Synthesizing Controllers for Multi-Lane Traffic Maneuvers [⋆]

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

[1] School of Electrical Engineering and Computer Science, University of Ottawa, Ottawa, Ontario, Canada
`bochmann@uottawa.ca`
[2] Department of Computing Science, University of Oldenburg, Oldenburg, Germany
`{hilscher, linker, olderog}@informatik.uni-oldenburg.de`

Abstract. The dynamic behavior of a car can be modeled as a hybrid system involving continuous state changes and discrete state transitions. However, 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.

Keywords. Multi-lane highway traffic, lane-change maneuver, safety, collision freedeom, hybrid systems, controller synthesis, interleaving and synchronous concurrency

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 part or pattern of the system that can be automated is of great help.

Well-known is the California PATH (Partners for Advanced Transit and Highways) project that developed automated highway systems for cars driving in groups called platoons [1]. The maneuvers include joining and leaving the platoon, and lane change. Lygeros et al. [2] sketch a safety proof for car platoons taking car dynamics into account, but admitting collisions at a low speed.

[⋆] 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 paper is motivated by previous work in [3], where an abstract model of highway traffic was introduced, consisting of so-called traffic snapshots. The main idea of [3] was that safety is a spatial property. Using a dedicated spatial logic called Multi-Lane Spatial Logic (MLSL) to describe spatial properties concisely, we presented two controllers for the lane-change maneuver on highways and proved that under certain assumptions the controllers guarantee safety. However, the controllers themselves were introduced in an ad-hoc manner.

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 [3] (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 [4]. 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 [5].
- 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 [3], 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 [6].

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

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.

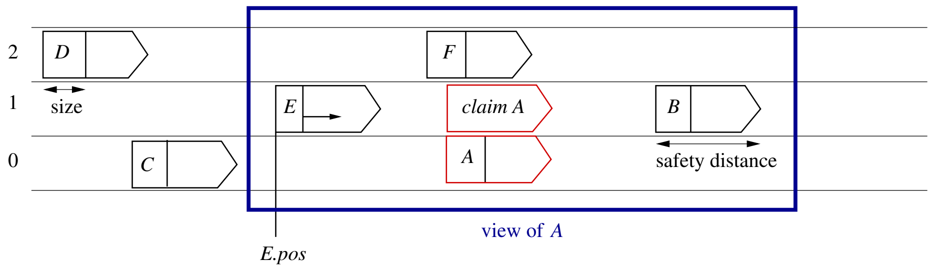

Fig. 1. A multi-lane highway with several cars. The large rectangle shows the view of car A, i.e., the part of the environment visible to A.

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 posssesses a set of sensors, which defines the part of the highway it may perceive, called its *view*. In Fig. 2, 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, since we assume that all cars behave alike, it is sufficient to analyse the interaction of two cars: if an accident happens, at least two cars are colliding. In this paper, we deal with the conflicting situation where two cars, say cars A and F in Fig. 1, claim space on the same lane. For conflicts between a claiming car and a car already on the claimed lane, say cars A and E in Fig. 1, we refer to our extended version [7].

2.2 A hybrid model of a car

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 are not passive, but have 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*, and (b) so-called *interaction interfaces* (dashed arrows). The static interconnection structure between components and

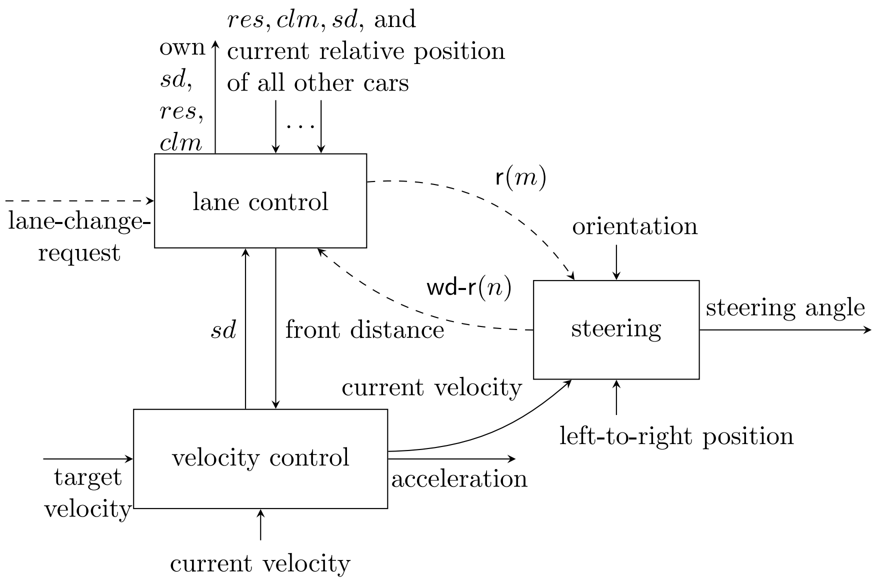

Fig. 2. Control components of a single car.

