Automating Verification of Cooperation, Control, and Design in Traffic Applications [⋆]

Werner Damm[1,2], Alfred Mikschl[1], Jens Oehlerking[1], Ernst-Rüdiger Olderog[1],
Jun Pang[1], André Platzer[1], Marc Segelken[2], and Boris Wirtz[1]

[1] Carl von Ossietzky Universität Oldenburg, Ammerländer Heerstraße 114-118,
26111 Oldenburg, Germany
[2] OFFIS, Escherweg 2, 26121 Oldenburg, Germany

Abstract. We present a verification methodology for cooperating traffic agents covering analysis of cooperation strategies, realization of strategies through control, and implementation of control. For each layer, we provide dedicated approaches to formal verification of safety and stability properties of the design. The range of employed verification techniques invoked to span this verification space includes application of pre-verified design patterns, automatic synthesis of Lyapunov functions, constraint generation for parameterized designs, model-checking in rich theories, and abstraction refinement. We illustrate this approach with a variant of the European Train Control System (ETCS), employing layer specific verification techniques to layer specific views of an ETCS design.

1 Introduction

Our society at large depends on the transportation sector to meet the increased demands on mobility required for achieving sustained economic growth. Major initiatives such as ERTRAC[3], eSAFETY[4] and the car2car consortium in automotive, ACARE[5] in avionics, and ERRAC[6], ETCS/ERMTS[7] in rail drive standards for inter-vehicle and vehicle to infra-structure cooperation, are thriving to push safety by enforcing cooperation principles between traffic agents.

Automatic collision avoidance systems form an integral part of such systems, with domain specific variants ranging from fully automatic protection to partial automation combined with warning/alerting, to warning combined with directives. For example, in the automotive domain, based on pre-crash sensing, close

[⋆] This work was partly supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS, `http://www.avacs.org/`).

[3] European Road Transport Research Advisory Council (`www.ertrac.org`)

[4] `http://ec.europa.eu/information_society/activities/esafety/`

[5] Advisory Council for Aeronautics Research in Europe (`www.acare4europe.com/`)

[6] European Rail Research Advisory Council (`www.errac.com/`)

[7] European Rail Traffic Management System (`www.ertms.com`)

distance warnings are automatically displayed, and hydraulic pressure for the braking system is built up, reducing the response time to a driver's reaction to such warnings. Full automation using brake-by-wire/steer-by-wire technology is technically feasible, and has been demonstrated early in research vehicles, e.g., within the California Path project. Anticipated future traffic scenarios include communication between cars and cars, and roadside infrastructure to guide coordinated maneuvers for collision avoidance. We use as running example in this paper a variant of the European Train Control System standard, which provides collision avoidance through a fully automated coordinated movement of trains, based on information obtained from track-side infrastructure called Radio Block Centers (RBC). An RBC is responsible for monitoring the position of all trains in its track segment, and provides authorities for trains to freely move ahead until so-called End-of-Authority points (EoA) are reached. As soon as the on-board Automatic Train Protection system ATP detects that a train risks to move beyond the current EoA, the ATP system takes control of the train's speed and enforces a braking curve leading to a complete stop of the train ahead of the EoA. Under ETCS level 3, EoAs are moved ahead by the RBCs, as soon as it has gained safe knowledge of the fact that the train ahead has reached a safe distance to the successor train. This "moving block principle", where each train is protected by an envelope surrounding and moving with the train, contrasts to classical interlocking principles, where tracks are partitioned statically into blocks, and trains are guaranteed exclusive access to blocks by interlocking protocols. Our running application is an extension of the moving block principle to include rail-road crossings, see Section 2 for more details.

In the avionics domain, the Traffic Alert and Collision Avoidance System (TCAS) provides directives to pilots how to avoid a near-collision situation using combined ascend/descent maneuvers, or through recently investigated "go-around" maneuvers, see [22].

This paper provides a formal verification methodology addressing such application classes. Specifically, we provide dedicated verification methods for establishing safety and stability requirements for three key design layers:

1. The *cooperation layer* addresses inter-vehicle (and infrastructure) cooperation, where traffic-agents and infrastructure elements negotiate and agree on maneuvers executed jointly to enforce safety while optimizing throughput.
2. The *control layer* focuses on the design of control-laws implementing the suit of maneuvers supported by a traffic agent.
3. The *design layer* focuses on the implementation of control-laws through digital controllers.

Jointly, the techniques presented here combine to a holistic system verification approach, ensuring that system-level requirements are guaranteed by the implementation of control-laws supporting the maneuver capabilities of cooperating traffic agents.

Technically, the verification methodology rests on techniques for the verification of hybrid systems developed by the large-scale foundational research project AVACS (`www.avacs.org`) on automatic verification and analysis of complex systems. The range of employed verification techniques invoked to span this verification space includes application of pre-verified design patterns, automatic synthesis of Lyapunov functions, constraint generation for parameterized designs, model-checking in rich theories, and abstraction refinement.

The verification of the correctness of collision avoidance system has been studied extensively, e. g., within the PATH project [36], by Leveson [31], Sastry *et al.* [53], Lynch *et al.* [32], Clarke [47], and Damm *et al.* [14] for various versions of the TCAS system, or by Peleska *et al.* [24] and Damm *et al.* [6] for train system applications. Sastry *et al.* presents in [53] a general approach of developing such distributed hybrid systems. More recently, R-Charon [30], a semi-conservative extension of Charon [1, 2] has been proposed for modular specification and dynamic reconfiguration of large distributed hybrid system based on hybrid automata.

This paper is structured as follows. We give a sufficiently detailed presentation of the variant of the ETCS level 3 protocol used as running example in Section 2. As unifying underlying formal model we use communicating hybrid automata presented in Appendix A. Section 3 describes the overall verification methodology as well as the underlying assumptions for each modelling layer. Section 4 shows how a pre-verified design pattern for collision avoidance protocols can be instantiated for our ETCS application. The focus of Section 5 is on generating constraints on design parameters for collision avoidance protocols ensuring collision freedom. Sections 6 and 7 discuss automatic verification methods for proving stability and safety, respectively, using as running example the drive train controller for maintaining the operator selected speed. Both sections discuss the local control as well as the design layer. We finally wrap by pointing out directions for further work in Section 8.

2 Extending ETCS Level 3 for Rail-Road Crossings

In this section we describe the model of a train system running under a variant of the ETCS level 3 protocol. We have extended the protocol to deal with the protection of track segments before a train gets access to enter this segment. As an example of an unsafe element inside a track segment we have chosen a rail road crossing. To be able to evaluate the different aspects of an embedded system we have developed a dynamic system model extended with different control levels. The system dynamics are modelled in Matlab-Simulink and the control parts of the ETCS protocol are modelled in Stateflow. The model of the dynamics consists of three parts. The first is the mechanical transmission, which converts the input torque into the angular velocities of the wheels. The second part consists of the outer conditions, used to produce the present train velocity. This velocity depends on the angular velocity of the wheels, the present adhesion between wheel and track, and other losses such as air resistance, rolling resistance etc. The third part of the model contains the control part of the ETCS protocol and communicates to the crossing station and to the radio control block.

2.1 Mechanical Transmission

The mechanical transmission consists of the engine which is coupled directly on the driven wheel where the dynamics of the shaft are neglected. The train dynamic contains the engine dynamics, the brake dynamics and the block which calculates the present velocity of the train as a function of the friction force and the angular momentum between the wheel-track system. The block produces the values of the present torque on a driven wheel T_w, the angular velocity of the wheel ω_w and the present velocity of the train v.

The angular momentum of the engine is modelled as a function of the drive current (I) as the controlled variable, and the present angular velocity of the engine (ω). The i_s and w_s are constants and the t_n and ω_{max} are parameters.

$$T_w = min(i_s \cdot \omega + t_n \cdot I,\ w_s \cdot \omega + w_s \cdot \omega_{max} \cdot I) \tag{1}$$

The drive current is driven by a PI controller with the desired velocity and the present velocity as input.

$$I(v, v_d) = P_d \cdot (v_d - v) + T_d \cdot \int (v_d - v)dt \tag{2}$$

To limit the maximum acceleration additional parts have been added. If the current acceleration (a) exceeds the max_acceleration (*max_acc*) the difference of them is multiplied by a scaling factor and then subtracted from the drive current. Equation 2 can then be rewritten to:

$$I(v, v_d) = \begin{cases} P_d \cdot (v_d - v) + T_d \cdot \int (v_d - v)dt & : \quad a \leq a_{max} \\ P_d \cdot (v_d - v) + T_d \cdot \int (v_d - v)dt - (a - a_{max}) \cdot a_{limit} & : \quad a > a_{max} \end{cases} \tag{3}$$

The dependency of the angular velocity ω and the toque is given by the formula

$$\omega = \int_0^t \frac{T_\omega - R_w \cdot (F_e + F_b)}{I}\, dt \tag{4}$$

where I is the moment of inertia of the rotating mass of the engine and the driven wheel, F_e is the resistance force of the environment, R_w denotes the radius of the wheel and F_b describes the braking force. The present velocity is calculated in the same way

$$v = \int_0^t \frac{F_t - F_e - F_b}{m}\, dt \tag{5}$$

where m is the mass of the train and F_t the traction force induced from the wheel into the track. The traction force is calculated by

$$F_t = \frac{T_w}{R_w} \cdot \mu_a \tag{6}$$

where μ_a denotes the adhesion coefficient between wheel and track.

2.2 Outer Losses

The outer losses are summarized in the resistance force F_e based on air resistance and roll resistance.

$$F_e = F_{air} + F_r + m \cdot g \cdot \sin\phi \tag{7}$$

The term $m \cdot g \cdot \sin\phi$ is the loss due to the lateral slope angle ϕ of the rail. The roll resistance is described by the formula

$$F_r = m \cdot c_r \tag{8}$$

where m is the total mass of the train and c_r the velocity independent roll resistance coefficient. The air resistance is described by

$$F_{air} = c_1^{air} \cdot v^2 + c_2^{air} \cdot v \tag{9}$$

The coefficient c_1^{air} depends on the density of air, the cross section of the train and such things, the coefficient c_2^{air} describes aerodynamic phenomenon which cannot be described as functions of v^2.

2.3 Brake

The main goal of this brake model is to bring up the train model in a non-moving state and not to study the brake behaviour in detail. For this reason the brake model is very simple. The brake model consists of two kinds of brake systems: an eddy current brake and an emergency brake. An eddy current brake consists of an electromagnetic shoe where the electromagnetic force is controlled by the brake current. The change of the magnetic field caused by the speed difference between the brake and the adjacent rail induces an eddy current in the rail. This eddy current leads to a resistance force which depends on the current and the speed difference. In high speed region ($v \geq 20\ m/s$) the resistance force is nearly linear to the speed difference. The resistance force tends to zero if the speed difference becomes zero. Therefore we need an additional brake mechanism for the low speed region. These two brake systems work as the service brake for the train model. For the emergency case there exists an additional brake called emergency brake. This brake is typically an electromagnetic rail brake. Both brake types, the eddy current brake and the electromagnetic rail brake, work directly between the train and the rail and do not depend on friction between the wheels and the train. This simplifies the brake model. The brake force F_b is modelled by

$$F_b = (I_b \cdot v + (v_{offset} - v)) \cdot sb_{sc} + eb_c \tag{10}$$

where I_b is the brake current to control the service brake, v is the present velocity of the train and v_{offset} is a constant to ensure brake force if the present velocity is close to zero. The constant sb_{sc} is a scaling factor for sufficient brake force. The emergency brake is modelled as a constant and denoted by eb_c. Deceleration is controlled through setting a proper brake current using a PI controller. The input of this controller is the present deceleration of the train and the coefficient for a comfortable deceleration of the service brake.

$$I_b = P_b \cdot (\frac{d\,v}{dt} - a_{sb}^d) + T_b \int_0^t \frac{d\,v}{dt} - a_{sb}^d \; dt \qquad (11)$$

We have presented here a sufficient precise description of the dynamical system. The equations are kept simple to get a linear system. The model can be extended mostly and a more detailed description can be found in [54].

2.4 ETCS Control Part

We consider a train system which is under the control of a variant of the ETCS level 3 protocol. The ETCS level 3 provides collision avoidance through a fully automated, decentralized interlocking scheme where the trains are moving in safe blocks. These blocks are controlled track-side by radio block centers (RBC) which are control centers to supervise and control train movements in a territory with radio based train control. One RBC is responsible for a fixed number of track segments and the trains currently on these track segments. The RBC grants movement authorities for trains to freely move ahead until so called *end of movement authority* (EoA) points. At each time there exist a certain EoA for each train, which is typically the end of a track segment, the position of a possibly unsafe point (e.g., a rail-road crossing), or the end of a train driving ahead. The granted movement authority defines a safety block surrounding the train. It is a moving block system which means that the signaling system will clear the track behind a train continuously.

This protocol is completely modelled in Stateflow and consists mainly of four states running in parallel. The *rbc_req* state shown in detail in Fig. 1 is responsible for the communication to the RBC. The *rbc_req* state is entered in

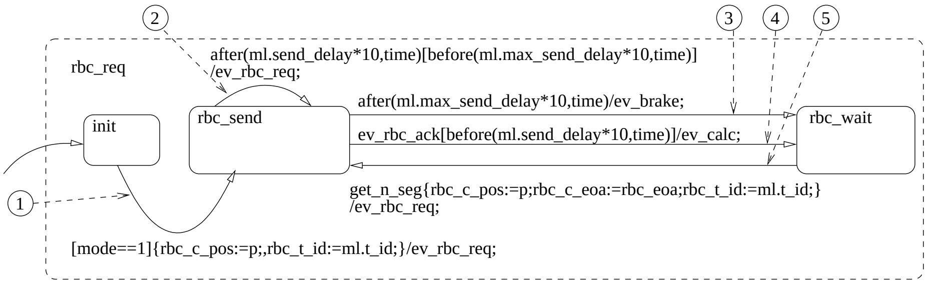

2
3
4
5
rbc_req
after(ml.send_delay*10,time)[before(ml.max_send_delay*10,time)]
/ev_rbc_req;
after(ml.max_send_delay*10,time)/ev_brake;
init
rbc_send
ev_rbc_ack[before(ml.send_delay*10,time)]/ev_calc;
rbc_wait
1
get_n_seg{rbc_c_pos:=p;rbc_c_eoa:=rbc_eoa;rbc_t_id:=ml.t_id;}
/ev_rbc_req;
[mode==1]{rbc_c_pos:=p;,rbc_t_id:=ml.t_id;}/ev_rbc_req;

Fig. 1. RBC communication control part

the *init* state. This state will be left by taking the transition ① only if the variable *mode* is equal to 1 which means that this train is under the supervision of the

ETCS. After enabling this transition and before entering the destination state *rbc_send*, the variables *rbc_c_pos* and *rbc_t_id* will be set to the current position p of the train and to the specific *id* of the requesting train. By taking this transition a request for a new EoA will be sent to the RBC and a timer will be started by entering the *rbc_send* state. After the time interval of a normal transmit action but before the upper bound of transmit actions has been reached, the request will be transmitted again. This behaviour is modelled in the transition on top ② of the *rbc_send* state. If no message arrived before the maximum send delay period is reached the service brake will be initialized by taking the transition ③ and generating a brake event *ev_brake* which will be consumed in the *moved* state in Fig. 6. If an acknowledge arrived before the max send delay the transition ④ will be enabled and an *ev_calc* event is generated, leading to the destination *rbc_wait* state. This state will be left after consuming a *get_n_seg* event and enabling the transition ⑤ to the *rbc_send* state. This transition is a model of a request for travelling to a new track segment after the current EoA which will be sent to the RBC. The parameters of this request are the current train position *rbc_c_pos*, the current EoA *rbc_eoa* of the train and the train id *t_id*. The RBC will then calculate the new EoA with respect to the current train position of train *t_id*, the current EoA of this train and the current positions of possible other trains moving ahead.

The second state is the *com_cross* state (Fig. 2) containing the communication model between the train and a rail-road crossing. If an *ev_com_cross* event is consumed the transition ① will be taken and a lock request will be sent to the rail-road crossing. If the level crossing has transmitted the safe state message before the train has reached the start-of-communication-point-to-rbc *x_c*, the transition ④ is taken, generating a request for a new EoA to the RBC. The variable *c_m* will be set to 1 which means that this train has already sent a lock message to the rail-road crossing. The new state is the *safe* state. If the train is behind the start-of-communication-to-rbc point and receives an unsafe

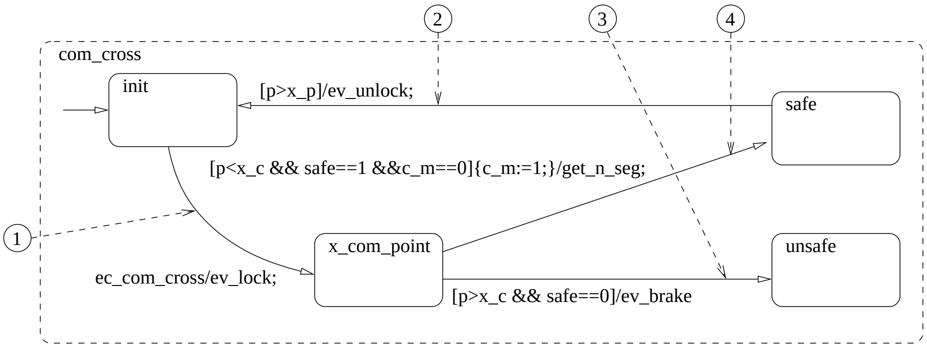


Fig. 2. Level crossing communication control part

message from the rail-road crossing the transition ④ is enabled and the service brake will be initiated by an *ev_brake* event. The *unsafe* state will only be left by switching off the automatic mode of this train. The *safe* state will be left if the train position is behind the position of the rail-road crossing and an unlock message is sent to the rail-road crossing ②. The new state is the *init* state and this train is ready to initiate a new request to a rail-road crossing.

