Under consideration for publication in Formal Aspects of Computing

Synthesizing and Verifying Controllers for Multi-Lane Traffic Maneuvers

Gregor v. Bochmann[1], Martin Hilscher[2], Sven Linker[3], Ernst-Rüdiger Olderog[2]

[1] School of Electrical Engineering and Computer Science, University of Ottawa, Ottawa, Ontario, Canada
[2] Department of Computing Science, University of Oldenburg, Oldenburg, Germany
[3] Department of Computer Science, University of Liverpool, Liverpool, UK

Abstract. The dynamic behavior of a car can be modeled as a hybrid system involving continuous state changes and discrete state transitions. We show that the control of safe (collision free) lane change maneuvers in multi-lane traffic on highways can be described by finite state machines extended with continuous variables coming from the environment. We use standard theory for controller synthesis to derive the dynamic behavior of a lane-change controller. Thereby, we contrast the setting of interleaving semantics and synchronous concurrent semantics. We also consider the possibility of exchanging knowledge between neighboring cars in order to come up with the right decisions. Finally, we address compositional verification using an assumption-guarantee paradigm.

Keywords: Multi-lane highway traffic, lane-change maneuver, collision freedom, controller synthesis, interleaving and synchronous concurrency, assumption-guarantee paradigm

1. Introduction

We consider the safety (collision freedom) of traffic on multi-lane highways. A means to avoid collisions in car maneuvers are advanced driver assistance systems (ADAS) onboard the cars. These systems require that each car is equipped with suitable controllers that interact with other cars by sensors and communication. The development of such a controller is difficult because the interaction of cars on a highway constitutes a distributed hybrid system, combining continuous car dynamics with discrete decisions of the controllers. Therefore every step in the process of developing and checking such a system that can be systematically performed or automated is of great help.

Most approaches investigate safety of traffic maneuvers as a hybrid verification problem with full details of the continuous dynamic of the cars. Examples for this come from the California PATH Project (Partners for Advanced Transit and Highways) [LGS98] for maneuvers on freeways. More recently, Loos et al. presented

Correspondence and offprint requests to: bochmann@uottawa.ca, olderog@informatik.uni-oldenburg.de, s.linker@liverpool.ac.uk

work for freeways using $Qd\mathcal{L}$ and the theorem prover KeYmaera [LPN11], also considering the full continuous dynamic of all cars. Similarly, motion planning algorithms for cooperative collision avoidance often take the continuous car dynamics into account. A comparison of such algorithms is given in [FB11], where the authors present simulations of a small number of cars in different traffic scenarios.

In [HLOR11], we realized that safety (collision freedom) on roads is primarily a spatial property. This led us to develop a new approach to car safety that *decomposes* spatial from dynamic reasoning. To this end, we introduced a dedicated Multi-Lane Spatial Logic (MLSL), which abstracts from the continuous car dynamics [HLOR11]. MLSL formulas express spatial properties of single traffic snapshots in a concise way. We proved safety of lane-change maneuvers on motorways when all cars are equipped with a suitable lane-change controller that uses MLSL formulas in its invariants and guards. However, the controllers themselves were introduced in an ad-hoc manner.

In [ORW15] we linked an abstract, discrete spatial model using MLSL to a concrete, continuous dynamic model using differential equations for traffic maneuvers of multiple vehicles on motorways. For the spatial model the safety (collision freedom) of lane-change maneuvers was already established in [HLOR11]. By formally linking the discrete and continuous model via suitable *linking predicates*, the safety carries over to the dynamic model.

Let us give an idea of the linking by an example. In the discrete spatial model we require for a distance controller of a car *ego* that it keeps the reserved space of *ego* disjoint from the reserved space of all cars on the same lane driving ahead of *ego* over all times. This invariant property can be expressed by an MLSL formula called $\neg ccf$, standing for "no forward collision". The safety proof in [HLOR11] solely depends on the disjointness of reserved (and so-called claimed) spaces. However, we do not specify how large these spaces are.

These details are kept for the continuous dynamic model. Only at this level, differential equations modelling the movements of the car *ego* enter the picture. A quantity derived from this model is the safety distance *sd* of *ego*, which covers the braking distance at the current velocity of the *ego*. Also, we stipulate a sensor measuring the front distance *fd* of *ego* to the car ahead.

The *linking predicate* relates the purely spatial property to a property about distances in the dynamic model: $\neg ccf \Longleftarrow sd < fd$. The reverse implication expresses *refinement* of the left-hand side by the right-hand side. This allows us to transfer the safety proof from the spatial model to the dynamic model, and thus justifies the decomposition of spatial from dynamic reasoning in our approach.

In this paper, we employ methods from discrete-event systems to *synthesize* the controllers, thus offering a systematic approach to construct such controllers. The achievement is that we connect methods from discrete-event systems with the application area of traffic maneuvers of multiple cars on highways. We also use methods from protocol derivation to obtain the specification of message exchanges in the case that certain cars need to communicate for their control decisions.

We describe the setting of multi-lane traffic as in [HLOR11] (however, without using MLSL), the control architecture, and the control components inside a single car with their interactions. As a formal representation of hybrid systems we consider a variant of Hybrid Input-Output Automata (HIOA), where assumptions on inputs are allowed [LSVW99]. However, we focus on the discrete actions needed for lane control, thereby assuming that the car maneuvers of speed control and steering are dealt with separately.

Our main contributions are as follows:

- We show that from a description of the set of all possible discrete behaviors during a lane change we can *synthesize constraints* that yield a safe lane change controller. This is achieved by applying a standard method for controller synthesis in discrete event systems [CW10].
- We investigate the impact of different semantic models of parallel composition: *interleaving* vs. *synchronous parallelism*. In the latter model more intricate safety risks of a lane change are revealed. We show that the method for controller synthesis can cope with both models.
- We investigate different sensor models that represent different knowledge a car may have about its neighboring cars during a lane change. In [HLOR11], the case that a car can sense only the lengths of other cars but not their braking distances was solved by stipulating a helper car and suitable communications with it. Here we show that these *communications can be synthesized* by applying methods for protocol synthesis [MvB83].
- We outline compositional verification using assumption-guarantee-style reasoning. This is presented for the velocity controller, the steering controller, and the lane controller.

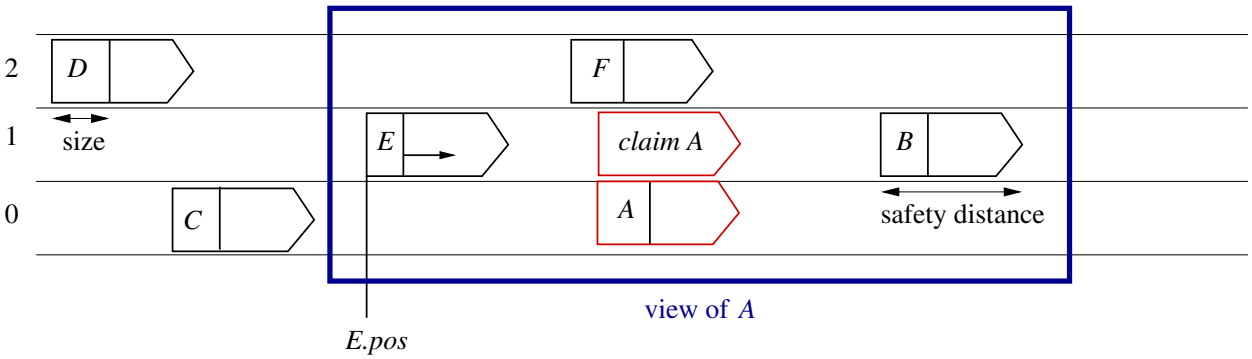

Fig. 1. A multi-lane highway with several cars. The large rectangle shows the view of car A, i.e., the part of the environment visible to A.

This paper extends the conference paper [vBHLO15] by several new (sub)sections:

- Within Section 3.5, we discuss potential interferences with cars on the target lane during a lane change maneuver, and present a solution.
- Section 3.7 contains a short description of the case when the gap on the target lane is not sufficiently large for an intended lane change.
- In Section 4, we discuss how our synthesized controller fits into a framework of compositional verification using an assumption-guarantee-style reasoning. To that end, we identify the necessary assumptions and guarantees and how the synthesized controller satisfies its contract.

This paper is structured as follows. In Section 2 we present the details of our car traffic modelling. In Section 3 we develop in several steps our approach to controller synthesis for multi-lane highway traffic. In Section 4 we outline compositional verification. Conclusions are presented in Section 5.

2. Car traffic modeling

2.1. The multi-lane highway

The development of a controller is based on models of the system to be controlled. In the case of car traffic, the system consists of the traffic infrastructure, such as roads, traffic lights, etc., and cars that drive within this infrastructure. The traffic infrastructure and the cars can be modelled as consisting of multiple components.