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 target-speed and acceleration; examples of actions are $\mathsf{r}(m)$ and $\mathsf{wd\text{-}c}(m)$.

Each component can be modelled as a Hybrid Input-Output Automaton. Its dynamic behavior is determined by (a) continuous state changes (called *trajectories* in [8]) which determine the values of output variables as a function of input variables and the evolving time, and (b) *discrete transitions*, associated with internal actions or interactions, which change the internal state (called mode) which determines the trajectories that are active in this mode and usually associated with an invariant that holds in this mode (see for example [9])

The roles of the components shown in Fig. 2 are 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 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 speed 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 [9,10].

The *steering* component controls the direction output variable, which acts via the steering angle on the front wheels of the car. It uses as input the orientation (angle of the car to the forward direction of the lane), 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. When the current lane has been changed, the component will perform a discrete transition with the output action $\mathsf{wd\text{-}r}(n)$ when the reservation of the old lane n is not needed any more. A possible way to construct such a component is described in [10].

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. 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).

2.3 Highway traffic with lane change

In this paper we concentrate on lane change on multi-lane highway which is handled by the lane control component. Its behavior does not involve any trajectories and can be described by a finit-state input-output automaton (IOA) where transitions may have guards that depend on variables.

The following lane change procedure was proposed in [3]: 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.

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.

The reserved lanes of the cars define the safety condition for the system. 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 dangerous situation of a collision is formalized by the following condition:

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

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.

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

- $\mathsf{c}(m)$: introduce a claim for lane m,
- $\mathsf{wd\text{-}c}(m)$: withdraw the claim for lane m,
- $\mathsf{r}(m)$: change a claim for lane m into a reservation for lane m,
- $\mathsf{wd\text{-}r}(m)$: withdraw the reservation for lane m.

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 [11]. Distributed control of systems consisting of several communicating components is described in [5]. It turns out that the method of submodule construction, as introduced in [6], can also be used for this purpose. In [12] this approach is formalized and described for different types of interactions between the controlled system, the environment and the controller. The approach of [12] is used in the following.

The typical system architecture for controlling a single component comprises the plant (to be controlled, called world model in [13]), 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 [13]. 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*: Safeness 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 [14].

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. However, in the examples discussed in this paper all interactions are visible, i.e., no blow-up occurs.

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 [5] and called distributed control. We note, however, that in general the problem of synthesizing distributed control is undecidable [15].

3.2 A simple algorithm for lane change

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 $\mathsf{r}(m)$ 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 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.

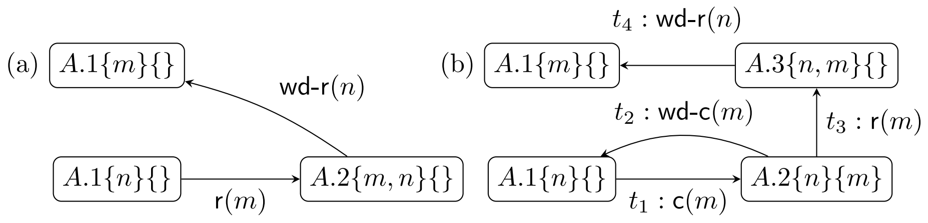

Fig. 3. Behavior of a car A changing from its current lane n to a neighboring target lane $m \in \{n-1, n+1\}$: (a) simple algorithm, (b) protocol with a claim transition $\mathsf{c}(m)$ as in [3].

Let us consider a given car *ego* with its 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 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. Consider now two arbitrarily chosen cars A and F. We are interested in understanding what happens if two cars want to reserve the same lane at the same time, as in Fig. 1. 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 $A.r(n+1)$ 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 $r(m)$ in Fig. 3(a) should be pruned if the following condition cc, called *collision check* in [3], is false:

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

This means that the predicate $A.cc(m)$ is an enabling condition for the transition $r(m)$ of car A. The same condition for car F restricts the transition $F.r(n{+}1)$ in Fig. 4 in such a way that the system remains safe.