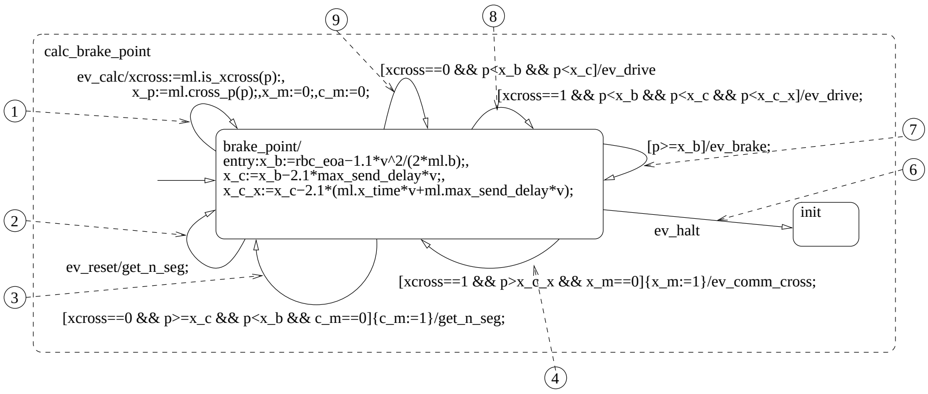

9
8
calc_brake_point
ev_calc/xcross:=ml.is_xcross(p):,
x_p:=ml.cross_p(p);,x_m:=0;,c_m:=0;
[xcross==0 && p<x_b && p<x_c]/ev_drive
[xcross==1 && p<x_b && p<x_c && p<x_c_x]/ev_drive;
1
7
brake_point/
entry:x_b:=rbc_eoa−1.1*v^2/(2*ml.b);,
x_c:=x_b−2.1*max_send_delay*v;,
x_c_x:=x_c−2.1*(ml.x_time*v+ml.max_send_delay*v);
[p>=x_b]/ev_brake;
6
init
2
ev_halt
ev_reset/get_n_seg;
[xcross==1 && p>x_c_x && x_m==0]{x_m:=1}/ev_comm_cross;
3
[xcross==0 && p>=x_c && p<x_b && c_m==0]{c_m:=1}/get_n_seg;
4

Fig. 3. Monitoring of safe motion

The main calculations to guarantee the safe motion of the train are done in the third state *calc_brake_point* shown in Fig. 3. The three variables x_b, x_c and x_c_x are updated every time the *brake_point* state is entered. To guarantee that the train stops before reaching the EoA, the train has to initialize the service brake some distance in front of the EoA. This distance depends on the current velocity of the train and the deceleration induced by the service brake. The point to initialize the service brake (x_b) is dynamically calculated every time step by

$$x\_b = EoA - \frac{1.1 \cdot v^2}{2 \cdot b} \tag{12}$$

where v is the actual velocity of the train and b is the typical deceleration of the service brake. The factor 1.1 guarantees a 10% safety margin. Typically, the train should not stop at every EoA, so the train has to ask the RBC for a new EoA before reaching the actual EoA. For comfort reason the new EoA should be received before the train has switched to the braking mode so the request has to be sent early enough in time before reaching the service brake initialization point x_b. The train will travel the distance $p = v \cdot t$ in time t with the velocity v. The delay for the request of the new EoA is two times the maximal send delay to the RBC plus the response time to serve this request. The point to initialize the request for the new EoA can then be calculated by

$$x\_c = x\_b - 2.1 \cdot max\_send\_delay \cdot v \tag{13}$$

In case that the EoA is a rail-road crossing the train has to lock the rail-road crossing before the train will reach this point. The train has to initiate a lock request to the rail-road crossing and the rail-road crossing has to lock the crossing and acknowledge the lock request. After receiving a *safe* message the train can send a request for a new EoA to the RBC. The point to initialize a lock request to a level crossing is calculated by

$$x\_c\_x = x\_c - 2.1 \cdot (x\_time + max\_send\_delay) \cdot v \tag{14}$$

The time to set up the rail-road crossing in a safe state is stored in the *x_time* variable. These three points (*x_b*, *x_c* and *x_c_x*) are updated every time the *brake_point* state is entered. A spatial view of this scenario is shown in Fig. 4, and

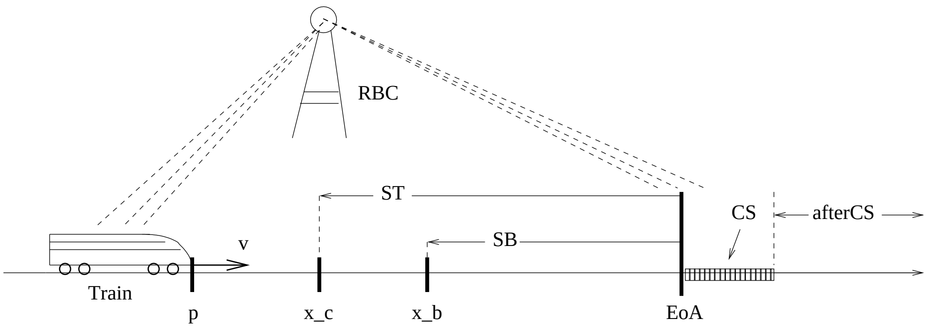


Fig. 4. Radio-based train control

a snapshot of the dynamical behaviour can be seen in Fig. 5. After explaining the main ideas to guarantee a safe motion, we continue to discuss Fig. 3. The transition ① is enabled after receiving an end-of-authority message from the RBC. By taking this transition the two variables which count the messages to the RBC and to the rail-road crossing are initialized to 0. The variable *xcross* picks up the information if a rail-road crossing is just in front of the current position *p* of the train. The position of the rail-road crossing itself is stored in the *x_p* variable. The information of the rail-road crossing is read out of the track data dictionary. If there is no rail-road crossing ahead and the current position of the train is before the point to initialize the service brake and before the point to initialize an EoA-request the transition ⑨ will be taken and an *ev_drive* event is generated to switch into the driving mode of the train. In case there is a rail-road crossing ahead the transition ⑧ will be enabled. If the train has passed the point to send an EoA-request to the RBC but in front of the *x_b* point the transition ③ will be taken and a *get_n_seg* event is generated to initialize the request of a new EoA. If there is a rail-road crossing ahead and the train has passed the *x_c_x* point a lock request is generated by the *ev_com_cross* event released by the transition ④. If the train has passed the *x_b* point the transition ⑦ is enabled which leads to the service brake mode by the *ev_brake* event. In

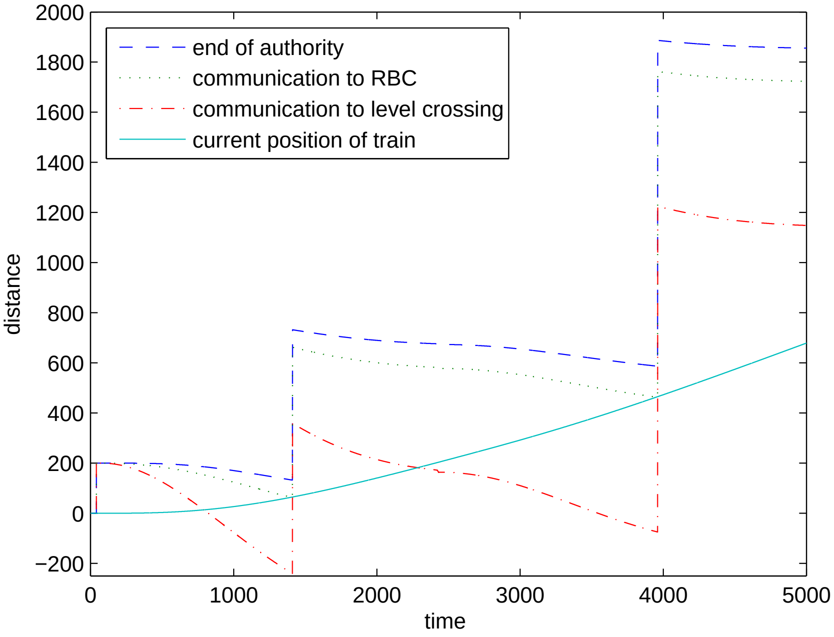


Fig. 5. Snapshot of dynamic calculations

case of an emergency halt indicated by the *ev_halt* event the *brake_point* state will be left by enabling the transition ⑥ to the *halt* state.

The supervision of the velocity of the train is modelled in the fourth state labelled *move* (Fig. 6). The previous value of the desired speed is set to 0 on entering the default *init* state and stored in the *o_v_d* variable. The drive mode is switched on by receiving the *ev_drive* event enabling the transition ① and the variable *d_c* (for drive control) is set, the desired speed at the current position of the train is read out of the track data dictionary and stored in *v_d* and the slope of the current track segment is stored in the variable *slope*. The destination state is *change*.

- If the current desired speed is different from the previous value the transition ② is enabled and this change is signalled through a reset on the *c_v_d* variable, state *switch* and the transition ③. The new value of the current desired speed is stored in the *move* state.
- If there is no change in the desired speed, the transition ④ is enabled and the *move* state is entered.
- If a brake event (*ev_brake*) occurs in the *change* state, the transition ⑫ is enabled, the drive mode is switched off ($d\_c = 0$), the service brake mode is switched on ($sb\_c = 1$) and finally the *init_brake* state is entered.

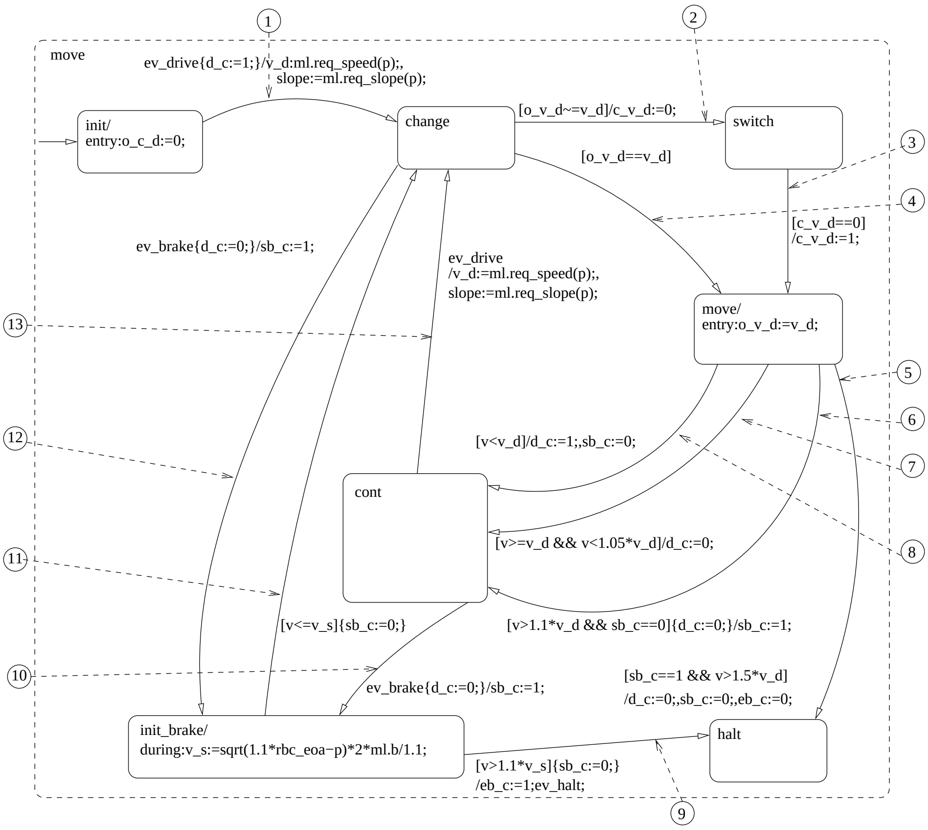


Fig. 6. Speed control

After reading the desired speed the current speed of the train is supervised in the *move* state.

- If the current velocity of the train is below the desired value the transition ⑧ is enabled, the drive mode is set, and the service brake mode is switched off.
- If the current velocity is equal or greater than the desired velocity but not greater than 5% of the desired velocity the transition ⑦ is taken and the drive mode is switched off.
- If the speed is greater than 10% of the desired speed and the service brake mode is not active, the mode is switched from driving mode to the the service brake mode by activating the transition ⑥.

  The destination state in all three cases is the *cont* state. This state is left either by receiving an *ev_drive* event taking the transition ⑬ to the *change* state and updating the slope and the desired speed variable or receiving an *ev_brake* event ⑩ and switching to the *init_brake* state.

- If in the *move* state the service break mode is active and the current speed is greater than 50% of the desired speed, then the emergency brake mode is switched on by taking the transition ⑤ to the *halt* state.

While in the *init_brake* state, the braking curve is supervised by updating the speed *v_s* at the current position. If the current speed is lower than this calculated speed the service brake is switched off ⑪ and the speed is supervised in state *change*. If the current speed is above the braking curve the emergency brake is switched on ⑨ and the *halt* state is entered.

The monitoring of the current velocity and the monitoring of the current EoA are done in parallel every time step.

3 A Verification Methodology for Cooperating Traffic Agents

This section proposes a verification methodology supporting current industrial practice in designing complex safety critical systems. We aim to exploit the typical layered structure in a model based design process of such systems to decompose the overall verification problem of establishing collision freedom of traffic agents into sub-verification problems which are within the range of automatic verification tools for subclasses of hybrid systems. While jointly the different sections of this paper will demonstrate the feasibility of this approach, we do not provide a consistent theory, but rather open a research direction which so far has gained little attention: the challenge of bridging the gap between design layers of safety critical systems. This gap is a direct consequence of the roles models of cooperating systems situated at different layers play in design processes:

- Models at the *cooperation layer* are focusing on how agents agree in finding strategies to resolve possible hazardous situations potentially leading to a collision, and demonstrating that such strategies are indeed capable of avoiding the collision. Strategies at this design level define trajectories to be followed by traffic agents, such as circular go-around, changing lanes, decelerating until a safe distance is achieved. The realization of such strategies is delegated to subsequent design steps – strategies are described directly in terms of dynamic models assuming direct control of speed and acceleration, and undisturbed immediate knowledge of location, speed, and acceleration.
- Models at the *control layer* provide a first step towards realization of strategies, in separating between control and plant, identifying sensors and actuators, and developing control laws ensuring stability and safety of such strategies. The focus at this stage is on getting the control laws right – typically assuming an ideal execution engine, which provides immediate visibility of sensor changes and impact of actuator settings, in a dense time model.
- Models at the *design layer* must deal with the inherent limitations of digital implementations of controllers. This includes the limited observability of the planned at defined sampling points, discretization errors, delayed impact

of actuator settings, physical distribution of sensors, controllers, and actuators, as well as addressing diagnostic and fault-tolerance. Idealized control laws must be made robust, in the sense that stability and safety must be guaranteed in spite of such impurities.

While such a layered approach is highly beneficial for a separation of concerns in design processes, the inherent differences between models situated at different layers make it a challenge to provide semantic bridges between these, as would be required for a complete verification methodology spanning all three levels.

Indeed, models at the control layer would hardly be able to exactly realize the trajectories prescribed by strategies at the cooperation layer. While highly elaborated plant models are available, e.g., capturing car dynamics, control designers typically work with simplified models which have been proven to be practically sufficient for validating stability and safety. Such simplified models ignore higher-order effects and use linear approximations whenever reasonably possible, trading exactness for simulation speed. Similarly, the induced limitations of digital control (cf. [3]) make it impossible for digital controllers to enforce the plant dynamics of control models; rather, robustness is designed into digital controllers to "sufficiently" approximate such dynamics.

Industrial design processes cater for these gaps, e.g., by enforcing design for robustness, re-validating stability and safety at each layer, in particular ensuring complete traceability of safety requirements throughout all design steps, and by rapid prototyping. Extensions to Matlab-Simulink such as the Jitterbug [10] allow for an early assessment of the impact of design level impurities on control-strategies.

However, as argued in Section 8, theoretical approaches to cover multi-layered designs, such as refinement and compositional reasoning, fail to provide semantic bridges across this design space, due to their inability to support the degree of deviations between models tolerated by industrial design processes. We thus leave the development of a theoretical approach building on compositional extensions of robust refinement to future research, and focus in this paper on a verification methodology which adds mathematical rigor to industrial approaches to bridge the gap between design layers. Specifically, we enhance established practices of providing full traceability for safety requirements in offering techniques of assigning responsibility of derived safety requirements jointly guaranteeing collision freedom to subsystems, and provide formal verification techniques tailored to the particularities of model classes at each level, to formally verify such delegated safety requirements. Regarding stability of local control, as well as stability of design models, we provide automated formal techniques establishing various notions of stability. The remainder of this section outlines the overall approach, which is then refined in individual sections.

Our verification methodology addresses the cooperation layer by formalizing design patterns for collision avoidance as a proof rule reducing collision freedom to locally dischargeable safety requirements on individual subsystems of the involved traffic agents. The design patterns builds on the following central concepts (see Section 4):

– Each agents is seen enclosed in a safety envelope, which must not be entered by other agents;
– A *criticality function* measures for each configuration of the (physical) state of involved agents its closeness to collisions;
– Thresholds of this function are chosen taking into account the dynamics of traffic agents to initiate negotiations on agreeing on strategies, as well as on their initiation;
– Strategies are reducing criticality of the agents state.

These design patterns have been proven to be expressive enough to cover rail, automotive, as well as avionics applications. A key feature of our approach to the verification of the coordination level is the capability to automate the generation of candidate criticality functions for linear strategies, by using linear matrix inequality (LMI) solvers to instantiate parameters in a suitably chosen quadratic generic form for candidate criticality functions taking into account the dynamics of the strategies. Moreover, the application of the design pattern generates safety requirements for the subsystem of the traffic agents responsible for inter-agent communication as well as strategy realization.

Such safety requirements are discharged at the control-level using symbolic reachability analysis (see Section 7). We have tuned the verification algorithms to cater for control models with non-trivial discrete control (e.g., resulting from the interaction between inter-agent coordination and control). This calls for a fully symbolic representation of the hybrid system state space in reachability analysis – in contrast to the explicit representation of discrete states used in hybrid system verification tools such as PHAVer [21], Checkmate [49], HyperTech [27]. The key to achieve this are recent results to lift techniques for compact discrete state space representations based on And-Inverter Graphs to the theory of linear arithmetic (Lin-AIGs) required to deal with hybrid system verification. The current prototype of our verification engine [11] uses substitution in backward image computation along jumps, providing for linear guards and linear expressions in assignments, and Loos-Weispfennig quantifier elimination for backward image computations at flows, and redundancy elimination for sets of linear-constraints, assuming linear hybrid automata. Future work will lift this extension to include plant models supporting linear differential equations. To cater for the transition to design models, we re-verify the safety requirements allocated to this subsystem, now using an abstraction refinement approach addressing discrete time reachability analysis for models with linear dynamics [48].

Stability of control models is demonstrated using LMI based candidate generation for Lyapunov functions for hybrid systems with linear differential equations, developed in [41]. As discussed in Section 6, we also show that stability can be re-proven after discretizing the system model with a given sampling rate, resulting in a discrete-time hybrid system with linear difference equations. This allows for the identification of safe sampling rates maintaining stability, which can in turn be used for discrete-time reachability analysis.

Jointly, these techniques allow a formal verification of stability and safety properties, with traceability of requirements from the coordination layer to design models.

4 Verification of the Cooperation Layer

In [13] we proposed a rule that decomposes the proof of the global property of collision avoidance of two traffic agents into simpler properties with the aim that they can be automatically verified. In this section we give a summary of this proof rule and show that an important ingredient of this rule, the criticality function, can indeed be found automatically. This is illustrated with the train case study.

4.1 A Design Pattern for Collision Avoidance

Proving the global safety property of collision freedom for a collection of traffic agent is extremely difficult because each traffic agent is (modelled by) a hybrid system with a number of (discrete) modes and a different continuous dynamics in each mode. To break down the complexity of this verification task we exploit that traffic agents typically cooperate using a certain pattern of operation modes that can be described as a generic *phase-transition diagram* shown in Fig. 7.

