Introducing Liveness into Multi-lane Spatial Logic lane change controllers using UPPAAL*

Maike Schwammberger
Department of Computing Science, University of Oldenburg
Oldenburg, Germany

schwammberger@informatik.uni-oldenburg.de

With Multi-lane Spatial Logic (MLSL) a powerful approach to formally reason about and prove safety of autonomous traffic manoeuvres was introduced. Extended timed automata controllers using MLSL were constructed to commit safe lane change manoeuvres on highways. However, the approach has only few implementation and verification results. We thus strenghen the MLSL approach by implementing their lane change controller in UPPAAL and confirming the safety of the lane change protocol. We also detect the unlive behaviour of the original controller and thus extend it to finally verify liveness of the new lane change controller.

Keywords. Autonomous cars, Multi-lane Spatial Logic, Automotive-Controlling Timed Automata, UPPAAL, Safety, Liveness.

1 Introduction

Nowadays, driving assistance systems and fully autonomously driving cars are increasingly capturing the market. For such autonomous systems, traffic safety and prevention of human casualties is of the utmost importance. In this context, safety means collision freedom and thus reasoning about car dynamics and spatial properties. A softer, but also highly desirable, requirement is liveness, meaning that a good state is finally reachable.

An approach to separate the car dynamics from the spatial considerations and thereby to simplify reasoning, was introduced in [13] with the Multi-lane Spatial Logic (MLSL) for expressing spatial properties on multi-lane motorways with one driving direction for all cars. The idea to separate dynamics from control laws follows the work by Raisch et al. [20] and Van Schuppen et al. [11].

The logic MLSL and its dedicated abstract model was extended for country roads with oncoming traffic [12] and urban traffic scenarios with intersecting lanes [14, 27]. The authors informally introduced respective controllers for safe lane change manoeuvres and safe turning manoeuvres at intersections. The respective safety of the controllers is proven with a semi-formal mathematical proof [13, 12, 14]. With *automotive-controlling timed automata (ACTA)*, a formal semantics for the previously informal controllers was later introduced [14]. (Un-) decidability results for (parts of) the logic MLSL were provided [10, 16, 23].

MLSL itself is a thoroughly researched and strong formal approach for proving properties of autonomous traffic manoeuvres. Recently, the first computer-based assistance for reasoning with a new hybrid extension of MLSL (HMLSL) was introduced by Linker [17]. The authors

*This research was partially supported by the German Research Foundation (DFG) in the Research Training Group GRK 1765 SCARE.

M. Gleirscher, S. Kugele, S. Linker (Eds.): 2nd International Workshop on Safe Control of Autonomous Vehicles (SCAV 2018)
EPTCS 269, 2018, pp. 17–31, doi:10.4204/EPTCS.269.3

successfully investigate safety constraints for the motorway traffic scenarios from [13] with Isabelle/HOL [22]. They outline an interesting extension of their work to liveness properties.

In this paper, we also focus on the motorway case. While [17] presents a strong implementation result focused directly on the spatio-temporal logic HMLSL, we instead investigate safety and liveness of the protocol of the lane change controller for highway traffic [13]. The controller can be formalised as an automotive-controlling timed automaton (ACTA) [14] and uses formulas of MLSL to reason about traffic situations and to decide, whether a car can safely change lanes.

As ACTA are extended timed automata [1], we implement the lane change controller in the tool UPPAAL [2], which allows for model-checking of timed automata. With this, we verify the correct behaviour of the considered lane change protocol and confirm the hitherto informally proven *safety* property in a preferably generic UPPAAL model. Thus, our goal is to show *unreachability* of a bad state with a collision in the overall system. With UPPAAL, we also detect the absence of *liveness* in the original lane change controller from [13]. We thus adapt the old lane change controller and show the liveness of the new controller with UPPAAL.

In Sect. 2, we briefly introduce the abstract model and logic MLSL from [13]. We also introduce the lane change controller and ACTA formalism. In Sect. 3, we explain the adaptions of the lane change controller for the implementation in UPPAAL and introduce our UPPAAL verification properties. We extend the original controller from [13] to a new live lane change controller in Sect. 3.4. Finally, we summarise our results in Sect. 4 and give ideas for future work.

2 Preliminaries

In this section, we briefly introduce the approach from [13]. For this, we start with an overview over the abstract model for highway traffic in Sect. 2.1 and introduce the Multi-lane Spatial Logic in Sect. 2.2. In Sect. 2.3, we introduce the automotive-controlling timed automata (ACTA) from [14], which serve to formalise the lane-change controller from [13], that we describe in Sect. 2.4.

2.1 Abstract model and local view

The abstract model for highway traffic consists of neighbouring infinite *lanes* $0,1,\ldots$ of continuous space, leading in the same direction from the set of all lanes $\mathbb{L}$. Every car has a unique *car identifier* $A,B,\ldots$ from the set $\mathbb{I}$ of all car identifiers and a real value for its position *pos* on a lane. An example for a traffic situation in our abstract model is depicted in Fig. 1. We use the concept of an *ego car* as the *car under consideration* and use the special variable ego to refer to this car. For Fig. 1, we assume E is our ego car and thus have the valuation $\nu(\text{ego}) = E$.

In the abstract model, the space a car E is currently occupying on a lane is represented by its *reservation* $res(\text{ego})$, while a *claim* $clm(\text{ego})$ is akin to setting the direction indicator (cf. dotted part of cars A and B in Fig. 1, showing the desire of A and B to change to lane 1). Thus, a claim represents the space a car plans to drive on in the future. For now we assume, that the size of a car includes its' physical size and its braking distance. With this, safety is already violated, if a car invades the braking distance of another car. The idea is that every car is supposed to be able to do an emergency brake at every moment, without causing a collision.

