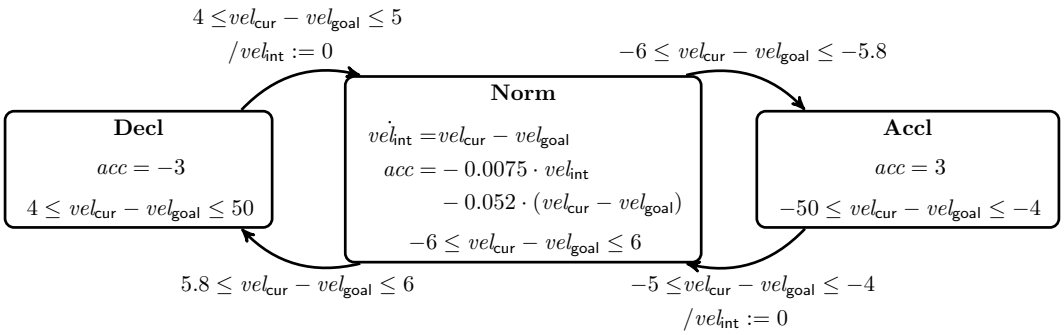


Figure 6: Hybrid Automaton Visualizing the VC_PI.

$\mathcal{E}.\varphi^{prom} \subseteq [-0.05, 0.05]$	$vel_{\mathsf{comf}} = vel_{\mathsf{crit}} = 50$
$\mathcal{E}.\varphi^{prom} \subseteq [-\frac{1}{15}, \frac{1}{15}]$	$vel_{\mathsf{comf}} = 45$
$\mathcal{E}.\varphi^{prom} \subseteq [-0.12, 0.12]$	$vel_{\mathsf{comf}} = 25$
$\mathcal{E}.\varphi^{prom} \subseteq [-0.2, 0.2]$	$vel_{\mathsf{comf}} = 15$
$\mathcal{E}.\varphi^{prom} \subseteq [-0.3, 0.3]$	$vel_{\mathsf{comf}} = vel_{\mathsf{safe}} = 10$

Table 1: Look up table for the comfortable velocity vel_{comf}.

$maxConvTime(vel_{\mathsf{cur}}, vel_{\mathsf{safe}})$ can also be specified via a lookup table listing for instance as safe velocities the minimal velocity, the maximal velocity and velocities that are chosen as reaction to events of SC (cf. Eq. 2, Tab. 1). For such pairs upper time bounds for convergence into an 1-region around vel_{goal} could be obtained using Lyapunov functions. This would yield safe time values, which may be minimized to make better use of events. An appropriate value of *maxConvTime* for a pair of $(vel_{\mathsf{cur}}, vel_{\mathsf{safe}})$ is then determined by choosing the safe best fit.

Property 1 *For every initial difference to the target velocity* $vel_{cur}' \in [-50, 50]$ *an upper bound maxConvTime on the time spent before finally reaching a* 1*-region around* vel_{goal} *exists.*

To obtain upper time bounds for convergence as required for Prop. 1 we use the Lyapunov functions to calculate an upper bound on the time spent outside a particular region. Such an upper bound is usually a very rough overapproximation and better bounds can be found by again combining safe boxes and model checking (cf. Sect. 5.1.2).

Property 1 implies that there exists a time *maxTimeToSafeVel* after which any trajectory from an initial state of VC will make vel_{cur} stay in a 1-region around a safe vel_{goal} according to Table 1.

For event $\mathcal{E}$, VC_ER sets vel_{goal} according to Eq. 1 to the maximal velocity that is comfortable for all announced steering actions, $\beta_{\mathsf{ori}} \in \mathcal{E}.\varphi^{prom}$ and additionally takes into account that VC_PI may overshoot the new vel_{goal} on its way to stabilizing on it. We refer to the maximal overshoot as the *peak overshoot*. The following equation summarizes how VC_ER determines the maximal velocity according to a given event $\mathcal{E}$:

$$velFor(\mathcal{E}) = \max\left(\frac{|F_{\mathsf{comf}}|}{|mass| \cdot \max(|\mathcal{E}.\varphi^{prom}|)} - peakOvershoot, vel_{\mathsf{goal}}^{\mathsf{min}}\right), \qquad (2)$$

where $\max(|\mathcal{E}.\varphi^{prom}|)$ is the maximum change of orientation according to $\mathcal{E}$ and $vel_{\mathsf{goal}}^{\mathsf{min}}$ is the least allowed velocity setpoint due to overshoots, i.e., any value in $[5 + peakOvershoot_{\mathsf{max}}, 10 - peakOvershoot_{\mathsf{min}}] = [8, 9]$. Based on *velFor*, $velFor(\mathcal{E}, vel_{\mathsf{des}}) = \min\{velFor(\mathcal{E}), vel_{\mathsf{des}}\}$ is defined to take into account the user's wishes vel_{des}, if applicable.

For the composition (cf. Sect. 6) we have implemented VC_ER (i.e. the *velFor*-function) using the lookup tables (cf. Table 1, Table 2). An received event implies that a velocity according to Table 1 is comfortable. The set point velocity then is adjusted taking into account the peak overshoot following Table 2.

Managing Events. Within a current validity duration new events can be received that may trigger themselves new setpoints and new validity durations. Such events are only exploited if their due time is reached before the current promise ends, so that contiguously promises can be exploited and the new promise is better than or equals the current promise. To simplify VC_ER, we assume a mail box component,

$\lvert vel_{\mathsf{cur}} - vel_{\mathsf{comf}} \rvert \geq 3$	$peakOvershoot = 3$
$\lvert vel_{\mathsf{cur}} - vel_{\mathsf{comf}} \rvert \in [2,3[$	$peakOvershoot = 2$
$\lvert vel_{\mathsf{cur}} - vel_{\mathsf{comf}} \rvert \in [0,2[$	$peakOvershoot = 1$

Table 2: Look up table for the peak overshoot *peakOvershoot*.

VC_MB, that preprocesses events. VC_MB sorts, stores and manages events and sends them to VC_ER in time. VC_MB updates the event's due time and duration as time progresses and in case event $\mathcal{E}_1$ is currently active allowing a high vel_{goal} and the next event $\mathcal{E}_2$ promises a lesser vel_{goal}, it adjusts due time and duration so that $\mathcal{E}_2$ is starts directly after $\mathcal{E}_1$ ends.

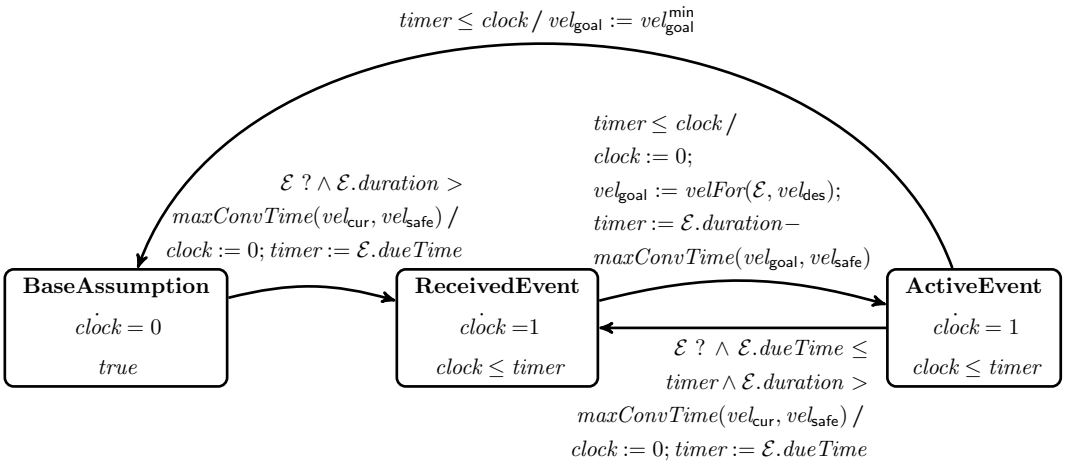


Figure 7: VC_ER adjusts the setpoint of the VC_PI.

3.1.3 Steering Controller

The steering controller SC consists of two subcomponents: a component for setting the actuator and a component for event generation.

Steering Control In the plant the change of the distance to the mid of the street evolves according to $\dot{dist} = vel_{\mathsf{cur}} \cdot \sin(\beta_{\mathsf{ori}})$. SC basically applies, $c_1 \cdot \beta_{\mathsf{ori}} + c_2 \cdot dist$, in all its modes but with different parameters. This PI-like control law exploits the distance as an integrator. SC's area of operation is partitioned into three regions, which correspond to mode names: **Keep**, **Left**, **Right** (cf. Fig. 8). Both, **Left** and **Right**, split up into three modes that represent distance categories (**Close**, **Mid**, **Far**). In each such mode the parameters c_1, c_2 are tweaked depending on the distance category. The implementation presented in Fig. 8 can be seen as a rough approximation of a more complex controller that performs a continuous adaptation of c_1, c_2 to the distance.

Event Generation SC has to generate events that conservatively overapproximate its steering for the announced window of time. As an event communicates guaranteed improvements, its due time may be greater or equals the actual start time of the

promised steering improvement, its duration may be smaller, but its end time has to be before the improvement ends.
In the following we sketch an approach to event generation in our setting. The steering angle will converge to the equilibrium, which exists since we assume a straight road. We can use this knowledge to derive an upper-bound for possible steering at a given state, i.e. orientation and distance the mid. But in case we deal with curved roads, event generation is a little bit more complicated, as the road's curves amount to disturbances that may perturb the SC on its way of stabilizing. In this case the SC may overapproximate the necessary steering by inspection of the road course within a horizon as visualized in Fig. 9 and described in the following. SC computes its future behavior within the perceivable horizon. Suppose it encounters a situation in which it will need to do only little steering. As SC needs to ensure that after $\mathcal{E}.dueTime$ the promise holds for $\mathcal{E}.duration$, it safely overapproximates the time the car needs to pass $horizon_{\mathsf{due}}$. This time becomes $dueTime$ of the generated event. All steering possibly done after the car has driven at least $dueTime$ and until the car reaches the end of $horizon$ constitute the interval φ^{prom}. $duration$ is set to the time the car needs at least to reach the end of $horizon$, since then the promise cannot be guaranteed any longer. So a sequel of events may be generated as the car drives along. In order to allow VC to make use of events, SC needs to generate events with a sufficiently long duration. Hence SC has to scan horizons of an appropriate extend. Also the degree of freedom implied by events increases if the events overlap by at least $maxConvTime\ (vel_{\mathsf{cur}}, vel_{\mathsf{safe}})$, so that instead of using the default setpoint vel_{safe} the event receiver can determine a new setpoint in time to continuously exploit the knowledge about the better (than worst case) future.