In this paper, we consider the infrastructure of multi-lane highways as shown in Fig. 1. In this case, the infrastructure consists of a fixed number of lanes, numbered 0 through L. This infrastructure is passive. It only serves as a coordinate system in which the cars evolve. Each car has a position along the road (from left to right in the figure) and the current lanes used, normally a single lane, but during a lane change a car uses two adjacent lanes.

Each car possesses a set of sensors, which defines the part of the highway it may perceive, called its *view*. In Fig. 1, a possible view of the car A is indicated by the rectangle surrounding A. The main motivation behind the concept of views is that safety of each car only depends on its local environment. The physical constraints on such a finite set of space ensure that only finitely many cars can be responsible for unsafe situations during each maneuver. Finally, we assume that all cars adhere to the same protocol with respect to lane-change maneuvers.

2.2. A component-based hybrid modeling framework

A car can also be modelled as consisting of several components. In this paper, we consider the components shown in Fig. 2: velocity control, steering, and lane control. These components have a dynamic behavior. In order to describe such behavior, one first has to define their communication with their environment.

The interfaces over which the components communicate are indicated in Fig. 2 by arrows. We distinguish between two types of interfaces: (a) shared (real-valued) *interface variables* (solid arrows), and (b) *interaction interfaces* (dashed arrows). The static interconnection structure between components and interface variables is such that each interface variable has exactly one component (possibly the environment) that determines the value of the variable – it represents the output of that component – while several components may read the value of the variable – it is the input for those components. The actions occurring over interaction interfaces are related to discrete transitions (see below). An action is initiated by the component for which the interface is output. Examples of interface variables in Fig. 2 are *fd* (front distance to car ahead), *target velocity* and *acceleration*; examples of actions are $\mathsf{r}(m)$ and $\mathsf{wd\text{-}c}(m)$.

The dynamic behavior of a component can be described by a Hybrid Input-Output Automaton (HIOA) as defined in [LSV03]. The state of such an automaton is defined by a set of variables (internal variables and external (interface) variables, which are either input or output for a given component) and there are two types of state changes: (1) *discrete transitions*, associated with an action, are instantaneous transitions from some internal state to another internal state, which may also have an effect on the values of output variables, and (2) continuous state changes (called *trajectories* in [LSV03] and sometimes *flows* in this paper) that determine the values of output variables as a function of the input variables and the evolving time. The possible trajectories depend on the internal state of the component, sometimes called *mode*, and on the value of input variables. The actions of discrete transitions are classified into internal, input and output actions, as for Input-Output Automata (IOA) [LT89]. In contrast to [LSV03], we assume in this paper that the enabling of a discrete transition (with internal or output action) may depend on the values of input variables, as in an earlier definition of HIOA [LSVW99].

A component communicates with the components in its environment in two ways, corresponding to the two types of state changes. First, when a trajectory is performed by the automaton, the trajectory determines how the internal and output variables evolve over time as a function of the values of the input variables. The value of an input variable is determined by that component for which the variable is output. Second, when a discrete transition performs an output action, those other components for which this action is an input action will perform at the same time a discrete transition associated with this action, thus leading to a joint discrete transition of several components.

We assume that the HIOA model of a component has a single internal discrete variable that defines the current *mode* of the component. For each possible mode, the trajectories are defined in terms of so-called *flow equations* which define the continuous evolution of the real-valued variables, and each mode is associated with an *invariant* that must remain true while the component is in that mode (see for example [DHO06]). Some time before the flow reaches the border of the state space where this predicate is true, a discrete transition must take place which leads the component into another mode for which the associated predicate is true.

2.3. Modeling the dynamic behavior of a car

Informally, the dynamic behavior of the car components shown in Fig. 2 can be described as follows. The "velocity control" component receives as input the *target velocity* set by the driver, the measured *current velocity* of the car, and the *front distance*, *fd*, to be maintained, as determined by the lane control component. The defined trajectories determine the value of the *acceleration* (output variable) as a function of the input variables and time. As dependent output variables, the component also produces the value of the *safety distance*, *sd*, which depends on the speed of the car. This distance is calculated such that the car could stop before that distance in case that a fixed obstacle suddenly occurs at that distance in front of the car. In the normal operation mode, the velocity control component will select a trajectory for the acceleration (or deceleration) such that the target speed will be attained under the condition that the front distance is larger than the safety distance. A possible way to construct such a component is described in [DHO06, DMR14, ORW15].

The "steering" component controls the *steering angle* output variable which affects the front wheels of the car. It uses as input the *orientation* (angle of the car in the forward direction of the car), the *current velocity* of the car, and the measured *left-to-right position* of the car over the different lanes. It has an internal variable which contains the target lane of the car. The value of this variable is set by the input action $\mathsf{r}(m)$ which sets the target lane to the value m and thereby reserves lane m. When the *current lane* has been changed,

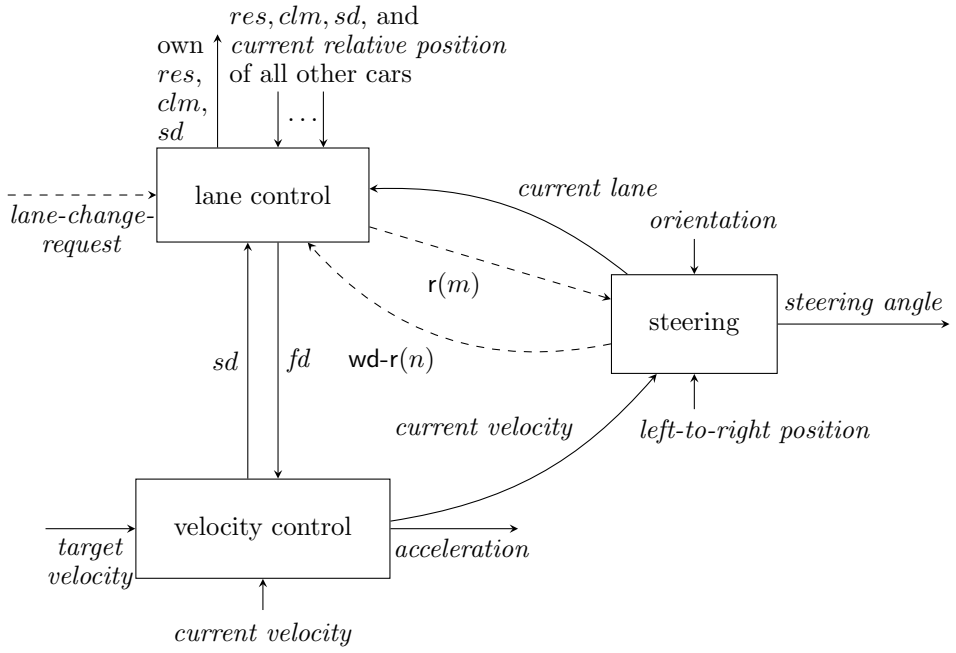

Fig. 2. Control components of a single car.

the component will perform a discrete transition with the output action wd-r(n) when the reservation of the old lane n is not needed any more. A possible way to construct such a component is described in [DMR14].

The "lane control" component is responsible for determining when a lane change maneuver can be performed. Such a lane change maneuver is requested by the driver through a discrete transition with the input action *lane-change-request* which tells the steering component to which lane it should move, taking the value of the variable *current lane* into account. Before performing such a maneuver, the component has to make sure that there is the necessary space on the new lane and that there is no conflict with other cars that may want to change their lane, as described in the following sections. For this purpose, there are a number of input and output variables through which the lane control component interacts with other cars in its environment (see Fig. 2: *res, clm, sd, current relative position*).

The lane control component is the focus of this paper. In the following, its behavior is described in more detail. It does not involve any continuous trajectories, but can be described by a finite-state input-output automaton where transitions may have guards that depend on variables.

The following lane change procedure was proposed in [HLOR11]: A car that wants to change lane, for instance the car A in Fig. 1, first "claims" the lane to which it wants to move (this corresponds to setting the turn signal ("blinker") in the manual car driving mode), and then "reserves" the new lane before it moves over on to the new lane. During this lane change, the car has two reserved lanes.

Each car has the following attributes, in addition to those mentioned above:

- *res*: the set of lanes reserved. It has at most two elements, namely the current lane n, and possibly an adjacent lane m to which the car wants to move.
- *clm*: the set of lanes claimed. It has at most one element. The claimed element must be a lane adjacent to the current lane n.

To describe the dynamic behavior of the lane change control component during lane change, the following interactions are introduced:

- c(m): introduce a claim for lane m,
- wd-c(m): withdraw the claim for lane m,
- r(m): change a claim for lane m into a reservation for lane m,
- wd-r(m): withdraw the reservation for lane m.

2.4. The safety condition

The lane control component shall ensure that cars do not collide during the lane change procedure. This means that the part of a lane that is reserved by a car should never overlap with a part that is reserved by another car. The meaning of a lane reservation by car c is that the lane is reserved for car c over the distance range from the current position of the car, $c.pos$, up to the point of its safety distance, $c.pos + c.sd$. We call this range the *safety envelope* of c. The safety distance, sd, is the distance over which the car may come to a full stop – clearly this distance depends on the current velocity of the car.