Static information about cars like their positions and their reserved or claimed lanes is captured in a *traffic snapshot* $TS = (res, clm, pos, spd, acc)$ from the set $\mathbb{TS}$ of all traffic snapshots. E.g. $res(A) = \{2\}$, $clm(A) = \{1\}$ and $pos(A) = 10$ for car A in Fig. 1. As lanes are of infinite

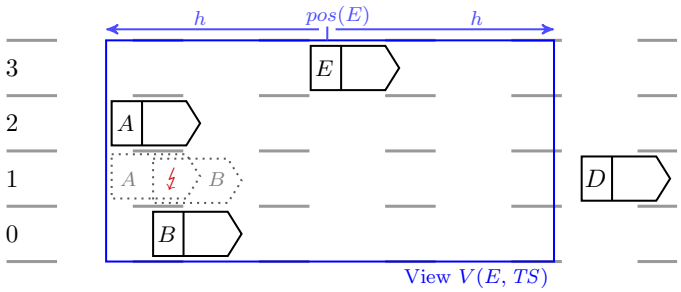

Figure 1: Abstract model with adjacent lanes 0 to 3 and cars A, B, E and D. Cars A and B both plan to change to lane 1, indicated with their resp. dotted claims on lane 1. Car D is too far away from car E to be considered in E's standard view $V(E,TS)$.

size, we also have an infinitely large traffic snapshot with infinitely many cars in it. However, for checking safety and liveness properties of our lane-change controller, only cars within some bounded *view* V around our ego car E are of interest.

Definition 1 (View)**.** *For an arbitrary traffic snapshot* TS*, the* view V*, owned by car* $E \in \mathbb{I}$*, is defined by* $V = (L,X,E)$*, where* $L \subseteq \mathbb{L}$ *is an interval of lanes visible in* V *and* $X = [r,t] \subseteq \mathbb{R}$ *is an interval of space along the lanes.*

We define the standard view *of car* E *by* $V(E,TS) = (\mathbb{L}, [pos(E)-h, pos(E)+h], E)$*, where* h *is a sufficiently large* horizon *for looking forwards resp. backwards from the position* $pos(E)$*, as given in the traffic snapshot* TS*.*

Note that we assume there exists a minimal positive value for the size of all cars, thus only finitely many cars are considered in a view. We furthermore assume that there exists a maximum velocity for all cars and the horizon h is big enough to consider the fastest car that could endanger E contained in its the standard view $V(E,TS)$. In the example in Fig. 1, car D is not considered in V, as it is to far away from E.

We use a car dependent sensor function $\Omega_E : \mathbb{I} \times \mathbb{TS} \to \mathbb{R}_+$ which, given a car identifier $C \in \mathbb{I}$ and a traffic snapshot $TS \in \mathbb{TS}$, provides the size $\Omega_E(C,TS)$ of C as perceived by E's sensors.

For a view $V = (L,X,E)$ and a traffic snapshot $TS = (res, clm, pos, spd, acc)$, we introduce the following abbreviations, used for the semantics definition of our logic MLSL in the next Sect. 2.2:

$$res_V : \mathbb{I} \to \mathbb{P}(L) \text{ with } res_V(C) = res(C) \cap L \tag{1}$$

$$clm_V : \mathbb{I} \to \mathbb{P}(L) \text{ with } clm_V(C) = clm(C) \cap L \tag{2}$$

$$len_V : \mathbb{I} \to \mathbb{P}(L) \text{ with } len_V(C) = [pos(C), pos(C) + \Omega_E(C,TS)] \cap X \tag{3}$$

The functions (1) and (2) restrict their counterparts $res(C)$ and $clm(C)$ from TS to the set of lanes considered in V. Function (3) defines the part of car C that E perceives with its sensors in the extension X of the considered view V.

2.2 Multi-lane Spatial Logic

With Multi-lane Spatial Logic (MLSL), we can reason about traffic situations in our local view V. As variables, we allow for *car variables* $c,d,\ldots$ from the set $CVar$, valuated with car identifiers from the set $\mathbb{I}$ and *lane variables* $n,l,\ldots$ from the set $LVar$, valuated with lanes from $\mathbb{L}$. We define $\text{ego} \in CVar$.

Definition 2 (Valuation of variables)**.** *A* valuation ν *is a function* $\nu\colon \mathit{Var} \to \mathbb{I} \cup \mathbb{L}$*, where* $\mathit{Var} = \mathit{CVar} \cup \mathit{LVar}$ *and* $\nu\colon \mathit{CVar} \to \mathbb{I}$ *and* $\nu\colon \mathit{LVar} \to \mathbb{L}$*.*

Formulae of MLSL are built from atoms, Boolean connectors and first-order quantifiers. As *spatial* atoms, we use *free* to represent free space on a lane and $re(c)$ (resp. $cl(c)$) to formalise the reservation (resp. claim) of a car. We also allow for the comparison of variables $u = v$ for variables $u, v, \in \mathit{Var}$ of the same type.

We use a horizontal *chop operator* similar to chop operations for timing intervals in Duration Calculus [6] or interval temporal logic [21], denoted by $\frown$. Also, we introduce a vertical chop operator given by the vertical arrangement of formulas. Intuitively, a formula $\varphi_1 \frown \varphi_2$ holds if we can split the view V vertically into two views V_1 and V_2 such that on V_1 the formula φ_1 holds and V_2 satisfies φ_2. Similarly a formula $\frac{\varphi_2}{\varphi_1}$ is satisfied by V, if the view can be chopped at a lane into two subviews, V_1 and V_2, where V_i satisfies φ_i for $i = 1, 2$.

Definition 3 (Syntax)**.** *The syntax of a* Multi-lane Spatial Logic *formula* ϕ_M *is defined by*

$$\varphi_M ::= \mathit{true} \mid u = v \mid \mathit{free} \mid re(c) \mid cl(c) \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \exists c\colon \varphi_1 \mid \varphi_1 \frown \varphi_2 \mid \tfrac{\varphi_2}{\varphi_1},$$

where $c \in \mathit{CVar}$ *and* $u, v \in \mathit{Var}$*. We denote the set of all MLSL formulas by* $\Phi_{\mathbb{M}}$*.*

The semantics of MLSL formulas is defined over a traffic snapshot TS, a view V and a valuation of variables ν. We denote the length of a real interval $X \subseteq \mathbb{R}$ by $|X|$.