3.1.4 Discussion: Realistic Controllers or Toy Examples?

This section's aim is to discuss the relevancy and representativeness of our case study. First of all, we believe that industrial examples are of larger size and higher complexity than this case study. Nevertheless, we think that the controllers already expose relevant features and we examine interesting properties in our case study. In particular, we demonstrate how compositional reasoning can be applied with a focus on industrial sized models.

Secondly, the controllers' behavior do not comply with industrial requirements as e.g. the SC's steering is not smooth: the steering wheel (β_{steer}) –not the car's orientation– is changing abruptly when the controller changes its mode. Although such behavior is by far the behavior of an industrial controller, the controller can be straightforwardly transformed into an industrial controller. Therefore, we suggest to add additional modes that smoothly connect the modes' individual vector fields by continuously raising or lowering β_{steer} via its derivative. Furthermore, we would like a minimal dwell time for each mode, since super-fast switching is unrealistic. Hence we suggest to add some hysteresis via overlapping invariants. These steps involve parameter tuning.

Parameter tuning is a time-consuming task even for experienced engineers and a task that is unfortunately scarcely tool-supported. We consider parameter analysis and tuning as a challenging research direction on its own and do not regard it as a focus of our current research. Once good parameters are found the implication for our case study are quiet simple: The model's size will be increased. There

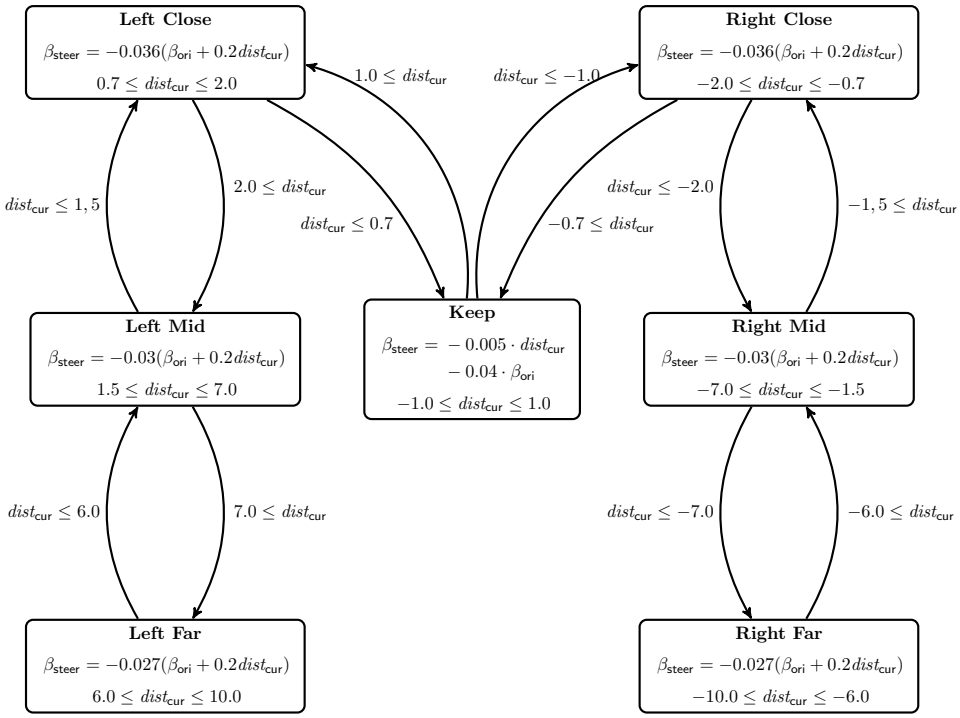


Figure 8: Hybrid automaton of SC.

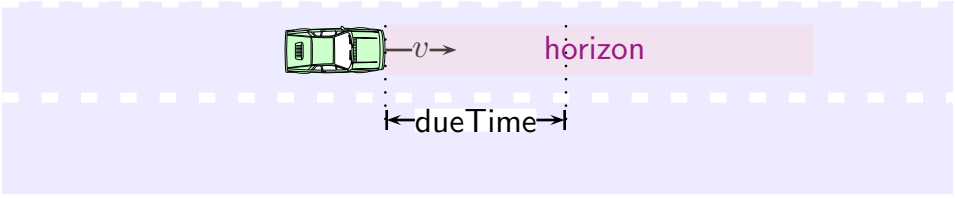


Figure 9: Generation of events encapsulating the steering actions.

will be more variables, more modes, and more transitions. As a result proving properties of the individual components might be harder and we believe that further decomposition can be used to compensate this fact. This does not prevent us from using our framework but is rather an argument to use a compositional framework to be able to deal component-wise with the complex, industrial sized controllers.

3.2 Interfaces

Our components will be accessed via their declared interfaces. In the following, we specify interfaces for our components in accordance with the framework we are developing and which we already briefly sketched in Sect. 2.2. Later, in Sect. 5, we will prove that the interfaces specified here get implemented by the automata specified above.

Within our framework a component declares base assumptions and base guarantees. For dynamic coordination components may exchange information via events. The sending component declares an (possibly strengthened) event guarantee and a (possibly relaxed) event assumption. A receiving component declares its reaction to an incoming event making the assumption that the event is truthful.

So for the time interval in which an event promise $\mathcal{E}.\varphi^{prom}$ holds, the receiving component assumes that its base assumption and the event promise hold. The receiving component has to deliver the guarantees declared as reaction to the event. The sending component assumes that its event assumption hold. Note that the event assumption may be weaker than the base assumption, so that the sender's event assumptions are satisfied even if the event gets lost or are ignored. So in any case the sending component is bound to deliver its event guarantees.

3.2.1 Interface of VC

In the sequel the interface of VC is described and summarized in Tab. 3. Base assumptions and base guarantees define assumptions and guarantees when there is no active event. The VC receives events of SC. It then may relax its guarantees and may strengthen its assumptions as appropriate for the received event. VC declares at its interface what it guarantees when it makes use of a received event (cf. Tab. 4). That way compatibility of components can be derived by checking compatibility of interface specifications. When SC tells VC via an event that steering is gentle, VC may increase its velocity.

Entry Condition VC can only be started when the car is driving slow, $vel_{\mathsf{cur}} \in [5, 10]$. This allows the VC to be ready for all steering actions.

Base Assumptions VC assumes that steering is restricted to $\beta_{\mathsf{steer}} \in [-0.3, 0.3]$. VC also allows only a limited range for user wishes on the velocity: The user may only choose $vel_{\mathsf{des}} \in \{10, \ldots, 45\}$.

Base Guarantees Given the above assumptions hold, VC has to deliver its guarantees, listed in the following. VC guarantees that the car's velocity vel_{cur} converges to the user chosen velocity, if comfortable. To be more precise, it approaches this velocity setpoint arbitrarily close, if the setpoint stays constant and given enough

time. Also annotated in its interface is that the VC keeps the car's velocity within certain bounds, i.e., always $vel_{\mathsf{cur}} \in [5, 10]$, if there is no active event.

Name	Property
VC.BA1	Initiallay holds $vel_{\mathsf{cur}} \in [5, 10]$.
VC.BA2	It always holds $\beta_{\mathsf{steer}} \in [-0.3, 0.3]$.
VC.BA3	It always holds $vel_{\mathsf{des}} \in \{10, \ldots, 45\}$.
VC.BG1	vel_{cur} converges to vel_{des} as close as possible under the base assumption.
VC.BG2	When there is no active event, it always holds $vel_{\mathsf{cur}} \in [5, 10]$.

Table 3: Base assumptions and guarantees declared at the interface of the VC.

Event Assumptions VC assumes received events are truthful. This means as long as an event promise $\mathcal{E}.\varphi^{prom}$ is active, VC assumes that β_{steer} is in $[\min(\mathcal{E}.\varphi^{prom}), \max(\mathcal{E}.\varphi^{prom})]$. As before VC assumes that the driver only chooses $vel_{\mathsf{des}} \in \{10, \ldots, 45\}$.

Event Guarantees When there is an active event $\mathcal{E}$, it always holds $vel_{\mathsf{cur}} \in [5, vel_{\mathsf{comf}}(\mathcal{E})]$, where $vel_{\mathsf{comf}}(\mathcal{E})$ is defined as $\frac{F_{\mathsf{comf}}}{mass \cdot \max(|\mathcal{E}.\varphi^{prom}|)}$ according to Eq. 1. With other words, VC promises to increase the velocity at most to a speed that is comfortable. The implied velocity range of an event is given in Tab. 5, derived from Tab. 1. Also VC continues to guarantee that the speed converges to the setpoint speed.

Name	Property
VC.EG1	vel_{cur} converges to vel_{des} as close as possible under the event assumptions.
VC.EG2	When there is an active event $\mathcal{E}$, then it always holds $vel_{\mathsf{cur}} \in [5, vel_{\mathsf{comf}}(\mathcal{E})]$.

Table 4: Event guarantees declared at the interface of the VC.

Event	Guarantee
$\mathcal{E}.\varphi^{prom} \subseteq [-0.05, 0.05]$	$vel_{\mathsf{cur}} \in [5, 50]$
$\mathcal{E}.\varphi^{prom} \subseteq [-\frac{1}{15}, \frac{1}{15}]$	$vel_{\mathsf{cur}} \in [5, 45]$
$\mathcal{E}.\varphi^{prom} \subseteq [-0.12, 0.12]$	$vel_{\mathsf{cur}} \in [5, 25]$
$\mathcal{E}.\varphi^{prom} \subseteq [-0.2, 0.2]$	$vel_{\mathsf{cur}} \in [5, 15]$
$\mathcal{E}.\varphi^{prom} \subseteq [-0.3, 0.3]$	$vel_{\mathsf{cur}} \in [5, 10]$

Table 5: Guarantees on velocities for incoming events at the interface of VC.

