Imperfect Knowledge in Autonomous Urban Traffic Manoeuvres*

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

schwammberger@informatik.uni-oldenburg.de

Urban Multi-lane Spatial Logic (UMLSL) was introduced in [13] for proving safety (collision freedom) in autonomous urban traffic manoeuvres with perfect knowledge. We now consider a concept of imperfect knowledge, where cars have less information about other cars. To this end, we introduce the concept of a multi-view and propose crossing controllers using broadcast communication with data constraints for turning manoeuvres at intersections.

Keywords. Urban traffic, autonomous cars, collision freedom, imperfect knowledge, broadcast communication, multi-view, timed automata, multi-dimensional spatial logic.

1 Introduction

In urban traffic, turning at an intersection is a challenge for autonomous cars, since other cars approach the crossing from various directions. To pass the intersection, the cars use possibly overlapping parts of the critical resource: the intersection. In a previous paper [13], we proposed *crossing controllers*, that can safely conduct turn manoeuvres with *perfect knowledge*. Here, perfect knowledge means that every car knows the physical size and braking distance of all other cars on the road. In our meaning, *safety* means collision freedom and thus reasoning about car dynamics and spatial properties.

An approach to separate the car dynamics from the spatial considerations and thereby simplify reasoning, was introduced in [11] with the *Multi-lane Spatial Logic* (MLSL) for expressing spatial properties on multi-lane motorways. This logic and its dedicated abstract model was extended with length measurement in [12] for country roads with oncoming traffic. We again extended this approach in [13] by introducing a generic topology of *urban traffic networks* and *Urban Multi-lane Spatial Logic* (UMLSL) for reasoning about traffic situations at intersections.

The key contribution of our paper is the adaption of the existing controller for perfect knowledge from [13] to a communicating crossing controller with *imperfect knowledge*, meaning that, besides its own braking distance, a car only perceives the physical size of other cars. To cope with this penalty, we extend the crossing controller by a concept of *broadcast communication with data constraints* to communicate with helper controllers which are located in other cars. We also define a *multi-view* covering all roads that meet at an intersection.

Our approach differs from the work of Ody [20] on monitoring of traffic situations, where the author also uses the abstract model and MLSL from [11]. There for single sequences of traffic snapshots it is automatically checked if a MLSL formula holds globally throughout the sequence.

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

L. Bulwahn, M. Kamali, S. Linker (Eds.): First Workshop on Formal Verification of Autonomous Vehicles (FVAV 2017).
EPTCS 257, 2017, pp. 59–74, doi:10.4204/EPTCS.257.7

We construct our crossing controller based on the design of the controllers in [11, 12, 13] and specifically for our urban traffic use case. Another approach is to synthesize controllers from given properties, which was already investigated for basic MLSL for highway traffic in [6]. However, their synthesized controllers abstract from a continuous time dimension.

We consider fully autonomous cars and thus do not model human drivers. Additionally, we do not consider cases where people invade the safety envelope of a car, but refer the approach of Althoff and Magdici [2] for this. There the authors compute an over-approximation of possible occupancies of traffic participants over time to ensure safety of autonomous cars. A different attempt to broaden the approach with MLSL for highway traffic and country roads to intersection scenarios was introduced by Xu and Li in [25]. Instead of our directed graph topology, the authors introduce a *space grid model*, where single grids may belong to horizontal lanes or vertical lanes or to no lane at all, e.g. because they are blocked by a building. The authors only apply their results to T-junctions and construct a controller for this special case. Moreover, for transitions between different traffic snapshots only a discrete time dimension is applied. Loos and Platzer investigate intersections of single lanes with one car on each lane in [17]. They use traffic lights as a control mechanism, where a car is not permitted to enter an intersection when the light is red. They verify safety of their hybrid systems with the tool KeYmaera.

This paper is structured as follows. In Sect. 2, we adapt the abstract model from [13] and focus on our extension to the concept of a *multi-view*. We introduce the broadcast communication with data constraints in Sect. 3. We introduce syntax and semantics of the crossing controllers with communication in Sect. 4, where we also introduce the concept of the crossing and helper controller for imperfect knowledge. We then construct the new crossing controller and its helper controllers. A conclusion, some further related work and ideas for proving safety of our controllers in future work are given in Sect. 5.

2 Abstract Model

We start with an informal introduction of the considered abstract model for urban traffic scenarios and give more formal details for central concepts in the respective subsections. Topics marked with * are from our previous paper [13] and only introduced briefly, while their formal definition from [13] can also be found in the appendix.

The abstract model contains a set $\mathbb{CS}$ of *crossing segments* $c_0, c_1, \ldots$ and a set $\mathbb{L}$ of *lanes (lane segments)* $0, 1, \ldots$ connecting different crossings. Each crossing segment and each lane (segment) has a finite length. Adjacent lanes are bundled to *road segments* $\{0,1\}, \{2,3\}, \ldots \in \mathbb{RS}$ such that $\mathbb{RS}$ is a subset of $\mathscr{P}(\mathbb{L})$. Typical representatives of $\mathbb{RS}$ are $r_0, r_1, r_2, \ldots$. Adjacent crossing segments form an intersection, e.g. named by cr. The connections of lane and crossing segments are defined by an underlying graph topology called *urban road network* $\mathscr{N}$ (cf. Sect. 2.1).

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. We use car E as the *car under consideration* (short *ego car* or *actor*) and introduce the special constant *ego* with valuation $\nu(ego) = E$ to refer to this car. While a *reservation* re(ego) is the space car E is actually occupying, a *claim* cl(ego) is akin to setting the direction indicator representing the space a car plans to drive on in the future (cf. dotted part of car G in Fig. 1, where G plans to change its lane). This static information about cars like position, reservation and claim is captured in a traffic snapshot TS (cf. Sect. 2.2). To simplify reasoning, only local parts of the traffic snapshot are considered as every car has its own

local *view*, cf. view $V_1(E)$ of car E in the example (cf. Sect. 2.3). For logical reasoning, we then evaluate formulae of the *Urban Multi-lane Spatial Logic* (UMLSL) over a view (cf. Sect. 2.4).

With *imperfect knowledge*, we assume, that the actor E only perceives those parts of other cars it can perceive with its sensors: The physical position and size of the car (cf. solid parts of cars in Fig. 1), but not the braking distance (cf. dashed parts). Only ego car E itself knows its own braking distance and thus its whole *safety envelope*, while the braking distances of the other cars are invisible to E. In our approach, the safety property is already violated, if a car invades the braking distance of another car and not only if a physical collision occurs. The idea is, that in case of an emergency braking manoeuvre our safety property is still valid.

We distinguish between the movement of cars on lanes and on crossings. We allow for two-way traffic on lanes of continuous space and finite length, where every lane has one direction and cars normally drive on a lane in the direction of increasing real values, but may temporarily drive in the opposite direction for overtaking. As a car's direction changes while turning at an intersection, we can not assign one specific direction to a crossing segment and consider them as discrete and either fully occupied by a car or empty. An example for this is car A in Fig. 1, where A occupies the whole discrete crossing segment c_3. When a car is about to drive onto a discrete crossing segment and time elapses, the car's safety envelope stretches to the whole crossing segment, while disappearing continuously on the lane it drove on.