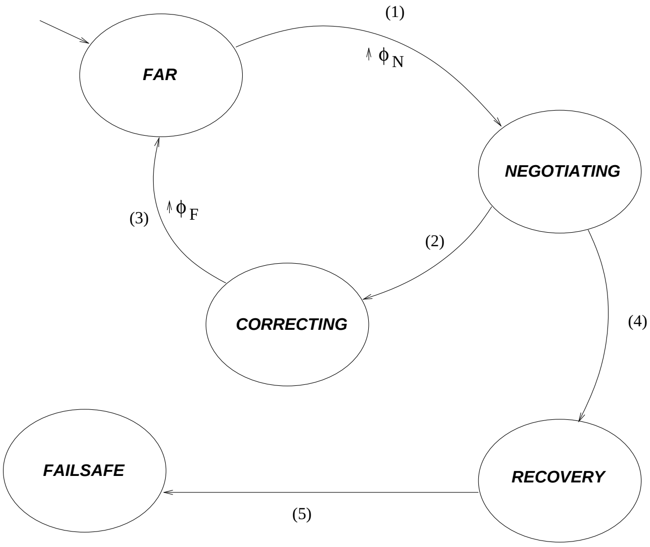


Fig. 7. Phase-transition diagram for proof rule

The phase *FAR* collects all controller modes that are not pertinent to collision avoidance. The protocol may only be in phase *FAR* if it is known to the controller that the two agents are "far apart". Determining conditions for entering and leaving phase *FAR* is thus safety critical. The *NEGOTIATION* phase is initiated as soon as the agents might evolve into a potentially hazardous situation. Within the negotiation phase the two agents determine the set of maneuvers to be performed. The *CORRECTING* phase is entered when matching correcting modes have been identified. During this phase, maneuvers associated with the correcting modes will cause the distance between traffic agents to increase, eventually allowing them to reenter the *FAR* phase. For instance, TCAS distinguishes maneuvers like “descent”, “maintain level”, and “climb” for aircrafts.

The cycle of transitions numbered (1) to (3) in the diagram thus characterizes successful collision avoidance maneuvers. Other phases and transitions shown in Fig. 7 increase the robustness of the protocol, by providing recovery actions in case of failures (e.g., disturbed communication channels) in the *NEGOTIATION* phase, and can only be offered by agents with fail-safe states (like trains). For instance, in its *RECOVERY* phase a train may initiate a braking maneuver to avoid a collision with a preceding train.

Stipulating the pattern in Fig. 7, we proposed a generic proof rule that decomposes the global safety proof of collision freedom (for the case of two traffic agents) into a number of simpler properties that involve only parts or limited aspects of the agent system. The proof rules employs two key concepts: a *safety envelope* surrounding each traffic agent and a *criticality function* providing an abstract measure of the distance between the traffic agents. The rule states that for all traffic agents A and all criticality functions cr satisfying the verification conditions VC of the rule, collision freedom is guaranteed:

$$A \models \mathrm{VC} \quad \Rightarrow \quad A \models \neg\ \mathit{collision}$$

where cr may occur in VC but not in the *collision* predicate. In an application we have to show that the verification conditions VC are satisfied by the concrete traffic agents A_0 and the concrete criticality function cr_0 substituted for cr:

$$A_0 \models \mathrm{VC}[cr_0/cr]$$

Thus the proof rule, when instantiated with A_0 and cr_0, yields the desired property of collision freedom:

$$A_0 \models \neg\ \mathit{collision}.$$

In Subsection 4.3 we show that criticality is a *Lyapunov-like function* and that (for certain dynamics) the concrete function cr_0 can be discovered automatically.

Formalization. Now we outline the formalisation of the approach as given in [13]. A traffic agent A is represented as the parallel composition of a plant P and a controller C, in symbols $A = P \mid\mid C$. Each of these components is modelled by a hybrid automaton $H = (\mathbb{M}, \mathit{Var}, R^d, R^c, m_0, \Theta)$ defining trajectories

$$\pi = (\ \hat{M},\ (\hat{X})_{X \in \mathit{Var}}\ )$$

where $\hat{M} : \mathit{Time} \rightarrow \mathbb{M}$ and $\hat{X} : \mathit{Time} \rightarrow \mathbb{R}$ for $X \in \mathit{Var}$. For details see Appendix A and for an example see Fig. 9.

To specify behavioural properties over time of hybrid automata and state the verification conditions of our proof rule for collision freedom, we use the State Transition Calculus [56], an extension of the Duration Calculus (DC) [57]. In DC, real-time interval properties of system observables *obs*, which are interpreted as functions $\mathit{obs}_{\mathcal{I}} : \mathit{Time} \rightarrow \mathit{Data}$, can be expressed and proven. For example,

$$\Box(\lceil M = \mathit{NEG} \rceil \Rightarrow \lceil \mathit{acc} = 0 \rceil)$$

expresses that in every interval ($\Box$) if the mode observable M has the value *NEG* (for *negotiating*) throughout this interval ($\lceil M = \mathit{NEG} \rceil$) then the observable *acc* (for *acceleration*) is zero throughout this interval ($\lceil \mathit{acc} = 0 \rceil$).

The State Transition Calculus can additionally express properties of instantaneous transitions. For example, $\uparrow M = \mathit{NEG}$ is true at $t \in \mathit{Time}$ if then the truth value of the assertion $M = \mathit{NEG}$ switches from false to true. In DC, this can be expressed as $\lceil \neg(M = \mathit{NEG}) \rceil;\ \lceil M = \mathit{NEG} \rceil$ which is to hold in an interval surrounding t. The chop operator ; is applied at time t and expresses the concatenation of two intervals where $\lceil \neg(M = \mathit{NEG}) \rceil$ holds in the first one and $\lceil M = \mathit{NEG} \rceil$ in the second one.

It can be defined that a hybrid automaton H is a *model* of a formula F, abbreviated as $H \models F$.

Correcting Modes. For each controller C_i we stipulate a set $\mathit{COR}(C_i)$ of *correcting maneuvers*. Then we assume a relation

$$\text{MATCH} \subseteq \mathit{COR}(C_1) \times \mathit{COR}(C_2)$$

of *matching modes* characterizing which pairs of maneuver are claimed to resolve hazardous states. For each pair $(m_1, m_2) \in \text{MATCH}$ there is an *activation condition* characterized by a state assertion $\Phi(m_1, m_2)$. Activation conditions of maneuvers must observe the following constraints:

- *timely activation*: the activation of the maneuvers occurs early enough to guide the traffic agents to a safe state using the associated control laws.
- *completeness*: for each possible approach to a hazardous state there is at least one matching pair of correcting modes whose activation condition is enabled in time.

For ground-based traffic agents like trains there is a special class of corrective modes, enforcing a complete stop of the traffic agent, thus reaching a *fail-safe state*. We refer to such corrective modes as *recovery modes* and assume that there is a single matching pair (r_1, r_2) of recovery modes.

Let Φ_{start} be the disjunction of all activation conditions of these modes:

$$\Phi_{\mathit{start}} \;\Leftrightarrow\; \Phi(r_1, r_2) \vee \bigvee\nolimits_{(m_1, m_2) \in \text{MATCH}} \Phi(m_1, m_2).$$

Intuitively, if any of these conditions becomes true, a hazardous state has been reached, which can compensated by the associated matching pair of correcting modes. By completeness of the set of activation conditions, Φ_{start} thus characterizes all hazardous states.

The flexibility of having multiple matching correcting modes entails the need for a *negotiation phase*, in which agents agree on which pair of maneuvers is to be activated. The design of this phase has to address the following critical issues:

- *limited time window*: the decision must be reached within a certain time Δ_N, catering for latencies occurred by inter-agent communication, as well as local computation times to perform the selection.
- *timely activation*: the activation of the negotiation phase must be early enough to guarantee timely activation of the chosen maneuvers.
- *adequacy of selection*: the negotiation phase may only choose among such matching pairs whose activation condition is known to become activated.

We cater in our generic scheme for the latter two items by the concept of *warnings* "announcing" that the activation condition for a matching pair will become true in Δ_C time units, with $\Delta_N < \Delta_C$. The warning Φ_N causing the *initiation of the negotiation phase* should thus be given as soon as it is known that the agent will in Δ_C time units hit one of these activation conditions. Formally, this is expressed as follows: $\Phi_N \Leftrightarrow pre(\Delta_C, \Phi_{start})$.

Safety Envelopes. We define collision freedom as maintaining disjointness of safety envelopes associated with each traffic agent. A safety envelope of an agent is a vicinity. Formally, safety envelopes are convex subspaces of $\mathbb{R}^3$ surrounding the current position, whose extent can depend on the valuation of plant variables.

Definition 1. *The* safety envelope *of an agent $A = P \parallel C$ is a continuous piecewise differentiable function*

$$\mathtt{SE}_A : \mathbb{R}^{Var(P)} \to \mathcal{P}(\mathbb{R}^3),$$

which is a convex subset of $\mathbb{R}^3$ including the current position. Given a run π, and a point in time t, the current safety envelope *is given by $\mathtt{SE}(\pi(t))$.*

Two traffic agents A_1 and A_2 are collision free if in all trajectories of the composed traffic system $A_1 \parallel A_2$ the safety envelopes associated with $A_1 = C_1 \parallel P_1$ and $A_2 = C_2 \parallel P_2$ have an empty intersection.

Definition 2. *Consider a run π of $A_1 \parallel A_2$. The state assertion* collision *holds in π at time $t \in Time$ if $\mathtt{SE}_{A_1}(\pi(t)) \cap \mathtt{SE}_{A_2}(\pi(t)) \neq \varnothing$. The two-agent system $A_1 \parallel A_2$ is* collision free *if $A_1 \parallel A_2 \models \lceil \neg collision \rceil$, i.e. if for all runs of $A_1 \parallel A_2$ and all intervals $\neg collision$ holds.*

Criticality. A central notion is that of *criticality* of plant states. Given valuations σ_1 and σ_2 of the plant variables of P_1 and P_2, respectively, the criticality $cr(\sigma_1, \sigma_2)$ measures the "distance" of (σ_1, σ_2) from unsafe states. A key property of such a criticality function is the separation of safe and unsafe states, in the following sense: whenever the criticality of plant states is below a fixed threshold c_{safe} the plant state is safe.

Formally, a *criticality measure* for given traffic agents $A_1 = P_1 \parallel C_1$ and $A_2 = P_2 \parallel C_2$ is a continuous piecewise differentiable function

$$cr : \mathbb{R}^{Var(P_1)} \times \mathbb{R}^{Var(P_2)} \longrightarrow \mathbb{R}_{\geq 0}$$

satisfying the implication $cr(\sigma_1, \sigma_2) < c_{safe} \;\Rightarrow\; \mathtt{SE}_{A_1}(\sigma_1) \cap \mathtt{SE}_{A_2}(\sigma_2) = \varnothing$.

A Proof Rule for Collision Freedom. For two cooperating traffic agents $A_1 = C_1 \parallel P_1$ and $A_2 = C_2 \parallel P_2$ the proof rule of [13] has the form

$$\frac{\textbf{(VC 1)}, \ldots, \textbf{(VC 18)}}{C_1 \parallel P_1 \parallel C_2 \parallel P_2 \models \lceil \neg \mathit{collision} \rceil}$$

where the verification conditions **(VC 1)** . . . **(VC 18)** require only verification tasks of the following types:

(A) off-line analysis of the dynamics of the plant in fixed modes,
(B) mode invariants for $C_1 \parallel C_2$,
(C) real-time properties for C_j,
(D) local safety properties, i.e., hybrid verification tasks for $C_j \parallel P_j$.

In the following we give a flavour of these verification conditions.

Criticality and safety. Our approach to establishing collision freedom is to reduce this to an analysis of the criticality of the system. The criticality measure separates safe from unsafe states: here a constant c_{safe} of type $\mathbb{R}_{>0}$ represents the level of criticality below which the two travel agents are *safe*, i.e., in no danger of a collision. The following type A verification condition checks this property.

(VC 1) Safety

$$Th(\mathbb{R}) \models cr < c_{safe} \;\Rightarrow\; \neg\ \mathit{collision}$$

This yields $P_1 \parallel P_2 \models \lceil cr < c_{safe} \rceil \Rightarrow \lceil \neg\ \mathit{collision} \rceil$.

Phase-transition diagram. It is straightforward to generate verification conditions enforcing compliance of the concrete protocol to the phase-transition system of Fig. 7. To this end, the phases *far away, negotiating, correcting, recovery* and *fail-safe* of the controllers C_i are represented as disjoint subsets $FAR(C_i)$, $NEG(C_i)$, $COR(C_i)$, $REC(C_i)$, $FSA(C_i) \subseteq \mathbb{M}_i$ of the set of modes. When specifying the behaviour of the controllers C_i in DC, we use $FAR(C_i)$ as a shorthand for $M(C_i) \in \mathit{FAR}(C_i)$ and analogously for the other phases. Then we check the following simple type B verification condition.

(VC 2) Controllers observe phase-transition diagram. For $i = 1, 2$

$$C_i \models_0 \Phi_{phase}(C_i)$$

Thus the controllers satisfy the above phase constraints from the start. Here $\Phi_{phase}(C_i)$ is a conjunction of formulae of the following type:

$$\begin{array}{lll} \textit{Initial phase}: & \lceil\rceil \vee \lceil \mathit{FAR}(C_i)\rceil;\ \text{true} & \text{for } i = 1, 2 \\ \textit{Phase sequencing}: & \lceil \mathit{FAR}(C_i)\rceil \longrightarrow \lceil \mathit{FAR}(C_i) \vee \mathit{NEG}(C_i)\rceil & \text{for } i = 1, 2 \\ & \ldots\ldots\ldots\ldots\ldots\ldots \text{etc} \ldots\ldots\ldots\ldots\ldots & \end{array}$$

Warnings. The following type A verification condition checks that each trajectory leading from a plant state without warning to a collision must cross an activation condition of one of the correcting modes, i.e., it checks whether the set of provided maneuvers is *complete.*

(VC 4) Completeness of maneuvers.

$$P_1 \parallel P_2 \models (\lceil \neg\phi_N\rceil;\ \text{true}) \wedge (\lceil \neg \mathit{collision}\rceil;\ \uparrow \mathit{collision}) \ \Rightarrow\ \Diamond \uparrow \Phi_{start}$$

The activation conditions for maneuvers must be chosen in such a way that criticality is below the critical threshold when maneuvers become enabled, leading to the following type A verification condition.

(VC 5) During warning period criticality is still low.

$$\mathit{Th}(\mathbb{R}) \models \forall\, \delta \leq \Delta_C : (\mathit{acc}_1 = \mathit{acc}_2 = 0 \wedge \uparrow \mathit{pre}(\delta, \Phi_{start}) \Rightarrow \mathit{cr} < c_{safe})$$

This yields $P_1 \parallel P_2 \models (\lceil \mathit{acc}_1 = \mathit{acc}_2 = 0\rceil \wedge \ell = \Delta_C;\ \uparrow \Phi_{start}) \Rightarrow \lceil \mathit{cr} < c_{safe}\rceil$. In particular, this ensures that maneuvers are not activated too late.

Negotiation phase. The negotiation phase must be initiated as soon as a first warning occurs. Recall that this event is represented by Φ_N becoming true. The following type B verification condition checks this.

(VC 6) Initiating negotiating phase. For $i = 1, 2$

$$C_i \parallel P_1 \parallel P_2 \models \lceil \mathit{FAR}(C_i)\rceil \xrightarrow{\uparrow\Phi_N} \lceil \mathit{NEG}(C_i)\rceil$$

Note that both controllers enter their negotiating phase simultaneously when the trigger $\uparrow \phi_N$ occurs.

The following type D verification condition guarantees that $\mathit{pre}(\Delta_C, \Phi_{start})$, the warning to start maneuvers according to Φ_{start}, was raised early enough. If the traffic agents changed their speeds during negotiation and selection (sub-) phase, the calculations of the warning would be wrong.

(VC 11) No acceleration during negotiation and selection. For $i = 1, 2$

$$C_i \parallel P_i \models \lceil \mathit{NEG}(C_i) \vee \mathit{SEL}(C_i)\rceil \Rightarrow \lceil \mathit{acc}_i = 0\rceil$$

The last type C verification condition for the negotiation phase checks, that indeed negotiation is completed within the given time window of length $\varDelta_N$.

(VC 12) Negotiation completes in time.

$$C_1 \parallel C_2 \models \lceil NEG(C_1) \vee NEG(C_2) \rceil \Rightarrow \ell \leq \varDelta_N$$

where $\varDelta_N < \varDelta_C$. Thus *both* controllers have left their negotiating phase after at most $\varDelta_N$ time units.

Adequacy. The following type A verification condition ensures that the criticality does not increase when collision avoidance maneuvers are activated. Note that these are part of the hybrid automata resulting from the restriction of the controller to the selected correcting mode.

(VC 15) Adequacy of matching modes. For all $(m_1, m_2) \in \mathrm{MATCH}$

$$\begin{aligned} & C_1 \upharpoonright m_1 \parallel P_1 \parallel C_2 \upharpoonright m_2 \parallel P_2 \ \models \\ & \qquad \uparrow \varPhi(m_1, m_2);\ \mathrm{true} \ \Rightarrow \ \forall\, c \in \mathbb{R}_{\leq 0} : \ \lceil cr \leq c \rceil \longrightarrow \lceil cr \leq c \rceil \end{aligned}$$

For the recovery maneuver, a similar verification conditions additionally requires that after a suitable braking time t depending on their speeds at the start of the maneuver, the traffic agents have come to a complete halt.

Far away phase. The following type B verification conditions enforces that the correcting phase can be left in favour of the phase *far away* only when there is no warning that new correcting maneuvers have to start in $\varDelta_C$ time.

(VC 17) Termination of maneuvers. For $i = 1, 2$ and $\phi_F \Leftrightarrow \neg\phi_N$

$$C_i \parallel P_1 \parallel P_2 \models \lceil COR(C_i) \rceil \xrightarrow{\uparrow \varPhi_F} \lceil FAR(C_i) \rceil$$

Fail-safe state. The following type B verification condition ensures that the recovery maneuver is concluded by entering the fail-safe state, when the agent has come to a complete stop. By **(VC 2)**, each traffic agent stays in this state. We require that in this state the traffic agent does not change its position, and that the criticality does not increase.

(VC 18) Fail-safe state. For $i = 1, 2$

$$\begin{aligned} & C_i \parallel P_i \models \lceil REC(C_i) \rceil \xrightarrow{\uparrow (spd_i = 0)} \lceil FSA(C_i) \wedge spd_i = 0 \rceil \\ & C_i \parallel P_1 \parallel P_2 \models \\ & \qquad \forall\, c \in \mathbb{R}_{\geq 0} : \lceil FSA(C_i) \wedge spd_i = 0 \wedge cr \leq c \rceil \longrightarrow \lceil spd_i = 0 \wedge cr \leq c \rceil . \end{aligned}$$

In [13] the following was shown.

Theorem 1 (Soundness). *The verification conditions* **(VC 1)**,...,**(VC 18)** *together imply*

$$C_1 \parallel P_1 \parallel C_2 \parallel P_2 \models \lceil \neg collision \rceil ,$$

i.e., the proof rule for collision freedom is sound.

4.2 Case Study: Movement Authority

Let us revisit the ETCS train control introduced in Subsection 2.4. However, instead of using Matlab-Simulink and Stateflow as modelling techniques, we shall now represent the scenario more abstractly by time-dependent observables and hybrid automata. We consider one train moving along a track and communicating with a radio block center (RBC) that grants movement authorities to the train. At each moment of time there is a certain end of movement authority (EoA) for the train because after the EoA a critical section begins, which may be a rail-road crossing or a track segment occupied by a preceding train (cf. Fig. 4).