Definition 4 (Semantics of MLSL)**.** *The* satisfaction *of MLSL formulas* φ *with respect to a traffic snapshot* TS*, a view* $V = (L, X, E)$ *with* $L = [l, n]$ *and* $X = [r, t]$*, and a valuation* ν *of variables is defined inductively as follows:*

$$
\begin{array}{ll}
TS, V, \nu \models \mathit{true} & \text{for all } TS, V, \nu \\
TS, V, \nu \models u = v & \Leftrightarrow \nu(u) = \nu(v) \\
TS, V, \nu \models \mathit{free} & \Leftrightarrow |L| = 1 \text{ and } |X| > 0 \text{ and } \forall i \in I_V\colon \mathit{len}_V(i) \cap (r,t) = \emptyset \\
TS, V, \nu \models re(c) & \Leftrightarrow |L| = 1 \text{ and } |X| > 0 \text{ and } \nu(c) \in I_V \text{ and } \mathit{res}_V(\nu(c)) = L \text{ and } X = \mathit{len}_V(\nu(c)) \\
TS, V, \nu \models cl(c) & \Leftrightarrow |L| = 1 \text{ and } |X| > 0 \text{ and } \nu(c) \in I_V \text{ and } \mathit{clm}_V(\nu(c)) = L \text{ and } X = \mathit{len}_V(\nu(c)) \\
TS, V, \nu \models \neg\varphi & \Leftrightarrow \text{not } TS, V, \nu \models \varphi \\
TS, V, \nu \models \varphi_1 \wedge \varphi_2 & \Leftrightarrow TS, V, \nu \models \varphi_1 \text{ and } TS, V, \nu \models \varphi_2 \\
TS, V, \nu \models \exists\colon \varphi_1 & \Leftrightarrow TS, V, \nu \models \exists \alpha \in I_V\colon TS, V, \nu \oplus \{c \mapsto \alpha\} \models \varphi_1 \\
TS, V, \nu \models \varphi_1 \frown \varphi_2 & \Leftrightarrow \exists s \in \mathbb{R}\colon r \le s \le t \text{ and } TS, V_{[r,s]}, \nu \models \varphi_1 \text{ and } TS, V_{[s,t]}, \nu \models \varphi_2 \\
TS, V, \nu \models \tfrac{\varphi_2}{\varphi_1} & \Leftrightarrow \exists m \in \mathbb{N}\colon l - 1 \le m \le n + 1 \text{ and } TS, V^{[l,m]}, \nu \models \varphi_1 \text{ and } TS, V^{[m+1,n]}, \nu \models \varphi_2
\end{array}
$$

Abbreviation. In the following we use the abbreviation $\langle\varphi\rangle$ to state that a formula φ holds *somewhere* in the considered view. For example, in Fig. 1 with valuation $\nu(\text{ego}) = E$, the formula $\langle\varphi\rangle \equiv \langle re(\text{ego})\rangle$ holds in $V(E, \mathit{Road})$, because there *somewhere* exists a reserved space for car E.

Example 1 (MLSL formulas). Consider Fig. 1 and assume a valuation of variables $\nu(\text{ego}) = E$, $\nu(a) = A$, $\nu(b) = B$ and $\nu(d) = D$. Consider the following MLSL formulas:

$$
\begin{aligned}
\varphi_1 &\equiv \langle re(\text{ego}) \frown \mathit{free}\rangle \\
\varphi_2 &\equiv \langle cl(a) \wedge cl(b) \frown \neg cl(a) \wedge cl(b)\rangle \\
\varphi_3 &\equiv \langle cl(b) \frown \mathit{free} \frown re(d)\rangle
\end{aligned}
$$

In view $V(E,TS)$ the formula φ_1 holds, as there is free space in front of car E. Equally φ_2 holds, as there is a claim of both cars A and B at the same spot on lane 1 and after this there is a space with only the claim of car B. Thus $TS,V(E,TS),\nu \models \varphi_1$ and $TS,V(E,TS),\nu \models \varphi_2$. However, $TS,V(E,TS),\nu \not\models \varphi_3$, as car D is not part of view $V(E,TS)$. △

2.3 Automotive-controlling timed automata

Before we introduce the actual lane change controller protocol from [13] in Sect. 2.4, we briefly define the extended timed automata type, introduced in [14] to formalise the controller. As variables these automotive-controlling timed automata (ACTA) use both clock and data variables. For clock variables $x,y \in \mathbb{X}$ and clock updates we refer to the definition of timed automata [1] and for data variables $u,v \in Var$ and data updates we refer to the extension of timed automata proposed for UPPAAL [9]. These clock and data updates ν_{act} are allowed on transitions of ACTA.

Further on, the controllers use MLSL formulas φ_M as well as clock and data constraints $\varphi_{\mathbb{X}}$ resp. φ_{Var} as guards φ on transitions and invariants $I(q)$ in states q. An example for a data constraint for a variable $l \in Var$ is $l > 1$. A guard or invariant φ from the set Φ of all guards and invariants is defined by $\varphi \equiv \varphi_M \mid \varphi_{\mathbb{X}} \mid \varphi_{Var} \mid \varphi_1 \wedge \varphi_2 \mid true$.

We express possible driving manoeuvres by *controller actions*, which may occur at the transitions of an ACTA. Controller actions e.g. enable a car to set or withdraw (`wd`) a claim (`c`) or a reservation (`r`) for a lane.

Definition 5 (Controller Actions)**.** *With $c \in CVar$, a* controller action c_{act} *is defined by*

$$c_{act} ::= \mathtt{c}(c,\psi_{\mathbb{D}}) \mid \mathtt{wd\ c}(c) \mid \mathtt{r}(c) \mid \mathtt{wd\ r}(c,\psi_{\mathbb{D}}) \mid \tau,$$

where $\psi_{\mathbb{D}} ::= k \mid l_1 \mid l_1 + l_2 \mid l_1 - l_2$ with $k \in \mathbb{N}$, $l_1,l_2 \in LVar$. The set of all controller actions is defined by $Ctrl_{Act}$.

2.4 Lane change controller

In this section, we introduce the lane change controller from [13], whose implementation into UPPAAL we introduce in Sect. 3. The overall goal of this controller is to safely change lanes in freeway traffic. Here, *safety* of ego car means collision freedom and thus disjunction of the reserved spaces of ego and other cars, expressed by the MLSL formula

$$Safe(ego) \equiv \neg\exists c\colon c \neq ego \wedge \langle re(ego) \wedge re(c)\rangle\,. \tag{4}$$

The main idea for the lane change controller is to first *claim* the space on a lane it wants to enter and *reserve* it only if no collision is detected. We assume a lane change to take at most t_{lc} time to finish. The lane change controller is constructed for the ego car ($\nu(\text{ego}) = E$ in the example from Fig. 1) but scales to all cars as ego can be substituted by an arbitrary car variable $c \in CVar$.