3.2.2 Interface of SC

In the sequel, the interface of SC is described and summarized in Tab. 6. We assume that SC sends justified events only and that there is a minimal separation time between two successive events, which we denote as $\Delta^{time}_{\mathsf{sep}}$. For now, we do not restrict event generation any further.

Entry Condition Basically the SC is allowed to start from all states for which it converges to $(\beta_{\mathsf{ori}}, dist_{\mathsf{cur}}) = (0, 0)$ and stays within its lane at the assumed velocity range. The entry condition here is a simple underapproximation of this set.

Base Assumptions As base assumption SC requires that the car's velocity is always low, that is vel_{cur} is in $[5, 10]$, so that SC has the freedom to choose its steering as necessary to stay on the road. SC will dynamically tell VC via events when it can tolerate a higher speed.

Base Guarantees SC promises to make the car drive towards the mid of its lane, i.e. given enough time the car approaches (β_{ori},$dist_{\mathsf{cur}}$)= (0,0) arbitrarily close. Also SC guarantees that the car never leaves the lane: it always holds that ($dist_{\mathsf{cur}} \in [-10, 10]$). Further the car's orientation stays within certain bounds, $\beta_{\mathsf{ori}} \in [-5°, 5°]$ and SC promises that its steering is restricted to satisfy $\beta_{\mathsf{steer}} \in [-0.2, 0.2]$.

Name	Property
SC.BA1	$dist_{\mathsf{cur}} \in [-9, 0] \wedge \beta_{\mathsf{ori}} \in [0°, 2.6°] \vee dist_{\mathsf{cur}} \in [0, 9] \wedge \beta_{\mathsf{ori}} \in [-2.6°, 0°]$
SC.BA2	It always holds that $vel_{\mathsf{cur}} \in [5, 10]$
SC.BG1	(β_{ori},$dist_{\mathsf{cur}}$) converges to (0,0).
SC.BG2	It always holds that $dist_{\mathsf{cur}} \in [-10, 10]$.
SC.BG3	It always holds that $\beta_{\mathsf{ori}} \in [-5°, 5°]$.
SC.BG4	It always holds that $\beta_{\mathsf{steer}} \in [-0.2, 0.2]$.

Table 6: Base assumptions and guarantees declared at the interface of the SC.

Event Assumptions and Event Guarantees SC basically assumes that the velocity remains comfortable that is the velocity may increase up to the maximal comfortable speed. We specify SC's event assumptions analogously to VC guarantees. SC strengthens via an event $\mathcal{E}$ only its guarantees wrt its steering, i.e. $\beta_{\mathsf{steer}} \in [\min(\mathcal{E}.\varphi^{prom}), \max(\mathcal{E}.\varphi^{prom})]$, otherwise it continues to satisfy (merely) its base guarantees (cf. Tab. 7). In Tab. 8, events of SC are declared along with the respective event assumptions, which weaken its base assumption while the event is active.

Name	Property
SC.EA1	When there is an active event $\mathcal{E}$, then it always holds $vel_{\mathsf{cur}} \in [5, velFor(\mathcal{E})]$.
SC.EG1	(β_{ori},$dist_{\mathsf{cur}}$) converges to (0,0).
SC.EG2	It always holds that $dist_{\mathsf{cur}} \in [-10, 10]$.
SC.EG3	It always holds that $\beta_{\mathsf{ori}} \in [-5°, 5°]$.
SC.EG4	It always holds that $\beta_{\mathsf{steer}} \in [\min(\mathcal{E}.\varphi^{prom}), \max(\mathcal{E}.\varphi^{prom})]$.

Table 7: Event assumptions and guarantees declared at the interface of the SC.

Event	Assumption
$\mathcal{E}.\varphi^{prom} \subseteq [-0.05, 0.05]$	$vel_{\mathsf{cur}} \in [5, 50]$
$\mathcal{E}.\varphi^{prom} \subseteq [-\frac{1}{15}, \frac{1}{15}]$	$vel_{\mathsf{cur}} \in [5, 45]$
$\mathcal{E}.\varphi^{prom} \subseteq [-0.12, 0.12]$	$vel_{\mathsf{cur}} \in [5, 25]$
$\mathcal{E}.\varphi^{prom} \subseteq [-0.2, 0.2]$	$vel_{\mathsf{cur}} \in [5, 15]$

Table 8: Declared events and respective assumptions on velocities at SC's interface.

4 Applied Tools

Before we actually show in Sect. 5 that the interfaces as given in Sect. 3.2 are implemented by the automata of Sect. 3.1.2, 3.1.3, we briefly introduce the tools applied to establish interface implementation. STABHYLI is a tool for certifying stability using Lyapunov functions and SOAPBOX is a symbolic model checking tool based on a new representation formalism for symbolic states.

4.1 Stabhyli

In this section we briefly describe the tool STABHYLI [12]. STABHYLI can be used to obtain Lyapunov functions, that certify stability for hybrid systems. It handles non-linear systems whose behavior is expressible in forms of polynomials. STABHYLI can be used to obtain a single common Lyapunov function, a piecewise Lyapunov function as well as to perform the (de-)compositional proof schemes presented in [14, 5]. These features are fully automatized and combined with pre- and postprocessing steps that simplify the design and counteract numerical problems. Furthermore, in case stability cannot be proven, STABHYLI returns a hint to the user.

In the sequel we briefly sketch the theoretical basis of Stabhyli and introduce some fundamental notions. A hybrid automaton is defined as follows

Definition 1 *A **Hybrid Automaton** is a quintuple*

$$H = (Var, \mathbb{M}, R^{dscr}, R^{cnt}, \Theta^{inv}, \varphi^{init}) \text{ where}$$

- *Var is a finite set of* variables *and* $\mathcal{S} = \mathbb{R}^{|Var|}$ *is the corresponding* continuous state space*,*
- $\mathbb{M}$ *is a finite set of* modes*,*
- R^{dscr} *is a finite set of* transitions (m_1, G, U, m_2) *where*
  - $m_1, m_2 \in \mathbb{M}$ *are the* source and target mode *of the transition, respectively,*
  - $G \subseteq \mathcal{S}$ *is a* guard *which restricts the valuations of the variables for which this transition can be taken,*
  - $U : \mathcal{S} \to \mathcal{S}$ *is the* update function *which specifies discrete changes of variable valuations,*
- $R^{cnt} : \mathbb{M} \to [\mathcal{S} \to \mathcal{P}(\mathcal{S})]$ *is the* flow function *which assigns a* flow *to every mode. A flow* $f \subseteq \mathcal{S} \to \mathcal{P}(\mathcal{S})$ *in turn assigns a closed subset of* $\mathcal{S}$ *to each* $x \in \mathcal{S}$*, which can be seen as the right hand side of a differential inclusion* $\dot{x} \in f(x)$*,*
- $\Theta^{inv} : \mathbb{M} \to \mathcal{S}$ *is the* invariant function *which assigns a closed subset of the continuous state space to each mode* $m \in \mathbb{M}$*, and therefore restricts valuations of the variables for which this mode can be active, and*
- $\varphi^{init} \subseteq \mathbb{M} \times \mathcal{S}$ *is a set of initial states.*

A trajectory *of H is an infinite solution in form of a function* $(x(t), m(t))$ *over time where* $x(\cdot)$ *describes the evolution of the continuous variables and* $m(\cdot)$ *the corresponding evolution of the modes.*

Intuitively, stability is a property expressing that all trajectories of the system eventually reach an equilibrium point of the sub-state space and stay in that point forever given the absence of errors. For technical reasons the equilibrium point is

usually assumed to be the origin of the continuous state space, i. e. **0**. This is not a restriction, since a system can always be shifted such that the equilibrium is **0** via a coordinate transformation. In the sequel, we focus on *asymptotic stability* which does not require the equilibrium point to be reached in finite time but only requires every trajectory to "continuously approach" it (in contrast to *exponential stability* where additionally the existence of an exponential rate of convergence is required).

Definition 2 ***Global Asymptotic Stability with Respect to a Subset of Variables [13].*** *Let $H = (Var, \mathbb{M}, R^{dscr}, R^{cnt}, \Theta^{inv})$ be a hybrid automaton, and let $Var' \subseteq Var$ be the set of variables that are required to converge to the equilibrium point* **0**. *A continuous-time dynamic system H is called* globally stable (GS) with respect to Var' *if for all functions $x_{\downarrow Var'}(\cdot)$,*

$$\forall \epsilon > 0 : \exists \delta > 0 : \forall t \geq 0 : ||x(0)|| < \delta \Rightarrow \left|\left|x_{\downarrow Var'}(t)\right|\right| < \epsilon.$$

H is called globally attractive (GA) with respect to Var' *if for all functions $x_{\downarrow Var'}(\cdot)$,*

$$\lim_{t\to\infty} x_{\downarrow Var'}(t) = \mathbf{0},\ i.e., \forall \epsilon > 0 : \exists t_0 \geq 0 : \forall t > t_0 : \left|\left|x_{\downarrow Var'}(t)\right|\right| < \epsilon,$$

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

In the following, we denote with $x_{\downarrow Var'} \in \mathbb{R}^{|Var'|}$ the sub-vector of vector $x \in \mathbb{R}^{|Var|}$ that represents the according valuation of $Var' \subseteq Var$. Also, in the following theorem, $\left\langle \frac{dV(x)}{dx} \middle| f(x) \right\rangle$ denotes the inner product between the gradient of a Lyapunov function V and a flow function $f(x)$.