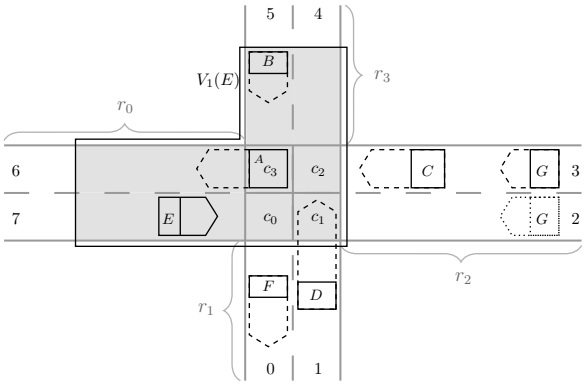

Figure 1: Car E perceives the physical size of other cars in its view $V_1(E)$. The dashed braking distances of other cars are invisible for E.

2.1 Topology

We restrict the abstract model to road segments with two lanes, one in each direction. Intersections are four connected crossing segments with four road segments meeting at a crossing (cf. example in Fig. 1). We describe connections between lanes and crossing segments by an *Urban Road Network** $\mathscr{N}$, whose nodes are from the set $\mathbf{V} = \mathbb{L} \cup \mathbb{CS}$ of lanes and crossing segments. As we are dealing with traffic that is evolving over time, we capture the (finite and real valued) length of lanes and crossing segments in our graph by assigning a weight $\boldsymbol{\omega}(v)$ to each node $v \in \mathbf{V}$. Adjacent crossing segments form strongly connected components I_{cs} (intersections, abbreviated with cr). Neighbouring lanes, connected with an undirected edge for bidirectional lane change manoeuvres, are components I_l (road segments, abbreviated with r). Edges from the sets $\mathbb{L} \times \mathbb{CS}$, $\mathbb{CS} \times \mathbb{L}$ and $\mathbb{CS} \times \mathbb{CS}$ are directed, whereby entry and exit points to the intersection are defined unambiguously. With these connected components, we can construct a coarser version $\mathscr{N}_I$ of $\mathscr{N}$, where a road segment r is connected with a crossing cr with a directed edge (r, cr) resp. (cr, r), iff there exists a matching directed edge in the underlying graph $\mathscr{N}$.

The corresponding road network $\mathscr{N}$ to Fig. 1 is depicted on the left side of Fig. 2 and the coarser version $\mathscr{N}_I$ on the right. A suitable *path* for car E in $\mathscr{N}$ is $pth(E) = \langle \ldots, 7, c_0, c_1, c_2, 4, \ldots \rangle$, where it plans on turning left. This *fine-grained path* is used later to determine the parts of lanes and crossing segments an arbitrary car occupies in a view.

The coarser version of this path is $pth(E)_I = \langle \ldots, r_0, cr, r_3, \ldots \rangle$, where cr is the name of the whole intersection. Such *coarse-grained paths* are used later to build the virtual lanes for our multi-view in Sect. 2.3.

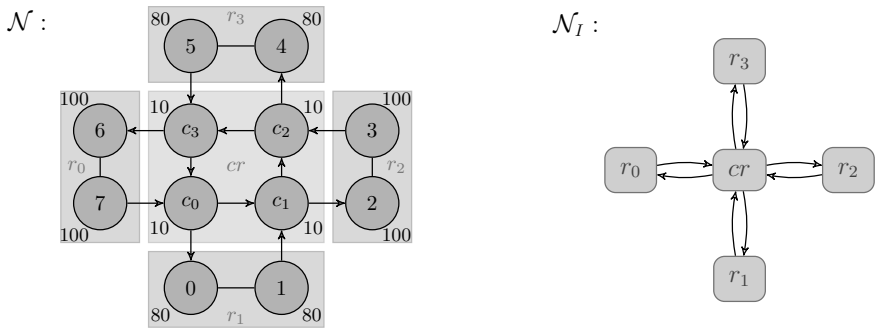

Figure 2: Urban road network $\mathcal{N}$ corresponding to Fig. 1 left and coarser version $\mathcal{N}_I$, only depicting the strongly connected components and their relations with each other, at the right.

2.2 Traffic Snapshot

A *traffic snapshot** TS captures the traffic on an urban road network $\mathcal{N}$ at a given point in time and is defined by the structure

$$TS = (\mathcal{N}, pth, curr, pos, res, clm, cres, cclm),$$

where $pth(C)$ is the path an arbitrary car C traverses in the urban road network $\mathcal{N}$. The index $curr(C)$ relates to the node $pth(C)_{curr(C)}$ the car C is currently driving on. With $pos(C)$ the real-valued position of the rear of car C on $pth(C)_{curr(C)}$ is defined. The set $clm(C)$ (resp. $res(C)$) is the set of all lanes C claims (resp. reserves) and $cclm(C)$ (resp. $cres(C)$) is the set of crossing segments C claims (resp. reserves). Amongst others, we demand the following *sanity conditions* to hold for an arbitrary traffic snapshot TS:

$$0 \leq |res(C)| \leq 2, \qquad 0 \leq |clm(C)| \leq 1, \qquad |res(C)| + |clm(C)| \leq 2, \tag{1}$$

$$1 \leq |res(C)| + |cres(C)|, \qquad |cclm(C)| \geq 1 \rightarrow |clm(C)| = 0 \wedge |res(C)| = 1. \tag{2}$$

With conditions 1, we only allow for one lane change manoeuvre at once, where either with $|clm(C)| = 1$ a lane is claimed or with $|res(C)| = 2$ the car is already changing lanes. With conditions 2, we state that at any point in time, a car reserves at least one lane or crossing segment and we only allow for a crossing claim, if car C is not involved in a lane change manoeuvre. Thus, a car may not enter an intersection with an active lane change manoeuvre.

To model the behaviour of cars, we allow *evolution transitions** between traffic snapshots which respect the sanity conditions. The node in $pth(C)$ that is reached after some time t elapses, we call $pth(C)_{next(C)}$. This node can either be a crossing or lane segment. Note that $pth(C)_{next(C)}$ is the node, where after t time units the position of the rear of car C is located, while it is possible, that the safety envelope of C stretches to more nodes. When approaching an intersection cr, we claim all needed crossing segments from cr that car C traverses in its path $pth(C)$ at once. Note that $curr(C) = next(C)$, iff C did not move far enough to leave its current node.

2.3 Imperfect Knowledge and Multi-View

For logical reasoning we consider only finite parts of the traffic snapshot TS. The idea is that the safety of a car depends only on its immediate surroundings. We therefore use the concept of a local *view*, which only contains those parts of lanes and crossing segments that are within some *horizon* h around the actor E. In previous work [11, 12] covering highway and country road traffic, the set of lanes L in a view was obtained by taking a subinterval of the global set of parallel lanes $\mathbb{L}$. This is no longer possible for the urban traffic scenario, since taking an arbitrary subinterval of lanes can yield a set of lanes which are not connected. We therefore construct a view from the urban road network, the current traffic snapshot, a given real-valued interval $X = [a,b]$ and the owner of the view E.

Definition 1 (View)**.** *For a road network $\mathscr{N}$ and its nodes* V *the* view $V(E) = (L,X,E)$ *of car E contains a set of* virtual *lanes $L \subseteq \mathscr{P}(\mathbf{V}^{\mathbb{Z}})$, an interval of space along the lanes $X = [a,b] \subseteq \mathbb{R}$ visible in $V(E)$ and $E \in \mathbb{I}$ as the car identifier of car E under consideration.*

If an intersection is within the horizon h, we deal with a bended view as cars are allowed to turn in any possible direction at the crossing (cf. view $V_1(E)$ in Fig. 1). To allow for spatial reasoning with our logic UMLSL, we flatten the view by constructing a straight *virtual view* from the urban road network $\mathscr{N}$ and the path $pth(E)$ of car E. As we currently only consider intersections of two by two lanes, one virtual view is also composed of two *virtual lanes*.