The dangerous situation of a collision is formalized by the following condition:

$$col = \exists\, c_1, c_2 : (c_1 \neq c_2 \wedge (c_1.res \cap c_2.res \neq \emptyset) \wedge safetyOverlap(c_1, c_2)), \tag{1}$$

where $safetyOverlap(c_1, c_2)$ is true if there is an overlap of the ranges from the current position up to the point of the safety distance for the two cars c_1 and c_2:

$$\begin{aligned} safetyOverlap(c_1, c_2) = {} & (c_1.pos \leq c_2.pos \leq c_1.pos + c_1.sd) \vee \\ & (c_2.pos \leq c_1.pos \leq c_2.pos + c_2.sd). \end{aligned}$$

We say that the system is *safe* if there is no overlap of the safety envelopes of any two cars on any given lane, that is, if the collision condition col is false. Therefore, the safety condition is

$$safe = \forall\, c_1, c_2 : (c_1 = c_2 \vee (c_1.res \cap c_2.res = \emptyset) \vee \neg safetyOverlap(c_1, c_2)), \tag{2}$$

We consider in this paper a car A, as shown in Fig. 1, which intends to change to the next-higher lane. In this situation, the safety condition may become false for the following reasons:

1. The distance between car A and car G may become too small and $safetyOverlap(A, G)$ becomes true. This is avoided by the behavior of the velocity control unit of car A (as explained in [ORW15]).
2. Similarly, the distance between car A and car C may become too small and $safetyOverlap(C, A)$ becomes true. This is avoided by the behavior of the velocity control unit of car C.
3. After having claimed lane 1, car A reserves this lane. At this point, the following problems may occur:
   a. The distance between car A and the cars B or E may become too small and $safetyOverlap(A, B)$ or $safetyOverlap(E, A)$ may become true. How this is avoided is discussed in Section 3.5.
   b. The car F may want to change lane to lane 1 at the same time. After claiming this lane, it may reserve it (while $safetyOverlap(A, F)$ is true. How this is avoided is the main topic of Section 3.

3. Controller synthesis for multi-lane traffic maneuvers

3.1. Overview of controller synthesis

The design of controllers for hybrid systems has to deal with two aspects: the control of the continuous flows, and the control of the discrete actions. In this paper we limit ourselves to the discrete aspects, since we concentrate the discussion on the lane control component, which has a behavior essentially characterized by discrete transitions, such as shown in Fig. 3. The synthesis of controllers for discrete event systems was first described in [RW87]. Distributed control of systems consisting of several communicating components is described in [CW10]. It turns out that the method of submodule construction, as introduced in [MvB83], can also be used for this purpose. In [vB13] this approach is formalized and described for different types of interactions between the controlled system, the environment and the controller.

The typical system architecture for controlling a single component comprises the plant (to be controlled, called world model in [DF11]), the environment, and the controller. The behavior of the plant is defined in terms of its interactions with the environment. These interactions are classified into controllable and uncontrollable interactions. The controller can observe a subset of these interactions, called the visible interactions, and it may prevent the occurrence of a visible controllable interaction, but it has no impact on uncontrollable or invisible interactions. In our modeling framework, we distinguish between input and output interactions. Plant inputs from the environment are in general uncontrollable, while input from the controller is controllable. The outputs of the plant to the controller are either controllable (can be prevented) or uncontrollable.

The environment provides input interactions to the plant, called disturbances in [DF11]. These inputs may depend on the outputs received from the plant previously. The order in which these inputs may arrive is sometimes called the *environment assumption*. The behavior of the environment may be described by a state machine model. In this case, the model explicitly describes in which state which input may be provided, thus defining the environment assumption. The environment model is also used to define *control objectives*: Safety objectives, namely that in certain states the plant should not provide certain specific outputs, can be modeled by including in the behavior of the environment a transition for such outputs into a **Fail** state – and the objective is that such a **Fail** state should never be reached.

We assume in the following that the set of possible sequences of interactions of the plant can be described by a finite automaton P where all its states s_P are accepting, and the set of interaction sequences of the environment are described by a finite automaton E where its states s_E are accepting, except the **Fail** states. In the case of full visibility, the most general controller behavior C that avoids the **Fail** states of the environment is obtained from the finite automaton $C_1 = P \times E$ (product of P with E where a state (s_P, s_E) of the product is accepting iff s_E is accepting in E). From this automaton, certain states must be pruned, that is, eliminated, in order to obtain the controller C.

Pruning is a recursive procedure. In each iteration, the following states are pruned: (a) any non-accepting states, (b) any states that have a transition with an uncontrollable interaction to a state that was pruned in an earlier iteration, and (c) any state that is a deadlock (that has no outgoing transition – we assume here that the plant and the environment, separately, do not have a final (deadlocking) state). A state is pruned by eliminating all outgoing transitions, all incoming transitions with controllable (and visible) interactions that lead into the state, and the state itself. The procedure stops when during the next iteration no further state is pruned.

If all states are eliminated by the pruning procedure, then there exists no suitable controller. However, it is important to note that, if a suitable controller is found, this controller may constrain the plant so much that the remaining behavior is not useful for the application at hand – in other words, the behavior satisfies the safety properties defined by the control objectives, but does not satisfy the liveness properties of the application. Controller synthesis including liveness objectives is discussed for instance in [ZS05].

In the case of partial visibility, the product automaton C_1 must first be projected onto the visible interactions. The resulting projected automaton, which is in general non-deterministic, must be determinized before the pruning operations can be performed. Hence, partial visibility introduces an exponential blow-up of the set of states. In the resulting automaton C_1', each state corresponds to a set of states of C_1. Any state of C_1' that includes a state (s_P, s_E) of C_1 for which s_E is non-accepting must be pruned (because if this state is reached, the environment may be in the non-accepting state s_E). In the examples discussed in this paper, all interactions are considered visible to the controller P. As explained in Section 2.2, we use in this paper Input-Output-Automata for the modeling of discrete transitions of our hybrid automata, where interactions are either input or output. The controller derivation theory can be adapted to this context (see for instance [vB13]) by considering the inputs to the controller as uncontrollable and the outputs from the controller as controllable.

For the application of multi-lane traffic control, as described in Section 2, we have a plant that consists of a large number of cars. We would like to obtain a controller per car that is able to control the controllable interactions of that car, and may possibly see some of the interactions of other cars, without being able to control them. In fact, in this paper we are mainly interested in deriving a controller for the lane control component of cars. For such a controller, all output interactions of its lane control component are controllable, but all other interactions – including output interactions of other cars – are uncontrollable. This situation is studied in [CW10] and is called distributed control. We note, however, that in general the problem of synthesizing distributed control is undecidable [Thi05].

3.2. A simple algorithm for lane change using interleaving semantics

Let us first assume that the lane control component has the simple behavior shown in Fig. 3 (a). In this case no claims are made. The notation $A.qRC$ means that car A is in control state q, it has reserved the lanes in the set R, and it claims the lanes in the set C. The car A in lane n starts with an action $\mathsf{r}(m)$, reserving one of its neighboring lanes $m \in \{n-1, n+1\}$, which is an output action that interacts with the steering component which will steer the car on to the new lane m. When this is done, that component will

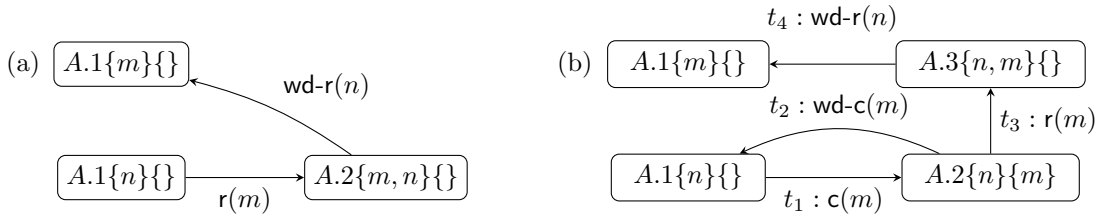

Fig. 3. Behavior of a car A changing from its current lane n to a neighboring target lane $m \in \{n-1, n+1\}$: (a) simple algorithm, (b) protocol with a claim transition $\mathsf{c}(m)$ as in [HLOR11].

withdraw from the previous lane by producing the $\mathsf{wd\text{-}r}(n)$ interactions which is received by the lane control component, and the car goes back to the normal driving condition.