Theorem 1 Discontinuous Lyapunov Functions for a subset of variables [13]. Let $H = (Var, \mathbb{M}, R^{\mathsf{dscr}}, R^{\mathsf{cnt}}, \Theta^{\mathsf{inv}})$ be a hybrid automaton and let $Var' \subseteq Var$ be the set of variables that are required to converge. If for each $m \in \mathbb{M}$, there exists a set of variables Var_m with $Var' \subseteq Var_m \subseteq Var$ and a continuously differentiable function $V_m : \mathcal{S} \to \mathbb{R}$ such that

1. for each $m \in \mathbb{M}$, there exist two class K^∞ functions α and β such that
$$\forall x \in \Theta^{\mathsf{inv}}(m) : \alpha(||x_{\downarrow Var_m}||) \preceq V_m(x) \preceq \beta(||x_{\downarrow Var_m}||),$$
2. for each $m \in \mathbb{M}$, there exists a class K^∞ function γ such that
$$\forall x \in \Theta^{\mathsf{inv}}(m) : \dot{V}_m(x) \preceq -\gamma(||x_{\downarrow Var_m}||)$$
for each $\dot{V}_m(x) \in \left\{ \left\langle \frac{dV_m(x)}{dx} \middle| f(x) \right\rangle \mid f(x) \in R^{\mathsf{cnt}}(m) \right\}$,
3. for each $(m_1, G, U, m_2) \in R^{\mathsf{dscr}}$,
$$\forall x \in G : V_{m_2}(U(x)) \preceq V_{m_1}(x),$$

then H is globally asymptotically stable with respect to Var' and V_m is called a *Local Lyapunov Function (LLF)* of m.

STABHYLI generates constraint systems using the so called *sums-of-squares* method [16] and the S-Procedure [3] to generate linear matrix inequalities which can then be solved by a *semi-definite program (SDP)*. To do this the numerical solver CSDP [2] is in charge. If such a constraint system is feasible then each solution represents a valid Lyapunov function.

4.2 SoapBox

This section briefly describes SoapBox, a tool for reachability analysis of hybrid systems, which is implemented in Matlab. SoapBox can handle hybrid systems with continuous dynamics described by linear differential inclusions and arbitrary linear maps for discrete updates. The invariants, guards, and sets of reachable states are given as convex polyhedra. The reachability algorithm of SoapBox is based on a novel representation class for convex polyhedra, the symbolic orthogonal projections (sops). On sops various geometric operations, including convex hulls, Minkowski sums, linear maps, and intersections, can be performed efficiently and exactly. The capability to represent intersections of convex polyhedra exactly is superior to support function-based approaches like the LGG-algorithm (Le Guernic and Girard [10]), which is implemented in SpaceEx [9].

4.2.1 Symbolic Orthogonal Projections

A *symbolic orthogonal projection* (*sop*) $\mathbf{P} = \mathbf{P}(A, L, \mathbf{a})$ encodes the orthogonal projection of a higher dimensional $\mathcal{H}$-polyhedron $\mathbf{P}((A\ L), \mathbf{a}) = \left\{ \binom{\mathbf{x}}{\mathbf{z}} \mid A\mathbf{x} + L\mathbf{z} \leq \mathbf{a} \right\}$ onto the vector space of dimension d:

$$\mathbf{P}(A, L, \mathbf{a}) = \left\{ \mathbf{x} \in \mathbb{K}^d \mid \exists \mathbf{z} \in \mathbb{K}^k : A\mathbf{x} + L\mathbf{z} \leq \mathbf{a} \right\}.$$

Obviously, the following properties hold for sops: a sop $\mathbf{P}(A, L, \mathbf{a})$ is empty if and only if its preimage $\mathbf{P}((A\ L), \mathbf{a})$ is empty; any $\mathcal{H}$-polyhedron $\mathbf{P} = \mathbf{P}(A, \mathbf{a}) \in \mathbb{K}^d$ can be represented by the sop $\mathbf{P}(A, \varnothing, \mathbf{a})$, where $\varnothing$ denotes the empty matrix; and the support function $h_{\mathbf{P}}(\mathbf{n})$ is given by the optimal value of the linear program

$$\text{maximize } \mathbf{n}^T\mathbf{x} \text{ subject to } A\mathbf{x} + L\mathbf{z} \leq \mathbf{a}.$$

The representation of polyhedra as orthogonal projections of higher dimensional polyhedra provides the freedom to introduce existentially quantified variables. For example, the Minkowski sum of two polyhedra $\mathbf{P}_1 = \mathbf{P}(A_1, \mathbf{a}_1)$ and $\mathbf{P}_2 = \mathbf{P}(A_2, \mathbf{a}_2)$ is defined as the set

$$\{\mathbf{z} \mid \exists \mathbf{x}, \mathbf{y} : A_1\mathbf{x} \leq \mathbf{a}_1, A_2\mathbf{y} \leq \mathbf{a}_2, \mathbf{z} = \mathbf{x} + \mathbf{y}\}.$$

Now, the Minkowski sum can be rewritten to

$$\begin{aligned} &\left\{\mathbf{z} \mid \exists \mathbf{y}' : A_1(\mathbf{z} + \mathbf{y}') \leq \mathbf{a}_1, -A_2\mathbf{y}' \leq \mathbf{a}_2\right\} \\ &= \mathbf{P}\left(\begin{pmatrix} A_1 \\ \mathrm{O} \end{pmatrix}, \begin{pmatrix} A_1 \\ -A_2 \end{pmatrix}, \begin{pmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \end{pmatrix}\right). \end{aligned}$$

By introduction of certain existentially quantified variables we obtain exact and efficient sop-based representations of the closed convex hulls, Minkowski sums, intersections and arbitrary linear transformations of polyhedra. In fact, most of these operations are realized as block matrices over the original representation matrices.

Overapproximation of Sops

Due to the introduction of additional variables the size of the representation matrices of the assembled sops grows monotonically. Hence, the evaluation of assembled sops by means of linear programming gets increasingly harder. The overapproximation of a sop by an $\mathcal{H}$-polyhedron with a fixed representation matrix (*template polyhedron*) helps to shrink the size but might induce a substantial loss of the facial structure of the sop. To recover at least some of the facial structure we use a post-processing technique called ray shooting: Let $\mathbf{P} = \mathbf{P}(A, L, \mathbf{a})$ be a non-empty sop in $\mathbb{K}^d$ which contains the origin $\mathbf{0}$ as a relative interior point and $\mathbf{r}$ be the direction of some ray $\lambda\mathbf{r}$ in $\mathbb{K}^d$, $\lambda \geq 0$. If $\mathbf{P}$ is neither unbounded nor flat in direction $\mathbf{r}$, i. e. a maximal $\lambda_0 > 0$ with $\lambda_0\mathbf{r} \in \mathbf{P}$ exists, then there is a linear program which provides a normal vector $\mathbf{n}$ of a supporting half-space $\mathbf{H} = \mathbf{H}(\mathbf{n}, 1)$ of $\mathbf{P}$ containing the boundary point $\lambda_0\mathbf{r}$.

4.2.2 Reachability Analysis Using Sops

Internally, SoapBox uses *symbolic states* to represent the reachable sets of the hybrid system. A *symbolic state* is a pair $(\mathbf{P}, m)$ of a polyhedron $\mathbf{P} \subseteq \mathcal{S}$ and a mode $m \in \mathbb{M}$. The reachable states from a symbolic state $(\mathbf{P}, m)$ are exactly those states which are reached by a trajectory emanating from $(\mathbf{P}, m)$.

Given a hybrid automaton H and the sets *Init*, *Safe*, and *Unsafe* as disjunction of finitely many symbolic states, the goal of SoapBox is to compute all reachable states from the initial states in *Init* until the trajectory enters *Safe*, touches *Unsafe*, or leaves the mode invariant $\Theta^{\mathsf{inv}}(m)$.

Let $(\mathbf{P}, m)$ be a symbolic state. Then the reachable states of $(\mathbf{P}, m)$ within the mode m are described by the linear system

$$\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + \mathbf{E} \in R^{\mathsf{cnt}},\ \mathbf{x}(0) \in \mathbf{P},\ \mathbf{x} \in \Theta^{\mathsf{inv}}(m),$$

where $\mathbf{E}$ is a convex polytope modelling the bounded input. A discrete transition $(m, \mathbf{G}, \mathbf{U}, m') \in R^{\mathsf{dscr}}$ can be taken for all solutions $\mathbf{x}(t)$ which satisfy the guard $\mathbf{G}$, i. e. $\mathbf{x}(t) \in \mathbf{G}$.

SoapBox computes an overapproximation of the reachable states in a step-wise manner. Let δ be a time-step parameter. First, a polyhedral overapproximation of the reachable states within the time-interval $[0, \delta]$ is computed using a safe first-order approximation of the continuous dynamics (initial bloating). The initial bloating procedure yields two sops, a bloated set $\mathbf{R}_0$ which contains all reachable states within the time-interval $[0, \delta]$ and a sop $\mathbf{V}$ which accounts for the influences of the bounded input $\mathbf{E}$ within the time-interval $[0, \delta]$. Now, the reachable states within the time-interval $[k\delta, (k+1)\delta]$ are computed according to the recursive formula

$$\mathbf{R}_{k+1} = (\mathrm{e}^{\delta A}(\mathbf{R}_k) + \mathbf{V}) \cap \Theta^{\mathsf{inv}}(m).$$

Then all intersections with possible guards are computed until the current flow segment leaves the invariant or completely lies in a symbolic *Safe* state. In the next step, SoapBox checks whether one of the resulting intersections is an intersection with one of the symbolic *Unsafe* states. In this case SoapBox stops the reachability analysis with an appropriate output. Otherwise, the discrete post-images of the guard intersections are computed and added to the initial states.

In fact, SoapBox combines the sop-based reachability computation with the reachability algorithm of Le Guernic and Girard (LGG-algorithm) [10], which computes a coarser overapproximation of the reachable states. While the sop-based part of the algorithm has the capability to provide an exact representation, the LGG-algorithm yields fast overapproximations in terms of template polyhedra, e. g. it almost completely ignores the influence of the invariant. Those operations of the reachability analysis which involve linear programming, e. g. the subset check for safe sets or the intersection tests for guards, are performed in the LGG-part. After a given number of computation steps, SoapBox uses ray shooting to improve the overapproximation computed in the LGG-part by adding some additional information of the facial structure gained by the sop part. Then the current sop is replaced by this overapproximation. This procedure efficiently shrinks the representation size of the sops, but leads to a loss of accuracy. Anyhow, since this method may only improve the state sets generated in LGG-part, SoapBox still archives tighter overapproximations than the pure LGG-algorithm.