It is important to note that the global system model uses the interleaving semantics [16], that is, there are never two transitions that occur at the same time. If, on the contrary, transitions may occur concurrently, it would be possible that the cars A and F in Fig. 1 would simultaneously perform a transition $\mathsf{r}(1)$, i.e., the dotted transition in the figure, resulting in a collision on lane 1. Interleaving semantics is widely used for modeling concurrent state machines. We note that interleaving semantics was also assumed in the safety proof of [3].

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 Interleaving semantics or synchronous models?

It can be argued that interleaving semantics is not a realistic assumption for distributed systems where transitions are controlled independently by different components. Suppose that cars A and F 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 car A does this just before car F, the question arises whether it is realistic to assume that car F will notice this change of reservation made by car A before it performs its own reservation?

A better modeling paradigm appears to be 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).

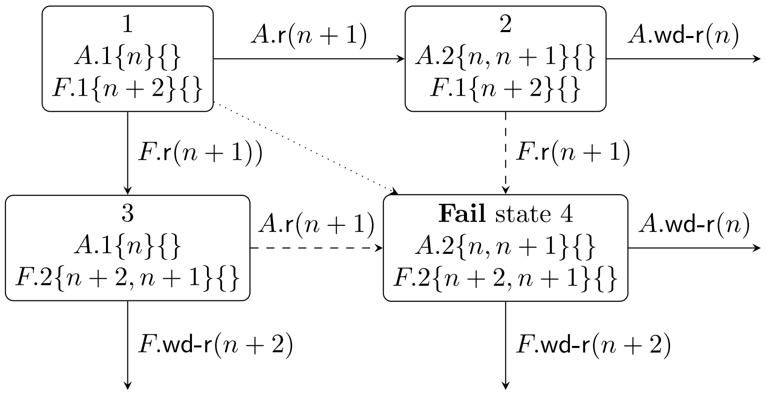

Fig. 4. Reachability analysis for two cars, A and F, behaving as in Fig. 3 (a). Solid and dashed arrows represent transitions in interleaving semantics. The dotted arrow represents an additional transition in the synchronous model.

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 parallel 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 [17] 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 [3] is based on this principle and represented in Fig. 3 (b). In case of a conflict between the two cars, both cars abandon their reservation and withdraw 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 [18].

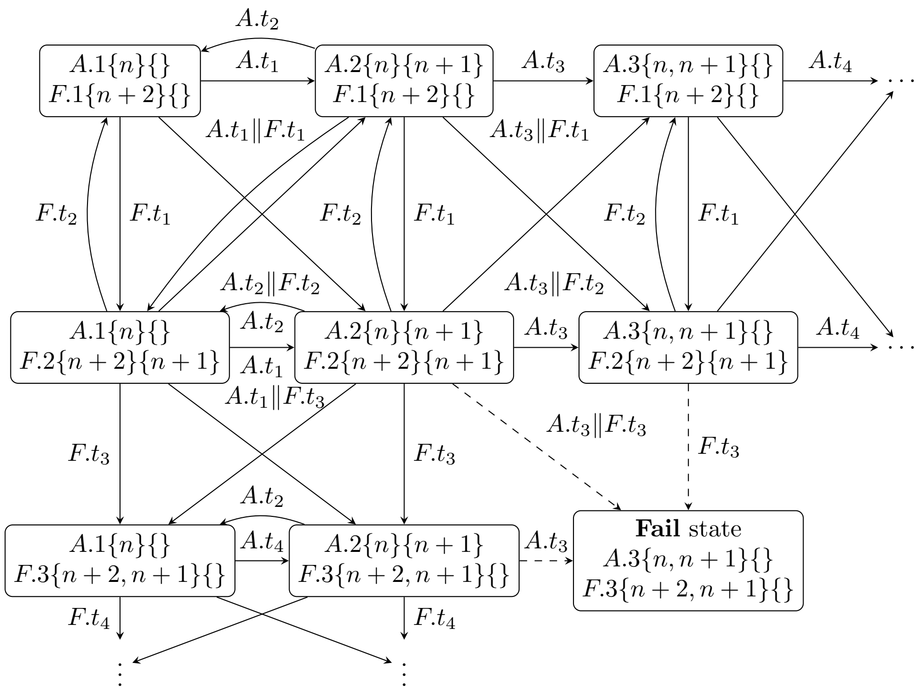
$A.1\{n\}\{\}$ $F.1\{n+2\}\{\}$
$A.2\{n\}\{n+1\}$ $F.1\{n+2\}\{\}$
$A.3\{n,n+1\}\{\}$ $F.1\{n+2\}\{\}$
$A.1\{n\}\{\}$ $F.2\{n+2\}\{n+1\}$
$A.2\{n\}\{n+1\}$ $F.2\{n+2\}\{n+1\}$
$A.3\{n,n+1\}\{\}$ $F.2\{n+2\}\{n+1\}$
$A.1\{n\}\{\}$ $F.3\{n+2,n+1\}\{\}$
$A.2\{n\}\{n+1\}$ $F.3\{n+2,n+1\}\{\}$
Fail state $A.3\{n,n+1\}\{\}$ $F.3\{n+2,n+1\}\{\}$
$A.t_1$
$A.t_2$
$A.t_3$
$A.t_4$
$F.t_1$
$F.t_2$
$F.t_3$
$F.t_4$
$A.t_1\|F.t_1$
$A.t_3\|F.t_1$
$A.t_2\|F.t_2$
$A.t_3\|F.t_2$
$A.t_1\|F.t_3$
$A.t_3\|F.t_3$