Let us consider a given car *ego* with this lane change controller. Its environment consists of all the other cars in the system and the requirement that the system should be safe in all instants. A safety condition can be proven by showing that it holds in the initial state and remains invariant under all transitions and flows that the system may make. In this example, the safety objective to be satisfied is the condition $\neg\, col$ (no collision). This can be modeled by an environment E with two states, one where $\neg\, col$ holds, and one where it is false. The latter is a **Fail** state. The plant P consists of all cars operating concurrently. In order to simplify the situation, we consider only two cars, say cars A and F in Fig. 1, since we are interested in understanding what happens if two cars want to reserve the same lane at the same time. In order to understand the situation in more detail, Fig. 4 shows the states of the plant, that is, the global reachability analysis involving the two cars A and F. Building the product $C_1 = P \times E$, we see that the state 4 in the figure is a **Fail** state if the two cars have a safety overlap, that is, if $\mathit{safetyOverlap}(A, F)$ is true (which implies *col*).

Therefore this **Fail** state must be pruned, if such an overlap exists. To this end, the transitions leading into this state should be pruned (see dashed arrows in the figure). The transition $A.\mathsf{r}(n+1)$ from state 3 to state 4 shown in the figure is performed by car A. It should be pruned if $\mathit{safetyOverlap}(A, F)$ is true, because car F has already reserved lane $(n+1)$ which makes the condition *col* true for the cars A and F. Generalizing from this example, we conclude that the transition $\mathsf{r}(m)$ in Fig. 3(a) should be pruned if the following condition cc, called *collision check* in [HLOR11], is false:

$$A.cc(m) = \neg\, \exists\, c : (c \neq A \wedge (m \in c.res) \wedge \mathit{safetyOverlap}(A, c)).$$

This means that the predicate $A.cc(m)$ is an enabling condition for the transition $\mathsf{r}(m)$ of car A. The same condition for car F restricts the transition $F.\mathsf{r}(n+1)$ in Fig. 4 in such a way that the system remains safe. It is important to note that here the global system model uses the interleaving semantics [Mil89], that is, there are never two transitions that occur at the same time. That is, this simple controller is safe under the assumption of interleaving semantics. Note that interleaving semantics was also assumed in the safety proof of [HLOR11].

We note that the output action $\mathsf{r}(m)$ also induces a mode change in the lane control component which determines the front distance used by the velocity controller for keeping safe distance with the cars in front. The function determining the front distance will have to change because the car must now keep safe distance to the preceding cars on both lanes. Similarly, a mode change occurs with the subsequent $\mathsf{wd\text{-}r}(m)$ action.

3.3. Safety problems in the synchronous model

Interleaving semantics is not a realistic assumption for distributed systems where transitions are controlled independently by different components. We show now that the simple control algorithm above is not safe under a synchronous semantic model where transitions may occur concurrently. Suppose that all cars are controlled by the simple controller of Sec. 3.2 and that A and F in Fig. 1 decide at the same time that they want to reserve lane number 1. They will check whether the lane is free and then perform the $\mathsf{r}(1)$ transition. If transitions may occur concurrently, it would be possible that the cars A and F would simultaneously perform a transition $\mathsf{r}(1)$, resulting in a collision on lane 1. Within the global system as described in Fig. 4, this would be the dotted transition. However, collisions, i.e., overlapping reservations, are exactly what we

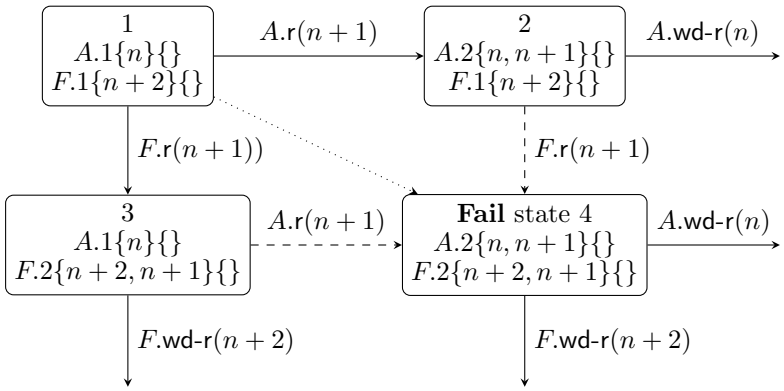

Fig. 4. Reachability analysis for two cars, A and F, behaving according to Fig. 3 (a). Solid and dashed arrows represent the transitions in the interleaving semantics. The dotted arrow represents an additional transition in the synchronous model.

intend to prevent. Furthermore, if car A creates its reservations immediately before car F, it is by no means realistic to assume that car F will notice this change before it performs its own reservation.

A better modeling paradigm to take such situations into account are synchronous systems with stuttering. In synchronous systems, all system components perform a transition in parallel during a transition period. Stuttering means that, in each transition period, a component may decide to do no transition, that is, remain in the same state. For the IOA modeling paradigm that we use for the discrete transitions of the lane control component, this means that an output transition of one component will proceed in parallel with the corresponding input transitions of those components receiving the output as input. Other components, during the same transition period, may remain in the same state or perform an internal discrete transition. For the example of lane changing cars considered in this paper, this means that a transition of the lane controller of one car may occur in parallel with a lane controller transition of another car (which is not possible in the context of interleaving semantics).

When this modeling paradigm is used for the simple lane change algorithm discussed above, there are problems as shown in Fig. 4 and discussed above. The dashed transitions are pruned by the *cc* enabling condition for the $\mathsf{r}(m)$ transition, but this condition does not prevent the possibility of simultaneous transitions of both cars from state ($A.1$, $F.1$) to state ($A.2$, $F.2$), as indicated by the dotted transition in the figure. Because of the independence of the distributed controllers in cars A and F, this dotted transition can only be pruned by also pruning the transitions from ($A.1$, $F.1$) to ($A.2$, $F.1$) and from ($A.1$, $F.1$) to ($A.1$, $F.2$), which means that no reservations can be made at all. Therefore, there *does not exist* a suitable controller for the simple algorithm for lane change when simultaneous transitions of different cars are allowed.

3.4. Lane change algorithm allowing for concurrent transitions

The problem of avoiding car collisions is an instance of the mutual exclusion problem. The space on the lane is the shared resource that must be managed in mutual exclusion by the different cars. One of the earliest mutual exclusion algorithms proposed by Dekker [Dij68] achieves this goal by introducing for each user a variable 'claimed' which can be read by the other user. Before using the resource, a user first has to set its own claimed variable to true, and then he can only use the resource if the claimed variable of the other user is false.

The lane reservation protocol proposed in [HLOR11] is based on this principle and represented in Fig. 3 (b). In case of a conflict between the two cars, both cars abandon their claim. In order to avoid infinite looping, it must be assumed that there is some random waiting before each user repeats his claim, similar to the behavior of agents in the ALOHA system [Abr70].

The proposed protocol proceeds in two steps as follows, as shown in Fig. 3 (b). First, car A claims a space on the target lane m adjacent to its current lane n by the action $\mathsf{c}(m)$. Note that an overlap of claimed spaces is not safety critical. Only the transition converting a claim into a reservation is safety critical. Therefore,

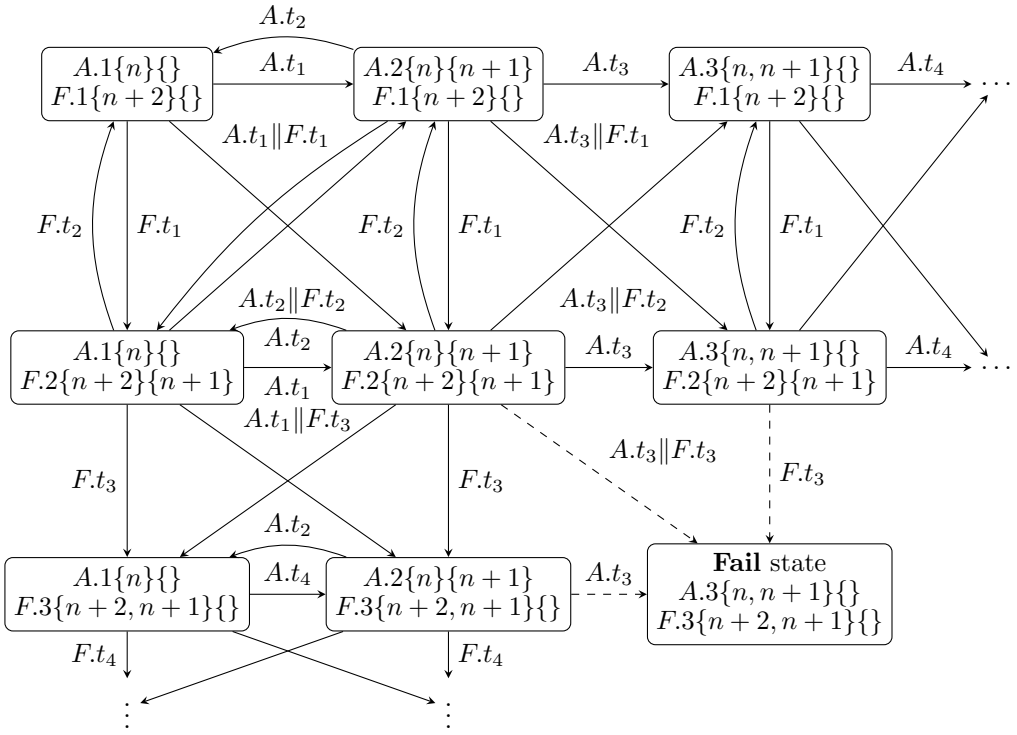

Fig. 5. Reachability analysis for two cars, A and F, behaving according to Figure 3 (b), with concurrent transitions, trying to reserve overlapping space on the same target lane $n+1$.

car A subsequently checks whether this claim intersects with the reservation or claim of any other car. If this is the case, A withdraws its claim by the action $\mathsf{wd\text{-}c}(m)$. Otherwise, A turns its claim into a reservation by the action $\mathsf{r}(m)$ so that it now reserves space on the two neighboring lanes m and n. During this double reservation A performs the lane change. Once this is completed, A withdraws its reservation on the original lane n by the action $\mathsf{wd\text{-}r}(n)$ and continues to drive on the target lane m.