For perfect knowledge, it was sufficient to consider only that virtual view for the actor E which corresponds to its path $pth(E)$ (cf. view $V_1(E)$ in Fig. 1, where E plans on turning left). With imperfect knowledge, E can not perceive whether the safety envelope of a car that is not (yet) physically driving on the crossing already stretches to some crossing segments.

Consider again the example from Fig. 1, where car E does not perceive the braking distance of car D which already stretches to the intersection. To cope with the imperfect knowledge, we propose, that car E communicates with all cars on the intersection and with all cars that are approaching the intersection from any direction. Therefore, we need to consider more than the previously introduced one bended view $V_1(E)$ and introduce the concept of a virtual *multi-view* $V_m(E)$. This view covers the already introduced view $V_1(E)$ as well as view $V_2(E)$ covering road segment r_0, the intersection and segment r_2 and view $V_3(E)$, covering r_0, the intersection and r_1. Note that we do not consider the u-turn direction of the intersection, because r_0 is already covered in all other virtual views. The constructed multi-view is depicted in Fig. 3.

To formally build the multi-view, we first identify the road segment E is currently driving on with the underlying graph topology (cf. Sect. 2.1). As in Sect. 2.2, the current path segment E is driving on is defined by $pth(E)_{curr(E)}$ (in the example: 7) and the related road segment $r_{curr(E)}$ is given through the strongly connected component $I_l(pth(E)_{curr(E)})$ (in the example: $I_l(7) = r_0$). When a crossing is ahead, the first crossing segment E will drive on when entering the intersection is given by $pth(E)_{next(E)}$ (in the example: c_0) and therefore the whole intersection cr is obtained through the connected component $I_{cs}(pth(E)_{next(E)})$ (in the example: $I_{cs}(c_0) = cr$).

Next we identify all road segments r_i apart from $r_{curr(E)}$ which are connected to the intersection cr with a directed edge (r_i, cr) in the coarser graph $\mathscr{N}_I$. We can simply do this by considering all crossing segments $c_i \in cr$ and identifying the lanes $l_i \in r_i$ which have a directed edge to c_i in $\mathscr{N}$. This way, we detect all road segments from which cars can enter the junction and derive pairs of virtual lanes, later needed for the construction of the respective virtual views.

Definition 2 (Virtual Lanes)**.** *Consider a car E, its current path element $\pi_i = pth(E)_{curr(E)}$ and its next path element $\pi_{i+1} = pth(E)_{next(E)}$. We derive the neighbouring lane $\pi_{i,n}$ to π_i from the*

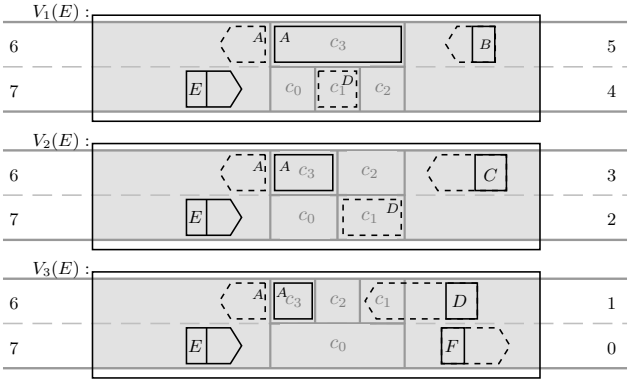

Figure 3: The virtual multi-view $V_m(E) = (V_1(E), V_2(E), V_3(E))$ of car E covers the road segment E is driving on, the intersection and all other road segments linked to the intersection.

urban network $\mathcal{N}$*, where it is the only node connected to* π_i *with an undirected edge. The current road segment is defined by* $r_{curr(E)} := I_l(\pi_i)$ *and the next intersection by* $cr := I_{cs}(\pi_{i+1})$.

We use the function $pre(cr)$ *to identify the set of all predecessor nodes* r_j *with an edge* (r_j, cr) *in the coarser graph* $\mathcal{N}_l$*. The* coarser virtual lanes $\mathbf{L}_j$ *are given through*

$$\forall r_j \in pre(cr) \wedge r_j \neq r_{curr(E)} : \mathbf{L}_j = (r_{curr(E)}, cr, r_j).$$

To identify the corresponding finer virtual lanes $\overrightarrow{\pi_j}$ *(driving direction according to* E*'s driving direction) and* $\overleftarrow{\pi_j}$ *(driving direction opposite to* E*'s driving direction) contained in each* $\mathbf{L}_j$*, we identify the shortest directed path forwards from* π_i *to an element* $\pi_{j_1} \in r_j$ *to build* $\overrightarrow{\pi_j}$*. For* $\overleftarrow{\pi_j}$*, we search the shortest directed path backwards from an element* $\pi_{j_2} \in r_j$ *to the neighbouring node* $\pi_{i,n}$*. We then derive for all coarser virtual lanes* $\mathbf{L}_j$ *the finer virtual lanes*

$$\overrightarrow{\pi_j} = [\pi_i, \overrightarrow{cs}, \pi_{j_1}] \text{ and } \overleftarrow{\pi_j} = [\pi_{i,n}, \overleftarrow{cs}, \pi_{j_2}],$$

where $\overrightarrow{cs}$ *and* $\overleftarrow{cs}$ *are the respective shortest directed subpaths through the intersection* cr*. The set of all* virtual lanes *is given by* $L_m = \{(\overrightarrow{\pi_1}, \overleftarrow{\pi_1}), \ldots, (\overrightarrow{\pi_n}, \overleftarrow{\pi_n})\}$*, where* $n := |pre(cr)| - 1$.

From definition 2 we obtained pairs L_j of virtual lanes $\overrightarrow{\pi_i}$ and $\overleftarrow{\pi_i}$, which each are used to build one virtual view $V_j(E, TS) = (L_j, X, E)$ for a traffic snapshot TS. All virtual views together lead to multi-view $V_m(E, TS) = (V_1(E, TS), \ldots, V_n(E, TS))$. To build these virtual views $V_j(E, TS)$ from the virtual lanes L_j, we need to define the size of the extension $X = [a, b]$ along the lanes. From the position $pos(E)$ of the car under consideration E, we look forwards and backwards up to a sufficient constant horizon h_f resp. h_b. We make sure that h_f is big enough, that a fast car approaching the intersection, that can already have a crossing claim or reservation on the intersection, is included in $h_f{}^{\star}$. We consider the same extension $X = [pos(E) - h_b, pos(E) + h_f]$ for each pair of virtual lanes. A virtual view is then defined from the pairs of virtual lanes with the described extension as follows.

Definition 3 (Virtual view and multi-view)**.** *For a car* E*, a traffic snapshot* TS*, a pair of virtual lanes* $L_i = (\overrightarrow{\pi_i}, \overleftarrow{\pi_i})$ *and the extension* $X = [pos(E) - h_b, pos(E) + h_f]$ *the* virtual view V_i *of* E *is defined by* $V_i(E, TS) = (L_i, X, E)$.