We should note that the current implementation of SoapBox internally uses floating point numbers. While most sop-based operations consist of block matrix operations and hence do not introduce additional numerical issues, there are two main sources of numerical issues regarding the use of floating point numbers: the usage of the matrix exponential $e^{\delta A}$ and the usage of the inexact linear solver Gurobi. Anyhow, SoapBox has only a few built-in functions, like scaling, to deal with potential numerical issues. Instead, the numerical values are passed over to the linear solver. Experiments with GLPK and Gurobi showed that Gurobi copes quite well with potential numerical issues. The numerical issues stemming from the usage of an inexact solver can be fixed by using exact arithmetic. Theoretically more challenging, is a numerical safe evaluation of the matrix exponential, which also SpaceEx has to face. This issue is regarded as a promising future research direction.

Handling Strict Inequalities

In order to handle linearizations of dynamics with non-linear differential inclusions properly, we also allow guards with strict inequalities. SoapBox handles strict inequalities in the following way: A transition involving a strict guard $\mathbf{n}^T\mathbf{x} < c$ is disabled as long as the current flow segment $\mathbf{P}$ does not contain a witness point $\mathbf{x}$ which satisfies the strict guard. Otherwise, the transition is enabled and we treat the strict inequality as a non-strict inequality. Hence, in this case, we compute a closed overapproximation of the actual intersection. The special treatment of strict inequalities avoids zeno-behavior which would occur if we handled strict inequalities as non-strict inequalities: Let x be a variable of a hybrid automaton with the two modes m_1 and m_2. Further, let $(m_1, x > 5, id, m_2)$ and $(m_2, x < 5, id, m_1)$ be two discrete transitions. Treating the strict inequalities as non-strict inequalities would allow zeno-behavior for $x = 5$. With our treatment of the strict inequalities, the transition from m_1 to m_2 is enabled for any set (m_1, X) containing at least one element $\mathbf{x} > 5$. The post-image of (m_1, X) is $(m_2, X \cap \{x \mid x \geq 5\})$ and does not contain a witness for the guard $x < 5$. Hence, an instantaneous transition back from m_2 to m_1 is disabled and zeno-behavior is avoided. For such cases this guarantees progress in the reach set computation.

5 Proving Interface Implementations for SC and VC

This section illustrates the proof steps to establish that the hybrid automata as presented in Sect. 3.1 implement the interface specifications as given in Sect. 3.2.

We assume that the maximal comfortable centrifugal force is $\frac{F_{\mathsf{comf}}}{mass} = \frac{240}{80} = 3.$ and the disturbance s is 1. We can easily drop the assumption on s and replace the differential equation for vel_{cur} with a differential inclusion.

As we did not formally specify an event generator for SC, we only assume that such a component sends events that can be mapped using Table 8 and that the sending of events is separated by at least $\Delta^{time}_{\mathsf{sep}}$.

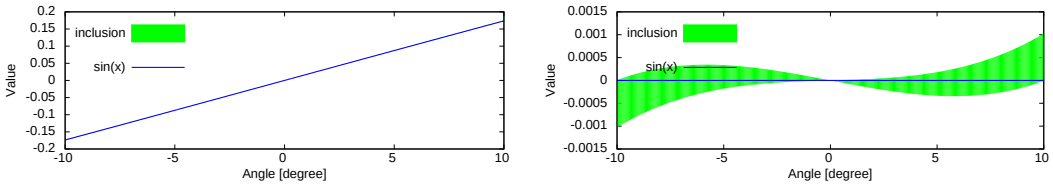
Basically, for successful verification we had to

- bridge the model-tool gap,
- simplify the model,
- simplify the proof goals and
- apply the tools.

5.1 Proving Interface Implementation for SC

5.1.1 Proving SC.BG1

First we showed that SC converges to $(\beta_{\mathsf{ori}}, dist) = (0, 0)$ using Stabhyli. In order to apply Stabhyli we used a simple overapproximation of the sine (cf. Fig. 10) and Stabhyli was able to certify convergence for the overapproximated system. We chose as template for the Lyapunov function a polynomial of degree two and Stabhyli generated the Lyapunov functions presented in Tab. 9.



(a) Actual Value (b) Absolute Error

Figure 10: Simple Linear Enclosing of the Sine. The line is the value of the sine and the shape the possible values of the inclusion.

$$
\begin{aligned}
V_{\textbf{Left Far}} &: +1.61995\beta_{\mathsf{ori}}\, dist_{\mathsf{cur}} + 0.231786 dist^2_{\mathsf{cur}} + 31.4842\beta^2_{\mathsf{ori}} - 0.44218 dist_{\mathsf{cur}} + 0.054299\beta_{\mathsf{ori}} + 4.72874\\
V_{\textbf{Left Mid}} &: +1.62776\beta_{\mathsf{ori}}\, dist_{\mathsf{cur}} + 0.305039 dist^2_{\mathsf{cur}} + 31.4844\beta^2_{\mathsf{ori}} - 0.281639 dist_{\mathsf{cur}} + 0.00300244\beta_{\mathsf{ori}} + 0.525766\\
V_{\textbf{Left Close}} &: +1.62936\beta_{\mathsf{ori}}\, dist_{\mathsf{cur}} + 0.414363 dist^2_{\mathsf{cur}} + 31.4845\beta^2_{\mathsf{ori}} - 0.275467 dist_{\mathsf{cur}} + 0.000106035\beta_{\mathsf{ori}} + 0.156752\\
V_{\textbf{Keep}} &: +1.62948\beta_{\mathsf{ori}}\, dist_{\mathsf{cur}} + 0.307842 dist^2_{\mathsf{cur}} + 31.4845\beta^2_{\mathsf{ori}}\\
V_{\textbf{Right Close}} &: +1.62948\beta_{\mathsf{ori}}\, dist_{\mathsf{cur}} + 0.414358 dist^2_{\mathsf{cur}} + 31.4845\beta^2_{\mathsf{ori}} + 0.275448 dist_{\mathsf{cur}} + 0.156739\\
V_{\textbf{Right Mid}} &: +1.62788\beta_{\mathsf{ori}}\, dist_{\mathsf{cur}} + 0.305033 dist^2_{\mathsf{cur}} + 31.4844\beta^2_{\mathsf{ori}} + 0.281608 dist_{\mathsf{cur}} - 0.00289793\beta_{\mathsf{ori}} + 0.52573\\
V_{\textbf{Right Far}} &: +1.62006\beta_{\mathsf{ori}}\, dist_{\mathsf{cur}} + 0.231782 dist^2_{\mathsf{cur}} + 31.4842\beta^2_{\mathsf{ori}} + 0.442058 dist_{\mathsf{cur}} - 0.0542052\beta_{\mathsf{ori}} + 4.72805
\end{aligned}
$$

Table 9: Automatically derived Lyapunov functions for the SC (rounded).

We could greatly benefit from having established convergence first in the following proof steps. In particular we used the Lyapunov functions (LFs) obtained from Stabhyli to alleviate the proof obligations for establishing SC.BG2.

5.1.2 Proving SC.BG2 and SC.BG3

In order to establish the safety property that it always holds $dist_{\mathsf{cur}} \in [-10, 10]$ (SC.BG2) and that it always holds that $\beta_{\mathsf{ori}} \in [-5°, 5°]$ (SC.BG3), we apply SoapBox after a sequence of model and proof goal transformations.

To reduce the automaton's complexity we used the observation that by design SC is symmetric. This yields a steering controller where either the left or right modes are omitted (cf. Fig. 11).

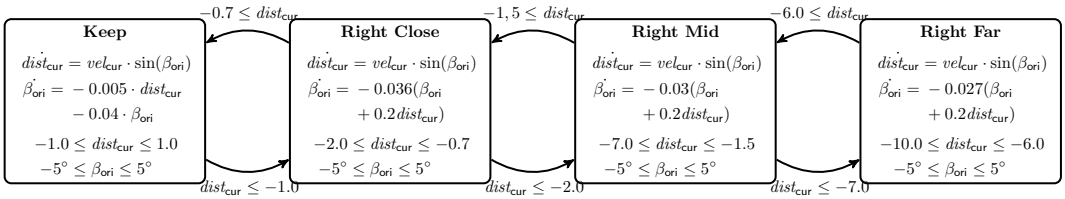


Figure 11: SC after Exploiting Symmetry

We also transformed the proof goal. If any trajectory of the reduced model emanating from the initial set finally re-enters the initial set in mode **Keep**, we can verify SC.BG2 (and also SC.BG3) on the reduced model, as the omitted part of the automaton does not inject additional behavior in the retained part.

In general, we can stop computation of a trajectory, when it re-enters the initial set. Any possible suffix of it is already covered as a trajectory directly emanating from the initial set.

In essence, we show on the reduced model that

1. any trajectory emanating from the initial set re-enters the initial set and
2. any reachable state along a trajectory from initial set up to re-entering the initial set satisfies SC.BG2.

To use SoapBox for verification on SC we have to transform the model, so that the bounded input vel_{cur} is additive instead of multiplicative and the sine function gets overapproximated. In the sequel we derive how to overapproximate the distance evolution by $\dot{dist}_{\mathsf{cur}} = m \cdot \beta_{\mathsf{ori}} + [\underline{e}, \overline{e}]$ where $[\underline{e}, \overline{e}]$ is an interval denoting a bounded error.

Let $[a, b]$ be the interval of all admissible values of β_{ori}. We linearize the differential inclusion $\dot{dist}_{\mathsf{cur}} = vel_{\mathsf{cur}} \cdot \sin(\beta_{\mathsf{ori}})$ for $vel_{\mathsf{cur}} \in [5, 10]$ over the interval $[a, b]$ as follows.