In order to derive the necessary control constraints, we proceed along the lines discussed in Section 3.2. Again, we consider the plant P consisting of two cars that want to reserve the same space on a given lane, for example the cars A and F in Fig. 1. The global plant behavior is shown by the state diagram of Fig. 5 which is the product of two state machines defined by Fig. 3 (b). The figure represents the uncontrolled behavior of two cars on lanes n and $n+2$ that both want to move to lane $m = n+1$. If we build the product of the plant behavior with the environment objective, $C_1 = P \times E$, we see that the lower right state becomes a **Fail** state where both cars collide if they have an overlap.

As in Section 3.2, we can introduce constraints (pruning) in the state machine of Fig. 3 (b) in order to eliminate the transitions into the **Fail** state. This means that we introduce a constraint on the r transition t_3 in Fig. 3 (b), such that this transition is not possible when the global system is in a state where the other car is in state 2 or 3, as shown in Fig. 5 by the dashed transitions. These states are characterized by the fact that the other car either has claimed or reserved an overlapping space on the same lane. Therefore the constraint for the transition of a car *ego* is the following condition *pcc*, called *potential collision check*:

$$ego.pcc(m) = \neg\,\exists\, c : (c \neq ego \wedge (m \in c.res \vee m \in c.clm) \wedge safetyOverlap(ego, c)).$$

If this constraint is implemented in both cars, then the joint transition from state ($A.2$, $F.2$) in Fig. 5 directly into the **Fail** state will also be eliminated and the system is safe.

We note that the lane change algorithm obtained by our derivation approach is very similar to the lane change controller proposed in [HLOR11]. There it was proven correct under an interleaving semantics. This paper shows that the algorithm of [HLOR11] is also correct under the synchronous model.

It is important to note, however, that the lane control component should remain in the state where a claim

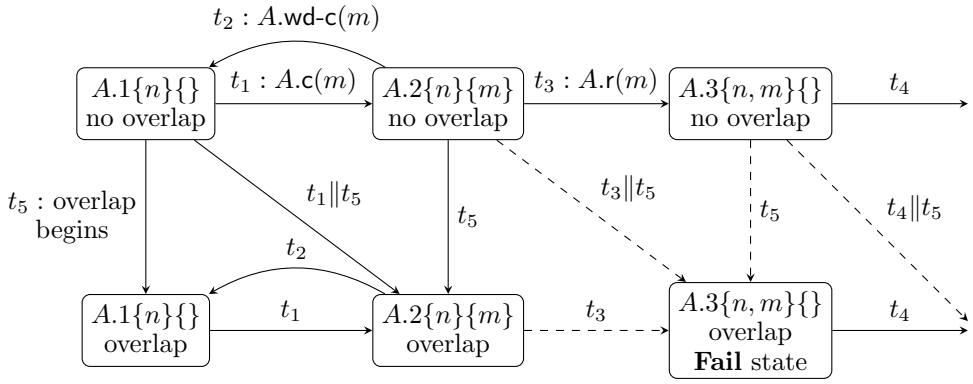

Fig. 6. Reachability analysis of car A making a reservation and another car driving on A's target lane and creating an overlap with the area to be reserved.

is made for a sufficiently long period such that all cars in the environment have seen this claim before the claim is converted into a reservation. In our modeling framework, we assume that all state changes happening during a set of parallel transitions have been seen by all system components before a next transition will be performed.

We note that in practice, one would probably introduce some constraint for the *claim* transition t_1 in Fig. 3 (b) in order to reduce the frequency of the looping behavior where a *claim* is directly followed by a $withdraw - claim$ transition. One could apply the same *pcc* condition that is applied to the *reservation* transition.

3.5. Interference with cars on the target lane

There is also the (potential) problem that a fast driving car (say car E in Fig. 1) may lead to a safety overlap with the car that changes lanes (such as car A in Fig. 1) just at the time when the latter makes the new reservation. And the same may happen when a car in front (say car B in Fig. 1) drives much slower than the lane changing car (such as car A in Fig. 1). These problems can be avoided, if we strengthen our condition for changing claims to reservations.

These situations are analyzed in Fig. 6. The figure shows the product of the behavior of the lane changing car of Fig. 3 (b) and the changing situation of the relative positions of the lane changing car with the car on the target lane behind or in front, e.g., car E and B in Fig. 1, respectively. We assume that initially (in the upper states of Fig. 6), there is no safety overlap, and at some point in time, the overlap begins – this is represented by the (vertical) transition t_5. Again, there is a **Fail** state that should never be reached. In the following, we discuss how the three transitions leading into the **Fail** state can be avoided:

- The transition t_5 from state ($A.3\{m,n\}\{\}$, no overlap) cannot occur in relation with a slow car in front, e.g. car B in Fig. 1, since in this state two lanes, m and n, are reserved by the lane changing car and the speed control component will use the minimum of the distance in the two reserved lanes to the cars in front for controlling its own speed, therefore avoiding any safety overlap. – In case of a conflict with a fast car behind, e.g. car E in Fig. 1, the latter will use the distance to the new reservation for its own speed control. We assume that this new reservation is visible to the car behind.
- The $\mathsf{r}(m)$ transition t_3 from state ($A.2\{n\}\{m\}$, overlap) cannot occur because the lane changing car uses the *pcc* constraint for this transition.
- If we allow that transitions of different cars occur concurrently, then we must also consider the possibility that the two transitions t_5 and t_3 occur simultaneously, as shown by the transition $t_3 \| t_5$ in Fig. 6. One way to avoid this joint transition is to strengthen the constraint *pcc* of t_3 by introducing a safety margin $\epsilon > 0$ between the reservations of different cars on the same lane, that is, we use in the definition of *pcc*

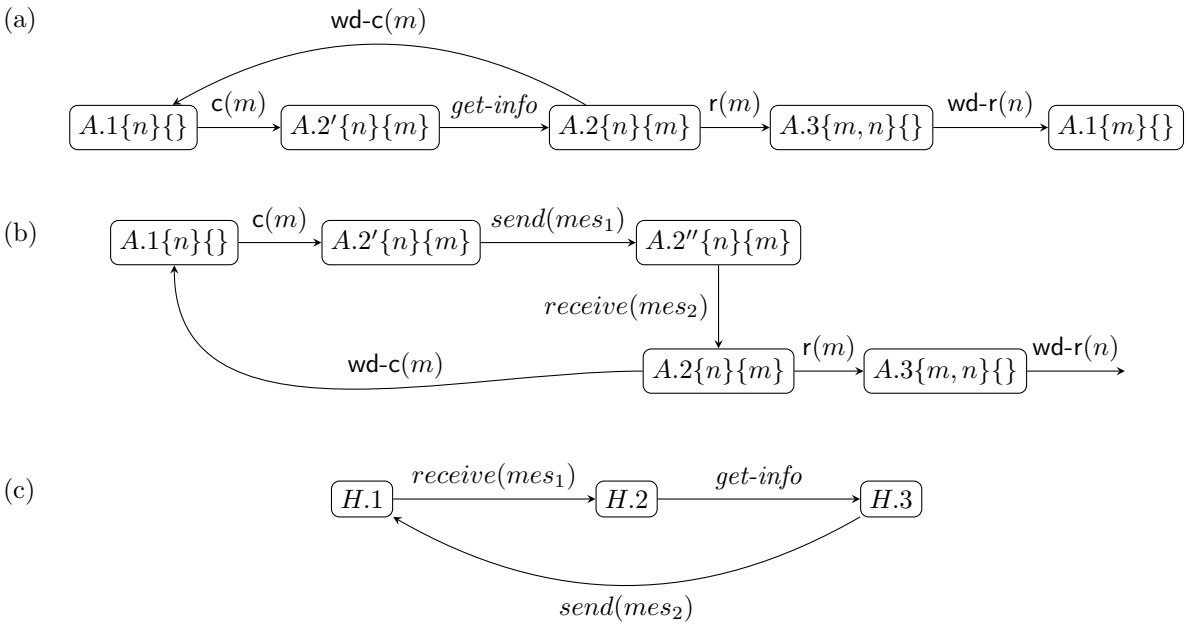

Fig. 7. (a) Global transition diagram involving a helper car that reads its own safety distance. Derived behavior for (b) lane changing car and (c) helper car.