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 $n+1$.

The proposed protocol proceeds as follows. First, car A claims a space on the target lane m adjacent to its current lane n by the action $\mathsf{c}(m)$. Subsequently, it checks whether this claim intersects with the reservation or claim of any other car. In that case, A withdraws its claim by the action $\mathsf{wd\text{-}c}(m)$. Otherwise, without any intersection, 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 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.

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*:

$$ego.pcc(m) = \neg \exists\, c : ((m \in c.res \lor m \in c.clm) \land safetyOverlap(ego, c)).$$

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 algorithm proposed in [3], which was verified for interleaving semantics. In fact, they are identical if the states q_1 and q_2 of Fig. 2 in [3] are combined by ignoring the time constraint for state q_2. However, this constraint does not concern safety, but was only included to obtain a upper time-bound for a lane-change maneuver. Therefore this paper shows that the algorithm of [3] is not only correct for interleaving semantics, but also in a synchronous model.

3.5 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 [3] 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

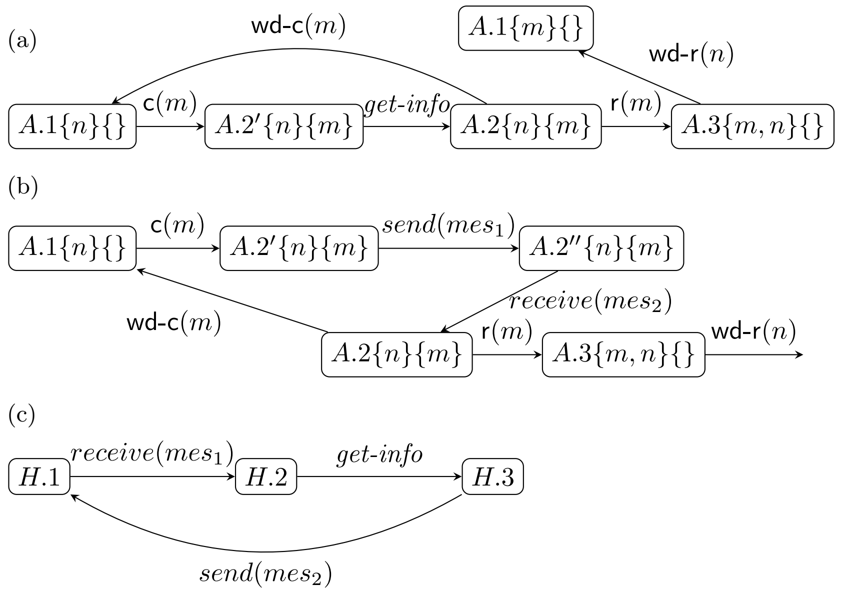

Fig. 6. (a) Global transition diagram involving a helper car that reads its own safety distance. Derived behavior for (b) lane changing car and (c) helper car.

information is available locally in each car. For this purpose, we can use the derivation algorithm described in [19] or use the approach described in [20]. 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 [19] 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 (enforcing sequencing only locally inside components [21]), 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 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. 6 (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 6 (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. 6 (c) should be performed in each car by the lane control component concurrently with its normal behavior described by Fig. 6 (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 than the algorithm proposed in [3]. The reason is that in [3], 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.