We explain the construction of the controller starting with the initial state. As we want to prevent different reservations from overlapping, we introduce a *collision check* for the ego car expressed by the MLSL formula

$$cc \equiv \neg\exists c\colon c \neq \text{ego} \wedge \langle re(\text{ego}) \wedge re(c)\rangle. \tag{5}$$

Formula (5) is evaluated to true, iff nowhere exists a car different from the ego car whose reservation overlaps with the actors reservation. We assume cc to hold in the initial state of our controller. Next the lane change controller can claim some space on either the lane to its left or right, provided such a lane exists. Here N is the lane identifier of the highest lane from the set of all lanes $\mathbb{L}$.

In order to transform a claim into a reservation and thus finally change lanes, a car first needs to check if there are overlaps of other cars' claims or reservations with its own claim. This is formalised by the *potential collision check*

$$pc(c) \equiv c \neq \text{ego} \wedge \langle cl(\text{ego}) \wedge (re(c) \vee cl(c)) \rangle. \tag{6}$$

Formula (6) evaluates to true, iff there exists a car different from the ego car whose claim or reservation overlaps with ego car's own claim. A (temporary) potential collision is allowed, because it does not endanger the safety property (4). However, if a potential collision is detected, the car must withdraw its claim immediately.

When $\exists c\colon pc(c)$ does not hold, the actor reserves the claimed lane and starts changing lanes. To prevent deadlocks, we set a time bound t in state q_2 for the time that may pass between claiming and reserving crossing segments. After t_{lc} time, the lane change is finished and the reservation of actor E is reduced to the new lane.

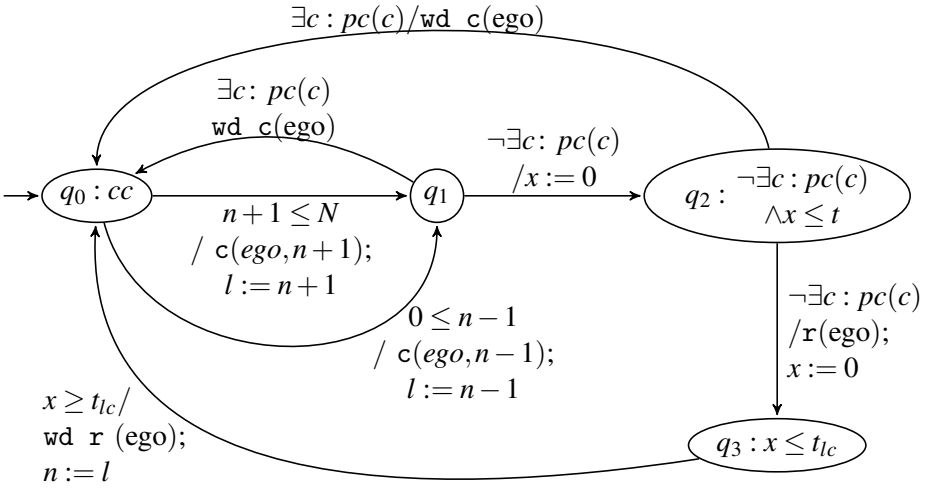

Figure 2: Lane change controller from [13].

3 UPPAAL Implementation and Verification

We first introduce the specific abstract model we examine with UPPAAL and the considered assumptions and restrictions for it in Sect. 3.1. We explain the adaptions of the lane change controller from Sect. 2.4 to the type of extended timed automata UPPAAL accepts in Sect. 3.2. We explain our verification method and show safety of the existing controller in Sect. 3.3. We detect liveness issues for the lane change controller from Sect. 2.4 and adapt it to a live controller in Sect. 3.4. We provide a summary of the goals and limitations of the current implementation and give an overview over scenario and UPPAAL model extensions in Sect. 3.5.

3.1 UPPAAL-Model and Assumptions

3.1.1 Overall scenario and data structure

The model we examine with UPPAAL is the traffic situation depicted in Fig. 1, where we consider lanes 0 to 3 and the cars A, B and E contained in view $V(E,TS)$. We encode the traffic snapshot TS, more precisely the positions, claims and reservations of the cars on the lanes, by a global data structure *pos_t*. For reservations *res* this is encoded as follows:

```
pos_t res[carid_t] = {
                      { {0,0,1,0}, 10, 5},
                      { {1,0,0,0}, 12, 5},
                      { {0,0,0,1}, 40, 5}
                     };
```

Here e.g. the first line represents car A and the Boolean lane list $\{0,0,1,0\}$ states that A has a reservation only on lane 2. The second parameter 10 is the position of A on lane 2 and the last parameter 5 is the size of A. Thus the space A occupies is the interval $[10,15]$ on lane 2. The other lines are the respective values for cars B and E, such that B initially occupies interval $[12,17]$ on lane 0 and E occupies interval $[40,45]$ on lane 3. We have a similar structure `pos_t clm[carid_t]` for the claims of the cars, where initially all Boolean lists for claims are empty, as all cars are supposed to start in the initial state of the controller without any claim.

3.1.2 Distance Controller

The lane change controller is not responsible for distance keeping. However, for cars with different acceleration and speed, a controller for distance keeping is inevitable to avoid rear-end collisions. Such a distance controller is outlined, but not formalised or constructed in [13]. Another possible distance controller is introduced and formally verified, but not yet implemented in [8]. Recently, the group of Kim Larsen synthesised an adaptive cruise control distance controller with the UPPAAL extension Stratego [15]. As the authors base their work on the spatial model of MLSL, this approach is of high interest for our implementation. However, they only consider a model consisting of one single lane without any neighbouring lanes and only two specific cars *ego* and *front* (cf. Fig. 3). Their idea is, that the *ego* car keeps track of its distance to the *front* car *always*. Additionally, their goal is to minimise the distance between *ego* and *front*. For this, one UPPAAL automaton for each *ego* and *front* is used, additional to a system controller.

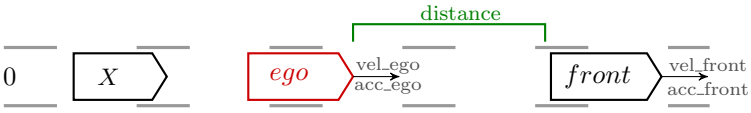

Figure 3: One-lane scenario with distance keeping from [15].

Consider on the other hand our multi-lane scenario, e.g. in Fig. 4. It is not enough to keep track of the distance to *front*, as cars A, B, C and D might change lanes and thus be in front of *ego* any time. Thus, we also need to keep track of the distances to these cars. A problem here is state space explosion, as the number of considered parallel timed automata for UPPAAL increases significantly, when using the approach from [15] directly. A second problem is the discretisation of space in their approach.