a larger "robust" safety overlap

$$\begin{aligned} \mathit{safetyOverlapR}(c_1, c_2) = {} & (c_1.pos - \epsilon \leq c_2.pos \leq c_1.pos + c_1.sd + \epsilon) \vee \\ & (c_2.pos - \epsilon \leq c_1.pos \leq c_2.pos + c_2.sd + \epsilon). \end{aligned}$$

That is, we add an additional length of ϵ to both sides of the reservation while checking for an overlap. Since this formula will be satisfied when transition t_3 occurs, the additional distance $\epsilon > 0$ between the new reservation and the other existing reservations will ensure that the transition t_5 (where this distance becomes zero) cannot occur concurrently with t_3.
Since *pcc* with *safetyOverlapR* is stronger than *pcc* with *safetyOverlap*, our reasoning concerning the transition t_3 from ($A.2\{n\}\{m\}$, overlap) still holds. Of course, safety with this approach depends on a sensible choice of ϵ, which will depend on the maximum difference of speeds of different cars and the time it takes for a car to determine (to "read") the reservations of the other cars.

3.6. Using a helper car

The preceding discussion assumes that a driving car has local knowledge about the reserved and claimed lanes of other cars in its environment and of the position and safety distance of these other cars. Among this information, the safety distance is probably the most difficult to obtain since it depends on the position and velocity of the other car. Therefore it is considered in [HLOR11] that this information may be obtained through message exchanges with another car in the environment, which is called a *helper car*. Such a car c should be on the target lane, but behind the lane changing car *ego*. It should provide information for the evaluation of the *safetyOverlap*(ego, c) predicate. This predicate must be evaluated in state 2 of Fig. 3 (b), before the transition $\mathsf{r}(m)$ can be performed. In Fig. 1, car E is a helper car for A in its lane change.

We would like to derive the behavior of the lane changing and helper cars from the behavior discussed in Section 3.4 for the case that the safety distance information is available locally in each car. For this purpose, we can use the derivation algorithm described in [GvB90] or use the approach described in [CvBB13]. In both cases, one starts out with a global specification of the different actions and their order of execution without being preoccupied by the question which components is responsible for executing each action, such as shown in Fig. 3 (b). After the different actions are allocated to the components that are responsible for

their execution, a so-called protocol derivation algorithm constructs the local behavior specification for each component which include, in addition to the actions for which the component is responsible, the exchanges of coordination messages that are required for assuring the orderly execution of all these actions.

The principle of the protocol derivation algorithm [GvB90] is to copy the control flow graph of the global specification for execution by each component, but to ignore all actions performed by other components and include instead the sending and reception of certain coordination messages. These coordination messages depend on the control flow operators in the global specification. For concurrency and weak sequencing, i.e., where sequencing is enforced only locally inside components [MR97], no coordination messages are required. However, they are essential for strict (i.e., global) sequencing, alternatives and loops. If a local action a_1 performed by component c_1 is followed (strictly) by an action a_2 performed by another component c_2, a coordination message will be send by c_1 to c_2, and the message will be received by c_2 before the local action a_2 is performed.

In the case of the system of a lane changing car with its helper car, we take as global behavior specification a modified version of Fig. 3 (b), where an additional *get-info* action is introduced before entering state 2, as shown in Fig. 7 (a). This action is executed by the helper car, while all other actions are executed by the lane changing car. If we apply the protocol derivation algorithm, we obtain the behaviors for the lane changing car and the helper car shown in Figures 7 (b) and (c). Sending and receiving the synthesized coordination messages mes_1 and mes_2 guarantee the right sequencing of the *get-info* action and the actions of the lane changing car.

The behavior of Fig. 7 (c) should be performed in each car by the lane control component concurrently with its normal behavior described by Fig. 7 (b). The message mes_1 is effectively a request to send the safety distance information, and the message mes_2 sent by the helper car contains the information.

The algorithm obtained here is quite different from the algorithm proposed in [HLOR11]. The reason is that in [HLOR11], the helper car makes the decision whether a lane change can be done and answers yes or no. In our approach, the helper car simply returns the value of a local variable; the decision whether the lane change can be done remains with the lane changing car. The algorithm proposed in this paper is simpler.

3.7. More complex lane change maneuvers

Following the approach presented in this paper a car can change to its target lane, only if there is currently a sufficiently large gap adjacent to the car.

This strategy limits the number of successful lane changes, e.g. a sufficiently large gap moving with the same speed as the car but misaligned can never be used for a lane change. To increase traffic flow by increasing the number of successful lane changes, one could extend the presented maneuver as described in detail in [WSRR11]:

- If there is a sufficiently large gap on the target lane in front of car E's current position, E accelerates and starts the maneuver presented here, as soon as the gap aligns with E.
- If there is no sufficiently large gap in the vicinity of car E, it negotiates with the surrounding cars to create a gap large enough for E to change lanes.

To implement either of these strategies would require more sensor input, e.g. measuring the relative speeds of the surrounding cars. The second maneuver additionally requires more communication, between the cars, e.g. for reaching consensus on where to make the gap. Both approaches would also require time to be represented in the model, since in both cases time has to elapse to reach (or create, respectively) the gap.

4. Compositional system verification

4.1. The framework

Compositional system verification and design using an assumption-guarantee-style reasoning have a long history in program verification, software engineering, and more recently, hybrid (cyber-physical) system design. For shared variable concurrent programs, work on rely-guarantee dates back to Jones [Jon83]. The book by de Roever et al. [dRdBH$^+$01] gives a detailed account on suitable semantics and proof rules. For reactive systems, the assume-guarantee paradigm for temporal properties was explored by Pnueli [Pnu85]

and systematically explored by Abadi and Lamport [AL93], where a suitable proof is proposed. For object-oriented software development, Meyer coined the notion of *design by contract* [Mey97], where a contract is a pair of assumptions and guarantees. For more general system design Benveniste et al. [BCF+08] develop a contract-based approach, and in [BCN+12] numerous pointers to the literature can be found. Specifically for hybrid systems, Damm et al. explore component-based design [DDOP10] and apply this in a case study involving concurrent velocity and steering control [DMR14].

Formally, a *contract* is a pair $C = (A, G)$ consisting of a set A of *assumptions* and a set G of *guarantees* or *goals.*. A system M *satisfies* a contract C, written $M \models (A, G)$, if under the assumptions A, the system M establishes the guarantees G. Consider a system M consisting of the parallel composition of components M_i for $i \in I = \{1, \ldots, n\}$. We stipulate that each component M_i satisfies the contract consisting of the assumptions A_i and the guarantees G_i, expressed by $M_i \models (A_i, G_i)$. To show that the system M satisfies the contract (A, G), the following proof rule dating back to Abadi and Lamport [AL93] is in widespread use:

$$\frac{\begin{array}{ll}(1) & \bigwedge_{i\in I} G_i \Rightarrow G,\\ (2) & A \wedge \bigwedge_{i\in I} G_i \Rightarrow \bigwedge_{i\in I} A_i\\ (3) & M_i \models (A_i, G_i) \text{ for } i \in I\end{array}}{M \models (A, G)}$$

where the premises (1)–(3) establish the following:

(1) shows that the goals of the system are implied by the guarantees of the components;

(2) shows that the assumptions of all components are implied by the guarantees of all components and the assumptions that the system makes about the environment; and

(3) shows that the implementation of each component establishes the guarantees of its contract if the corresponding assumptions are satisfied.

Abadi and Lamport [AL93] showed that for discrete reactive systems, such specifications avoid circular definitions. They assumed that each discrete interaction is produced as output by a single component, but it may be consumed by several components as input. The guarantees of a component are concerned with properties of the outputs produced by that component (under which circumstances they may occur, what values of parameters they may have, and what effect they may have on shared variables), and assumptions are concerned with similar properties of the inputs received by that component. The compositional verification can then be performed by induction over the length of the interaction sequence. For each subsequent interaction, the guarantees of the outputting components must imply the assumptions that all inputting components make about this interaction.

This is presented in [AL93] for discrete systems. In the case of hybrid systems, there are in addition assumptions about the continuous evolution of input variables during flow transitions, and guarantees for the continuous evolution of output variables. That is, we need to distinguish between continuous flows and discrete transitions. Such a distinction is possible using extensions of temporal logic, e.g., differential dynamic logic [Pla10], or the temporal variant of MLSL [Lin15]. In order to avoid notational overhead, we do not state the temporal aspects of the assumptions and guarantees in a fully formal way. Whenever we make use of temporal operators like $\Box$ (*always*), they refer to traces of both continuous and discrete behavior in a similar fashion to HRELTL [CRT15] or temporal dynamic logic [Pla07]. That is, $\Box F$ is true for a trace t of the system, if and only if F is true at all discrete transitions, and at all time points in the continuous flows within t.