The set of all virtual views for car E*, built for one intersection* cr *is named the* multi-view $V_m(E, TS) = (V_1(E, TS), \ldots, V_n(E, TS))$*, where* n *is the amount of pairs of virtual lanes* L_i *constructed through Def. 2. We abbreviate* $V(E, TS) = V(E)$ *if* TS *is clear from context.*

Sensor Function. The car dependent sensor function $\Omega_E : \mathbb{I} \times \mathbb{TS} \to \mathbb{R}$ yields, given an arbitrary car C and a traffic snapshot TS the physical size of a car C as perceived by E's sensors.

Visible Segments of Cars in a View*. For both virtual lanes, we need to find all segments $seg_V(C)$ which are (partially) occupied by a car C and visible in the view of E. Considering highway traffic on continuous lanes, it is easy for a car E to obtain the interval of space $[a_c, b_c]$ another car C occupies in its view $V(E)$ through $a_c := pos(C)$ and $b_c := pos(C) + \Omega_E(C)$.

For urban traffic with intersections, it is a lot more complicated to address this task, because lanes as well as crossing segments are of finite length. Thus, the perceived size $\Omega_E(C)$ of a car C may stretch over several (connected) lane and crossing segments in the road network $\mathscr{N}$ (cf. car A in Fig. 1, whose physical part will occupy crossing segment c_3 and a part of lane segment 6 when it leaves the intersection in the near future). We therefore construct the set of segments $seg_V(C)$ another car C occupies in the virtual view of car E by taking the position of C, its size $\Omega_E(C)$ and the weight of nodes as defined in the road network $\mathscr{N}$ into account. For details for the construction of $seg_V(C)$, we refer to [13].

2.4 Urban Multi-lane Spatial Logic

Using car variables $c \in \mathbf{CVar} \cup \{\mathbf{ego}\}$ ranging over car identifiers and variables $u, v \in \mathbf{CVar} \cup \mathbf{RVar}$ with $\mathbf{RVar}$ ranging over the real numbers the syntax of UMLSL formulae is defined by

$$\phi ::= true \mid u = v \mid free \mid cs \mid re(c) \mid cl(c) \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \exists c \bullet \phi_1 \mid \phi_1 \frown \phi_2 \mid \begin{smallmatrix}\phi_2\\ \phi_1\end{smallmatrix}.$$

We use the atom *free* to represent free space and *cs* for crossing segments. Hereby, we can e.g. state that car E claims ($cl(\mathbf{ego})$) or reserves ($re(\mathbf{ego})$) a crossing segment ($cs \wedge (cl(ego) \vee re(\mathbf{ego}))$) or that a crossing segment is free ($cs \wedge free$). We can formalise the size of a horizontal interval in UMLSL, where e.g. $free \wedge \ell > d$ holds, if there is an interval of free space on a lane exceeding the size $d \in \mathbb{R}^+$. Besides these atoms, Boolean connectors and first-order quantifiers, formulae of UMLSL use two *chop operators*. One for a horizontal chop, denoted by $\phi_1 \frown \phi_2$ like for interval temporal logic [18] and one for a vertical chop given by the vertical arrangement of formulae $\begin{smallmatrix}\phi_2\\ \phi_1\end{smallmatrix}$. Intuitively, a formula $\phi_1 \frown \phi_2$ holds if we can split the view V horizontally into two views V_1 and V_2 such that on V_1 ϕ_1 holds and V_2 satisfies ϕ_2. Similarly a formula $\begin{smallmatrix}\phi_2\\ \phi_1\end{smallmatrix}$ is satisfied by V, if V can be chopped at a lane into two subviews, V_1 and V_2, where V_i satisfies ϕ_i for $i = 1, 2$.

In a part of view $V_1(E)$ (cf. Fig. 1) the formula $\phi \equiv re(\mathbf{ego}) \frown free \frown cs \wedge free$ holds. Here, $re(\mathbf{ego})$ is the space car E reserves on lane 7, the atom *free* represents the free space in front of car E, and $cs \wedge free$ stands for the unoccupied space on crossing segment c_0.

In case of a single (possibly virtual) view $V(E)$ of car E, the semantics* of UMLSL formulae is evaluated over a traffic snapshot TS, the view $V(E)$ and a valuation ν, which defines the current valuation $\nu(u)$ of variables u with elements from $Var = \mathbb{I} \cup \mathbb{R} \cup \mathbb{CS}$. In case of a multi-view V_m, we define the following satisfaction of a formula ϕ over V_m.

Definition 4 (Multi-view semantics of UMLSL formulae)**.** *For a multi-view $V_m = \{V_0, \ldots, V_n\}$, a traffic snapshot TS and a valuation ν the* satisfaction *of a formula ϕ is defined by*

$$TS, V_m, \nu \models \phi \;\Leftrightarrow\; \forall V_i \in V_m : TS, V_i, \nu \models \phi.$$

Existential satisfaction over a multi-view is possible with $\exists V_i \in V_m : TS, V_i, \nu \models \phi$.

Abbreviations. We use the abbreviation $\langle\phi\rangle$ to state that a formula ϕ holds *somewhere* in the considered view. We use abbreviations like $\phi^{<d}$ or $\phi^{>d}$ for $\phi \wedge \ell < d$ resp. $\phi \wedge \ell > d$.

Twisted views and the evaluation of UMLSL formulae. For highway traffic and country roads [11, 12], spatial formulae of MLSL are evaluated from "left to right". In urban traffic, a car C builds up the virtual multi-view from its own perspective, to evaluate formulae of the UMLSL. Consider again Fig. 3. In view $V_1(E)$, the formula $\phi \equiv \langle re(E) \frown free \frown cs \rangle$ holds. Now consider the respective view $V_1(B)$, comprising the same lane and crossing segments as $V_1(E)$, but build up from the sight of car B. This view is comparable with $V_1(E)$, twisted around by 180 degrees. In view $V_1(B)$, the formula $\phi \equiv \langle re(E) \frown free \frown cs \rangle$ does not hold, whereby its inverse version $\phi^{-1} \equiv \langle cs \frown free \frown re(E) \rangle$ holds.

3 Broadcast Communication with Data Constraints

In our abstract model, the autonomous cars can be understood as nodes in a *Vehicular ad-hoc network* (VANET), without a fixed wireless infrastructure and without taking roadside units into account. In [22], we proposed a concept of broadcast communication with data constraints for the there introduced hazard warning controllers. We reuse this communication concept for the controllers we introduce in Sect. 4 and which are modelled as extended timed automata [3]. One extension is the use of data variables and data constraints in guards, invariants and variable updates, as described by Behrmann et al. in [5] for UPPAAL. We broaden this use of data constraints in timed automata even more by sending data via broadcast channels.

Alrahman et al. propose a *Calculus for Attribute-based Communication* in [1]. The authors consider systems with a large amount of dynamically adjusting components that interact via broadcast channels. Components broadcast valuations of data variables u via an attribute-based output $(u)@\Pi$ to all processes whose attributes satisfy the predicate Π. By using updates $a := u$ of local attributes a, the received data u can be used locally by these processes. Other components only then synchronise with an output $(u)@\Pi$ when they have an input $\Pi(x)$ and their local attributes a, together with the received message x, satisfy the predicate Π. We adapt this concept of synchronisation in the definition of input and output actions for our controllers.