We start from domains $\mathit{Position} = \mathbb{R}_{\geq 0}$ with typical element p for the position of the train on the track, $\mathit{Speed} = \mathbb{R}_{\geq 0}$ with typical element v for the speed of the train, and $\mathit{Acc} = \mathbb{R}$ with typical element a for the acceleration of the train. Let v_{max} denote the maximal speed of the train and $-b$ the *braking force* of the train, represented as a negative acceleration, i.e., with $-b < 0$. The current *end of authority* for the train is modelled by an observable

$$\mathit{EoA} : \mathit{Time} \to \mathit{Position}$$

which is maintained by the RBC. We require

$$\forall\, t_1, t_2 \in \mathit{Time} : t_1 \leq t_2 \Rightarrow \mathit{EoA}(t_1) \leq \mathit{EoA}(t_2).$$

The *critical section CS* behind the *EoA* is represented by an interval of positions

$$[\mathit{CS}.s, \mathit{CS}.e] \subseteq \mathit{Position}$$

starting at $\mathit{CS}.s$ and ending at $\mathit{CS}.e$, with a fixed positive length $\mathit{CS}.e - \mathit{CS}.s$. A predicate describing all positions *after the critical section* is

$$\mathit{afterCS} : \mathit{Position} \to \mathbb{B} \quad \text{with} \qquad \forall\, p \in \mathit{Position} : \mathit{afterCS}(p) \Leftrightarrow \mathit{CS}.e \leq p.$$

When the train approaches the current EoA it has to *start talking* to the RBC to get permission to extend the EoA. The distance relative to EoA where start talking has to be initiated is modelled by a function

$$\mathit{ST} : \mathit{Speed} \times \mathit{Time} \to \mathit{Position}$$

depending on the train's speed, the maximal time delay needed to communicate with the RBC, and implicitly on the fixed braking force $-b$. If the permission is not granted by the RBC the train has to *start braking* with the braking force $-b$. The distance relative to EoA where the braking has to be initiated is modelled by a function

$$\mathit{SB} : \mathit{Speed} \to \mathit{Position}$$

depending on the train's speed and implicitly on the fixed braking force $-b$. These positions and distances are illustrated in Fig. 4. We require

$$\forall\, v \in \mathit{Speed}, \mathit{\Delta} \in \mathit{Time} : \mathit{ST}(v, \mathit{\Delta}) \geq \mathit{SB}(v).$$

Safety. For the train's safety envelope $\mathtt{SE}_{Train}(p)$ we choose an extension around its current front position p which encompasses the length of the train, independent of mode and speed:

$$\mathtt{SE}_{Train}(p) = [p - L_T, p] \subseteq Position$$

where L_T is the length of the train. The critical section's safety envelope depends on the position of the *EoA*:

$$\mathtt{SE}_{CS} = \begin{cases} \varnothing & \text{if } CS.e < EoA \\ [CS.b, CS.e] & \text{otherwise} \end{cases}$$

The choice of $\varnothing$ as the extension of the safety envelope caters for the case that the RBC has granted an extension of the *EoA* beyond the critical section. This permits the train to pass the critical section without safety violation.

We define $inCS(p) \Leftrightarrow CS.s \leq p \leq CS.e + L_T$. The predicate *inCS* describes all positions where the safety envelopes of the train and the critical section overlap, i.e.,

$$\mathtt{SE}_{Train}(p) \cap \mathtt{SE}_{CS} \neq \varnothing \;\; \Leftrightarrow \;\; inCS(p) \wedge EoA \leq CS.e.$$

Thus *collision freedom* is equivalent to $inCS(p) \Rightarrow CS.e < EoA$, i.e., whenever the front position p is in the critical section, the *EoA* has been extended beyond the critical section.

Traffic agents. The scenario movement authority is modeled as a system *MA* with two traffic agents:

$$MA = \text{Train} \,\|\, \text{RBC},$$

one train consisting of plant and controller interacting with an RBC consisting of a controller only. Fig. 8 shows how these agents are represented by real-valued variables *pos* (position), *spd* (speed), *acc* (acceleration) and *EoA* (end of authority), and which (other) variables the four components share for communication with each other.

We assume that the *train plant* has knowledge of its position on the track and controls its speed depending on requests from the train controller. It will react to speed control commands from the train controller. Thus we consider the variables below. We do not distinguish between the (syntactic) variables of the automaton and the corresponding trajectories in runs. So we take for the type of a variable the type of its time-dependent trajectory, and we permit variables with discrete ranges without explicitly coding them in reals.

Variables:		Train plant
input	$sc : Time \to \{Keep, Brake\}$	(speed control)
output	$pos : Time \to Position$	(position of the train)
	$spd : Time \to Speed$	(speed of the train)
	$acc : Time \to Acc$	(acceleration of the train)

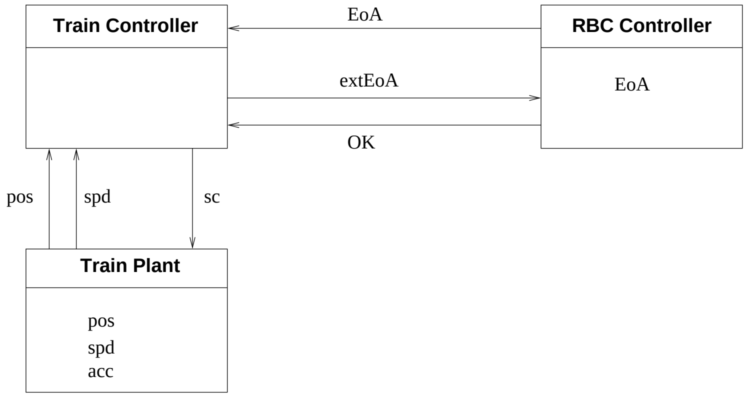


Fig. 8. Communication between train and RBC

For the dynamics of the train we assume the continuous transition relations $pos^\bullet = spd$ and $spd^\bullet = acc$ and the invariants $-b \leq acc$ and $spd \leq v_{max}$. Here we are interested only in the change of speed during braking:

$$acc = \begin{cases} 0 & \text{if } sc = \mathit{Keep} \vee (sc = \mathit{Brake} \wedge spd = 0) \\ -b & \text{if } sc = \mathit{Brake} \wedge spd > 0 \end{cases}$$

The *train controller* monitors the position and speed of the train. When approaching the current end of authority *EoA* (guarding a critical section) it requests for an extension from the RBC by sending an *extEoA* signal. If the RBC sends a signal *OK* the controller requests the train plant to keep the (desired) speed. If the RBC does not reply in time and instead the train passes the position *SB* the controller forces the train plant to brake. Thus the train controller has the following time dependent variables.

Variables:		Train controller
input	$pos : \mathit{Time} \to \mathit{Position}$	(position of the train)
	$spd : \mathit{Time} \to \mathit{Speed}$	(speed of the train)
	$EoA : \mathit{Time} \to \mathit{Position}$	(current EoA)
	$OK : \mathit{Time} \to \mathbb{B}$	(EoA is extended)
local	$CS.s : \mathit{Time} \to \mathit{Position}$	(begin of critical section)
output	$extEoA : \mathit{Time} \to \mathbb{B}$	(request to extend EoA)
	$sc : \mathit{Time} \to \{\mathit{Keep}, \mathit{Brake}\}$	(speed control)
Modes:	**Far, Appr, SafeAppr, Braking, FailSafe**	

The dynamics of the train controller is described by the automaton in Fig. 9. Initially, the controller is in the mode **Far**. When the predicate Φ_N abbrevi-

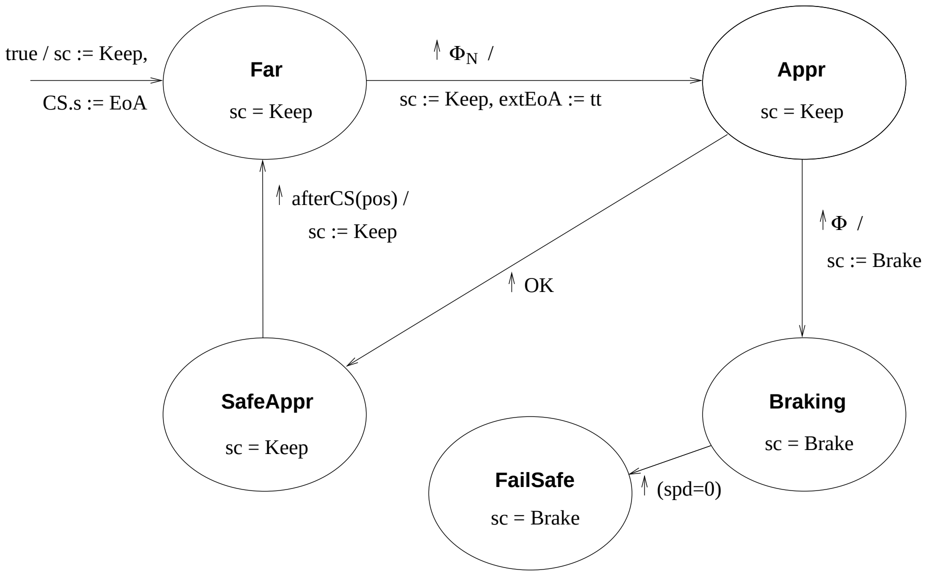


Fig. 9. Train controller

ating $pos \geq EoA - ST(spd, \Delta_C)$ becomes true the controller switches to the mode **Appr**. On occurrence of a signal OK the controller switches to the mode **SafeAppr** indicating that the train can safely approach the critical section. In this mode the train continues to keep its speed. If the predicate Φ abbreviating $pos \geq EoA - SB(spd)$ becomes true the controller switches to the mode **Braking** where it forces the train to brake until a complete stop. If the train's speed is zero, the controller enters the mode **FailSafe**. In the terminology of Fig. 7, the mode **Appr** is the phase *NEGOTIATION*, **SafeAppr** is *CORRECTING*, and **Braking** is *RECOVERY*.

The RBC is modelled only as far as the communication concerning the extension of EoA is concerned. It outputs of current EoA to the train and if requested to extend it by an $extEoA$ signal may grant an OK signal. Thus the *RBC controller* has the following time dependent variables.

Variables:		RBC controller
input	$extEoA : Time \to \mathbb{B}$	(request to extend EoA)
local	$x : Time \to Time$	(clock)
output	$EoA : Time \to Position$	(current EoA)
	$OK : Time \to \mathbb{B}$	(EoA is extended)
Modes:	**Idle, Check, Unsafe**	

The dynamics of this simplified RBC controller is described by the automaton in Fig. 10. The expression *update*(*EoA*) abbreviates an assignment of a new, larger value to the variable *EoA*. The clock x with upper bound ε in mode **Check** models the maximum delay it takes for the RBC to answer the request for extending the *EoA*.

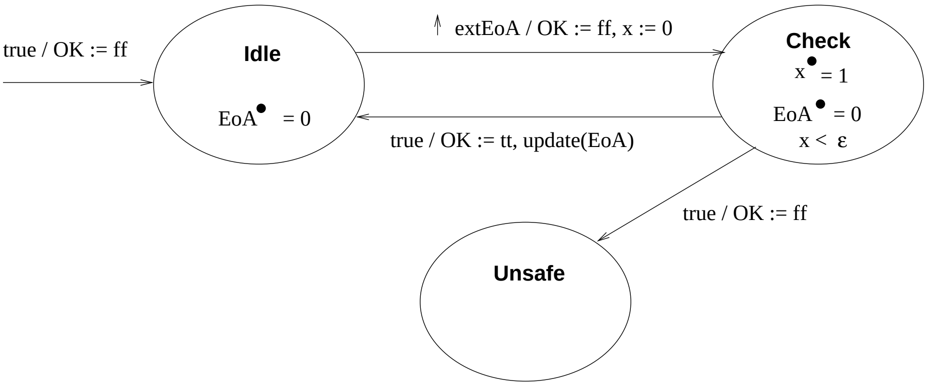


Fig. 10. RBC controller

4.3 Automatic Discovery of the Criticality Functions

It is critical for a system according to Fig. 7 that a recovery maneuver will always lead into a fail-safe state without violating any safety constraints. To ensure this, recovery needs to be initiated in time, so that potentially hazardous situations can be avoided. For the train example given in the previous sections, we will now demonstrate how to determine states which lead to a safe recovery maneuver. In this particular case we will ensure that the train will always come to a stop before an end-of-authority point associated with a critical section. In particular, we will construct a predicate Φ guaranteeing that the train system is safe in the sense that no critical section can be passed, unless the RBC sent the signal *OK* to the train passing it. In other words, once the train system enters the **Braking** mode, the safety condition $pos \leq EoA$ will not be violated and the train will come to a stop in the **FailSafe** mode — braking is always initiated in time.

We will now show that the criticality function in the verification conditions from Subsection 4.1 can be seen of an instance from a generic class Lyapunov-like functions. Methods for synthesis of Lyapunov functions can then be adapted to automatically compute a suitable criticality function. Since contour lines of the function can be used to separate reachable and non-reachable states, we call this class *Lyapunov-like boundary functions.*

Definition 3. *Let $x(t) \in \mathbb{R}^n$ be a hybrid system's* state vector *(the vector of the valuations of all variables* [8]*) at time t. Given a set of initial states vectors $S \subseteq \mathbb{R}^n$ and a set of unsafe state vectors $U \subseteq \mathbb{R}^n$, a* Lyapunov-like boundary function *of the hybrid system is a function $V : \mathbb{R}^n \to \mathbb{R}$, such that:*

- *for all runs of the system and all reachable states $x \in \mathbb{R}^n$:*

  $$V^\bullet(x) := \tfrac{dV}{dx}\tfrac{dx}{dt} \leq 0$$

- $\exists\, k \in \mathbb{R} : (x \in S \Rightarrow V(x) < k) \wedge (x \in U \Rightarrow V(x) > k)$

The function V has Lyapunov-like properties, as it will never increase throughout the evolution of any trajectory due to the condition $V^\bullet(x) \leq 0$, which forces the function's time derivative to be non-positive. Furthermore, there exists a contour line, given by the points x with $V(x) = k$, such that the possible initial states S lie on one side of this line, while the unsafe states U lie on the other (see Fig. 11). Due to the Lyapunov-like property it is then impossible for a trajectory beginning in the set of initial states to cross into the unsafe region, as this would require an increase of $V(x)$.

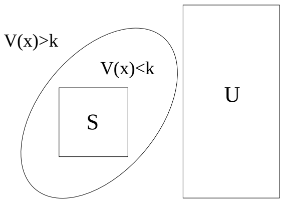


Fig. 11. Criticality function contour line with initial set S and unsafe set U

Since such a Lyapunov-like criticality function is a variant of a Lyapunov function, computational approaches for Lyapunov function synthesis can be adapted for this case. For instance, linear matrix inequalities can be employed to *automatically* compute a suitable quadratic V, and then the maximal k such that $x \in U \Rightarrow V(x) > k$. The computation procedure is very similar to the one that will be described in detail in Section 6.

Such a Lyapunov-like boundary function is a special case of a criticality function as described in Subsection 4.1. The function V can be used as criticality function cr and the contour line value k represents the maximal admissible criticality value c_{safe}. Setting $cr = V$ and $c_{safe} = k$, the verification condition **(VC 1)** is fulfilled since $(x \in U \Rightarrow V(x) > k)$ implies $(x \in U \Rightarrow V(x) \geq k)$, which is equivalent to **(VC 1)** by contraposition. For condition **(VC 5)**, in the

[8] For the ease of mathematical treatment, the state of the system is represented as a *vector* of real numbers, instead of a function $\sigma : Var \to \mathbb{R}$ like in Subsection 4.1.

case of $\delta = 0$, the set S assumes the role of $pre(0, \Phi_{start})$. The condition $x \in S$ means that x is an admissible state vector for initiating the maneuver, which is equivalent to the requirement that the variables at time of initiation fulfill $pre(0, \Phi_{start})$. If $\delta > 0$, a backward reachability computation is needed to show that $V(x) < k$ for the entire negotiation period. Since verification condition **(VC 5)** requires an acceleration of zero during negotiation, this simplifies the computation. Condition **(VC 15)** is implied by the Lyapunov-like condition $V^{\bullet}(x) \leq 0$ stating that V cannot increase over time.

Therefore, a criticality function as needed in **(VC 1)**, **(VC 5)** and **(VC 15)** can be computed automatically, using methods for Lyapunov function synthesis. The dynamics for the given tuple of maneuvers is needed as an input, as is the set of unsafe states. Condition **(VC 4)** needs to be checked separately, since a Lyapunov-like function does not guarantee that the set $\{x \mid V(x) < k\}$ is always entered before a trajectory can pass into the unsafe region U. It is then possible to synthesize a Lyapunov-like boundary function (serving as the criticality function) and a contour line value k (serving as the maximal admissible criticality level) such that initiating the maneuver with criticality lower than k guarantees safety. Each admissible set of initial state vectors S for the maneuver corresponds to a possible safe condition for the maneuvers.

For the rail-road crossing case study, will now employ Lyapunov-like boundary functions to identify a safe guard Φ, such that $pos \leq EoA$ is always guaranteed. Therefore we put $U = \{pos > EoA\}$. As Φ is not given, but to be derived, we define $\Phi := V(x) < k$. All states with this property are separated from U by the contour line $V(x) = k$.

Since a system can potentially have many admissible criticality functions, this even holds for any state within a contour line of *any* criticality function with respect to the same unsafe region U. Therefore, we are not restricted to one function, but can use many. The predicate Φ is then the disjunction of the predicates $V_i(x) < k$ for all such criticality functions V_i and associated contour line values k_i. Using many criticality functions instead of one can result in a weaker, and therefore less conservative, predicate Φ.

For the case study, it was sufficient to use just one criticality function, as the use of several functions brought no significant improvement. As a result we obtained the following criticality function cr and boundary value c_{safe}:

$$cr = 0.0014616 * (pos - EoA + 2000)^2 + spd^2 \tag{15}$$

$$c_{safe} = 5846.445 \tag{16}$$

Figure 12 shows the position of the train in meters before the EoA point on the horizontal axis and its velocity in m/s on the vertical axis. The shaded set of states is safe set $\{x \mid V_i(x) < k_i\}$. Initiating the braking within this set guarantees that the unsafe region to the right of the vertical line cannot be entered. For this particular example, where the speed is decreasing at a fixed rate, this implies an eventual transition to the **FailSafe** phase, without breaching any safety requirements. Furthermore, assuming a maximal speed $v_{max} = 76.46 m/s$, condition **(VC 4)** is also fulfilled, since system trajectories could not enter the unsafe region without first passing through the ellipsoid. Any predicate Φ which evaluates to false everywhere outside this set is admissible as a guard for the transition between the *Appr* and *Braking* modes.