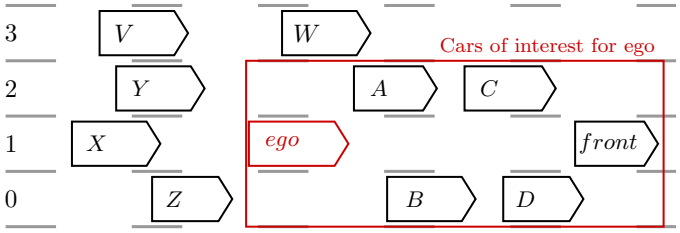

Figure 4: Cars of interest for ego car for distance keeping in multi-lane highway scenario [13].

However, for examining the safety and liveness solely of the lane change manoeuvres with the controller from [13], we do not need to consider a scenario with cars with different speed and acceleration. We restrict all cars to have the same constant speed whereby the relative distances between the cars along the lanes never change. Although this is a strong restriction, it is reasonable, as our goal is to show safety and liveness of lane change manoeuvres, where collision freedom while changing lanes is considered, not rear-end collisions.

Nonetheless, as a constant speed for all cars is a strong assumption, we plan to implement a version of the adaptive cruise controller from [15] in future work for a more realistic model.

3.1.3 Generic model

Despite the speed limitation, we encode a preferably general behaviour. In our model, the expected behaviour of car E is that it is *always* able to change lanes and that there can *never* occur a *potential collision or collision* with E, as there is no conflicting car on any neighbouring lane. In contrast, cars A and B can *not always* change lanes, as their position intervals $[10, 15]$ and $[12, 17]$ would intersect if the cars had reservations or claims on the same lane. Thus, we *expect potential collisions* between A and B, but show that the lane change controller *always* prevents *actual collisions*.

3.2 Implementation

For the UPPAAL implementation, we adapt the lane change controller from Fig. 2 to UPPAAL syntax, as neither formulas of Multi-lane Spatial Logic (cf. Def. 3, p. 20) nor controller actions for claiming or reserving lanes (cf. Def. 5, p. 21) are directly implementable in UPPAAL. The resulting UPPAAL lane change controller `LCP` is depicted in Fig. 5. Each of the cars A, B and E in our model owns one instance `LCP(i)` of the controller `LCP`, where `i` ranges over A, B and E. Note, that Fig. 5 already contains the adaptions to a live controller, we explain later in Sect. 3.4.

We start with the UPPAAL representation of MLSL formulas. The only MLSL formulas used by the lane change controller are the collision check cc (cf. formula (5), p. 21) in the initial state q_0 and the potential collision check $pc(c)$ (cf. formula (6), p. 22) used in several guards and invariants of the controller. Our solution for implementing formulas (5) and (6) in UPPAAL bases on checking the intersection of position intervals of cars with the Boolean UPPAAL function

```
bool intersect(const pos_t p1, const pos_t p2) {
  return exists(lane: laneid_t)
    p1.lane[lane] and p2.lane[lane]
```

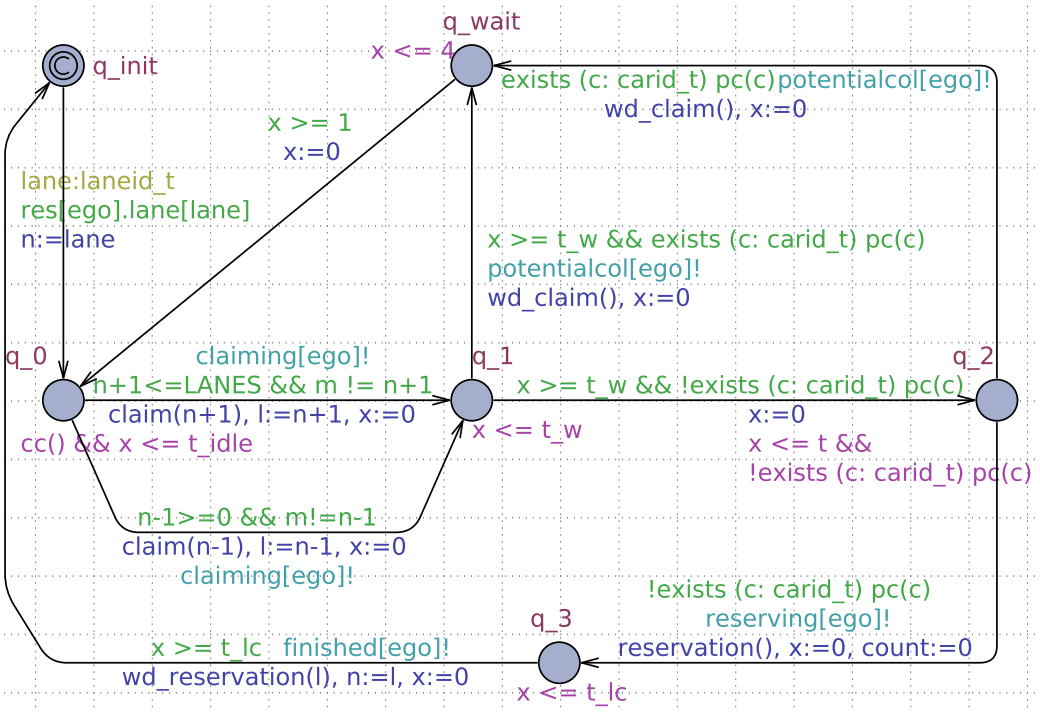

Figure 5: Lane-change controller implementation `LCP` in UPPAAL

```
        and not (p1.pos > p2.pos+p2.size or p2.pos > p1.pos+p1.size);
}
```

The function `intersect` checks for two position parameters `pos_t` (cf. Sect. 3.1) if their position intervals intersect and if both positions are on the same lane. If e.g. car A and B both claim lane 1 with $\texttt{clm[A]} = \{\{0,1,0,0\},10,5\}$ and $\texttt{clm[B]} = \{\{0,1,0,0\},12,5\}$, the function call $\texttt{intersect(clm[A],clm[B])}$ returns `true`.