4 Conclusion

This paper revisits the traffic maneuvers on multi-lane highways as discussed in [3]. 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. [5]) can be used for synthesizing controllers for the discrete transitions of hybrid systems. Corresponding algorithms exist for interleaving semantics, synchronous systems, and IOA [12]. (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 [19]. (For a detailed discussion, see Section 3.5).

We note that an additional difficulty may occur during the lane change maneuver if a fast driving car approaches the claimed space just when the claim is being made. It could happen that the *safetyOverlap* condition with this car becomes false at the same time as the claim is set. It is shown in an extended version of this paper [7] that a collision can be avoided by requiring that the front distance, which is used for the calculation of the safety distance by the approaching car, should take into account the distance to the lane changing car as soon as the claim is made.

References

1. Varaija, P.: Smart cars on smart roads: problems of control. IEEE Trans. on Automatic Control **AC-38** (1993) 195–207
2. Lygeros, J., Godbole, D.N., Sastry, S.S.: Verified hybrid controllers for automated vehicles. IEEE Trans. on Automatic Control **43** (1998) 522–539
3. Hilscher, M., Linker, S., Olderog, E.R., Ravn, A.P.: An abstract model for proving safety of multi-lane traffic manoeuvres. In: Proc. ICFEM, Springer (2011) 404–419
4. Lynch, N.A., Segala, R., Vaandrager, F.W., Weinberg, H.: Hybrid i/o automata. Technical Report Report CSI-R9907, April 1999, Computing Science Institute, University of Nijmegen (1999)
5. Cai, K., Wonham, W.: Supervisor localization: A top-down approach to distributed control of discrete-event systems. IEEE Trans. Autom. Control **55** (2010) 605–618
6. Merlin, P., v. Bochmann, G.: On the construction of submodule specifications and communication protocols. ACM Trans. Program. Lang. Syst. **5** (1983) 1–25
7. v. Bochmann, G., Hilscher, M., Linker, S., Olderog, E.R.: Synthesizing and verifying controllers for multi-lane traffic maneuvers. Technical Report 109, AVACS (2015) see `www.avacs.org` under ‘Papers’.
8. Lynch, N.A., Segala, R., Vaandrager, F.W.: Hybrid i/o automata. Inf. Comput. **185** (2003) 105–157
9. Damm, W., Hungar, H., Olderog, E.R.: Verification of cooperating traffic agents. Intern. Journal of Control **79** (2006) 395–421
10. Damm, W., Möhlmann, E., Rakow, A.: Component based design of hybrid systems: A case study on concurrency and coupling. In: Proc. 17th Intern. Conf. on Hybrid Systems: Computation and Control. HSCC ’14, ACM (2014) 145–150
11. Ramadge, P., Wonham, W.: Supervisory control of a class of discrete event processes. SIAM J. Control Optim. **25** (1987) 206–230
12. v. Bochmann, G.: Using logic to solve the submodule construction problem. Discrete Event Dynamic Systems **23** (2013) 27–59
13. Damm, W., Finkbeiner, B.: Does it pay to extend the perimeter of a world model? In: Proc. 17th Intern. Conf. on Formal Methods. FM’11, Springer (2011) 12–26
14. Ziller, R., Schneider, K.: Combining supervisor synthesis and model checking. ACM Trans. Embed. Comput. Syst. **4** (2005) 331–362
15. Thistle, J.G.: Undecidability in decentralized supervision. Systems & Control Letters **54** (2005) 503–509
16. Milner, R.: Communication and Concurrency. Prentice-Hall (1989)
17. Dijkstra, E.W.: Cooperating sequential processes. In Genuys, F., ed.: Programming Languages: NATO Advanced Study Institute, Academic Press (1968) 43–112
18. Abramson, N.: The ALOHA system: Another alternative for computer communications. In: Proc. Fall Joint Computer Conf. AFIPS ’70, ACM (1970) 281–285
19. Gotzhein, R., v. Bochmann, G.: Deriving protocol specifications from service specifications including parameters. ACM Trans. Comput. Syst. **8** (1990) 255–283
20. Castejón, H.N., v. Bochmann, G., Bræk, R.: On the realizability of collaborative services. Software and System Modeling **12** (2013) 597–617
21. Mauw, S., Reniers, M.A.: High-level message sequence charts. In: SDL 1997: Time for Testing – SDL, MSC and Trends, Elsevier Science B.V. (1997) 291–306