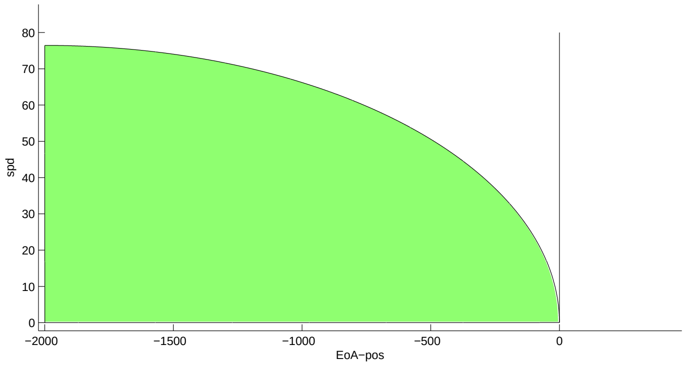


Fig. 12. Safe region for initiating the braking

5 Parameterized Verification of the Cooperation Layer

In this section, we present results for verifying parameterized instances of traffic protocols. On the one hand, system safety in systems like ETCS crucially depends on the right choice of parameter values. For instance, whether a train can keep its speed safely depends on the relationship of EoA to the current velocity v and maximum braking power b. If these values are imbalanced then the train protocol is no longer guaranteed to avoid crashes. Hence, it is utterly important to analyze and discover safety constraints between such parameters or state variables and adjust design parameters in accordance with those parametric constraints.

On the other hand, once those constraints have been discovered, all instances of the traffic scenario that satisfy the parametric safety constraints can be verified at once. Generally, safety statements do not only hold for a particular setting but generalise to a broader range of possibilities. For instance, train control is not only safe for a particular initial speed $v \leq 10$ and a specific braking force $b = 0.1$ with remaining EoA-distance of $5km$. Instead, the system remains safe for *all* choices of parameters that satisfy a corresponding constraint. Using our techniques from $[43, 45, 44, 46]$, all such instances of the system can be verified at once, and the required safety constraints on the free parameters can be discovered.

5.1 Parameterized Hybrid Systems

Parameters naturally arise from the degrees of freedom of how a part of the system can be instantiated or how a controller can respond to input. They include both external system parameters like the braking force b of a train, and design parameters of internal choice like SB, i.e., when to start braking before approaching EoA in order to ensure that the train cannot possibly run into an open gate or preceeding train.

The major challenge in dealing with symbolic parameters is that they lead to nonlinearities: Even comparably simple flow constraints like $2b(EoA - p)$ become nonlinear when b is considered as a symbolic parameter rather than instantiated with some specific value like 0.1.

To handle parameters, we follow a fully symbolic deductive approach. We have introduced a logic, $\mathsf{d}\mathcal{L}$, for verifying hybrid systems with parameters and a corresponding verification calculus [43]. It generalizes dynamic logic from the discrete case [23] to hybrid systems. Our $\mathsf{d}\mathcal{L}$ calculus can be used both for verifying correctness statements about parametric hybrid systems and for deriving constraints on their free parameters that are required for safety [43]. Thus, with $\mathsf{d}\mathcal{L}$, it is possible to zoom in to a subset of the system, typically at the coordination layer, and find safety constraints for the parameters.

5.2 Technical Approach: Differential Logic

To illustrate how our techniques for parameterized hybrid systems work, we provide a short survey of the $\mathsf{d}\mathcal{L}$ principles. The full details of the theory behind $\mathsf{d}\mathcal{L}$ are reported in [43, 45, 44, 46].

The logic $\mathsf{d}\mathcal{L}$ provides modal formulae like $[MA]\phi$, which express that all runs of the parametric hybrid system MA (see Subsection 4.2) lead to states which satisfy some safety constraint ϕ. Further, such formulae can be combined propositionally, by quantifiers, or modalities $[\beta]$ about other automata β. With this, safety of parametric hybrid systems can be stated as formulae in the logic $\mathsf{d}\mathcal{L}$, for instance:

$$b > 0 \land \epsilon \geq 0 \Rightarrow [MA](p \leq EoA) \ . \tag{17}$$

This $\mathsf{d}\mathcal{L}$ formula states that all runs of the hybrid system MA are such that the train position p remains within the movement authority EoA, provided that the braking force b is non-zero and the maximum reaction-cycle-time is some $\epsilon \geq 0$. Both symbols, b and ϵ, are train model parameters and given externally. Their values depend on the specific characteristics of the actual train and should be handled symbolically for a thorough analysis of all trains. From the perspective of a single train automaton, EoA can also be considered as an external parameter. In the full system MA, which involves trains and RBCs, it can also be considered as a state variable instead.

Using the $\mathsf{d}\mathcal{L}$ calculus, such a formula can be analyzed systematically in order to find out if it holds or under which parameter constraints the system is safe. For instance, for the safety constraint (17), the $\mathsf{d}\mathcal{L}$ calculus reveals that the system is only safe when the initial velocity does not exceed the braking capabilities and the control parameters are chosen in accordance with the movement authorities, speed, and reaction times.

To make our calculus compositional and simplify its step-wise symbolic processing principle, we use a textual notation of hybrid automata as *hybrid programs* [43]. As hybrid automata [25] can be embedded in hybrid programs by a simple canonical construction [43], we identify hybrid automata and their corresponding program rendition, here. With this embedding, parametric safety statements can be easily expressed using d$\mathcal{L}$ formulae of the form (17) and analyzed in the d$\mathcal{L}$ calculus.

Given a safety statement like (17), the d$\mathcal{L}$ calculus performs a symbolic analysis of the parametric hybrid system and identifies safety constraints on the free parameters. Figure 13 contains a corresponding abbreviated proof outline for a part of the system analysis in the d$\mathcal{L}$ calculus. At this point, we only want to remark that the proof starts at the bottom with the full *MA* controller and splits into sub-goals that symbolically analyze a part of the ETCS behavior each. For instance, the left branch analyzes the train behavior in the recovery mode, the right branch investigates acceleration cases, see [43] for details. The calculus works by successive symbolic decomposition of the system, which can be understood to follow a symbolic case analysis. As a basis, our implementation uses an integration of the KeY prover [5, 4] with quantifier elimination in Mathematica for arithmetic reasoning about the continuous dynamics.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dots \vdash v^2 \leq 2b(EoA{-}p)}{\dots \vdash \dots}}{\psi, EoA{-}p{<}SB \vdash [a := -b][\mathit{drive}]\psi}}{\psi, EoA{-}p{<}SB \vdash [\mathit{recover}][\mathit{drive}]\psi} \qquad \dfrac{\dfrac{\dfrac{\dots \vdash SB \geq \frac{v^2}{2b} + \varepsilon v \dots}{\dots \vdash \dots}}{\psi, EoA{-}p{\geq}SB \vdash [a{\leq}a_{max}][\mathit{drive}]\psi}}{\psi, EoA{-}p{\geq}SB \vdash [\mathit{accel}][\mathit{drive}]\psi}}{\psi \vdash [\mathit{nego}][\mathit{drive}]\psi}}{\psi \vdash [MA]\psi}}{\vdash \psi \Rightarrow [(MA)^*](p \leq EoA)}$$

Fig. 13. Proof outline for ETCS protocol in d$\mathcal{L}$

5.3 Analysis of Parameters in ETCS Protocol Phases

In the d$\mathcal{L}$ calculus, we can derive constraints on the parameters for a safe operation of train control. These parameters are limits in the ideal-world model of the coordination level, hence, general engineering principles advise using additional safety margins to compensate for inaccuracies and disturbances.

From an analysis of the braking behavior in recovery mode, we can automatically determine a *controllability constraint* for the train, see [43]. If the following constraint is violated, no safe control of the train is possible at all, because its speed exceeds the braking power b for the remaining movement authority $EoA - p$:

$$v^2 \leq 2b(EoA - p) \ . \tag{18}$$

Assuming that condition (18) holds, it remains to show that the particular train control choices maintain safety. Especially, the controllers must maintain (18) invariably during all possible driving behavior.

The two most crucial control parameters for the cooperation protocol MA in ETCS are ST and SB. Both are design parameters of internal choice by the controllers and, thus, need an adequate instantiation to ensure safety. The parameter ST determines when the train enters negotiation mode to ensure that it can get an EoA-extension from the RBC before reaching EoA. The control parameter SB is the safety distance at which the speed supervision needs to initiate braking when no positive EoA extension has occurred yet (recovery mode). Both parameters are formulated as points on the track in terms of distances from EoA (see Fig. 4).

The parameter SB is a very important safety parameter that needs to be chosen adequately such that the train can guarantee to remain within its movement authority, regardless of the behavior of other traffic agents like preceeding trains or gates at critical sections as mediated by the RBC agent. Especially, if SB is chosen right, the system remains safe, whatever the outcome of the RBC communication may be.

The safety constraint for parameter SB can be derived from an analysis of the hybrid program rendition of the MA-automata using a proof of the form in Fig. 13, see [43] for details. In addition, the underlying RBC and train models bridge the gap from cooperation layer models to design layer models as they take maximum controller response times into account. Similar to the notion of lazy hybrid automata [51], we account for the fact that controller implementations react with a processing delay and that the effect of actuators like brakes can be delayed as well.

An acceleration $a \leq a_{max}$ is permitted in case $EoA - p \geq SB$, when adaptively choosing SB depending on the current speed v and the parameters of maximum braking force b and maximum speed supervision response time ϵ in accordance with the following constraint:

$$SB \ \geq \ \frac{v^2}{2b} + \left(\frac{a_{max}}{b} + 1\right)\left(\frac{a_{max}}{2}\varepsilon^2 + \varepsilon v\right) \ . \tag{19}$$

This constraint expresses that it is only safe to keep on driving when the controllability constraint (18) is maintained even after a maximal acceleration of a_{max} during a maximum period of ϵ time units. In particular, constraint (19) makes the controllability constraint (18) inductive.

Observe that constraint (19) is a refined and parameterized version of the (12) (remember that x_b is the point on the track corresponding to the distance SB from EoA). The actual symbolic constraints in (19) identify what needs to be captured by the 10% safety margin in (12). It also clearly identifies under what conditions a 10% safety margin is sufficient. Likewise, constraint (19) explains the shape of the safety region given in Fig. 12 and gives insights about a systematic symbolic generalization of the numerical criticality function in (15). It identifies fully symbolic constraints as opposed to specific real numbers that only hold for a particular scenario.

Parameter ST is a liveness parameter. Depending on the expected maximum RBC communication latency L, which again is a parameter for the train analysis, it ensures that the RBC can still respond in time before the train needs to decelerate. That is, when the train enters negotiation at ST, it does not need to brake unless an EoA extension cannot be granted by the RBC within L at all. For instance, an RBC may not be able to grant an EoA extension despite an early request because other traffic agents occupy the track segment beyond EoA.

Constraints on the parameter ST can be derived [45] from an analysis of a single negotiation and correction phase. A proof yields the following necessary constraint depending on the expected maximum RBC communication latency L:

$$ST \geq Lv + \frac{v^2}{2b} \ . \tag{20}$$

Again, (20) corresponds to a version of (13) that has been synthesized from the system model deductively.

The constraints (19) and (20) can be used to find out how dense a track can be packed with trains in order to maximize throughput without endangering safety, and how early a train needs to start negotiation in order to minimize the risk of having to reduce speed when the MA is not extendable in time.

6 Proving Stability of Local Control and Design Models

Stability is a property of a dynamic system that subsumes its ability to withstand, and eventually compensate for, outside disturbances that affect a system. For a local closed-loop control system, this is a very desirable property, because stability ensures that the controller is actually able to keep the controlled parameter close to the desired value. Furthermore, if one requires asymptotic stability, there cannot be any undamped oscillations or cyclic behavior in the closed-loop system. For instance, one would expect from a speed controller for a train, that it forces the speed to converge toward a desired value, without producing needless cycles of acceleration and deceleration. Very little controller activity should be needed, once the train is close to this desired speed. In this section, we will apply methods based on the concept of *Lyapunov-functions* [35] to the speed controller of the train model from Section 2. Lyapunov functions are functions that map each system state onto a nonnegative real value. For every run of the system, the sequence of values this function attains is required to be decreasing, eventually converging to zero at the desired control point. If a function with these properties is found, then the system is *asymptotically stable*. We will detail how methods for automatic computation of these functions can be applied to a model of a speed controller.

Definition 4. *Consider a continuous-time dynamic system with state vector* $x \in \mathbb{R}^n$*. Let* $x(t)$*,* $t \geq 0$*, denote its state at time* t *during a run of the system. The system is called* globally asymptotically stable *if the following two properties hold for all possible runs:*

a) $\forall\, \epsilon > 0 \quad \exists\, \delta > 0 \quad \forall\, t \geq 0 : \| x(0) \| < \delta \Rightarrow \| x(t) \| < \epsilon$ (stability)
b) $t \to \infty \Rightarrow x(t) \to 0$ (global attractivity)

If a) and b) hold only on a bounded set containing 0*, the system is called* locally asymptotically stable*.*

Without loss of generality we assume that the origin of the continuous state space $\mathbb{R}^n$ is the equilibrium point all trajectories converge to. If one wants to show asymptotic stability with respect to a different equilibrium – as is the case in the drive train example – the state space of the hybrid system can simply be “shifted” to move this point into the equilibrium.

Intuitively, the stability property guarantees that there is an upper bound on how far the system can stray from the equilibrium point, depending on its initial state. Moreover, the global attractivity property tells us that the system will eventually converge to the equilibrium point. Together, this implies that there is an upper bound on the temporary change of state a disturbance can cause, relative to the size of the disturbance, and that eventually the system will have compensated for the disturbance.

We will consider hybrid systems with a finite number of discrete modes. With each mode m, we associate an affine differential equation $x^\bullet = A_m x + b_m$ with $A_m \in \mathbb{R}^{n \times n}, b_n \in \mathbb{R}^n$ for describing the continuous evolution of the system's state variables. A possible transition between a pair of modes m_1 and m_2 is given as a quantifier-free first-order predicate over the continuous variables. No discrete updates of continuous variables are allowed. We also allow for an invariant in each mode, given by a quantifier-free first-order predicate[9] on the continuous variables. The system may only stay in a mode while its invariant is satisfied. We assume that the system does not exhibit Zeno or blocking behavior, so that all trajectories are continuous and unbounded in time (cf. Appendix A).

Since the state space of such a hybrid system is $\mathbb{R}^n \times \mathbb{M}$, the cartesian product of the continuous and discrete state (mode) space, one is usually interested in local stability. The invariants specify which continuous states can be active with which modes – combinations violating the invariants need not be considered. Therefore the stability property is local as defined by the invariants. Furthermore, we only expect the continuous variables to converge, but for all permissible initial hybrid states $(x(0), m(0))$.

For systems of this kind, local asymptotic stability can be shown with the help of a *common Lyapunov function.* It is defined as follows (see [29, 9]).

[9] In principle, any quantifier-free predicate over the continuous variables is admissible for mode transitions or invariants. If the resulting invariant set is not a convex polyhedron, it will need to be over-approximated for the actual computation, increasing conservativeness.

Definition 5. *Consider a hybrid system with state vector $x \in \mathbb{R}^n$ and mode $m \in \mathbb{M}$, where $\mathbb{M}$ is the finite set of modes. Assume that the dynamics in mode m are given as $x^\bullet = f_m(x)$ and that the invariant belonging to mode m is the predicate I_m. A* (common) Lyapunov function *for this system is then a function $V : \mathbb{R}^n \to \mathbb{R}$ such that:*

a) $V(x) = 0$ *if* $x = 0$ *and* $V(x) > 0$ *otherwise*
b) for all m: $V_m^\bullet(x) := \frac{dV}{dx}(x) f_m(x) < 0$ *if* $0 \neq x \models I_m$
c) $0 \models I_m \Rightarrow V_m^\bullet(0) = 0$
d) $V(x) \to \infty$ *when* $\| x \| \to \infty$

A Lyapunov function maps each state of the system onto a nonnegative real number, such that the value of the function is decreasing *at all times for all possible trajectories*, eventually converging to zero at the origin of the state space. Condition a) enforces a global minimum of V at 0. Conditions b) and c) imply that V is decreasing over time in every mode, whenever its invariant is true, except at the equilibrium, where $V_m^\bullet(0) = 0$ for all applicable modes. Condition d) is needed to enforce the stability property a) in Definition 4.

Theorem 2 ([9]). *Consider a hybrid system as in Definition 5. The existence of a common Lyapunov function for such a system implies local asymptotic stability for all initial hybrid states that are covered by at least one invariant.*

There are also refinements to the common Lyapunov function approach, using piecewise continuous functions instead [9, 28, 42, 16]. This allows the use of different functions for each mode. However, for the train controller application in this paper, this extension was not necessary. Lyapunov functions can be found automatically via numerical optimization [28, 42, 16]. We will demonstrate this on the following example from the train control context.

6.1 The Drive Train Subsystem

The proof techniques outlined above will now be applied to the drive train part of the train model from Section 2. The drive train is generally active in the *Far* phase of the system when no full braking action is imminent. In this part of the system, the actual velocity of the train should be kept in line with the desired velocity, in the presence of outside disturbances. Furthermore, a change of desired velocity should result in an adequate convergence of the actual velocity towards this new value.

This is achieved by closed-loop control of the drive train via a PI-Controller, i.e. a linear controller with proportional and integral part. This controller takes the difference between current and desired velocity as an input and outputs a current that is used to accelerate/decelerate the train.

In Equations 2-8, all constants and parameters have been instantiated with sensible values, to represent a concrete drive train system. Braking force is assumed constant, as is the environment force F_e. All these equations have then

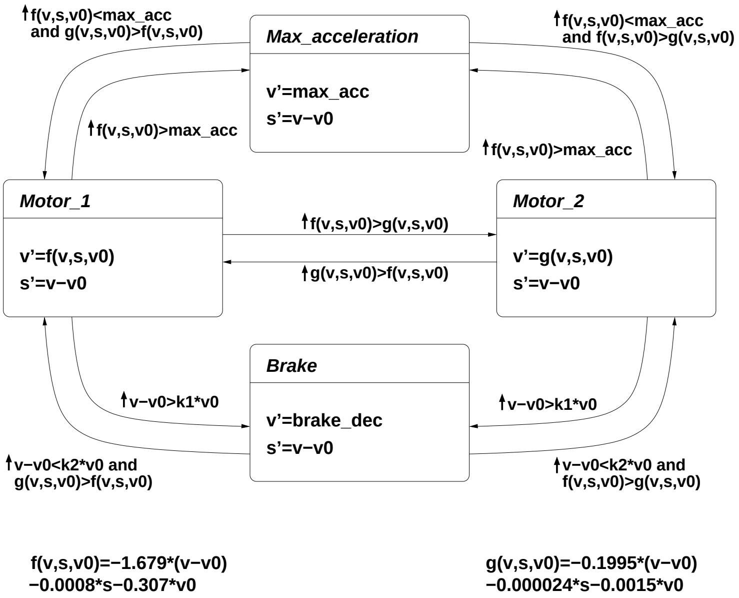

f(v,s,v0)<max_acc and g(v,s,v0)>f(v,s,v0)
Max_acceleration
v'=max_acc
s'=v−v0
f(v,s,v0)<max_acc and f(v,s,v0)>g(v,s,v0)
f(v,s,v0)>max_acc
f(v,s,v0)>max_acc
Motor_1
v'=f(v,s,v0)
s'=v−v0
f(v,s,v0)>g(v,s,v0)
g(v,s,v0)>f(v,s,v0)
Motor_2
v'=g(v,s,v0)
s'=v−v0
Brake
v−v0>k1*v0
v−v0>k1*v0
v'=brake_dec
s'=v−v0
v−v0<k2*v0 and g(v,s,v0)>f(v,s,v0)
v−v0<k2*v0 and f(v,s,v0)>g(v,s,v0)
f(v,s,v0)=−1.679*(v−v0) −0.0008*s−0.307*v0
g(v,s,v0)=−0.1995*(v−v0) −0.000024*s−0.0015*v0

Fig. 14. Hybrid automaton of drive train subsystem

been collapsed into a set of two differential equations per mode, through elimination of superfluous variables and exploitation of variable dependency. The functions f an g are therefore the representation of Equations 2-8 for these fixed values. The three relevant unknowns that remain in the drive train model given in Fig. 14 are the desired speed v_0, the actual speed v and the integral value in Equation 2, denoted as s. Since Equation 2 describes dynamics modelled as the minimum of two affine functions (Equation 1), there are two corresponding modes, *Motor_1* and *Motor_2*, in the closed-loop hybrid system, each with affine dynamics. The mode *Max_acceleration* is used to model the cutoff at maximum acceleration in Equation 3. If the current speed is far beyond the desired speed, we activate the brakes, which are assumed to produce constant negative acceleration. This is represented by mode *Brake.*

6.2 Synthesizing Lyapunov Functions

To compute a function V that fulfills the conditions in Definition 5, we use a fixed parameterized function template: quadratic functions of the form $V(x) = x^T Px, P \in \mathbb{R}^{n\times n}$. In this representation, the parameters are isolated in the symmetric matrix P. This means we have to compute matrix entries for P, such that conditions a) to d) are satisfied.