With the intersect function, we encode the collision check *cc* from MLSL formula (5) by the function

```
bool cc () {
   return not exists(c:carid_t) c != ego
          and intersect(res[ego],res[c]);
}
```

and the potential collision check $pc(c)$ from MLSL formula (6) with

```
bool pc (carid_t c) {
   return c != ego
          and (intersect(clm[ego],res[c])
               or intersect(clm[ego],clm[c]));
}
```

We use the functions `cc()` and `pc(c)` in the UPPAAL controller `LCP` in Fig. 5 exactly in the same manner as we use the respective MLSL formulas in the original lane change controller from

Fig. 2. Besides MLSL formulas, we also encode controller actions for claiming and reserving lanes and their respective withdrawal actions with UPPAAL methods. For claiming a lane for the ego car, the related lane change controller calls the method

```
void claim(laneid_t lane) {
    clm[ego].lane[lane] = true;
}
```

where in the Boolean list $\{0,0,0,0\}$ for claims, the value of the forwarded lane `lane` is set to `true`. Upon a reservation request from a lane change controller, we have to check if there exists a claim for the related car and only then transform the claim into a reservation. Thus,

```
void reservation(){
    for (i:laneid_t)
    {
        if (clm[ego].lane[i]) {
            res[ego].lane[i] = true;
            clm[ego].lane[i] = false;
        }
    }
}
```

changes the value of the respective lane in the reserved lanes for the ego car to `true`, while setting the value for the transformed claim for the same lane to `false`.

3.3 Verification of Safety with UPPAAL

The requirement queries for the verifier in UPPAAL are formulated in a computation tree logic (CTL) [7, 26] style specification language. The first query we successfully check is

$$\texttt{A[] not deadlock}, \tag{7}$$

with which we globally exclude deadlocks in an arbitrary run of our system. We checked the query on a normal work station in 48 to 49 seconds with a memory usage peak of roughly 140KB.

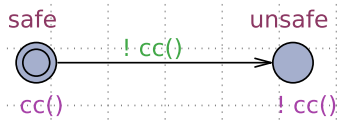

Figure 6: `Observer1` checking for a collision.

For the second query, we introduce the Observer automaton `Observer1`, depicted in Fig. 6. This Observer automaton uses a slightly adapted version of the collision check `cc()` to check for a collision between any two arbitrary cars at any moment. We use the query

$$\texttt{A[] not Observer1.unsafe} \tag{8}$$

to show in averagely less than 4 seconds with a memory usage peak of 46KB, that there exists no example trace where the formula *cc* does not hold. With this query, we verify the safety property (4) (p. 21) for the lane change controller from [13].

3.4 Adaptions for constructing a live controller

With query (7), we exclude deadlocks in our system. However, the original controller in Fig. 1 is not truly live, as e.g. *livelocks* exist, where no car ever changes lanes, even though in our model at least car E, should always be able to change lanes.

To analyse liveness, we introduce a second Observer automaton `Observer(i)`, as depicted in Fig. 7. For every instance `LCP(i)` of the lane change controller, we require an automaton `Observer(i)` which synchronises with `LCP(i)` over communication channels. E.g. on claiming a lane for car A, `LCP(A)` sends over the channel `claiming[A]` with which `Observer(A)` synchronises, such that both controllers simultaneously change to a new state. Upon reserving a lane, `LCP(A)` sends over `reserving[A]` and the Observer changes to a state `success`. We check the query

```
A<> (Observer(A).success or Observer(B).success or Observer(E).success),   (9)
```

which states, that *finally in every trace*, at least one of the controllers `LCP(i)` is successful in changing a lane. Remember, that we generally expect query (9) to be successfully verified, as in our model at least car C should be able to finally change a lane in every possible trace.

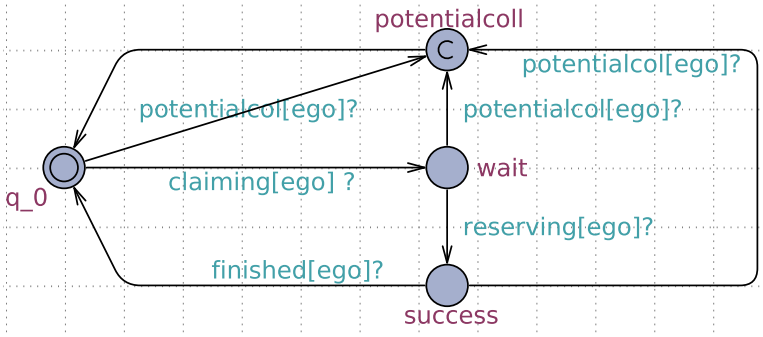

Figure 7: `Observer(i)` checks for every instance of the lane change controller `LCP(i)`, if whenever car i claims a lane, it finally changes lanes, or if a potential collision occurs.

3.4.1 Adaption 1

Without a time invariant in state q_1 of the original controller from [13] and without respective time guards on the outgoing edges, query (9) *does not hold*.

The reason is that there exists a trace, where cars A and B both infinitely often claim lane 1 without any elapse of time and thus both circle between their respective states q_0, q_1 and q_2 in a *livelock*. As no time elapses, `LCP(E)` has no possibility of executing any transition and thus starves. This problem is easily solvable by introducing the invariant $x <= t_w$ to state q_1 and placing the guard $x >= t_w$ on the outgoing edges of q_1, as done in the UPPAAL automaton depicted in Fig. 5. With these adaptions, we successfully show query (9) in less than 0.5 seconds with a memory usage peak of 40KB.

3.4.2 Adaption 2

The verification query (9) is already a *weak liveness* property, as it shows that in every simulation trace, at least one of the controllers finally changes lanes. We refine this property to

$$\texttt{A<> Observer(i).success}, \tag{10}$$

which states for an arbitrary car identifier i, that the related car finally changes lanes. When considering only the first adaption, as anticipated, this property only holds for `LCP(E)`. The reason is, that there still exists a trace, where cars A and B both unsuccessfully try to change to lane 1 infinitely often and thus creating a potential collision infinitely often, preventing both controllers from ever transforming their claim into a reservation.

To solve this, we introduce an additional state `q_wait`, in which the controller is forced to wait for a bounded non-deterministic time. For now, we delimit this waiting time in `q_wait` by its invariant $x \leq 4$ and the guard $x \geq 1$ on its outgoing edge. With this, cars A and B do not permanently block each other from changing a lane and we verify both

$$\texttt{A<> Observer(A).success} \quad \text{and} \quad \texttt{A<> Observer(B).success} \tag{11}$$

in each less than 2.7 seconds with a memory usage of each less than 76KB.