4.2. Verification of multi-lane traffic maneuvers

The system discussed in this paper consist of many cars, each containing the three components velocity controller VC, steering controller SC, and lane controller LC (see Fig. 2). Their specification contract is given by the following assumptions Ai and guarantees Gj. We note that each assertion either is about the continuous flow or about a discrete state transition that happens in reaction to one of the discrete interactions within the system. For example, for the velocity controller, VC.A1 and VC.G1 below talk about

any continuous flow, while VC.A2 relates to a state change resulting from a discrete interaction $\mathsf{r}(m)$ or $\mathsf{wd\text{-}r}(n)$.

For the velocity controller VC we consider:

- VC.A1 : During a continuous flow, the front distance *fd* (input) does not decrease faster than the *current velocity cv* (that is, there is no car in front that goes in the opposite direction): $fd' > -cv$, where fd' denotes the timewise derivation of *fd*.
- VC.A2 : After a discrete change of the front distance *fd* (which may be caused by an interaction $\mathsf{r}(m)$ or $\mathsf{wd\text{-}r}(n)$ of the lane controller), the front distance is still larger than the safety distance: $sd < fd$.
- VC.G1 : During a continuous flow, the safety distance *sd* (output) remains smaller than the front distance *fd* (input): $sd < fd$. Note that this guarantee ensures, if VC.A2 is also satisfied, that the condition $sd < fd$ is always true.

For the steering controller SC we consider:

- SC.A1 : When an input $\mathsf{r}(m)$ is received, $|m - current\_lane| = 1$ holds.
- SC.G1 : After the input of an $\mathsf{r}(m)$ interaction, the current *left-to-right position* will become centered around the middle of lane m, and then an output $\mathsf{wd\text{-}r}(n)$ will be sent (where n is the previous current lane).

For the lane controller LC we consider:

- LC.A1 : After an output $\mathsf{r}(m)$ is produced, an input $\mathsf{wd\text{-}r}(current\_lane)$ will eventually be received. Using the *eventuality* operator $\Diamond$ of temporal logic, we express this as: $\mathsf{r}(m) \Longrightarrow \Diamond\ \mathsf{wd\text{-}r}(current\_lane)$.
- LC.G1 : When an output $\mathsf{r}(m)$ is produced, $\forall c : c \neq ego \rightarrow m \notin c.res \vee \neg safetyOverlap(ego, c)$, that is, there is no overlap between the safety envelope of the current car *ego* and the space reserved for the safety envelope of any other car on lane m.
- LC.G2 : The front distance output *fd* is always equal to the distance to the next car reservation in front on the current lane, or when, in a lane change maneuver, a target lane is reserved, to the minimum of this distance and the distance to the reservation in front on the target lane.
- LC.G3 : When an output $\mathsf{r}(m)$ is produced, $|m - current\_lane| = 1$ holds.

For the multi-lane car traffic system, the goal G is that the safety condition

$$safe = \forall\, c_1, c_2 : (c_1 = c_2 \vee (c_1.res \cap c_2.res = \emptyset) \vee \neg safetyOverlap(c_1, c_2))$$

stated in Section 2.4 should always hold. In terms of temporal logic, we express this as:

$$G = \Box\ safe.$$

The assertion A which the multi-lane car traffic system assumes to be satisfied by the environment is as follows:

$A = safe$ holds initially and VC.A1 holds during all continuous flows.

We now turn to the compositional verification.

Verification of the safety goal – premise (1) *of the proof rule:*

The assertion *safe* may become false in the following situations: (1) when the intersection $c_1.res \cap c_2.res$ becomes non-empty, and (2) when the $safetyOverlap(c_1, c_2)$ between two cars c_1 and c_2 with reservations on the same lane becomes true. The first situation may only occur when c_1 (or c_2) executes an $\mathsf{r}(m)$ interaction. However, in this case the guarantee LC.G1 states that for all other cars c_2 (or c_1) either m is not included in $c_2.res$ (or $c_1.res$, and therefore $c_1.res \cap c_2.res$ remains empty) or $safetyOverlap(c_1, c_2)$ is false. Therefore *safe* remains true.

In the second situation (two cars with reservations on the same lane), we consider the car further behind for which the VC component controls the speed such that the safety distance *sd* is always smaller than the front distance *fd* (guarantee VC.G1). Since the front distance *fd* is the minimum of the distances to the reservations on the lanes for which the car has its own reservations (guarantee LC.G2), the distance to all these cars remains larger than the safety distance *sd*, and the assertion *safe* remains true during the continuous flow.

Verification of the component assumptions – premise (2) *of the proof rule:*

- VC.A1 is trivially implied by the assumptions A of the multi-lane car traffic system.
- VC.A2 refers to the discrete interactions $\mathsf{r}(m)$ and $\mathsf{wd\text{-}r}(n)$. LC.G2 provides guarantees about the resulting values of the *fd* variable. Only in case of an $\mathsf{r}(m)$ interaction is there any danger of violating VC.A2. For this interaction, LC.G1 (for this car and all other cars in the system) ensures that VC.A2 is satisfied after the interaction.
- SC.A1 is implied by LC.G3.
- LC.A1 is implied by the second part of SC.G1.

Verification of component implementations – premise (3) *of the proof rule:*

A possible implementation of the lane controller is discussed in detail in Section 3. The main issue there was the proof of the guarantee LC.G1. LC.G2 was discussed in Section 3.5. LC.G3 follows directly from the state machine shown in Figure 3(b) because the transition $t_1 : \mathsf{c}(m)$ can only occur when the condition $m \in \{n-1, n+1\}$ is satisfied, where n refers to the current lane.

4.3. Other considerations

Liveness:

We showed above that safety is always satisfied. In addition, one may also be interested in proving a liveness condition, namely that some lane change will actually occur after a *lane-change-request* is received. In fact, SC.G1 states that some lane change will occur if an $\mathsf{r}(m)$ input is received. However, there is no guarantee that the lane controller will generate the $\mathsf{r}(m)$ interaction after a *lane-change-request* is received.

As discussed on Section 3.4, the t_3 transition of the LC state machine of Figure 3(b), which generates the $\mathsf{r}(m)$, has the enabling condition that the car has no safety overlap with any reservation made by another car. This condition may remain continuously false in very heavy traffic conditions, or there may be contention with another car that wants to use the space (e.g. car F in Fig. 1), This contention may lead to an infinite loop, which may be avoided by given priority to one of the cars, or by introducing a random delay before the claim is repeated.

An extension:

An interesting extension of the car behavior is described in [DMR14] which concerns the reduction of the velocity when a curved trajectory is followed in order to keep the passenger comfortable with the exerted centrifugal force. This involves a collaboration between the steering controller and the velocity controller, as expressed by the following additional guarantees and assumptions:

SC.A2 : When the car gets into a curve, the *current velocity* is sufficiently small to guarantee SC.G2 can be satisfied.

SC.G2 : The centrifugal force due to steering is always smaller than the driver's comfort limit.

VC.G2 : equivalent to SC.A2.

VC.G3 : The *current velocity* (output) is close to the *target velocity* (input).

The assumption SC.A2 could be assured by a corresponding guarantee of the velocity controller (namely VC.G2). However, to realize such a guarantee, the velocity controller requires some additional inputs. For instance, one could introduce an additional sensor that signals speed restrictions and provides two additional inputs to the velocity controller: maximum velocity and distance where this speed restriction applies (such restrictions may be due to highway curvature or to speed limit signs).

We note that SC.G2 (which requires VC.G2) and VC.G3 are conflicting goals, that is, it may be impossible to realize them simultaneously. In [DMR14] it is proposed to order conflicting goals according to their priority. In this example, SC.G2 would have higher priority than VC.G3. The paper [DMR14] also discusses the development of appropriate control procedures for conflicting goals.

5. Conclusion

This paper revisits the traffic maneuvers on multi-lane highways as discussed in [HLOR11]. The main conclusions of the discussions in this paper, which apply to the control of hybrid systems in general, are as follows:

(1) For verifying the safety of systems consisting of several loosely coupled components, where the behavior of a component may depend on the state of other components and where there may be some (even small) delay of communication, a modeling paradigm using interleaving semantics is not suitable. The possibility that different discrete transitions of several components occur in parallel must be considered, which can be modeled by synchronous modeling paradigms. (For a detailed discussion, see Section 3.3).

(2) Well-known algorithms for synthesizing controllers for discrete event systems (e.g. [CW10]) can be used for synthesizing controllers for the discrete transitions of hybrid systems. Corresponding algorithms exist for interleaving semantics, synchronous systems, and IOA [vB13]. (For a detailed discussion, see Section 3.1).