For data types on our channels, we use the Z notation [24] of sequences: $seq\, X$ is the set of all finite sequences of elements from a given set X. A sequence s consisting of elements A, B, C is written as $s = \langle A, B, C \rangle$. It stands for a function $s = \{1 \mapsto A, 2 \mapsto B, 3 \mapsto C\}$ from indices $1, 2, 3$ to elements A, B, C. Thus the ith element of s is denoted by function application $s(i)$, e.g., $s(2) = B$. The *length* of s is derived by $\#s$, here $\#s = 3$. For the empty sequence $\langle\rangle$ the length is 0.

Definition 5 (Input and Output actions)**.** *For a finite list of data variables $d = \langle d_1, \ldots, d_n \rangle$ and a UMLSL formula φ we define an* output action *OUT on a broadcast channel a by $OUT := a!d$ and a related* input action *IN by $IN := a?d : \varphi$. The set of data variables $d_i \in \mathbb{D}$ ranges over the set of all car identifiers $\mathbb{I}$, the power set $\mathscr{P}(\mathbb{L})$ (resp. $\mathscr{P}(\mathbb{CS})$) of the set of all lanes $\mathbb{L}$ (resp. all crossing segments $\mathbb{CS}$), and finite sequences* seq $\mathbb{I}$, seq $\mathbb{L}$ *and* seq $\mathbb{CS}$.

Abbreviation. We abbreviate $\langle d_1 \rangle = d_1$ for a single data variable d_1.

Example. A request of car E for some crossing segments is sent via broadcast channel *cross* with the output $cross!\langle ego, cs_{ego} \rangle$. Here, cs is the set of crossing segments car E claims for its turning manoeuvre and $\nu(\text{ego})$ is the senders car identifier. Consider a corresponding input $cross?\langle c, cs \rangle : a \neq c \wedge cs \cap cs_a = \emptyset$, where $\nu(a)$ is the car identifier of the request receiving controller and cs_a is the set of crossing segments this car reserves or claims itself. The received data is stored by the receiver in local variables: $\nu(c) = \nu(ego)$ and $\nu(cs) = \nu(cs_{ego})$. This input synchronises with the output iff the UMLSL formula $a \neq c \wedge cs \cap cs_a = \emptyset$ evaluated over valuation ν holds.

4 Controllers for Safe Crossing Manoeuvres

In [13], we introduced a *crossing controller* to perform turn manoeuvres at intersections with perfect knowledge. This controller made driving decisions according to the current view and traffic snapshot, where it was able to perceive the whole safety envelope of other cars and thus had information about all reserved or claimed lanes and crossing segments of other cars. With imperfect knowledge, ego car E is not able to perceive if the braking distance of another car stretches up to the crossing segments E plans to reserve for itself. Therefore, ego car E has to actively communicate with those cars to prevent collisions. For this purpose we adapt the crossing controller for perfect knowledge from [13] with broadcast communication elements as introduced in Sect. 3 and introduce a helper controller. This helper concept roughly follows the helper approach for imperfect knowledge for highway traffic from [11].

In previous works [11, 12, 13] we showed that if every car is equipped with the respective proposed controllers for the different traffic scenarios, safety in the sense of disjointedness of reservations is preserved under all time and action transitions. We check the property

$$Safe(ego) \equiv \neg\exists c\colon c \neq ego \wedge \langle re(ego) \wedge re(c)\rangle \tag{3}$$

from the viewpoint of ego car E and use the somewhere operator $\langle\rangle$. $Safe(ego)$ states, that there is never a spatial overlap of the reservation of E with the reservation of another car. Note that by demanding the disjointedness of (the speed-dependent) reserved spaces, the formula indirectly requires that E lowers its speed (to shorten its reserved space) when a car ahead of it starts breaking. To maintain $Safe\text{-}re(ego)$ under time transitions, each car has a *distance controller* as proposed by Damm et al. in [8]. Rizaldi et al. [23] examine safety distances for autonomous vehicles, which is useful for such a distance controller. For urban traffic we additionally demand that the distance controller keeps a positive distance to an intersection, if the car does not get permission to enter the intersection. In worst case the car comes to a standstill in front of the crossing until permission to conduct its planned turn manoeuvre is granted. The described distance controller initiates acceleration and braking manoeuvres for the car, which means setting inputs for the actuators on a lower level of controllers. A good example for such a controller on the dynamics level is given by Damm et al. in [9], where the authors introduce a velocity controller. In our approach, we explicitly separate our controllers from these car dynamics level and focus on a decision making level. That is, our controllers, e.g., decide how and whether a lane change or a crossing manoeuvre is conducted. This approach allows for a purely spatial reasoning. However, a link between the spatial and dynamic reasoning is formalised in [21].

Road segments between intersections are structurally comparable to country roads, wherefore we refer to [12], where a lane change controller for these roads was presented. We only modify this *road controller* by the requirement, that as soon as a crossing is ahead within some distance d_c, any claim must be withdrawn immediately and no new claim or reservation might be created until the crossing is passed. However, the car may finish an already begun overtaking manoeuvre, wherefore we make sure the distance d_c is big enough to do so. We assume crossings to be at least d_c apart from each other to guarantee correct functionality of our controllers.

4.1 Automotive-controlling Timed Automata

In [13], we introduced extended time automata, called *automotive-controlling timed automata* (ACTA)*, to formalise the controllers for different traffic scenarios from [11, 12, 13]. As variables

these controllers 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 and for data variables $d_i \in \mathbb{D}$ and data updates we refer to the extension of timed automata proposed for UPPAAL. These clock and data updates ν_{act} are allowed on transitions of the automata. Note that we allow for the same set of data variables $\mathbb{D}$ we introduced in Def. 5 for input and output actions, including sets and lists.

Further on, the controllers use UMLSL formulae φ_U as well as clock and data constraints $\varphi_{\mathbb{X}}$ resp. $\varphi_{\mathbb{D}}$ as guards φ on transitions and as invariants $I(q)$ in states q. An example for a data constraint for a variable $l \in Var$ is $l > 1$. We extend the data constraints for single variables from Var by set operations, which e.g. allows for $cs \cap cs' = \emptyset$ as a guard or invariant, where $cs, cs' \in \mathscr{P}(\mathbb{CS})$. The set Φ of all guards and invariants is defined by $\varphi \equiv \varphi_{\mathbb{U}} \mid \varphi_{\mathbb{X}} \mid \varphi_{\mathbb{D}} \mid \varphi_1 \wedge \varphi_2 \mid true$.

We use the broadcast communication as defined in Sect. 3. Remember that we consider output actions OUT which can synchronise with appropriate input actions IN in another controller. We also use *controller actions* c_{act} to commit lane change manoeuvres on road segments and turning manoeuvres at crossings, where e.g. `rc(ego)` is a *crossing reservation* action for ego car E and `wd rc(ego)` is the respective withdrawal action for a crossing reservation.

A transition in an ACTA comprises the elements depicted in Fig. 4. The guard $\varphi \wedge IN$ shown before the separator / has to hold with respect to the current traffic snapshot TS, the view $V(E)$ of ego car E and the valuation ν in order to execute the output, controller and update actions shown after the separator /, yielding a successor state q' and a valuation ν'. The invariant $I(q')$ has to hold in q'.