3.5 Summary and extendability of the current implementation

With the traffic situation from Fig. 1 and the corresponding implementation, as described in this section, we presented one very specific scenario, designed for the following purposes:

- Showing the absence of collisions between any cars (i.e. proving safety (4)) and
- Identifying and analysing the existing livelocks (cf. location q_1) and
- Eliminating the livelocks and showing liveness of the new controller.

For this, the restrictions for the scenario, e.g. on 3 cars and 4 lanes were reasonable. However, we also tried different scenarios, with different numbers of lanes and cars. Our liveness and safety properties were not violated for any of the considered numbers of lanes and cars. Up to 16 parallel lanes were considered without any problems. However, we observed the following run-time issues when adding cars.

While run-time seemes to increase only linear by about 50 ms each time when we add one lane, it appears to increase exponentially when adding a car. This observation is not surprising, as adding only one car `i` means adding two timed automata and one clock variable to the system: One timed automaton `LCP(i)` with its clock x and one observer automaton `Observer(i)`. Consider for example the model from Fig. 1 with one additional car. Now for property (11), UPPAAL takes 1025 seconds to verify the query instead of the previously observed 2.7 seconds for the three car scenario. While 1025 seconds for four cars is still acceptable, after including a fifth car, UPPAAL could not finish the verification of query (11) within one day.

Thus, for future considerations of our implementation where more than four cars should be considered, we would have to optimise our implementation first.

4 Conclusion

We strengthened the MLSL approach from the group of Olderog [13, 12, 14], by implementing their lane change controller for highway traffic in UPPAAL and successfully verified their safety property. We additionally optimised their controller by examining and implementing liveness properties into it.

Related Work

There exist several approaches for analysis and control of traffic using intelligent transportation systems, where e.g. in [18] traffic lights are used as a central control mechanism at intersections. The authors verify safety of their hybrid systems with the tool KeYmaera. There also exists an approach to synthesise intelligent traffic light control mechanisms with the UPPAAL extension Stratego [3]. The key idea of this approach is to minimise waiting times and energy waste.

Also various approaches for safe and autonomously driving systems were implemented during the DARPA Grand Challenge, where e.g. finite state machines were used to describe the autonomous behaviour of the cars [25, 28].

For a hazard warning extension of MLSL, a dedicated *hazard warning controller* was implemented in UPPAAL [24]. However, the hazard warning controller was focused on a timely warning message delivery via broadcast channels and did not use MLSL formulas. A combined proof of UPPAAL verification queries with a formal proof by induction was used to prove the timely warning delivery.

Future Work

In the end of Sect. 3.4, we observe that cars A and B block each other on lane 1 and suggest an adaption ensuring the liveness of the controllers. However, this adaption does not guarantee *fairness*, as one of the cars could get the right of changing lanes arbitrarily more often than the other car. To overcome this problem, we could implement a notion of fairness into `LCP(i)`, where either car A or car B lets the other car go first, when they already got the right of way often enough. Also, we could use the UPPAAL extension for stochastical model checking (UPPAAL SMC) [4], to analyse the probabilities of unfair behaviour. We could add prices to the transitions of our controller, which increase, when a car unsuccessfully claims too often.

In this paper, we only considered the lane change controller for highway traffic [13]. An implementation of their lane change controller for country-roads [12] and the crossing controller for intersections [14] would be highly interesting. Also, they published results on a relaxation of their assumption of *perfect knowledge*, where the controllers communicate, to cope for the missing information. Also, for future considerations with more cars or different controllers, an optimisation of our implementation is of high interest, as described in Sect. 3.5.

Last but not least, for now we have the assumption of a constant speed. To verify properties in a more realistic scenario, our cars should be able to dynamically change their speed. To this end, we plan to implement an adaption of the existing UPPAAL Stratego distance controller from [15], as described in Sect. 3.1. Their adaptive cruise control implementation also minimises the distance between the *ego* car and the car in front, which we could use to optimise the traffic flow in our scenario. With this, we could even extend our MLSL scenario to a platooning scenario (cf. PATH Project [19] and the European SARTRE project [5]). However, as outlined in Sect. 3.1, the adaption of the distance controller from [15] poses some non-trivial challenges.

Acknowledgements. I would like to thank Marius Mikučionis for his help with starting the UPPAAL implementation.

References

[1] Rajeev Alur & David L. Dill (1994): *A Theory of Timed Automata*. *Theoretical Computer Science* 126(2), pp. 183–235, doi:10.1016/0304-3975(94)90010-8.

[2] G. Behrmann, A. David & K. G. Larsen (2004): *A Tutorial on* Uppaal. In Marco Bernardo & Flavio Corradini, editors: *4th Intern. School on Formal Methods for the Design of Computer, Communication, and Software Systems*, Springer, doi:10.1007/978-3-540-30080-9_7.

[3] Andreas Berre Eriksen, Chao Huang, Jan Kildebogaard, Harry Lahrmann, Kim G. Larsen, Marco Muniz & Jakob Haahr Taankvist (2017): *Uppaal Stratego for Intelligent Traffic Lights*. In: *12th ITS European Congress*.

[4] Peter Bulychev, Alexandre David, Kim Guldstrand Larsen, Axel Legay, Marius Mikučionis & Danny Bøgsted Poulsen (2012): *Checking and Distributing Statistical Model Checking*, pp. 449–463. Springer Berlin Heidelberg, Berlin, Heidelberg, doi:10.1007/978-3-642-28891-3_39.

[5] E. Chan, A. Ekfjorden, P. Jootel, J. Gidney, A. DÃąvila, M. BrÃd'nnstrÃűm, D. Skarin & L. WahlstrÃűm (2012): *SAfe Road TRains for the Environment (SARTRE): Project final report*. Technical Report. Available at `www.sartre-project.eu/en/publications/Documents/SARTRE_Final-Report.pdf`.

[6] Zhou Chaochen, C. A. R. Hoare & Anders P. Ravn (1991): *A calculus of durations*. *Information Processing Letters* 40(5), pp. 269–276, doi:10.1016/0020-0190(91)90122-X.

[7] Edmund M. Clarke & E. Allen Emerson (1982): *Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic*. In: *Logic of Programs, Workshop*, Springer-Verlag, London, UK, UK, pp. 52–71, doi:10.1007/BFb0025774. Available at `http://dl.acm.org/citation.cfm?id=648063.747438`.