(3) When the global behavior involving several system components is known, for instance the actions that should be performed by different controllers of different components, then the behavior of each controller, including the exchange of coordination messages, can be synthesized using an algorithm described in [GvB90]. (For a detailed discussion, see Section 3.6).

(4) A modular modeling of hybrid systems is possible, however, the different system components are normally not independent of one another. The mutual dependencies should be described using an assumption-guarantee paradigm where the specification of each component describes (a) the assumptions that the component makes about the behavior of the other components in its environment, and (b) the guarantees that the component makes about its own behavior. In the HIOA specification paradigm described in [LSV03], the assumptions involve constraints on the possible trajectories of input variables and assumptions about possible discrete input actions.

Acknowledgements. This research was partially supported by the German Research Council (DFG) in the Transregional Collaborative Research Center SFB/TR 14 AVACS, and the Natural Science and Engineering Research Council of Canada. This journal version is an extended and revised version of [vBHLO15].

References

[Abr70] N. Abramson. The ALOHA system: Another alternative for computer communications. In *Proc. Fall Joint Computer Conf.*, AFIPS '70, pages 281–285. ACM, 1970.

[AL93] M. Abadi and L. Lamport. Composing specifications. *ACM Trans. Program. Lang. Syst.*, 15(1):73–132, 1993.

[BCF+08] A. Benveniste, B. Caillaud, A. Ferrari, L. Mangeruca, R. Passerone, and C. Sofronis. Multiple viewpoint contract-based specification and design. In F.S. de Boer, M.M. Bonsangue, S. Graf, and W.P. de Roever, editors, *Formal Methods for Components and Objects (FMCO 2007)*, volume 5382 of *LNCS*, pages 200–225. Springer, 2008.

[BCN+12] A. Benveniste, B. Caillaud, D. Nickovic, R. Passerone, J.-B. Raclet, P. Reinkemeier, A. Sangiovanni-Vincentelli, W. Damm, T. Henzinger, and K. Larsen. Contracts for systems design. Technical Report 8147, INRIA Research Center Rennes – Bretagne Atlantique, November 2012. 64 pp.

[CRT15] A. Cimatti, M. Roveri, and S. Tonetta. HRELTL: A temporal logic for hybrid systems. *Information and Computation*, 245:54 – 71, 2015.

[CvBB13] H. N. Castejón, G. v. Bochmann, and R. Bræk. On the realizability of collaborative services. *Software and System Modeling*, 12(3):597–617, 2013.

[CW10] K. Cai and W.M. Wonham. Supervisor localization: A top-down approach to distributed control of discrete-event systems. *IEEE Trans. Autom. Control*, 55(3):605–618, 2010.

[DDOP10] W. Damm, H. Dierks, J. Oehlerking, and A. Pnueli. Towards component based design of hybrid systems: Safety and stability. In Z. Manna and D.A. Peled, editors, *Time for Verification, Essays in Memory of Amir Pnueli*, volume 6200 of *LNCS*, pages 96–143. Springer, 2010.

[DF11] W. Damm and B. Finkbeiner. Does it pay to extend the perimeter of a world model? In M.J. Butler and W. Schulte, editors, *Formal Methods (FM 2011)*, volume 6664 of *LNCS*, pages 12–26. Springer, 2011.

[DHO06] W. Damm, H. Hungar, and E.-R. Olderog. Verification of cooperating traffic agents. *Intern. Journal of Control*, 79(5):395–421, 2006.

[Dij68] E. W. Dijkstra. Cooperating sequential processes. In F. Genuys, editor, *Programming Languages: NATO Advanced Study Institute*, pages 43–112. Academic Press, 1968.

[DMR14] W. Damm, E. Möhlmann, and A. Rakow. Component based design of hybrid systems: A case study on concurrency and coupling. In *Hybrid Systems: Computation and Control (HSCC)*, pages 145–150. ACM, 2014.

[dRdBH+01] W.P. de Roever, F.S. de Boer, U. Hannemann, J. Hooman, Y. Lakhnech, M. Poel, and J. Zwiers. *Concurrency Verification – Introduction to Compositional and Noncompositional Methods*. Cambridge University Press, 2001.

[FB11] C. Frese and J. Beyerer. A comparison of motion planning algorithms for cooperative collision avoidance of multiple

cognitive automobiles. In *IEEE Intelligent Vehicles Symposium (IV), 2011, Baden-Baden, Germany, June 5-9, 2011*, pages 1156–1162, 2011.

[GvB90] R. Gotzhein and G. v. Bochmann. Deriving protocol specifications from service specifications including parameters. *ACM Trans. Comput. Syst.*, 8(4):255–283, 1990.

[HLOR11] M. Hilscher, S. Linker, E.-R. Olderog, and A.P. Ravn. An abstract model for proving safety of multi-lane traffic manoeuvres. In *Proc. ICFEM*, pages 404–419. Springer, 2011.

[Jon83] C.B. Jones. Specification and design of (parallel) programs. In R.E.A. Mason, editor, *Information Processing*, volume 83 of *IFIP*, pages 321–332. Elsevier (North-Holland), 1983.

[LGS98] J. Lygeros, D.N. Godbole, and S.S. Sastry. Verified hybrid controllers for automated vehicles. *IEEE Trans. on Automatic Control*, 43(4):522–539, 1998.

[Lin15] Sven Linker. *Proofs for traffic safety – combining diagrams and logic*. PhD thesis, Carl von Ossietzky Universität Oldenburg, 2015.

[LPN11] S.M. Loos, A. Platzer, and L. Nistor. Adaptive cruise control: Hybrid, distributed, and now formally verified. In Michael Butler and Wolfram Schulte, editors, *Formal Methods (FM 2011)*, volume 6664 of *LNCS*, pages 42–56. Springer, 2011.

[LSV03] N.A. Lynch, R. Segala, and F.W. Vaandrager. Hybrid I/O automata. *Inf. Comput.*, 185(1):105–157, 2003.

[LSVW99] N.A. Lynch, R. Segala, F.W. Vaandrager, and H.B. Weinberg. Hybrid I/O automata. Technical Report CSI-R9907, Computing Science Institute, University of Nijmegen, April 1999.

[LT89] N.A. Lynch and M.R. Tuttle. An introduction to input/output automata. *CWI Quarterly*, 2(3):219–246, 1989.

[Mey97] B. Meyer. *Object-Oriented Software Construction*. Prentice Hall, 2nd edition, 1997.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[MR97] S. Mauw and M.A. Reniers. High-level message sequence charts. In A. Cavalli and A. Sarma, editors, *SDL 1997: Time for Testing – SDL, MSC and Trends*, pages 291–306. Elsevier Science B.V., 1997.

[MvB83] P. Merlin and G. v. Bochmann. On the construction of submodule specifications and communication protocols. *ACM Trans. Program. Lang. Syst.*, 5(1):1–25, January 1983.

[ORW15] E.-R. Olderog, A. P. Ravn, and R. Wisniewski. Linking spatial and dynamic models for traffic maneuvers. In *54th IEEE Conf. on Decision and Control (CDC), Osaka, Japan*, pages 6809–6816. IEEE, Dec 2015.

[Pla07] A. Platzer. A temporal dynamic logic for verifying hybrid system invariants. In S.N. Artemov and A. Nerode, editors, *Logical Foundations of Computer Science: International Symposium (LFCS 2007)*, volume 4514 of *LNCS*, pages 457–471. Springer, 2007.

[Pla10] A. Platzer. *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*. Springer, 2010.

[Pnu85] A. Pnueli. In transition from global to modular reasoning about programs. In K.R. Apt, editor, *Logics and Models of Concurrent Systems*, volume 13 of *NATO ASI Series F*, pages 123–144. Springer, 1985.

[RW87] P.J. Ramadge and W.M. Wonham. Supervisory control of a class of discrete event processes. *SIAM J. Control Optim.*, 25(3):206–230, 1987.

[Thi05] J.G. Thistle. Undecidability in decentralized supervision. *Systems & Control Letters*, 54(5):503–509, 2005.

[vB13] Gregor v. Bochmann. Using logic to solve the submodule construction problem. *Discrete Event Dynamic Systems*, 23(1):27–59, 2013.

[vBHLO15] G. v. Bochmann, M. Hilscher, S. Linker, and E.-R. Olderog. Synthesizing controllers for multi-lane traffic maneuvers. In X. Li, Z. Liu, and W. Yi, editors, *1st Symp. on Dependable Software Engineering: Theories, Tools and Applications (SETTA)*, volume 9409 of *LNCS*, pages 1–16. Springer, 2015.

[WSRR11] B. Wirtz, T. Strazny, J. Rakow, and A. Rakow. A lane change assistance system: Cooperation and hybrid control. Technical Report 78, SFB/TR 14 AVACS, July 2011. ISSN: 1860-9821, http://www.avacs.org.

[ZS05] R. Ziller and K. Schneider. Combining supervisor synthesis and model checking. *ACM Trans. Embed. Comput. Syst.*, 4(2):331–362, May 2005.