The values $vel_{\mathsf{cur}}(t)$ of any admissible evolution of the velocity are bounded by the interval $[5, 10]$. Hence, the values of $\dot{dist}_{\mathsf{cur}} = vel_{\mathsf{cur}} \sin(\beta_{\mathsf{ori}})$ are bounded by $[5, 10] \cdot \sin(\beta_{\mathsf{ori}})$. Let $m \cdot \beta_{\mathsf{ori}} + [\underline{e}, \overline{e}]$ be a safe enclosure of $\sin(\beta_{\mathsf{ori}})$ over the interval $\beta_{\mathsf{ori}} \in [a, b]$, where $\underline{e}$ and $\overline{e}$ depend on the interval bounds a and b. Then

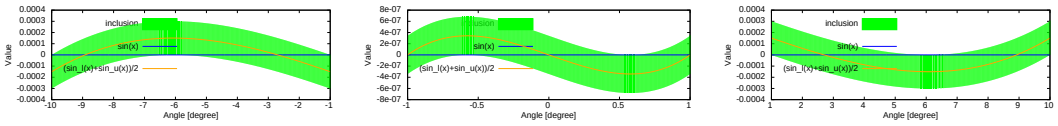
$$\begin{aligned} \dot{dist}_{\mathsf{cur}} \in [5, 10] \cdot \sin(\beta_{\mathsf{ori}}) &\subseteq [5, 10] \cdot (m \cdot \beta_{\mathsf{ori}} + [\underline{e}, \overline{e}]) \\ &= [5, 10] \cdot m \cdot \beta_{\mathsf{ori}} + [5, 10] \cdot [\underline{e}, \overline{e}] \\ &= 7.5 \cdot m \cdot \beta_{\mathsf{ori}} + [-2.5, 2.5] \cdot m \cdot \beta_{\mathsf{ori}} + [5, 10] \cdot [\underline{e}, \overline{e}]. \end{aligned}$$

Hence, solely using the safe enclosure of the sine, we obtained the inclusion $\dot{dist}_{\mathsf{cur}} \in 7.5 \cdot m \cdot \beta_{\mathsf{ori}} + [-2.5, 2.5] \cdot m \cdot \beta_{\mathsf{ori}} + [5, 10] \cdot [\underline{e}, \overline{e}]$. While the second interval expression, $[5, 10] \cdot [\underline{e}, \overline{e}]$, only contains constants and thus can be modeled as an additive input, the first interval expression, $[-2.5, 2.5] \cdot m \cdot \beta_{\mathsf{ori}}$, still depends on β_{ori}. In the final

step we use $\beta_{\mathsf{ori}} \in [a, b]$, and obtain the following differential inclusion for $\dot{dist}_{\mathsf{cur}}$

$$\begin{aligned}\dot{dist}_{\mathsf{cur}} &\in 7.5 \cdot m \cdot \beta_{\mathsf{ori}} + [-2.5, 2.5] \cdot m \cdot \beta_{\mathsf{ori}} + [5, 10] \cdot [\underline{e}, \overline{e}] \\ &\subseteq 7.5 \cdot m \cdot \beta_{\mathsf{ori}} + [-2.5, 2.5] \cdot [a, b] \cdot m + [5, 10] \cdot [\underline{e}, \overline{e}].\end{aligned}$$

Since a single overapproximation over all possible values, i.e., $\beta_{\mathsf{ori}} \in [-5°, 5°]$, is too coarse, we choose to split the state space along β_{ori}, that is we partition the interval $[a, b]$ into $a = a_0 < a_1 < \cdots < a_n = b$. For each subinterval $[a_i, a_{i+1}]$ we derived a linear system with an individual differential inclusion, hence yielding a better fitting overapproximation. The linear systems are joined by discrete transitions as follows. Assume, β_{ori} is initially in the interval $[a_i, a_{i+1}]$. If the value of β_{ori} reaches the bound a_{i+1} and surely will exceed the bound, i. e. $\beta_{\mathsf{ori}} = a_{i+1}$ and $\dot{\beta}_{\mathsf{ori}} > 0$, then a transition to the subinterval $[a_{i+1}, a_{i+2}]$ has to be taken.



(a) Absolute Error (b) Absolute Error (c) Absolute Error

Figure 12: Piecewise Linear Enclosing of the Sine using Additive Error. The blue line is the value of the sine, the orange line is the mean value and the green shape the possible values of the inclusion.

Side effect of such a state space partitioning is that the SC's modes are split up, as illustrated in Figure 13 for a partition at $\beta_{\mathsf{ori}} = 0$. Drawbacks of such partitioning hence are, that a lot of new transitions are introduced and some of them can even lead to zeno-behavior. To prove SC.BG2 we used a finer partition which results in a large automaton. It is easy to see that some of its transitions do not correspond to the original flow of the system and can hence be removed (marked in red in Figure 13).

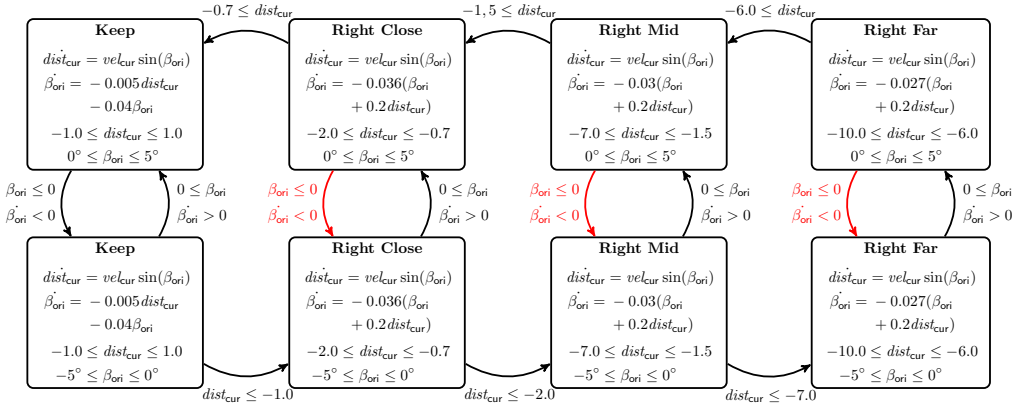


Figure 13: SC after partitioning β_{ori} at 0

To apply a partition, SoapBox has to deal with strict inequalities. Strict inequalities allow us to express that subsets are disjoint, while the complete state space is

covered. A division into disjoint subset allows finer overapproximation. As discussed in Sect. 4.2.2 SoapBox supports strict inequalities and thereby rules out some forms of zeno-behavior. Note that transition removal described above was not applicable to the **Keep** mode.

We further simplified the proof goal to show the safety property SC.BG2 and SC.BG3. We used two different arguments that allowed us to derive that all future states of a trajectory satisfy SC.BG2 and SC.BG3.

The first argument refers to the automaton. Analysis on the vector field reveal that (a) $dist_{\mathsf{cur}} \in [-10, 10] \wedge \beta_{\mathsf{ori}} \geq 2$ implies $\dot{\beta_{\mathsf{ori}}} \leq 0$ and (b) $\beta_{\mathsf{ori}} \geq 0$ implies $\dot{dist_{\mathsf{cur}}} \geq 0$. Thus we can conclude that the upper copies of **RightClose**, **RightMid**, and **RightFar** in Figure 13 are safe wrt SC.BG2 and SC.BG3 if SC.BG2 and SC.BG3 hold on entering these modes. We instead established only that any trajectory that enters these modes already satisfies SC.BG2 and SC.BG3.

The second argument introduces so-called *safe sets* that are subsets of the state space and guarantee that any trajectory entering such a set will not reach an *unsafe set* anymore.

Level Sets and Safe Sets

Systems of damped oscillating behavior like the SC are often difficult to deal with when it comes to determining their trajectories near the equilibrium due to numerical issues. In the remainder of the section we sketch an argument that alleviates the burden of examining every reachable state wrt a safety property by a sound argument based on the system's Lyapunov function. We combine Lyapunov functions with bounded model-checking by first using Lyapunov functions to calculate a barrier certificate. Roughly speaking, a barrier certificate can be used to establish a border between two state sets such that one set cannot be reached via any trajectory starting in the other set. Each level set of a Lyapunov function can be used as barrier certificate [15]. Then we applied a model checker to prove that the safety property globally holds until either an upper time bound derived from the Lyapunov function or that set is reached.

Basically, this argument allows us to neglect the future of a trajectory as soon as the trajectory is sufficiently close to the equilibrium. We further describe its concrete application on the case study to establish SC.BG2 and SC.BG3.

A Lyapunov function assigns a value to any state of the state space and this LF's value along any trajectory decreases. Each *level set* $\mathcal{L}_{V,s} = \{x \mid V(x) \leq s\}$ is hence an invariant set as defined in Def. 3.

Definition 3 ***Invariant Set.*** *A set of states $\mathcal{S}' \subseteq \mathcal{S}$ is called an* invariant set *of a hybrid automaton H if for all trajectories $\mathbf{x}(t)$ holds $\mathbf{x}(0) \in \mathcal{S}' \Rightarrow \forall t \geq 0 : \mathbf{x}(t) \in \mathcal{S}'$.*

Now, the alleviating argument for establishing SC.BG2 and SC.BG3 is, as soon as a level set has been entered that satisfies SC.BG2 and SC.BG3 we can stop further computing the trajectory. As we want to stop trajectory computation as early as possible, we try to find a level set as big as possible. Therefore we determine the minimal LF's value over all states violating SC.BG2 $\wedge$ SC.BG3. A state with a lesser LF's value is hence guaranteed not to be in the *unsafe set*. Such a state is element of the *safe set* (cf. Def. 4).

Definition 4 ***Safe Set.*** *Given a set of states Unsafe* $\subseteq \mathcal{S}$*, a set* $Safe_{Unsafe} \subseteq \mathcal{S}$ *is called a* safe set *wrt Unsafe for a hybrid automaton H, if for all trajectories* $\mathbf{x}(t)$ *holds* $\mathbf{x}(0) \in Safe_{Unsafe} \Rightarrow \forall t \geq 0 : \mathbf{x}(t) \notin Unsafe$*.*

Note, that any level set $\mathcal{L}_{V,s}$, which has no intersection with the set *Unsafe*, serves as a safe set $Safe_{Unsafe}$.

To use safe sets for trajectory truncation in a tool, we have to be able to express when a state is element of the safe set, i. e. when its LF's value is sufficiently low. As we obtained quadratic Lyapunov functions for SC from Stabhyli, we cannot refer to the Lyapunov function directly in SoapBox, as it does not support quadratic constraints. As SoapBox supports linear constraints only, we underapproximated the safe set via safe boxes. This yields the following basic algorithm:

1. Determine the lowest LF's value $b := \min\{V(x) \,|\, x \in Unsafe\}$ of all states in the unsafe set.
2. Subtract a safety margin ϵ on the LF's value, $g := b - \epsilon$, to build the safe set.
3. Guess a box within the level set $\mathcal{L}_{V,g}$.