As detailed in [28, 42], this can be done with the help of *linear matrix inequalities* [8], as long as the differential equations for all modes are affine. Linear matrix inequalities are optimization problems with constraints given as definiteness constraints on matrices. They will be formally defined in the following. Phrasing the problem to find an adequate P as a linear matrix inequality allows the use of convex optimization software like CSDP [7] to identify suitable matrix entries.

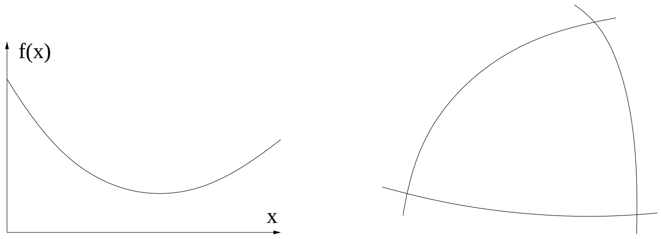
Definition 6. *A matrix $P \in \mathbb{R}^{n \times n}$ is called* positive semidefinite *if $x^T P x \geq 0$ for all $x \in \mathbb{R}^n$. This is also denoted $P \succeq 0$. For given matrices $M_1, \ldots, M_j \in \mathbb{R}^{n \times n}$, a* linear matrix inequality *is a problem of the form:*

Find $x_1, \ldots, x_j \in \mathbb{R}$ such that $x_1 M_1 + \ldots + x_j M_j \succeq 0$.

Define I as the $n \times n$ identity matrix. The problem of finding a Lyapunov function as in Definition 5 corresponds to the following linear matrix inequality [42]. Find P, μ_m^i such that:

$$\begin{aligned} P &\succeq \alpha * I \\ \forall\, m \in \mathbb{M} : A_m^T P + P A_m - \sum_i \mu_m^i\, Q_m^i + I &\preceq 0 \end{aligned}$$

The matrices $Q_m^i \in \mathbb{R}^n$ are the result of the so-called *$\mathcal{S}$-procedure* [55]. They are computed a priori from the invariants I_m such that $I_m \Rightarrow x^T Q_m^i x \geq 0$ for all i. The details of this computation, which only involves basic algebra in the case of polytopic invariants, can be found in [42].



(a) Convex function (b) Convex set

Fig. 15. Convex set and function

Intuitively, this linear matrix inequality can be visualized as follows. Figure 15(b) shows an illustration of the parameter space of the Lyapunov function candidate. Note that the parameter space will generally be high-dimensional (for example 10 dimensions in case of 4 continuous variables, plus the $\mathcal{S}$-procedure variables μ_m^i), so the parameter space for an actual system can not be represented visually in a meaningful way. Each linear matrix inequality constraint bounds the set of feasible Lyapunov functions with a convex (that is, "curving inward", see Fig. 15(a)) constraint, resulting in a convex solution set. Each point in this solution set corresponds to one admissible Lyapunov function for the system, and identifying one is a *convex feasibility problem*, which can be solved with standard nonlinear optimization software [7]. Additionally, it is possible to identify an optimal feasible point, with respect to a convex constraint. This is for instance used to maximize the volume of the ellipsoid or the value of k in Section 4. One can also use this to obtain an estimate on the convergence rate of an asymptotically stable system [42]. As opposed to linear optimization, the optimum will not generally lie on the edge of the feasible set – therefore interior point algorithms [40] are used. Here the convexity of the solution set can be exploited.

6.3 Stability of the Drive Train with Continuous-Time Controller

For the drive train with continuous controller, as described above, the solver CSDP [7] gives the following solution

$$P = \begin{bmatrix} 0.0021 \, 0.0021 \\ 0.0021 \, 8.4511 \end{bmatrix}$$

leading to a Lyapunov function

$$V(v - v_0, s) = 0.0021 * s^2 + 0.0042 * (v - v_0) * s + 8.4511 * (v - v_0)^2.$$

The contour lines of V are visualized in Fig. 16. These contour lines are only passed "outside-in" by all trajectories, resulting in convergence to the center, which represents $v - v_0 = 0$ and $s = 0$. Therefore, the velocity v will converge to the desired velocity v_0 and the integral value s of the PI-controller will converge to 0.

The existence of this Lyapunov function is sufficient to prove global asymptotic stability for the drive train system. Using the YALMIP [33] frontend under Matlab, this computation took around 0.65 seconds. The problem consists of 17 scalar constraints and 6 three-by-three matrix inequality constraints, on a total of 23 scalar variables. Therefore, the convex search space visualized in Fig. 15(b) is 23-dimensional and bounded by $17 + 6 = 23$ constraint surfaces.

6.4 Stability of the Discretized Drive Train

For a time-discretized version of the drive train, stability can be shown in a very similar manner. The discrete-time system is obtained by choosing an appropriate sampling rate. Too slow sampling might destroy stability, while too fast sampling increases the computational cost for proving safety properties (see Section 7). For linear/affine dynamics, the discretized system can then be computed through the matrix exponential $e^{A\tau}$, where τ is the sampling rate and A the matrix representing the dynamics (i.e., $x^{\bullet} = Ax$).

For such a discrete-time hybrid system with dynamics given as difference equations, asymptotic stability can be shown using the same methods as for the

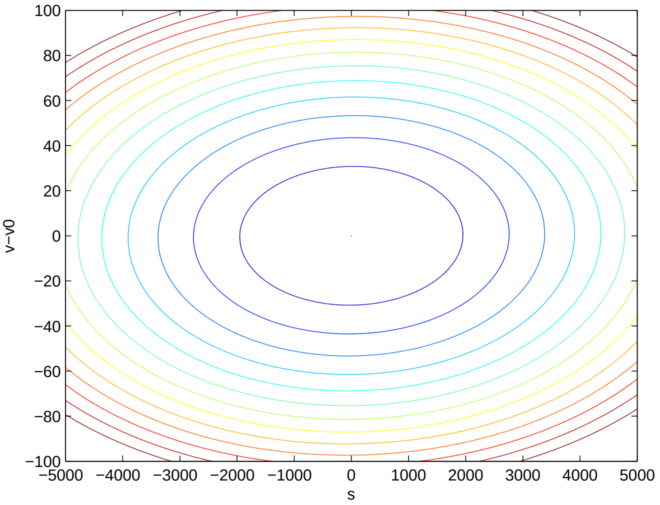


Fig. 16. Lyapunov function contour lines

continuous case [16]. Again a (slightly different) set of LMIs can be obtained and solved through convex optimization. For instance, for a sampling rate of 0.1 seconds, the following Lyapunov function was obtained:

$$V(v - v_0, s) = 0.0105 * s^2 + 0.0172 * (v - v_0) * s + 6.0591 * (v - v_0)^2$$

6.5 Stability of the Sampled-Data Drive Train

Stability analysis of discrete-time hybrid systems can also be used to shed light on stability properties of sampled-data systems, that is control loops with continuous plant and discrete controller. In this case sensor measurements are sent to the controller in periodic intervals. The actuators will also periodically receive updates from the controller.

In case of linear plant dynamics, stability analysis can be conducted on a purely discrete-time system which is obtained by also discretizing the plant via zero-order-hold discretization. This procedure is lossless in case of a non-hybrid linear plant because the state of the plant at each sampling instant can be exactly computed from its state at the previous sampling instant and the current controller output. If the plant is hybrid, but with linear dynamics, the dynamics can still be discretized exactly, but the switches are possibly inexact. With the absence of so-called “grazing switches” [15], it is usually possible to approximate the sampled system closely enough.

For the drive train system, we have performed this kind of analysis for a fixed sampling rate (0.1 seconds) and a discrete controller obtained by a textbook discretization method for linear systems (zero-pole matching transformation [17]). The resulting sampled-data system consists of the continuous-time drive train dynamics given in Subsection 2.1 and the discretized controller, and it can still be proven stable by this method.

7 Proving Safety of Local Control and Design Models

In this section, we present our approach of model checking safety properties of local control and design models of the example. We first outline our general methods for verification of hybrid systems with non-trivial discrete behaviour (Subsections 7.1 and 7.3); then we build both continuous-time and discrete-time models of the system based on its Matlab-Simulink description and show model checking results of these models (Subsections 7.2 and 7.4).

7.1 Model Checking Hybrid Systems with Large Discrete State Spaces

We have proposed an approach for verification of hybrid systems, which contain large discrete state spaces and simple continuous dynamics given as constants [11] (methods dealing with richer dynamics, e.g., given as differential inclusions, are currently under development). Large discrete state space arise naturally in industrial hybrid systems, due to the need to represent discrete inputs, counters, sanity-check bits, possibly multiple concurrent state machines *etc*, which typically jointly with properties of sensor values determine the selection of relevant control laws. Thus this non-trivial discrete behavior cannot be treated by considering discrete states one by one as in tools based on the notion of hybrid automata. We have developed a model checker dealing with ACTL properties for this application class.

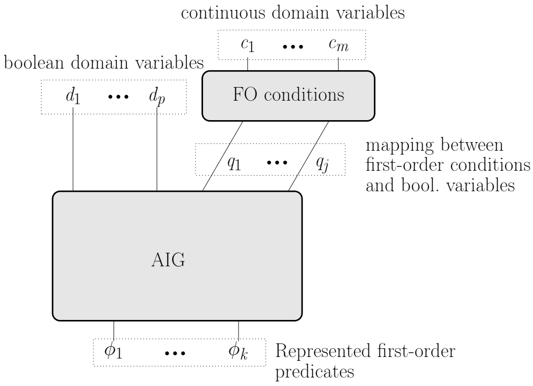

continuous domain variables
c_1 ••• c_m
boolean domain variables
d_1 ••• d_p
FO conditions
q_1 ••• q_j
mapping between first-order conditions and bool. variables
AIG
ϕ_1 ••• ϕ_k
Represented first-order predicates

Fig. 17. The Lin-AIG structure

Representation of state-sets. In our setting, the state-sets of hybrid systems consist of both discrete states, represented by Boolean formulas, and continuous regions, represented by a Boolean combination of linear constraints. We use an extension of And-Inverter-Graphs [39] with linear constraints (Lin-AIGs) as a compact representation format (see Fig. 17). In Lin-AIGs Boolean formulas are represented by Functionally Reduced And-Inverter Graphs (FRAIGs), which are basically Boolean circuits consisting only of AND gates and inverters. In contrast to BDDs, FRAIGs are not a canonical representation for Boolean functions, but they are "semi-canonical" in the sense that every node in the FRAIG represents a unique Boolean function. To be able to use FRAIGs to represent continuous regions, we introduce a set of new (Boolean) *constraint variables* Q as encodings for linear constraints, where each occurring linear constraint is represented by some $q_\ell \in Q$ as illustrated in Fig. 17. Thus we arrive at state-sets encoded by Boolean formulas over Boolean variables and Q, together with a mapping of Q into linear constraints.

Step computation. Our model checker can handle continuous-time models, which contains both discrete transitions and continuous flows. Discrete transitions are given in the form of *guarded assignments*, while continuous flows are given in the form of *modes*, which define the evolution of continuous variables by constants. For each mode there is a boundary condition, the mode is left as soon as the boundary condition is satisfied.

For checking an *invariance* property, the model checker performs a symbolic backward reachability analysis, to ensure that no state in the complement of the property is reachable from the initial state set. The key achievement lies in the capability of reducing this backward analysis to pure substitution. It can be done easily for discrete transitions as detailed in [12]. The capability of representing as well the effect of continuous flows through substitution rests on our ability to perform pre-image computations for arbitrary Boolean combinations of linear constraints using the Loos-Weispfenning quantifier elimination method [34]. In contrast to other verification methods for hybrid systems, this allows us to handle non-convex polyhedra directly [11]. Note that during each step computation new linear constraints can be introduced, thus the set Q is dynamically updated.

Example 1 (Discrete transitions). Assume that we want to check an invariance property $\neg FAIL$, stating that the failure state can never be reached. In the model, one discrete transition can set the Boolean variable *FAIL* to true:

$$REC \wedge (v \leq 0.0) \wedge (p > EoA) \rightarrow FAIL := true;$$

The transition says that if currently the cooperation protocol is in the *REC* phase, the train has come to a stop, and the position of the train is beyond the end of authority point (EoA), then the system reports a failure. One backward step computation of such transition leads to the following state-set (after optimizations in Lin-AIGs): $\neg FAIL \wedge \neg(REC \wedge v \leq 0.0 \wedge \neg(p <= EoA))$.

Example 2 (Continuous flows). The pre-image of the state-set in the previous example under the mode with continuous evolution $v^\bullet = 0.0 \wedge p^\bullet = v_d$ and boundary condition $p < EoA - ST$ (start talking) will remain the same.

Fix-point detection. We need to perform subsumption checks to detect whether a fixpoint has been reached during model checking. In our approach, since linear constraints enter the state-set descriptions, one has to check implications between two state-set representations. We use HySAT [18] for this purpose.

Optimizations. Using efficient methods for keeping the state-set representation as compact as possible is the key point for our approach. This is achieved by integration of different techniques, ranging from purely Boolean methods to (increasingly) exploiting knowledge about linear constraints.

- We use inexpensive methods such as simulation, test-vector generation, detection of implications between linear constraints, and propagation of learned implications to simulation vectors to identify inequivalent Lin-AIG nodes.
- We use HySAT [18] for reducing the size of the Lin-AIG representation by detecting equivalent Lin-AIG nodes, which is applied only to candidates obtained by inexpensive methods.
- We extract "don't cares" from conflicts of calls of HySAT to remove redundant linear constraints in the state set representations. This allows us to restructure Lin-AIGs based on internal node equivalences modulo "don't cares", and to achieve new compact representation as a Boolean combination of a minimal subset of the original set of linear constraints.

We have demonstrated [12, 11] that the tuned combination of these deeply integrated methods leading to significant performance improvements.

7.2 Continuous-Time Models and Verification Results

The models of the system mainly consists of two parts: (1) a cooperation protocol between the train and the rail-road crossing for collision avoidance, (2) a speed supervision of the train. Compared to its original description in Matlab-Simulink, we have made some simplifications.

The cooperation protocol. The protocol distinguishes a number of phases as shown in Fig. 7. Additionally, we add one more transition from the *RECOVERY* phase: If the train stops in front of the crossing, and the position of the train is beyond the end of authority point (EoA), then a *FAILURE* phase is entered. The safety property of collision freedom is equivalent to prove that this *FAILURE* phase of the cooperation protocol can never be entered.

The speed supervision. Figure 6 gives an overview of the train speed control. Here, we summarize the modes and the switching conditions between them in our models, as derived from Fig. 6. For each segment of the track, there is a pre-defined desired train speed. The modes for driving the train are decided by the relation of the desired speed v_d and the current speed v. In the *NormalMove* mode ($v < v_d \leq 1.05 \cdot v$), the train is driven under a PI controller. Once the condition $1.05 \cdot v_d < v \leq 1.1 \cdot v_d$ holds, the train switches to the mode *MotorOff*, where the drive force for the train becomes zero, and the train decelerates on a constant $-0.05m/s^2$. From the *MotorOff* mode, the train can re-enter the *NormalMove* mode as soon as $v \leq v_d$. If the difference between v_d and v is large, the mode *EmergencyBraking* (*ServiceBraking*) will be entered if $v \geq 1.5 \cdot v_d$, ($v \geq 1.1 \cdot v_d \wedge v \leq 1.5 \cdot v_d$) from the *NormalMove* mode. In modes *EmergencyBraking* and *ServiceBraking*, the train decelerates on constants $-3.0m/s^2$ and $-1.0m/s^2$, respectively. During *ServiceBraking*, the mode *NormalMove* (*MotorOff*) is re-entered if $v \leq v_d$ ($1.05 \cdot v_d < v \leq 1.1 \cdot v_d$). There is one special case when the train enters the *EmergencyBraking* mode. Since the train is desired to stop, it is not possible to re-enter the modes *NormalMove* and *MotorOff*. Constants in the conditions are derived from the Matlab-Simulink model.

Approximations. Currently, our model checker only supports models with dynamics given by constants. In the train example, the dynamics $v^\bullet$ in mode *NormalMove* (controlled by a PID controller) relies on the difference between v_d and v (see Fig. 6), and the evolution for the position p is normally defined as $p^\bullet = v$. Therefore, both $v^\bullet$ and $p^\bullet$ are linear if v is a variable. So the train system cannot be described and checked directly by our approach. We need to have an over-approximation of the train's behavior using the method developed in [26]. First, the mode *NormalMove* is split into a set of sub-modes, we define accelerations $v^\bullet$ as constants, depending the relation between v and v_d. Second, for each mode defined in the previous section (together with sub-modes for *NormalMove*), we divide the speed into several regions, and use this information to safely over-approximate the evolution of the position $p^\bullet$ by its possible maximal changing rate. Therefore, the number of modes depends on the concrete approximation. The constraints on the speed and the desired speed and the constraints whether the train has reached the positions $EoA - ST$ (start-talking) or $EoA - SB$ (start-braking), are treated as the boundary conditions for each mode. An appropriate mode is selected depending the phase of the cooperation protocol, current velocity v and its relation to v_d. For instance, if $v_d \geq 1.5 \cdot v$, $30.0 \leq v \leq 40.0$, and the cooperation protocol is in the *FAR* phase, then the speed controller of the train will choose a mode with $v^\bullet = 2.0$ (a fast acceleration) and $p^\bullet = 40.0$ (the maximal changing rate of p). The condition $v_d \geq 1.5 \cdot v$ and $30.0 \leq v \leq 40.0$ will be part of the boundary condition for such mode. The condition whether the crossing is secured is treated as a discrete input. The decisions in the cooperation protocol constitute discrete transitions in the model.