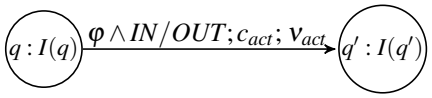

Figure 4: Syntax elements of an ACTA with communication

4.2 Imperfect Knowledge

In order to enter a crossing, a car first needs to claim a path through the crossing for its turn manoeuvre and check whether there is an overlap of this claim with the claim or reservation of another car, formalised by the *potential collision check*

$$pc(c) \equiv c \neq \text{ego} \wedge \langle cl(\text{ego}) \wedge (re(c) \vee cl(c)) \rangle. \tag{4}$$

If a potential collision is detected, the ego car must withdraw its claim. However, with imperfect knowledge the ego car is not able to detect a potential collision with the whole safety envelope of another car, but only with its physical size. Therefore, ego car E has to communicate with cars that might cause a potential collision. Following [11], we call those cars *helper cars*.

In urban traffic, a helper car for the ego car either has an own reservation on at least one crossing segment of the considered intersection or is approaching it from any direction. The case where a car is driving on a crossing segment is formalised by the *on crossing check*

$$oc(c) \equiv \langle re(c) \wedge cs \rangle. \tag{5}$$

For the second case, we first introduce the abbreviation *one lane*

$$ol \equiv (true \frown free \frown true) \vee \exists c : (re(c) \vee cl(c)),$$

stating, that there is *exactly one lane* occupied with something. While tempting, it is not sufficient to use only $true$ instead of ol because the formula $true$ also holds for zero lanes. If the ego car is approaching an intersection within the distance d_c, its crossing controller is supposed

to start claiming crossing segments. For an arbitrary other car C approaching the intersection from the opposite side of the intersection, we do not know the braking distance and therefore add the maximum safety envelope $se\,max(C)$ to d_c, yielding the distance $d'_c = d_c + max\,se(C)$. We formalise that a car approaches an intersection from the opposite side of the intersection within the distance d'_c with the *opposing car approaching the crossing check*

$$ocac(c) \equiv \left\langle \begin{array}{c} ol \\ re(\text{ego}) \end{array} \right\rangle \frown \left\langle \begin{array}{c} cs \frown \neg\langle cs\rangle \wedge free^{<d'_c} \frown re(c) \\ ol \end{array} \right\rangle \wedge dir(c). \tag{6}$$

The atom $dir(c)$ states whether a car drives in the direction of its lane or not, which the ego car is able to perceive with its sensors. This atom is needed to exclude the special case, that $ocac(c)$ comprises a car spatially driving on the requested lane but driving away from the intersection. A car that is driving away from the intersection is not of interest, as its own braking distance can not stretch to the intersection and as it might leave the view of ego car E soon anyway.

We generally forbid a car entering an intersection while changing lanes as the directed edges in our topology do not allow this (cf. Sect. 2.1). Therefore, we introduce the *lane change check*

$$lc(c) \equiv \left\langle \begin{array}{c} re(c) \\ re(c) \end{array} \right\rangle. \tag{7}$$

With formulae (5), (6) and (7), the ego car identifies all described suitable helper cars with the *potential helper check*

$$ph(c) \equiv c \neq \text{ego} \wedge (oc(c) \vee ocac(c)) \wedge \neg lc(c). \tag{8}$$

4.3 Crossing Controller

We now construct the crossing controller $\mathscr{A}_{cc}$ for turning manoeuvres on crossings with imperfect knowledge. The overall goal of the crossing controller is to perform turn manoeuvres at intersections while always maintaining the safety property (3). A coarser version of the detailed crossing controller $\mathscr{A}_{cc}$ depicted in Fig. 6 is shown in Fig. 5.

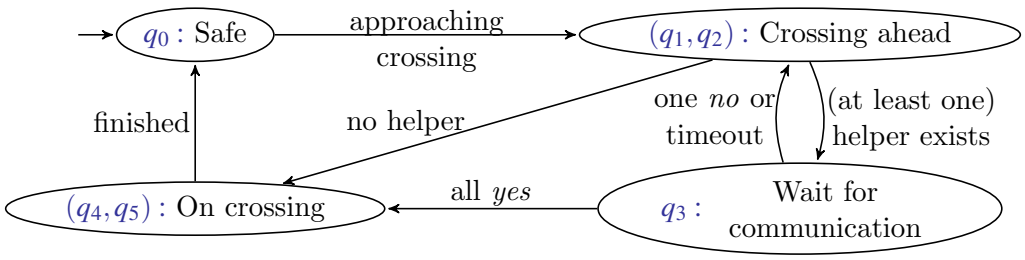

Figure 5: Overview over crossing controller protocol.

Overview (cf. Fig. 5). We assume the initial state of the controller to be Safe, i.e. no collision exists. When a crossing is ahead, the car may enter the intersection by itself, iff no helper exists (e.g. the multi-view is empty except for the ego car). If at least one potential helper exists, the actor needs to communicate with the helpers. If one helper sends a *no*-message or one helper

does not answer, the actor withdraws the crossing claim and may try to enter the intersection later again (somewhen the conflicting other car will have left the intersection). Iff all helpers send a *yes*-message, the ego car can safely enter the intersection and finish the crossing manoeuvre.

Details (cf. Fig. 6). We introduce a *collision check* $col(\text{ego})$ whose negation $\neg col(\text{ego})$ holds invariantly in the initial state of our crossing controller and is expressed by the UMLSL formula

$$col(\text{ego}) \equiv \exists c : c \neq \text{ego} \wedge \langle re(\text{ego}) \wedge re(c) \rangle. \tag{9}$$

The crossing controller only becomes active and leaves its initial safe state, if E approaches an intersection within less than the previously introduced distance d_c with no other car between the actor and the intersection. For this, we formalise the *crossing ahead check*

$$ca(\text{ego}) \equiv \langle re(\text{ego}) \frown free^{<d_c} \wedge \neg \langle cs \rangle \frown cs \rangle. \tag{10}$$

The crossing controller *claims* the crossing segments needed for the turn manoeuvre with the controller action `cc(ego)`. Then it checks for a potential collision $pc(c)$ (4) with an arbitrary car c and possibly withdraws the crossing claim. Else with the potential helper check $ph(c)$ (8) it evaluates if a helper for the manoeuvre is available, where we observe two possible results:

1. No helper car is available or
2. At least one helper car exists.

In the first case, the controller proceeds without help. If $lc(\text{ego})$ (7) and $pc(c)$ (4) do not hold, the actor reserves the claimed crossing segments and starts the crossing manoeuvre. To prevent deadlocks, we set a time bound t_o for the time that may pass between claiming and reserving crossing segments. If the actor reserves crossing segments, the on crossing check $oc(\text{ego})$ holds invariantly. We assume a crossing manoeuvre to take at most t_{cr} time to finish. Once the actor has left the last crossing segment and is driving on a lane, the crossing manoeuvre is finished. The reservation of E is then reduced to the next segment after the intersection in $pth(E)$.

If helper cars are available, the crossing controller needs to communicate because of the missing information about the braking distances of the helpers. E sends the output message $cross!\langle ego, cs \rangle$, where cs is the set of crossing segments the ego car claims according to $pth(E)$. If E receives its own car identifier via channel *no*, it immediately withdraws its claim and changes back to q_1. While only one *no*-message is sufficient to abort the crossing manoeuvre, it is not enough to receive only one *yes*-message. Therefore, the controller waits t_w time units for the answers of the helpers, where we assume t_w to be a worst case time bound in which all helpers are technically able to answer. For realistic worst case time bounds in real-time broadcast communication, we e.g. refer to the work of Asplund et al. [4].