Such a box lies within the safe set, if the LF's value for any of its corner points is less then or equal to g. Multiple safe boxes can be found by applying multiple optimization functions such as maximizing the length of an edge or the length of the diagonal. This procedure can be automatized and multiple optimization functions can be combined. We achieved the best results on the SC model via manual optimization, though. This was easily feasible, as we had to deal with two dimensions only. The result is shown in Figure 14.

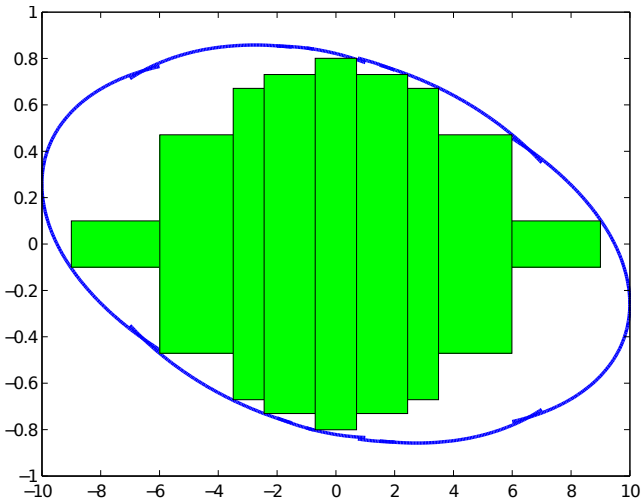


Figure 14: Piecewise Level Sets (blue) and Safe Sets (green) of the Steering Controller

In principle, safe boxes can be encoded into the model. As SoapBox and many other tools require convex modes, a safe box within a mode hence implies that the surrounding of the safe box has to be split into several convex submodes. SoapBox supports convex safe sets directly, though, as outlined in Sect. 4.2.2.

5.1.3 Proving SC.BG4

That SC.BG4 holds can be seen by examining the differential equation for β_{steer} as defined by SC. Figure 15 shows a heat map for all combinations of ($dist_{\mathsf{cur}}$,β_{ori}) within bounds as in SC.BG2 and SC.BG3. The color encodes the (maximum) value of the differential equation for steering at the respective mode. It hence follows that the value of the differential equation ranges over $[-0.2, 0.2]$ and thus implies SC.BG4 as long as SC.BG2 and SC.BG3 hold.

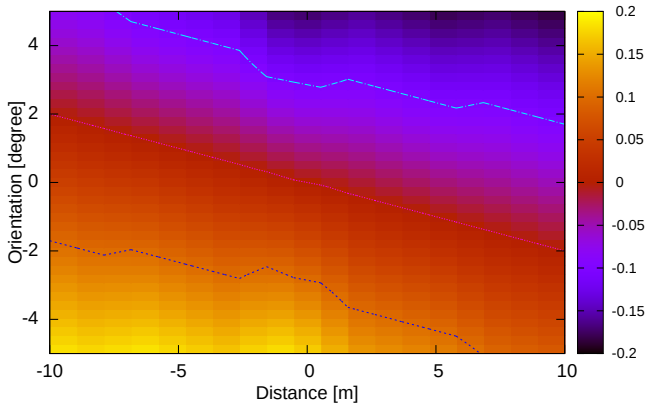


Figure 15: Possible values of β_{steer} within mode invariants

5.2 Proving Interface Implementation for VC

To analyze the VC, we use that VC has finitely many possible setpoints. The number of setpoints is the product of the number of receivable events of VC (cf. Table 1), the two values of *peakOvershoot* (cf. Table 2) and the number of possible values for vel_{des}. For each of VC's setpoints, i.e. for each value of vel_{goal}, we obtain an instance $\mathsf{VC}(vel_{\mathsf{goal}_i})$. Appropriate discrete switching between these instances, lead to an automaton chaining the $\mathsf{VC}(vel_{\mathsf{goal}_i})$. We instead analyze an equivalent but more compact version, that adopts the mode invariants in the original VC and resets vel_{goal}.

5.2.1 Proving VC.BG1 and VC.EG1

To show that VC converges to vel_{goal}, VC is shifted by setting $vel_{\mathsf{cur}}' = vel_{\mathsf{cur}} - vel_{\mathsf{goal}}$. We obtained Lyapunov functions from Stabhyli (cf. Table 10) that certify convergence of the shifted VC towards $vel_{\mathsf{cur}}' = 0$. These Lyapunov functions certify that VC converges starting from any state. Hence convergence is implied also when a new vel_{goal} has been chosen.

$$V_{\mathbf{Accl}} : -5.3788e^{-06}\, vel_{\mathsf{cur}} + 1.89209e^{-05}$$
$$V_{\mathbf{Norm}} : +1.64382e^{-07}\, vel_{\mathsf{cur}} vel_{\mathsf{int}} + 1.64569e^{-08}\, vel_{\mathsf{int}}^2 + 1.83258e^{-06}\, vel_{\mathsf{cur}}^2$$
$$V_{\mathbf{Decl}} : +5.3788e^{-06}\, vel_{\mathsf{cur}} + 1.89209e^{-05}$$

Table 10: Automatically derived Lyapunov functions for the VC (rounded).

5.2.2 Proving VC.BG2 and VC.EG2

To show that, *when there is an active event* $\mathcal{E}$*, then it always holds* $vel_{cur} \in [5, vel_{comf}(\mathcal{E})]$ *and when there is no active event, it always holds* $vel_{cur} \in [5, 10]$, we first analyze the timing behavior of VC, in particular we analyze the automaton in Fig. 7.

VC_ER will always only set a new vel_{goal} as based on the event assumption of $\mathcal{E}$ after its dueTime has elapsed. Also VC_ER will determine a new safe vel_{goal} at least t time units before the (current) promise ends, where $t = maxTimeToSafeVel(vel_{\mathsf{cur}}, vel_{\mathsf{goal}}^{\mathsf{min}})$. By construction (cf. Prop. 1) t is sufficiently large. Thus VC will always stay below the new setpoint, before the end of an event promise is reached.

If a new setpoint is chosen, then vel_{goal} is always set according to Eq. 2 or to $vel_{\mathsf{goal}}^{\mathsf{min}}$. We show that the peak overshoot is sufficiently high:

Property 2 *If* vel_{goal} *is (re)set so that* $vel_{goal} \leq vel_{comf} - peakOvershoot$ *holds, then* $vel_{cur} \leq vel_{comf}$ *always holds.*

For the proof of Prop. 2 we determined safe boxes for VC using the Lyapunov functions from Stabhyli. We then applied SoapBox to prove that Prop. 2 also holds on all reachable states outside the safe boxes.

6 Deducing properties of the composed system

We now use the properties proven for the isolated SC and VC to derive properties for the composed ADAS controller. The ADAS has to make assumptions that are sufficient to imply the assumptions of its subcomponents, so that they are bound to deliver their guarantees. The assumptions of the ADAS controller are weaker than the conjunction of assumptions of SC and VC, as it takes into account, that its subcomponents contribute their guarantees.

As an intermediate step for proving the composite properties of the ADAS, we derive local properties for VC and SC in form of assumed properties and guaranteed properties. These are in a convenient form to derive the composite properties for the ADAS. In order to make this paper self-contained, we also enrich the list of local properties with the requirements implied by our framework, which do not explicitly appear in Sect. 3.2.

6.1 Local Properties of the Controllers

In the sequel, we summarize as local properties the interface properties, assumptions that we made for simplification, as well as requirements resulting from our framework. We give pointers to the origin of the local properties and also present small proofs to derive additional local properties.

6.1.1 Local Properties of VC

VC.LG3 *The current velocity of the car is always in* $vel_{cur} \in [5, 50]$.

This follows from VC.BG2 and since the reaction to events is guaranteed as VC.EG2 and Tab. 5.

VC.LG2 *The current velocity is always at most the maximum comfortable velocity,* $\mathsf{G}\,(vel_{cur} \leq vel_{comf})$.

According to VC.BG2, without an active event it holds that $vel_{\mathsf{cur}} \in [5, velFor(\mathcal{E})]$ and according to VC.BA2) steering has to be in $\beta_{\mathsf{steer}} \in [-0.3, 0.3]$. So that vel_{cur} is comfortable according to Eq. 1. In case there is an active event, vel_{cur} is comfortable according to VC.EG2. Hence the current velocity never exceeds the comfortable velocity.

VC.LG4 *VC is lock-free.*

VC is lock-free (time always progresses), since the union of all invariants covers the component's state space i.e. $[-50 + vel_{\mathsf{goal}}, 50 + vel_{\mathsf{goal}}] \supseteq [5, 50]$ cf. Fig. 6, and there is at least one enabled transition at the mode boundaries. VC does not stall time by firing discrete transitions, since for a given constant vel_{des} it converges towards vel_{goal} and by VC.LA7, it follows that there is no discrete divergence.

Name	Property	Pointer to **Origin**
VC.LA1	Events are truthful.	required by the frame work
VC.LA2	$vel_{\mathsf{cur}} \in [5, 10]$	equals VC.BA1
VC.LA3	$\mathsf{G}\,(\beta_{\mathsf{steer}} \in [-0.3, 0.3])$	equals VC.BA2
VC.LA4	$\mathsf{G}\,(vel_{\mathsf{comf}} \in [10, 50])$	implied by VC.BA2
VC.LA5	$\mathsf{G}\,(s = 1)$	simplifying restriction, can be lifted cf. Sect. 5
VC.LA6	$\mathsf{G}\,(vel_{\mathsf{des}} \in [10, 45])$	equals to VC.BA3
VC.LA7	There is a lower time bound > 0 on event separation	required by the frame work
VC.LG1	vel_{cur} converges to vel_{goal}	VC.EG1 and VC.BG1
VC.LG2	$\mathsf{G}\,(vel_{\mathsf{cur}} \leq vel_{\mathsf{comf}})$	implied by VC.BG2 and VC.EG2
VC.LG3	$\mathsf{G}\,(vel_{\mathsf{cur}} \in [5, 50])$	implied by VC.EG2, VC.BG2 assuming VC.BA2
VC.LG4	VC is lock free	proven above, required by framework

Table 11: Summary of local assumptions and local guarantees of the VC.