[8] W. Damm, H. Hungar & E.-R. Olderog (2006): *Verification of Cooperating Traffic Agents*. *International Journal of Control* 79(5), pp. 395–421, doi:10.1080/00207170600587531.

[9] Alexandre David, Peter Gjøl Jensen, Kim Guldstrand Larsen, Marius Mikučionis & Jakob Haahr Taankvist (2015): *Uppaal Stratego*. In Christel Baier & Cesare Tinelli, editors: *Tools and Algorithms for the Construction and Analysis of Systems*, *Lecture Notes in Computer Science* 9035, Springer Berlin Heidelberg, pp. 206–211, doi:10.1007/978-3-662-46681-0_16.

[10] Martin Fränzle, Michael R. Hansen & Heinrich Ody (2015): *No Need Knowing Numerous Neighbours*. In Roland Meyer, André Platzer & Heike Wehrheim, editors: *Correct System Design*, *LNCS* 9360, Springer, pp. 152–171, doi:10.1007/978-3-319-23506-6_11.

[11] Luc C. G. J. M. Habets, Pieter J. Collins & Jan H. van Schuppen (2006): *Reachability and control synthesis for piecewise-affine hybrid systems on simplices*. *IEEE Trans. Automat. Contr.* 51(6), pp. 938–948, doi:10.1109/TAC.2006.876952.

[12] Martin Hilscher, Sven Linker & Ernst-Rüdiger Olderog (2013): *Proving Safety of Traffic Manoeuvres on Country Roads*. In Zhiming Liu, Jim Woodcock & Huibiao Zhu, editors: *Theories of Programming and Formal Methods*, *LNCS* 8051, Springer, doi:10.1007/978-3-642-39698-4_12.

[13] Martin Hilscher, Sven Linker, Ernst-Rüdiger Olderog & Anders P. Ravn (2011): *An Abstract Model for Proving Safety of Multi-lane Traffic Manoeuvres*, pp. 404–419. Springer, doi:10.1007/978-3-642-24559-6_28.

[14] Martin Hilscher & Maike Schwammberger (2016): *An Abstract Model for Proving Safety of Autonomous Urban Traffic*. In Augusto Sampaio & Farn Wang, editors: *Theoretical Aspects of Computing (ICTAC)*, *LNCS* 9965, Springer, pp. 274–292, doi:10.1007/978-3-319-46750-4_16.

[15] Kim Guldstrand Larsen, Marius Mikučionis & Jakob Haahr Taankvist (2015): *Safe and Optimal Adaptive Cruise Control*, pp. 260–277. Springer International Publishing, Cham, doi:10.1007/978-3-319-23506-6_17.

[16] Sven Linker (2015): *Proofs for Traffic Safety – Combining Diagrams and Logic*. Ph.D. thesis, University of Oldenburg.

[17] Sven Linker (2017): *Spatial Reasoning About Motorway Traffic Safety with Isabelle/HOL*. In Nadia Polikarpova & Steve Schneider, editors: *Integrated Formal Methods*, Springer International Publishing, Cham, pp. 34–49, doi:10.1007/978-3-319-66845-1_3.

[18] Sarah M. Loos & André Platzer (2011): *Safe Intersections: At the Crossing of Hybrid Systems and Verification*. In Kyongsu Yi, editor: *Intelligent Transportation Systems (ITSC)*, pp. 1181–1186, doi:10.1109/ITSC.2011.6083138.

[19] J. Lygeros, D.N. Godbole & S.S. Sastry (1998): *Verified hybrid controllers for automated vehicles*. *IEEE Transactions on Automatic Control* 43(4), pp. 522–539, doi:10.1109/9.664155.

[20] Thomas Moor, Jörg Raisch & Siu O'Young (2002): *Discrete Supervisory Control of Hybrid Systems Based on l-Complete Approximations*. *Discrete Event Dynamic Systems* 12(1), pp. 83–107, doi:10.1023/A:1013339920783.

[21] Ben Moszkowski (1985): *A Temporal Logic for Multilevel Reasoning About Hardware*. *Computer* 18(2), pp. 10–19, doi:10.1109/MC.1985.1662795.

[22] T. Nipkow, L.C. Paulson & M. Wenzel (2003): *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*. Lecture Notes in Computer Science, Springer Berlin Heidelberg. Available at `https://books.google.de/books?id=xwdqCQAAQBAJ`.

[23] Heinrich Ody (2015): *Undecidability Results for Multi-Lane Spatial Logic*. In Martin Leucker, Camilo Rueda & Frank D. Valencia, editors: *Theoretical Aspects of Computing - ICTAC*, *LNCS* 9399, Springer, pp. 404–421, doi:10.1007/978-3-319-25150-9_24. Available at `http://theoretica.informatik.uni-oldenburg.de/~sefie/files/mlsl-undec-ictac15.pdf`.

[24] Ernst Rüdiger Olderog & Maike Schwammberger (2017): *Formalising a Hazard Warning Communication Protocol with Timed Automata*. In Luca Aceto, Giorgio Bacci, Giovanni Bacci, Anna IngÃşlfsdÃşttir, Axel Legay & Radu Mardare, editors: *Models, Algorithms, Logics and Tools*, *LNCS* 10460, Springer, pp. 640–660, doi:10.1007/978-3-642-39698-4_12.

[25] U. Ozguner, C. Stiller & K. Redmill (2007): *Systems for Safety and Autonomous Behavior in Cars: The DARPA Grand Challenge Experience*. *Proceedings of the IEEE* 95(2), pp. 397–412, doi:10.1109/JPROC.2006.888394.

[26] J. P. Queille & J. Sifakis (1982): *Specification and verification of concurrent systems in CESAR*, pp. 337–351. Springer, doi:10.1007/3-540-11494-7_22.

[27] Maike Schwammberger (2017): *Imperfect Knowledge in Autonomous Urban Traffic Manoeuvres*. In: *Proceedings First Workshop on Formal Verification of Autonomous Vehicles, FVAV@iFM 2017, Turin, Italy, 19th September 2017.*, pp. 59–74, doi:10.4204/EPTCS.257.7.

[28] M. Werling, T. Gindele, D. Jagszent & L. Groll (2008): *A robust algorithm for handling moving traffic in urban scenarios*. In: *2008 IEEE Intelligent Vehicles Symposium*, pp. 1108–1112, doi:10.1109/IVS.2008.4621260.