E collects all identifiers of helpers that answered via channel *yes* in a set $\mathbb{H}$. After t_w time, it compares $\mathbb{H}$ with the available potential helpers with $\neg \exists c \in \mathbb{I} \backslash \mathbb{H} : ph(c)$. Then it either reserves the claimed crossing segments, or withdraws the claim, if at least one potential helper did not answer. Once the crossing controller entered state q_3 and thus started the communication, it informs the helpers when it either withdraws a claim or successfully finishes the manoeuvre via broadcast channel *finish*. The constructed crossing controller is depicted in Fig. 6.

4.4 Helper Cars and Helper Controller

As introduced in Sect. 4.2, a helper car is either driving on the crossing or approaching it from a different direction than the ego car. An arbitrary car is allowed to be helper for more than one

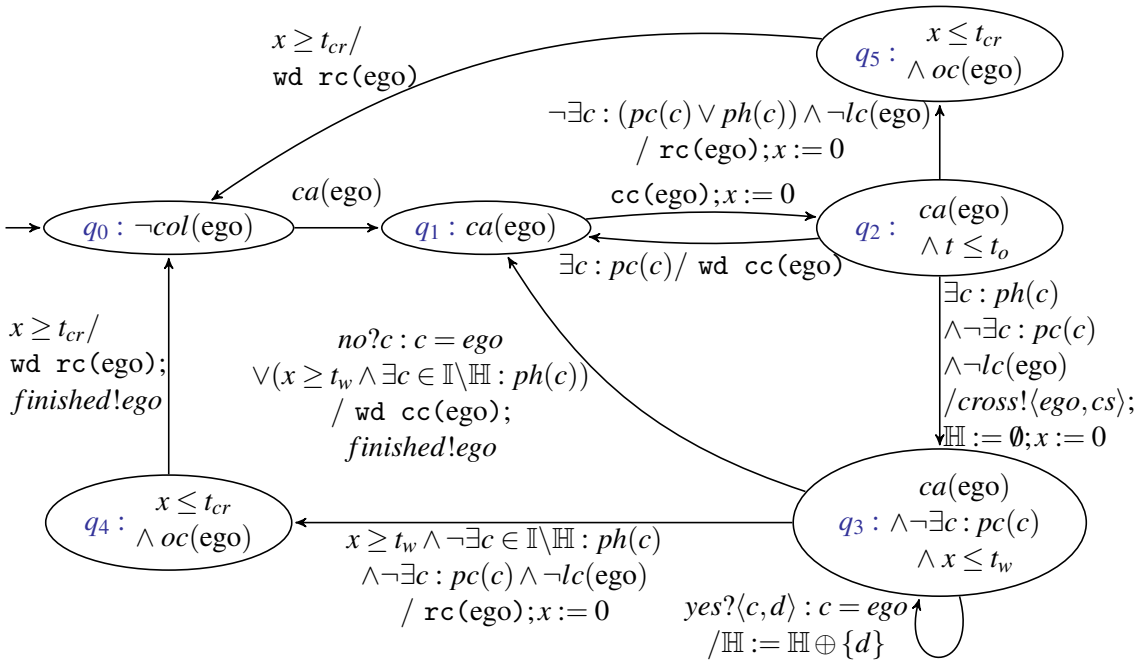

Figure 6: Crossing controller $\mathscr{A}_{cc}$

requesting car, e.g. needed if four cars turn simultaneously right at an intersection. We therefore assume that every car owns several clones of the helper controller, but only one of the helper controllers assist one specific car at once. A coarser version of the detailed helper controller $\mathscr{A}_{hc}$ depicted in Fig. 8 is shown in Fig. 7.

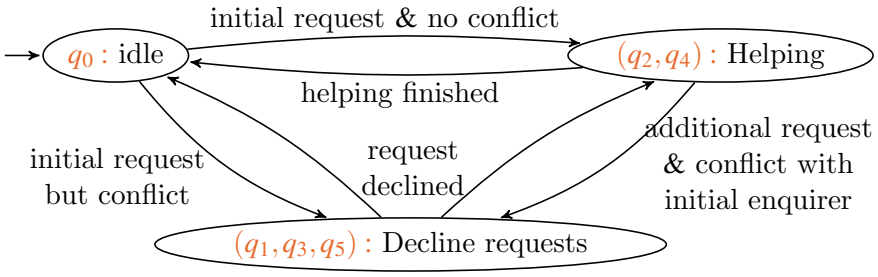

Figure 7: Overview over helper controller protocol.

Overview (cf. Fig. 7). Whenever an idle helper controller receives a crossing request it checks if it meets the helper requirements (on crossing or approaching crossing) and if there exist no potential collision of the request with its own crossing claim or reservation. Then it either declines the request or starts to help the enquirer. If the helper controller receives a conflicting request from another car during the helping process, it declines this request immediately.
Details (cf. Fig. 8). In the helper controller we use the unique variable a to identify the helper controller and we call a car searching for a helper *enquirer* or *enquiring car*. The set

$cs_a := cclm(a) \cup cres(a)$ denotes the claimed and reserved crossing segments of the helper car. If a car receives a broadcast request $cross![c,cs]$, its helper controller first checks if it is a potential helper for c with the *inverse potential helper check*

$$ph^{-1}(c,cs) \equiv a \neq c \wedge (oc(a) \vee ca(a)) \wedge \neg lc(a) \wedge (cs_a \cap cs = \emptyset). \tag{11}$$

With the first part of the formula, the potential helper checks if its position is suitable and whether it is currently changing lanes. With the latter part of the formula the potential helper checks for disjointedness of its own segments cs_a and the received crossing segments cs. Note that this check resembles the potential collision check for lanes. If the controller detects a potential collision, it immediately sends a *no*-message to the enquiring car.

If it is a potential helper, the controller sends a *yes*-message to the enquiring car in less than t time units, where $t < t_w$ and t_w is the time bound the crossing controller waits for the answers of the helpers. While helping, it additionally declines crossing requests from a third car whose request overlaps with the crossing segments of the car the helper already assists. If the helper left the intersection or if the crossing manoeuvre of the enquirer is finished, the helping process is finished. The resulting helper controller is depicted in Fig. 8.