6.1.2 Local Properties of SC

SC.LG4 *The SC is lock-free.*

SC is lock-free that is time always progresses, since the union of all invariants covers the component's state space, which is never left (SC.LG2) and since there is always one transition enabled at the mode boundaries and time is not stalled by firing infinitely many discrete transitions, as modes are always entered with a fixed least distance to a guard.

6.2 Composing SC and VC to build the ADAS controller

To compose SC and VC to build a composite controller, the two components need to be compatible under assumptions of the ADAS. Basically this means that the SC's guarantees together with the ADAS assumptions have to imply the assumptions of

Name	Property	Pointer to **Origin**
SC.LA1	init conf. restricting β_{ori} depending on *dist*	equals SC.BA1
SC.LA2	$\mathsf{G}\,(vel_{\mathsf{cur}} \in [5, 50])$	implied by SC.BA2 and SC.EA1
SC.LA3	$\mathsf{G}\,(vel_{\mathsf{cur}} \leq vel_{\mathsf{comf}})$	implied by SC.BA2 and SC.EA1
SC.LG1	converg. to $(\beta_{\mathsf{ori}}, dist_{\mathsf{cur}}) = (0,0)$	implied by SC.BG1 and SC.EG1
SC.LG2	$\mathsf{G}\,(dist_{\mathsf{cur}} \in [-10, 10] \wedge \beta_{\mathsf{ori}} \in [-5°, 5°])$	implied by SC.BG2, SC.EG2, SC.BG3, SC.EG3
SC.LG3	$\mathsf{G}\,(\beta_{\mathsf{steer}} \in [-0.3, 0.3])$	implied by SC.BG4,SC.EG4, Tab. 8
SC.LG4	SC is lock free	proven above, required by framework

Table 12: Summary of local assumptions and local guarantees of the SC.

VC, and vice versa, the SC's assumptions have to implied by VC's guarantees and the ADAS assumptions. This has to hold for base assumptions as well as for event assumptions. So for a valid composition, we can derive assumptions for the ADAS as follows:

- VC.BA1 (initially $vel_{\mathsf{cur}} \in [5, 10]$) is propagated to the ADAS interface,
- VC.BA2 ($\mathsf{G}\ \beta_{\mathsf{steer}} \in [-0.3, 0.3]$) is guaranteed by SC.BG4,
- VC.BA3 ($\mathsf{G}\ vel_{\mathsf{des}} \in \{10, \ldots, 45\}$) is propagated to the ADAS interface,
- SC.BA1 is propagated to the ADAS interface,
- SC.BA2 ($\mathsf{G}\ vel_{\mathsf{cur}} \in [5, 10]$) is guaranteed by VC.BG2,
- SC.EA1 ($\mathsf{G}\ vel_{\mathsf{cur}} \in [5, velFor(\mathcal{E})]$) is guaranteed by VC.EG2.

Additionally, for composition of VC and SC, we have to assure that the requirements of the framework on the components are satisfied. Thus we have to ensure that it holds that

- events are truthful (VC.LA7),
- events have a minimal separation time (VC.LA7),
- VC does not stall time (VC.LG4), and
- SC does not stall time (SC.LG4).

We already established these properties in the previous section. In summary, it follows that we can compose VC and SC to an ADAS controller with the following assumptions and guarantees:

Name	Property
ADAS.BA1	Initially holds that $vel_{\mathsf{cur}} \in [5, 10]$.
ADAS.BA2	It always holds that $vel_{\mathsf{des}} \in \{10, \ldots, 45\}$.
ADAS.BA3	Initially holds that $dist_{\mathsf{cur}} \in [-9, 0] \wedge \beta_{\mathsf{ori}} \in [0°, 2.6°] \vee dist_{\mathsf{cur}} \in [0, 9] \wedge \beta_{\mathsf{ori}} \in [-2.6°, 0°]$
ADAS.BG1	$(\beta_{\mathsf{ori}}, dist_{\mathsf{cur}})$ converges to (0,0).
ADAS.BG2	It always holds that $dist_{\mathsf{cur}} \in [-10, 10]$.
ADAS.BG3	It always holds that $\beta_{\mathsf{ori}} \in [-5°, 5°]$.
ADAS.BG4	It always holds that $\beta_{\mathsf{steer}} \in [-0.2, 0.2]$.
ADAS.BG5	vel_{cur} converges to vel_{des} as close as possible at the current steering state.
ADAS.BG6	It always holds $vel_{\mathsf{cur}} \in [5, 50]$.
ADAS.BG7	It always holds that the centrifugal force is comfortable, $\mathsf{G}\,(\lvert F\rvert \in [0, F_{\mathsf{comf}}])$.

Table 13: Base assumptions and guaranties at the interface of the ADAS controller.

The guarantees listed above are all but ADAS.BG7 directly implied by respective guarantees of VC and SC. It is easy to see that also the composite guarantee ADAS.BG7 holds:

ADAS.BG7 *It always holds that the centrifugal force is comfortable.*

This holds, since VC.LG2 holds and since events are truthful and due to SC.BG4. VC guarantess to never reach an uncomfortable velocity (VC.LG2), that is neither under base assumptions VC.BG2 or under event assumptions VC.EG2. Also SC sends only truthful events and otherwise exposes steering as declare via its base assumptions.

ADAS.BG5 *vel_{cur} converges to vel_{des} as close as possible at the current steering state.*

We did not formally specify the event generation of SC, so that we do not formally specify this guarantee. Basically SC declares for a generated event its assumptions on the street. These are propagated to the ADAS, so that it can declare speed increase in terms of the road condition and vel_{des}.

Moreover, it holds that the resulting ADAS component is lock-free and hence can be used for further composition:

Property 3 *The ADAS controller is lock-free.*

The ADAS controller is lock-free, i.e., guarantees time to progress, since its subcomponents are lock-free under their respective assumptions (cf. SC.LG4, VC.LG4). The assumptions capture all possible interactions with an admissible environment. So VC and SC remain to be lock-free when composed to build the ADAS, since VC refines the assumption of SC and SC refines VC's assumptions.

7 Conclusion

We have presented a design methodology based on an exemplary composition of a steering controller (SC) and a velocity controller (VC) together implementing a simple ADAS. We gave an overview on the system's architecture, specified the controllers as hybrid automata and explained their event-based communication. We established that each controller fulfills its interface specification. Therefore we combined overapproximations, state space partitioning, Lyapunov functions and bounded reachability analysis. According to their interface specifications, VC controls the velocity to be safe but stabilizes to a user chosen velocity, if this is safe, and SC brings and then keeps the car to the middle of its lane. We demonstrated, how to derive the interface specification of the composite ADAS from the interface specifications of the subcomponents. As VC and SC are compatible, they still achieve their objectives when composed and together they also maintain the global objective, *the centrifugal force a passenger is experiencing does not exceed a certain threshold.*

In the future, we plan to extend our case study towards curved lanes, for which we have to consider reference tracking. However, our main target is to develop a formalized design methodology which allows library-based design of controllers, so that the properties of a composed controller can be compositionally derived from properties annotated in the subcomponents' interface –without examining the global system.

References

[1] L. J. Alun Foster, Iris Hamelink, editor. *ARTEMIS Book of Successes*. ARTEMIS, 2013.

[2] B. Borchers. Csdp, a C library for semidefinite programming. 10:613–623, 1999.

[3] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, Mar. 2004.

[4] W. Damm, H. Dierks, J. Oehlerking, and A. Pnueli. Towards component based design of hybrid systems: Safety and stability. In Manna and Peled [11].

[5] W. Damm, H. Dierks, J. Oehlerking, and A. Pnueli. Towards Component Based Design of Hybrid Systems: Safety and Stability. In Manna and Peled [11], pages 96–143.

[6] W. Damm, A. Mikschl, J. Oehlerking, E.-R. Olderog, J. Pang, A. Platzer, M. Segelken, and B. Wirtz. Automating Verification of Cooperation, Control, and Design in Traffic Applications. In *Formal Methods and Hybrid Real-Time Systems, Essays in Honor of Dines Bjørner and Chaochen Zhou on the Occasion of Their 70th Birthdays*, 2007.

[7] W. Damm, T. Peikenkamp, and B. Josko. Contract Based ISO CD 26262 Safety Analysis. In *SAE World Congress – Session on Safety-Critical Systems*, 2009.

[8] W. Damm, A. Votintseva, A. Metzner, B. Josko, and E. Peikenkamp, Thomas; Böde. Boosting re–use of embedded automotive applications through rich components. In *Proceedings of FIT 2005*, 08 2005.

[9] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler. SpaceEx: Scalable verification of hybrid systems. In *Computer Aided Verification*, pages 379–395, 2011.

[10] C. Le Guernic and A. Girard. Reachability analysis of hybrid systems using support functions. In *Computer Aided Verification*, pages 540–554, 2009.

[11] Z. Manna and D. Peled, editors. *Time for Verification, Essays in Memory of Amir Pnueli*, volume 6200 of *LNCS*. Springer, 2010.

[12] E. Möhlmann and O. E. Theel. Stabhyli: a Tool for Automatic Stability Verification of Non-Linear Hybrid Systems. In C. Belta and F. Ivancic, editors, *HSCC*. ACM, 2013.

[13] J. Oehlerking. *Decomposition of Stability Proofs for Hybrid Systems*. PhD thesis, Carl von Ossietzky University of Oldenburg, Department of Computer Science, Oldenburg, Germany, 2011.

[14] J. Oehlerking and O. E. Theel. Decompositional Construction of Lyapunov Functions for Hybrid Systems. In *HSCC*, volume 5469 of *LNCS*, pages 276–290. Springer, 2009.

[15] S. Prajna and A. Jadbabaie. Safety verification of hybrid systems using barrier certificates. In R. Alur and G. J. Pappas, editors, *HSCC*, volume 2993 of *LNCS*. Springer, 2004.

[16] S. Prajna and A. Papachristodoulou. Analysis of Switched and Hybrid Systems - Beyond Piecewise Quadratic Methods, 2003.

[17] A. L. Sangiovanni-Vincentelli, W. Damm, and R. Passerone. Taming Dr. Frankenstein: Contract-based design for cyber-physical systems. *Eur. J. Control*, 18(3), 2012.