Experimental results. The safety property for the models is that the *FAIL* phase can never been entered, i.e., the train comes to a complete stop in front of the crossing if it is not secured. For the continuous-time models, we have successfully proven the given safety invariant for a model with 16 modes in 376 seconds. The final state set representation contained 3906 Lin-AIG nodes with 2358 linear constraints. During model checking, up to 7798 Lin-AIG nodes were used, 36683 HySAT calls occurred and 2582 redundant linear constraints were removed. Experiments are performed on a PC with an AMD Opteron Processor with 2.6 GHz and 16 GB RAM.

7.3 Iterative Abstraction Refinement for Step-Discrete Linear Hybrid Systems

Alternatively, an iterative abstraction refinement approach called ω-Cegar [48] is being developed with the focus on open-loop systems, exploiting the characteristics of huge discrete state spaces by ruling out comprehensive classes of spurious counterexamples for subsequent iterations, so that counterexamples with common reasons of invalidity cannot occur again. With the incremental construction of an omega-automaton and its parallel composition with a course abstraction of the model, all runs containing already detected reasons of being invalid are excluded. Since the reasons are fully independent from the discrete behavior, the approach converges fast also for huge discrete state spaces.

The implementation is currently restricted to step-discrete linear hybrid models being represented as a discrete transition graph where the transitions are guarded by linear constraints (*guard expressions*) and extended with linear transformations (*computations*) on the set of continuous variables. This corresponds exactly to the semantics of linear reactive systems being modeled in industrial contexts based on CASE-tools, where executable target code can be generated from. Thanks to an appropriate compiler and the realization of the algorithm on a symbolic representation level, the procedure can be applied to the generated C code of such models even for very large systems.

Initial abstraction. To verify a property φ, the procedure starts by creating an *initial abstraction* A_0 of a hybrid automaton H by removing all guard expressions and computations on continuous items, but fully preserving the discrete structure of the model. This entails a translation of φ to a new property $\hat{\varphi}$ to hold for some corresponding states in A_0.

Analysis phase. A_0 can be analyzed by any *finite state model checker* being able to generate a *counterexample* $\hat{\pi}$. The counterexample $\hat{\pi}$ consisting of a sequence of discrete states is analyzed by projecting it to the hybrid automaton H to retrieve the corresponding guard expressions and computations (*regulation laws*) that have to be fulfilled or performed in H, respectively, for $\hat{\pi}$ to be a valid counterexample. For linear hybrid automata, this analysis consists of solving conjunctions of linear constraints directly derived from the projected regulation laws. The result of this analyzation is either a valid sequence of valuations of continuous state variables or a *generalized conflict* $(\rho_1, \rho_2, \ldots, \rho_k)$ consisting of a minimized sequence of partial regulation laws for which no solution exists in the corresponding conjunctive formula.

Since such conflicts are fully independent of the discrete state sequence they occurred with, there is a high probability that they apply to many other fragments of the discrete transition system as well, especially for huge discrete state spaces combined with only few regulation laws exhibited by the model.

Construction of ω-automaton. Thus we follow a strategy of completely ruling out generalized conflicts by constructing an ω-automaton A_C that accepts all runs not containing any known conflict as a subsequence. Considering partial regulation laws as atomic characters and C as the set of all previously detected generalized conflicts, the behavior of A_C can be described by an LTL formula:

$$A_C \models \neg \mathbf{F} \bigvee_{(\rho_1,\rho_2,\ldots,\rho_k)\in C} (\rho_1 \wedge \mathbf{X}(\rho_2 \wedge \mathbf{X}(\ldots \wedge \mathbf{X}\rho_n))) \tag{21}$$

Instead of using standard algorithms to translate LTL formulae to Büchi-automata, we apply an efficient automaton construction algorithm dedicated to the structure of LTL formulae as presented above, resulting in rather small automata, especially in comparison to general Büchi-automata construction algorithms [50].

Abstraction refinement. A parallel composition $A = A_0 \times A_C$ ensures that any (infinite) run not accepted by A_C cannot be exhibited by A. With A_C being incrementally extended to not accept conflicts found in subsequent model checking iterations, we get a sequence $A_1, A_2, \ldots, A_n$ of refined abstract transition systems, where the model checker can finally prove that either $A_k \models \hat{\varphi}$ from which can be concluded that $H \models \varphi$ or that a counterexample $\hat{\pi}$ violates φ with $\hat{\pi}$ having a valid projection to a path π in H as computed in the analysis phase.

Remarks. The finite state model checker used to verify the abstract system in each iteration can be freely exchanged even in between iterations. Thus, advantages of different technologies can be combined by

1. starting with (faster) bounded model checking (BMC) while counterexamples within the given bound can be generated and
2. switching to unbounded model checking (e.g., CTL model checker) if no counterexamples within a given bound k are found anymore.

This way computation times of iteration cycles can be kept short while being able to prove if a property φ holds for a model (*certification*). However, since the approach is a semi-decision one, affirmation of properties might fail even by using unbounded model checkers.

The restriction to step-discrete linear hybrid models is due to the implementation only and does not follow from the approach. Currently, only safety properties can be verified. An extension to CTL-formulae is possible with the limitation, that valid infinite counterexamples cannot be confirmed as such.

7.4 Discrete-Time Modes and Verification Results

Our abstraction-refinement approach deals with step-discrete linear hybrid systems modelled as discrete transition graphs, in which assignments and transition

Proof	dimensions	regulation laws	conflicts	iterations	$\mid \pi \mid$	A_C	time
$\neg(v = 0 \Rightarrow p \leq EoA)$	0+10	34	31	15	$\nexists$	3	3 min

Table 1. It shows: number of continuous dimensions (*inputs*+*state-based*), number of exhibited regulation laws/generalized partial regulation laws, number of conflicts, iterations, final path length, size of A_C in terms of statebits, and total runtime.

guards may use linear arithmetical expressions, this subsumes the capability to describe the evolvement of plant variables by linear equations. Hence, the approximations in Subsection 7.2 are not necessary. The discrete-time models of our example can be derived from the Matlab-Simulink model with a given sampling rate δ. The discrete transitions for the cooperation protocol and for mode-switchings in the speed supervision are encoded exactly the same as in the continuous-time models (see Subsection 7.2). In this part, we focus on time-discretization of the plant behavior. Our main assumption is that the acceleration of the train during a discrete time step keeps unchanged. If the train is in the modes of *MotorOff*, *EmergencyBraking* and *ServiceBraking*, the velocity and position of the train can be simply updated by $v' = v + \delta \cdot a$ and $p' = p + v \cdot \delta + a \cdot \delta^2/2$, where a is the constant deceleration for those modes. If the train is in the *NormalMove* mode, the formulas for computing acceleration are given in the form of f and g in Fig. 14. Hence, we can calculate the train's new acceleration using $a' = c_1 \cdot (v - v_d) + c_2 \cdot s + c_3 \cdot v_d$ at each discrete time step (c_1, c_2, c_3 are constants in Fig. 14). Updates of the velocity and position can be done in the same way as for other modes. Here, s denotes the integration part of the PI controller (see Equation (2)), and it is accumulated at each step by $s' = s + (v_d - \bar{v}) \cdot \delta$, where $\bar{v} = v + a \cdot \delta$ denotes the average velocity during the time interval δ.

Table 1 shows statistical data of the application of the ω-Cegar approach to a central part of the train system presented in Section 2: to prove that the train always stops before the crossing if it is not secured. The property was certified within 3 minutes and only 15 iterations by first using BMC (Prover-CL V5.0.6) for the iterations and switching to an unbounded model checker (VIS Version 2.0) to finally prove the unreachability on the refined model, which consisted of 16 state bits.

8 Conclusion and Future Work

Industrial design processes for cooperating traffic agents exploit a layered design structure to separate concerns in addressing cooperation strategies, control design, and design implementation. We have provided a verification methodology covering these design steps, where safety and stability properties resulting from the overall safety objective of collision freedom are traced to design entities at all levels, and have provided layer specific verification approaches to establish such derived safety and stability requirements at each layer. The feasibility of the approach has been established using a variation of the ETCS level 3 protocol enforcing collision freedom of trains following the moving block principle.

Theoretical approaches to cover multi-layered designs, such as refinement and compositional reasoning, fail to provide semantic bridges across this design space, in particular due to their inability to support the degree of deviations between models tolerated by industrial design processes. Horizontal composition theories provide the semantic foundation for compositional verification, deducing properties of composed systems in terms of their constituents. Vertical composition theories exploit the layered structure of designs, allowing to abstract from design aspects manifest at lower levels when verifying safety properties such as collision freedom for a "higher" level model.[10] Vertical composition theories typically built on refinement relations.

Of particular relevance to our application domain are approaches for refinement and compositional verification of hybrid systems. Compositional verification techniques for hybrid systems have only recently being investigated, e.g., in [37, 38]. Frehse [19] provides an assume-guarantee based approach for hybrid systems which do not share variables. The restriction to event-based communication is reasonable for controller models; however, as soon as closed loop models are considered, plant models will typically share system states; in particular, for collision avoidance protocols, it is exactly the shared physical state space which is subject to the analysis. The extension to shared variables provided in [20] requires unique statically assigned owners of shared variables – only owners are allowed to write on shared variables.

Frehse's approach only addresses refinement of specifications. For hybrid systems with shared variables, the notions of refinements presented do not track continuous evolutions, but only require matching continuous states at end-points of continuous evolutions. Stauner [51, 52] studies more general notions of refinement, which in particular aim at bridging the gap between local control and design models. The key concepts of using bounded perturbations to provide room for discretization and inter-sampling errors in the transition to design models are promising. Systematic methods are proposed for constructing a discrete time model refining the relaxed local control model under certain conditions are provided. However, a general notion of refinement, applicable to design models not constructed using Stauner's approach, is not given. Since design models also must cater for additional aspects such as fault-tolerance and diagnosis, a general theory for refinement between design models and local control models is desirable. Additionally, a compositional extension of this framework is needed.

We plan to elaborate our research along multiple dimensions. First, we will extend the model at the cooperation layer emphasizing the dynamic aspect of traffic applications, in which traffic agents enter and leave "interaction areas", and lift the technique of reducing collision freedom to arguments on criticality functions and local control properties to this richer semantic setting. Secondly, while we demonstrated the scalability of our AIG based verification methods to linear hybrid automata with large discrete state spaces (e.g., to a flap controller with 2^{20} discrete states [11]), future work will address support for plant

[10] We assume that the cooperation layer is "higher" than the local control layer, which in turn is "higher" than the design layer.

dynamics governed by linear differential equations. Thirdly, we plan to research into robust refinement relations and non-standard semantics of hybrid automata to extend compositional refinement techniques to a theory providing semantic bridges across the layered design space of cooperating traffic agents.

References

1. R. Alur, R. Grosu, Y. Hur, V. Kumar, and I. Lee. Modular specification of hybrid systems in CHARON. In *Proc. 9th Workshop on Hybrid Systems: Computation and Control*, volume 1790 of *LNCS*, pages 6–19. Springer, 2000.
2. R. Alur, R. Grosu, I. Lee, and O. Sokolsky. Compositional modeling and refinement for hierarchical hybrid systems. *Journal of Logic and Algebraic Programming*, 68(1-2):105–128, 2006.
3. A. Balluchi, L. Benvenuti, S. Engell, T. Geyer, K. Johansson, F. Lamnabhi-Lagarrigue, J. Lygeros, M. Morari, G. Papafotiou, A. Sangiovanni-Vincentelli, F. Santucci, and O.Stursberg. Hybrid control of networked embedded systems. *European Journal on Control, Fundam. Issues in Control*, 11(4-5):478–508, 2006.
4. B. Beckert, M. Giese, R. Hähnle, V. Klebanov, P. Rümmer, S. Schlager, and P. H. Schmitt. The KeY System 1.0 (deduction component). In F. Pfenning, editor, *Proc. 21st Intern. Conf. on Automated Deduction*, LNCS. Springer, 2007.
5. B. Beckert, R. Hähnle, and P. H. Schmitt, editors. *Verification of Object-Oriented Software: The KeY Approach*, volume 4334 of *LNCS*. Springer, 2007.
6. J. Bohn, W. Damm, J. Klose, A. Moik, and H. Wittke. Modeling and validating train system applications using Statemate and live sequence charts. In *Proc. Conference on Integrated Design and Process Technology*. Society for Design and Process Science, 2002.
7. B. Borchers. CSDP, a C library for semidefinite programming. *Optimization Methods and Software*, 10(1):613–623, 1999.
8. S. Boyd, L. E. Ghaoui, E. Feron, and V. Balakrishnan. *Linear Matrix Inequalities in System and Control Theory*. SIAM, 1994.
9. M. S. Branicky. Multiple Lyapunov functions and other analysis tools for switched and hybrid systems. *IEEE Transactions on Automatic Control*, 43(4), April 1998.
10. A. Cervin, D. Henriksson, B. Lincoln, J. Eker, and K. Arzén. How does control timing affect performance? *IEEE Control Systems Magazine*, 23(2):16–30, 2003.
11. W. Damm, S. Disch, H. Hungar, S. Jacobs, J. Pang, F. Pigorsch, C. Scholl, U. Waldmann, and B. Wirtz. Exact state set representations in the verification of linear hybrid systems with large discrete state space. Technical report, AVACS, 2007.
12. W. Damm, S. Disch, H. Hungar, J. Pang, F. Pigorsch, C. Scholl, U. Waldmann, and B. Wirtz. Automatic verification of hybrid systems with large discrete state space. In *Proc. 4th Symposium on Automated Technology for Verification and Analysis*, volume 4218 of *LNCS*, pages 276–291. Springer, 2006.
13. W. Damm, H. Hungar, and E.-R. Olderog. Verification of cooperating traffic agents. *International Journal of Control*, 79(5):395 – 421, May 2006.
14. W. Damm, G. Pinto, and S. Ratschan. Guaranteed termination in the verification of LTL properties of non-linear robust discrete time hybrid systems. *International Journal of Foundations of Computer Science*, 18(1):63–86, 2007.
15. V. Donde and I. A. Hiskens. Shooting methods for locating grazing phenomena in hybrid systems. *Intern. Journal of Bifurcation and Chaos*, 16(3):671–692, 2006.

16. G. Feng. Stability analysis of piecewise discrete-time linear systems. *IEEE Transactions on Automatic Control*, 47(7):1108–1112, 2002.
17. G. F. Franklin, J. D. Powell, and M. Workman. *Digital Control of Dynamic Systems*. Pearson International, 1998.
18. M. Fränzle and C. Herde. HySAT: An efficient proof engine for bounded model checking of hybrid systems. *Formal Methods in System Design*, 30(3):179–198, 2007.
19. G. Frehse. Compositional verification of hybrid systems with discrete interaction using simulation relations. In *Proc. 13th IEEE Conference on Computer Aided Control Systems Design*. IEEE Computer Society, 2004.
20. G. Frehse. *Compositional Verification of Hybrid Systems using Simulation Relations*. PhD thesis, Radboud Universiteit Nijmegen, 2005.
21. G. Frehse. PHAVer: Algorithmic verification of hybrid systems past HyTech. In *Proc. 8th Workshop on Hybrid Systems: Computation and Control*, volume 3414 of *LNCS*, pages 258–273. Springer, 2005.
22. G. Hager. European ACAS operational evaluation – Final report. Technical Report EEC Report No. 316, Eurocontrol, 1997.
23. D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic*. MIT Press, 2000.
24. A. E. Haxthausen and J. Peleska. Formal development and verification of a distributed railway control system. *IEEE Transactions on Software Engineering*, 26(8):687–701, 2000.
25. T. A. Henzinger. The theory of hybrid automata. In *Proc. 11th IEEE Symposium on Logic in Computer Science*, pages 278–292. IEEE Computer Society, 1996.
26. T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. Algorithmic analysis of nonlinear hybrid systems. *IEEE Transactions on Automatic Control*, 43(5):540–554, 1998.
27. T. A. Henzinger, B. Horowitz, R. Majumdar, and H. Wong-Toi. Beyond HyTech: Hybrid systems analysis using interval numerical methods. In *Proc. 3rd Workshop on Hybrid Systems: Computation and Control*, volume 1790 of *LNCS*, pages 130–144. Springer, 2000.
28. M. Johansson and A. Rantzer. Computation of piecewise quadratic Lyapunov functions for hybrid systems. *IEEE Transactions on Automatic Control*, 43, 1998.
29. H. K. Khalil. *Nonlinear Systems*. Prentice-Hall, 2nd edition, 1996.
30. F. Kratz, O. Sokolsky, G. J. Pappas, and I. Lee. R-Charon, a modeling language for reconfigurable hybrid systems. In *Proc. 9th Workshop on Hybrid Systems: Computation and Control*, volume 3927 of *LNCS*, pages 392–406. Springer, 2006.
31. N. G. Leveson. *Safeware: System Safety and Computers*. Addison-Wesley, 1995.
32. C. Livadas, J. Lygeros, and N. A. Lynch. High-level modeling and analysis of TCAS. *Proceedings of IEEE – Special Issue on Hybrid Systems: Theory & Applications*, 88(7):926–947, 2000.
33. J. Lofberg. YALMIP: a toolbox for modeling and optimization in Matlab. In *IEEE Intern. Symp. Computer Aided Control Systems Design*, pages 284–289, 2004.
34. R. Loos and V. Weispfenning. Applying linear quantifier elimination. *The Computer Journal*, 36(5):450–462, 1993.
35. M. A. Lyapunov. Problème général de la stabilité du movement. *Ann. Fac. Sci. Toulouse*, 9:203–474, 1907. (Translation of a paper published in Comm. Soc. Math. Kharkow, 1893, reprinted Ann. Math. Studies No. 17, Princeton Univ. Press, 1949).
36. J. Lygeros, D. N. Godbole, and S. S. Sastry. Verified hybrid controllers for automated vehicles. *IEEE Transactions on Automatic Control*, 43(4):522–539, 1998.
37. N. A. Lynch, R. Segala, and F. W. Vaandrager. Hybrid I/O automata revisited. In *Proc. 4th Workshop on Hybrid Systems: Computation and Control*, volume 2034 of *LNCS*, pages 403–417. Springer, 2001.

38. N. A. Lynch, R. Segala, and F. W. Vaandrager. Hybrid I/O automata. *Information and Computation*, 185(1):105–157, 2003.
39. A. Mishchenko, S. Chatterjee, R. Jiang, and R. K. Brayton. FRAIGs: A unifying representation for logic synthesis and verification. Technical report, EECS Dept., UC Berkeley, 2005.
40. Y. Nesterov and A. Nemirovskii. *Interior Point Polynomial Algorithms in Convex Programming*. SIAM, 1994.
41. J. Oehlerking, H. Burchardt, and O. Theel. Fully automated stability verification for piecewise affine systems. In *Proc. 10th Workshop on Hybrid Systems: Computation and Control*, volume 4416 of *LNCS*, pages 741–745. Springer, 2007.
42. S. Pettersson. *Analysis and Design of Hybrid Systems*. PhD thesis, Chalmers University of Technology, Gothenburg, 1999.
43. A. Platzer. Differential dynamic logic for verifying parametric hybrid systems. In N. Olivetti, editor, *Proc. 16th TABLEAUX*, LNCS. Springer, 2007.
44. A. Platzer. Differential logic for reasoning about hybrid systems. In *Proc. 10th Workshop on Hybrid Systems: Computation and Control*, volume 4416 of *LNCS*, pages 746–749. Springer, 2007.
45. A. Platzer. A temporal dynamic logic for verifying hybrid system invariants. In *Proc. International Symposium on Logical Foundations of Computer Science*, volume 4514 of *LNCS*, pages 457–471. Springer, 2007.
46. A. Platzer. Towards a hybrid dynamic logic for hybrid dynamic systems. In P. Blackburn, T. Bolander, T. Braüner, V. de Paiva, and J. Villadsen, editors, *Proc. LICS Intern. Workshop on Hybrid Logic*, ENTCS, 2007.
47. A. Platzer and E. Clarke. The image computation problem in hybrid systems model checking. In *Proc. 10th Workshop on Hybrid Systems: Computation and Control*, volume 4416 of *LNCS*, pages 473–486. Springer, 2007.
48. M. Segelken. Abstraction and counterexample-guided construction of omega-automata for model checking of step-discrete linear hybrid models. In *Proc. 19th Conference on Computer Aided Verification*, LNCS. Springer, 2007.
49. B. I. Silva, K. Richeson, B. H. Krogh, and A. Chutinan. Modeling and verification of hybrid dynamical system using CheckMate. In *Proc. 4th Conference on Automation of Mixed Processes*, 2000.
50. F. Somenzi and R. Bloem. Efficient Büchi Automata from LTL Formulae. In *Proc. 12th Conf. on Computer Aided Verification*, volume 1855 of *LNCS*, pages 248–263. Springer, 2000.
51. T. Stauner. *Systematic Development of Hybrid Systems*. PhD thesis, Technische Universität München, 2001.
52. T. Stauner. Discrete-time refinement of hybrid automata. In *Proc. 5th International Workshop on Hybrid Systems: Computation and Control*, volume 2289 of *LNCS*, pages 407–420. Springer, 2002.
53. C. Tomlin, G. J. Pappas, and S. S. Sastry. Conflict resolution for air traffic management: A case study in multi-agent hybrid systems. *IEEE Transactions on Automatic Control*, 43(4):509–521, 1998.
54. D. Wende. *Fahrdynamik des Schienenverkehrs*. Teubner, 2003.
55. V. Yakubovich. S-procedure in nonlinear control theory. *Vestnik Leningrad University*, pages 62–71, 1971.
56. C. Zhou and M. Hansen. *Duration Calculus: A Formal Approach to Real-Time Systems*. Springer-Verlag, 2004.
57. C. Zhou, C. Hoare, and A. Ravn. A calculus of durations. *Information Processing Letters*, 40(5):269–276, 1991.

Appendix

A Communicating Hybrid Automata

For the sake of completeness, we include from [13] the description of communicating hybrid automata which we use as a model for cooperating traffic agents in this paper. We assume that the signature of the real numbers is given with function and predicate symbols like $0, 1, +, \cdot, <, =$ interpreted on the domain $\mathbb{R}$ in the usual way. (Real-valued) expressions, Boolean expressions, and first-order formulas over this signature are defined as usual. By $Th(\mathbb{R})$ we denote the theory of the real numbers, i.e., the set of all first-order formulas that hold in $\mathbb{R}$.

Definition 7 (Hybrid Automaton). *A* hybrid automaton *is a tuple* $H = (\mathbb{M}, Var, R^d, R^c, m_0, \Theta)$ *where*

1. $\mathbb{M}$ *is a finite set of* modes*, with typical element* $m \in \mathbb{M}$ *and with a distinguished* mode observable M *ranging over* $\mathbb{M}$*,*
2. *Var is a set of* variables *ranging over the set* $\mathbb{R}$ *of real numbers. Typical elements of Var are* X, Y*. Var is partitioned into disjoint sets of* input, local *and* output *variables:* $Var = Var^{in} \cup Var^{loc} \cup Var^{out}$*, where local variables cannot be accessed by other hybrid automata in a parallel composition (see Subsection A.2),*
3. $m_0 \in \mathbb{M}$ *is the* initial mode*,*
4. Θ *is a mapping that associates with each mode* $m \in \mathbb{M}$ *a* local invariant $\Theta(m)$*, which is a quantifier-free first-order formula over Var,*
5. R^d *is the* discrete transition relation *with elements* $(m, \uparrow \Phi, \mathcal{A}, m')$ *called* transitions*, which are graphically represented as* $m \xrightarrow{\uparrow\Phi/\mathcal{A}} m'$*, where*
   - $m, m' \in \mathbb{M}$,
   - *the* trigger $\uparrow\Phi$ *guarding the transition describes the event that a quantifier-free formula* Φ *over Var becomes true,*
   - $\mathcal{A}$ *is a (possibly empty) set of (disjoint)* assignments *of the form* $X := e$ *with* $X \in Var^{loc} \cup Var^{out}$ *and* e *an expression over Var.*
6. R^c *is the* continuous transition relation*, i.e., a mapping that associates with each mode* $m \in \mathbb{M}$ *and each variable* $X \in Var^{loc} \cup Var^{out}$ *an expression* $R^c(m)(X)$ *over Var, which is taken as the right-hand side of the differential equation* $X^\bullet = R^c(m)(X)$ *describing the evolution of* X *over time while* H *is in mode* m*.*

Valuations or *states* of the variables in *Var* are given by functions $\sigma : Var \to \mathbb{R}$. A valuation σ assigning to each variable X the value $v_X \in \mathbb{R}$ is denoted by $\sigma = \{X \mapsto v_X \mid X \in Var\}$. For a valuation σ and a set of assignments $\mathcal{A}$ let $\mathcal{A}(\sigma) : Var \to \mathbb{R}$ denote the *update* of σ according to $\mathcal{A}$ defined by

$$\mathcal{A}(\sigma) = \{X \mapsto \sigma(e) \mid \exists\, e : X := e \in \mathcal{A}\} \cup \{X \mapsto \sigma(X) \mid \neg \exists\, e : X := e \in \mathcal{A}\}.$$

For a valuation σ and a formula Φ let $\sigma \models \Phi$ denote that σ *satisfies* Φ.

We require of the discrete transition relation that the execution of one transition does not immediately enable a further transition.

Definition 8 (Transition Separation). *The discrete transitions in a hybrid automaton H are* separated*, if for any two transitions* $(m_1, \uparrow \Phi_1, \mathcal{A}_1, m_1')$ *and* $(m_2, \uparrow \Phi_2, \mathcal{A}_2, m_2')$ *in* R^d *of* H *with* $m_1' = m_2$ *the following condition holds:*

$$\forall\ \sigma : \mathit{Var} \to \mathbb{R} :\ (\sigma \models \Phi_1 \ \Rightarrow\ \mathcal{A}_1(\sigma) \not\models \Phi_2)\ .$$

Separation implies that at any given point in time during a run, at most one discrete transition fires. Thus our models have dense time but not *superdense* time, where a sequence of discrete transitions is permitted to fire at one instant in time.

Discrete variables may be included into hybrid automata according to our definition via an embedding of their value domain into the reals, and associating a derivative of constantly zero to them (locals and outputs). Timeouts are easily coded via explicit local timer variables with a derivative taken from $\{-1, 0, 1\}$.

Note that this general model subsumes both controller and plant models, by choosing the set of variables appropriately and enforcing certain modeling restrictions. For our plant models, we require the absence of discrete transitions. This entails that plant variables only evolve continuously and cannot be changed by discrete jumps. This is convenient for the formulation of our approach but not essential.

Definition 9 (Restriction). *For a hybrid automaton H and a mode $m \in \mathbb{M}$ let the* restriction $H \restriction m$ *be defined as H, but with the mode fixed to m. Formally, this is the following hybrid automaton:*

$$H \restriction m = (\{m\}, \mathit{Var}, R^d \restriction m, R^c \restriction m, m, \Theta \restriction m)$$

where $R^d \restriction m = \{(m, \uparrow\Phi, \mathcal{A}, m') \in R^d \mid m = m'\}$ *and* $R^c \restriction m$ *and* $\Theta \restriction m$ *are the restrictions of the mappings* R^c *and* Θ *to the singleton set* $\{m\}$.

A.1 Behaviour

We will in the definition of *runs* of a hybrid automaton interpret all transitions as urgent, i.e., a mode will be left as soon as the triggering event occurs. This can either be the expiration of a time-out or a condition (e.g., on the plant sensors) becoming true. Valid runs also avoid Zeno behavior and time-blocks, i.e., each run provides a valuation for each positive point in time. We did not take provisions to ensure the existence of such a run, nor the property that each initial behavior segment can be extended to a full run. Such might be added via adequate modeling guidelines (e.g., by including the negation of an invariant as trigger condition on some transition leaving the mode). As these properties are not essential to the purpose of this paper we left them out.

We now give the formal definition of runs of a hybrid automaton H capturing the evolution of modes and the real-valued variables over time. To this end, we consider the continuous time domain $\mathit{Time} = \mathbb{R}_{\geq 0}$ of non-negative reals, for the mode observable M a function $\hat{M} : \mathit{Time} \to \mathbb{M}$, and for every variable $X \in \mathit{Var}$ a corresponding function $\hat{X} : \mathit{Time} \to \mathbb{R}$ describing for each time point $t \in \mathit{Time}$ the current mode $\hat{M}(t) \in \mathbb{M}$ and the current value $\hat{X}(t) \in \mathbb{R}$, respectively. Further on, for $\hat{X}$ and $0 < t \in \mathit{Time}$ we define the *previous value* of $\hat{X}$ at t by $\mathit{prev}(\hat{X}, t) = \lim_{u \to t}(\hat{X}(u))$. Satisfaction of a condition containing *prev* entails that the respective limes does exist.[11]

Definition 10 (Runs of a Hybrid Automaton). *A* run *of a hybrid automaton $H = (\mathbb{M}, \mathit{Var}, R^d, R^c, m_0, \Theta)$ is a tuple of trajectories*

$$\pi = \left(\hat{M},\ (\hat{X})_{X \in \mathit{Var}}\right),$$

with $\hat{M} : \mathit{Time} \to \mathbb{M}$ and $\hat{X} : \mathit{Time} \to \mathbb{R}$ for $X \in \mathit{Var}$, iff

$$\exists\ (\tau_i)_{i \in \mathbb{N}} \in \mathit{Time}^{\mathbb{N}} :\ \tau_0 = 0\ \wedge\ \forall\ i \in \mathbb{N} : \tau_i < \tau_{i+1},$$

a strictly increasing sequence of discrete switching times, satisfying the following conditions:

1. non-Zeno*:* $\forall\ t \in \mathit{Time}\ \exists\ i \in \mathbb{N} : t \le \tau_i$

2. mode switching times*:* $\forall\ i \in \mathbb{N}\ \forall\ t \in [\tau_i, \tau_{i+1}) : \hat{M}(t) = \hat{M}(\tau_i)$

3. continuous evolution*:*

$$\forall\, i \in \mathbb{N}\ \forall\, t' \in [\tau_i, \tau_{i+1})\ \forall\, X \in \mathit{Var}^{loc} \cup \mathit{Var}^{out} : \sigma \models X^{\bullet} = R^c(\hat{M}(\tau_i))(X)$$

where σ is the valuation $\sigma = \{X^{\bullet} \mapsto \frac{d\hat{X}(t)}{dt}(t')\} \cup \{Y \mapsto \hat{Y}(t') \mid Y \in \mathit{Var}\}$. Thus in σ the variable $X^{\bullet}$ gets the value of the derivative of the function $\hat{X}$ at t' and all other variables $Y \in \mathit{Var}$ get the value of the function $\hat{Y}$ at t'.

4. invariants*:* $\forall\ t \in \mathit{Time} : \{X \mapsto \hat{X}(t) \mid X \in \mathit{Var}\} \models \Theta(\hat{M}(t))$

5. urgency*:*

$$\begin{array}{l}\forall\, i \in \mathbb{N}\ \forall\, t \in [\tau_i, \tau_{i+1})\ \forall\, (m, \uparrow\!\Phi, \mathcal{A}, m') \in R^d \text{ we have that} \\ \hat{M}(t) = m\ \Rightarrow \{X \mapsto \hat{X}(t) \mid X \in \mathit{Var}\} \not\models \Phi\end{array}$$

6. discrete transition firing*:* $\forall\ i \in \mathbb{N}$ *we have that*

$$\begin{array}{l}(\hat{M}(\tau_{i+1}) = \hat{M}(\tau_i)\ \wedge\ \forall\ X \in \mathit{Var}^{loc} \cup \mathit{Var}^{out} : \hat{X}(\tau_{i+1}) = \mathit{prev}(\hat{X}, \tau_{i+1})\,) \\ \vee \\ (\exists\ (m, \uparrow\!\Phi, \mathcal{A}, m') \in R^d : \hat{M}(\tau_i) = m\ \wedge\ \hat{M}(\tau_{i+1}) = m'\ \wedge \\ \quad \exists\ \sigma \in \mathit{Var} \to \mathbb{R} :\ \forall\ X \in \mathit{Var}^{loc} \cup \mathit{Var}^{out} : \\ \qquad \sigma(X) = \mathit{prev}(\hat{X}, \tau_{i+1})\ \wedge\ \sigma \models \Phi \\ \qquad \wedge\ \forall\ X \in \mathit{Var}^{in} : \hat{X}(\tau_{i+1}) = \sigma(X) \\ \qquad \wedge\ \forall\ X \in \mathit{Var}^{loc} \cup \mathit{Var}^{out} : \hat{X}(\tau_{i+1}) = \mathcal{A}(\sigma)(X)\,)\end{array}$$

[11] In fact, our definition of a run implies that these limits do exist for all local and output variables in any run.

For a run $\pi = \left(\hat{M},\ (\hat{X})_{X \in Var}\right)$ *of* H *and* $t \in Time$ *let* $\pi(t)$ *denote the state*

$$\pi(t) = \{M \mapsto \hat{M}(t)\} \cup \{X \mapsto \hat{X}(t) \mid X \in Var\}.$$

assigning to the mode observable M *and the all variables* $X \in Var$ *the values in the run* π *at time* t*.*

The time sequence $(\tau_i)_{i \in \mathbb{N}}$ identifies the points in time, at which mode-switches may occur, which is expressed in Clause (2). Only at those points discrete transitions (having a noticeable effect on the state) may be taken. On the other hand, it is not required that any transition fires at some point τ_i, which permits to cover behaviors with a finite number of discrete switches within the framework above. Our simple plant models with only one mode provide examples. As usual, we exclude zeno behavior (in Clause (1)). As a consequence of the requirement of transition separation, after each discrete transition some time must elapse before the next one can fire. Clause (3) forces all local and output variables (whose dynamics is constrained by the set of differential equations associated with this mode) to actually obey their respective equation. Clause (4) requires, for each mode, the valuation of continuous variables to meet the local invariant while staying in this mode. Clause (5) forces a discrete transition to fire when its trigger condition becomes true. The effect of a discrete transition is described by Clause (6). Whenever a discrete transition is taken, local and output variables may be assigned new values, obtained by evaluating the right-hand side of the respective assignment using the previous value of locals and outputs and the current values of the input. If there is no such assignment, the variable maintains its previous value, which is determined by taking the limit of the trajectory of the variable as t converges to the switching time τ_{i+1}. Values of inputs may change arbitrarily. They are not restricted by the clauses, other that they obey mode invariants and contribute to the satisfaction of discrete transitions when those fire.

A.2 Parallel Composition

The parallel composition of two such hybrid automata H_1 and H_2 presupposes the typical disjointness criteria for modes, local variables, and output variables. Output variables of H_1 which are at the same time input variables of H_2, and vice versa, establish communication channels with instantaneous communication. Those variables establishing communication channels become local variables of $H_1 \parallel H_2$ (in addition to the local variables of H_1 and H_2), for other variable sets we simply take the union of those not involved in communication. Modes of $H_1 \parallel H_2$ are the pairs of modes of the component automata. One may define the set of runs of H as those tuples of trajectories which project to runs of H_1 and H_2, respectively. It is not always possible to give a hybrid automaton for $H_1 \parallel H_2$, because of problems with cycles of instantaneous communications. Therefore, we impose the following additional condition on the composability of hybrid automata.

Definition 11 (Composable Hybrid Automata). *Let two hybrid automata H_i, $i = 1, 2$, with discrete transition relations R_i^d, $i = 1, 2$, be given. For a pair of transitions $s_i = (m_i, \uparrow \Phi_i, \mathcal{A}_i, m_i') \in R_i^d$, $i = 1, 2$, the transition s_1 is* unaffected *by s_2, if each variable for which there is an assignment in $\mathcal{A}_2$ appears neither in Φ_1 nor in $\mathcal{A}_1$ (on any of the right-hand sides).*

The two transition relations are composable*, if for each pair of transitions $s_i \in R_i^d$, $i = 1, 2$, either s_1 is unaffected by s_2 or vice versa.*

Composability establishes essentially a direction on instantaneous communications – communications may have an immediate effect on the output and thus the partner automaton, but they must not immediately influence the originator of the information. Assuming composability, the rest of the construction of the parallel composition automaton is rather standard.

For a mode (m_1, m_2), the associated invariant condition is the conjunction of the invariance conditions associated with m_1 and m_2. Similarly, the set of differential equations governing the continuous evolution while in mode (m_1, m_2) is obtained by simply conjoining the set of differential equations attached to m_1 and m_2, respectively – note that the disjointness conditions on variables assure, that this yields a consistent set of differential equations. Finally, the discrete transition relation consists of the following transitions:

1. $((m_1, m_2), \Phi_1 \wedge \mathcal{A}_1(\Phi_2), \mathcal{A}_1 \cup \mathcal{A}_1(\mathcal{A}_2), (m_1', m_2'))$
   for each pair of transitions $s_i = (m_i, \uparrow \Phi_i, \mathcal{A}_i, m_i') \in R_i^d$, $i = 1, 2$ where s_1 is unaffected by s_2,
2. $((m_1, m_2), \Phi_1 \bigwedge \{\neg \mathcal{A}_1(\Phi_2) \mid \Phi_2 \text{ trigger in } R_2^d\}, \mathcal{A}_1, (m_1', m_2))$
   for each $(m_1, \uparrow \Phi_1, \mathcal{A}_1, m_1') \in R_1^d$, and
3. transitions of the forms (1) and (2) with the role of H_1 and H_2 interchanged,

where $\mathcal{A}(\Phi)$ denotes the substitution into Φ of e for v for each assignment $X := e \in \mathcal{A}$, and $\mathcal{A}_1(\mathcal{A}_2)$ denotes the substitution of the assignments of $\mathcal{A}_1$ into the right-hand terms of $\mathcal{A}_2$.

Composability ensures that the simultaneous transitions of Clause (1) indeed capture the combined effect of both transitions. The separation of transitions in the resulting automaton is inherited from separation in the component automata by the way single-automata transitions (Clause (2)) are embedded.