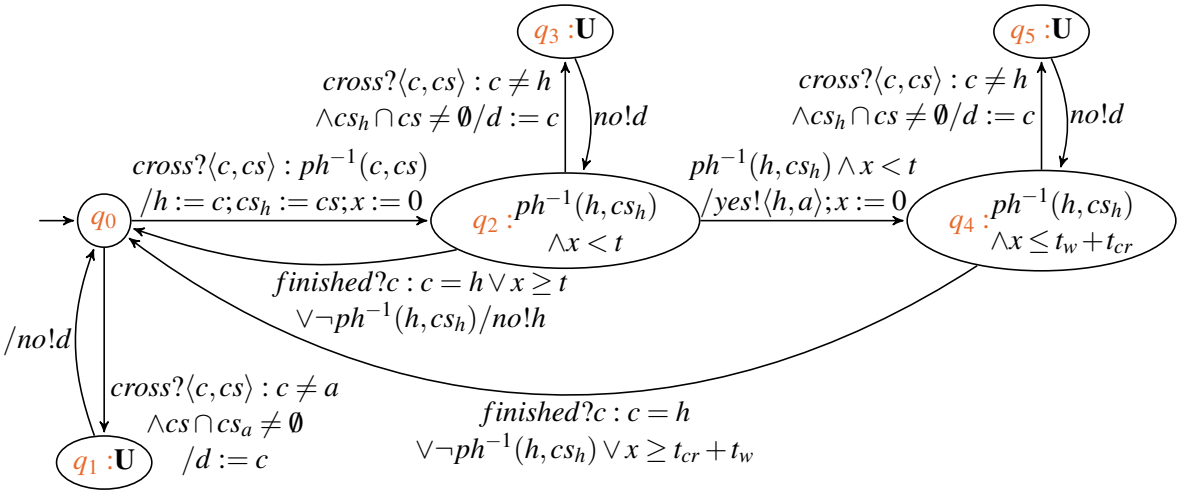

Figure 8: Helper controller $\mathscr{A}_{hc}$

5 Conclusion

We extend our approach for urban traffic manoeuvres with perfect knowledge from [13] by a more realistic concept of imperfect knowledge, where autonomous cars have no information about speed and braking distances of other cars. To this end, we introduce a multi-view semantics for UMLSL formulae. We propose broadcast communication with data constraints to specify our communicating crossing controllers, which can autonomously perform turn manoeuvres at intersections with the help of controllers in other cars at the intersection.

More on related work. Linker [16] and Ody [19] present undecidability results of the spatial part of MLSL, which unfortunately apply for our extension UMLSL, too. However, Fränzle et

al. [10] prove that MLSL is decidable, when considering only a bounded *scope* around the cars. This is a constraint motivated by reality because actual autonomous cars can only process state information of finitely many environmental cars in real-time.

Future work. The purely formal specification of our controllers, detached from the car dynamics, allows for formal verification of the safety condition (3) from p. 67 as future work. The proof idea is as follows: we show safety (3) from the viewpoint of an arbitrary actor E with that approaches an intersection and thus generates a multi-view $V_m(E)$ (cf. Sect. 2.3). We assume an initial safe traffic snapshot TS_0 and inductively show for every traffic snapshot TS_k, reachable from TS_0 by k evolution transitions, that it is also safe. For this purpose, we propose to separate the proof of spatial properties in UMLSL guards and invariants in the controllers from the proof for their timing and communication behaviour. The spatial part can be shown either by exploiting directly the semantics of guards and invariants in the controllers or by using an adaptation of the proof system introduced for standard MLSL in [16]. For the time and communication part of the extended timed automata controllers, we aim for a proof with assistance of UPPAAL [5].

The here proposed crossing controller is safe, but not deadlock free, wherefore it is interesting to examine a *(timed) liveness* property. By extending UMLSL with operators from Koymans *metric temporal logic* [15], we could express, that a car approaching an intersection ($ca(c)$) and that desires to cross it ($pth(c)_{next(c)} \in \mathbb{CS}$), *finally* ($\mathbf{F}$) passes it in less than t time units ($< t$):

$$Life \equiv \forall c : (ca(c) \wedge pth(c)_{next(c)} \in \mathbb{CS} \rightarrow \mathbf{F}_{<t} oc(c)).$$

The relation of our work to game theoretical approaches is interesting. We could e.g. use UPPAAL TiGa [7] for our purposes, where an extended timed automaton represents two players: the system itself and the environment. As environmental part, we could model the time out transitions of our controllers. The systems' goal is to reach a specific state (e.g. a state where *on crossing* ($oc(c)$) holds invariantly) or avoid a specific state (e.g. a *bad state* with a time out).

For now, we conveniently assumed broadcast communication as we already used it in previous approaches. For future work it is interesting to link our communication requirements more detailed to communication standards from Car2Car Communication (cf. Kenney [14]).

References

[1] Yehia Abd Alrahman, Rocco De Nicola, Michele Loreti, Francesco Tiezzi & Roberto Vigo (2015): *A Calculus for Attribute-based Communication.* In: *Proc. 30th Annual ACM Symp. on Applied Computing (SAC)*, ACM, pp. 1840–1845, doi:10.1145/2695664.2695668.

[2] Matthias Althoff & Silvia Magdici (2016): *Set-Based Prediction of Traffic Participants on Arbitrary Road Networks. IEEE Trans. Intelligent Vehicles* 1(2), pp. 187–202, doi:10.1109/TIV.2016.2622920.

[3] 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.

[4] Mikael Asplund & Simin Nadjm-Tehrani (2012): *Worst-case Latency of Broadcast in Intermittently Connected Networks. Int. J. Ad Hoc Ubiquitous Comput.* 11(2/3), pp. 125–138, doi:10.1504/IJAHUC.2012.050281.

[5] Gerd Behrmann, Alexandre David & Kim G. Larsen (2004): *A Tutorial on Uppaal*, pp. 200–236. Springer Berlin Heidelberg, doi:10.1007/978-3-540-30080-9_7.

[6] Gregor v. Bochmann, Martin Hilscher, Sven Linker & Ernst-Rüdiger Olderog (2017): *Synthesizing and verifying controllers for multi-lane traffic maneuvers. Formal Aspects of Computing* 29(4), pp. 583–600, doi:10.1007/s00165-017-0424-4.

[7] Franck Cassez, Alexandre David, Emmanuel Fleury, Kim G. Larsen & Didier Lime (2005): *Efficient On-the-Fly Algorithms for the Analysis of Timed Games*, pp. 66–80. Springer Berlin Heidelberg, Berlin, Heidelberg, doi:10.1007/11539452_9.

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

[9] Werner Damm, Eike Möhlmann & Astrid Rakow (2014): *Component Based Design of Hybrid Systems: A Case Study on Concurrency and Coupling.* In: *Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control*, HSCC '14, ACM, New York, NY, USA, pp. 145–150, doi:10.1145/2562059.2562120.

[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] 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 Berlin Heidelberg, doi:10.1007/978-3-642-24559-6_28.

[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 & 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.

[14] J. B. Kenney (2011): *Dedicated Short-Range Communications (DSRC) Standards in the United States. Proceedings of the IEEE* 99(7), pp. 1162–1182, doi:10.1109/JPROC.2011.2132790.

[15] Ron Koymans (1990): *Specifying real-time properties with metric temporal logic. Real-Time Systems* 2(4), pp. 255–299, doi:10.1007/BF01995674.

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

[17] 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.

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

[19] 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.

[20] Heinrich Ody (2017): *Monitoring of Traffic Manoeuvres with Imprecise Information.* FVAV17.

[21] Ernst Rüdiger Olderog, Anders P. Ravn & Rafael Wisniewski (2017): *Linking spatial and dynamic models, applied to traffic maneuvers.* In M. Hinchey, J. P. Bowen & E.-R. Olderog, editors: *Provably Correct Systems*, NASA Monographs in SSE, Springer, pp. 95–120, doi:10.1007/978-3-319-48628-4_5.

[22] Ernst Rüdiger Olderog & Maike Schwammberger (Springer 2017, to appear): *Formalising a Hazard Warning Communication Protocol with Timed Automata.*

[23] Albert Rizaldi, Fabian Immler & Matthias Althoff (2016): *A Formally Verified Checker of the Safe Distance Traffic Rules for Autonomous Vehicles.* In: *NASA Formal Methods – 8th International Symposiuim*, pp. 175–190, doi:10.1007/978-3-319-40648-0_14.

[24] J. Woodcock & J. Davies (1996): *Using Z – Specification, Refinement, and Proof.* Prentice Hall.

[25] Bingqing Xu & Qin Li (2016): *A Spatial Logic for Modeling and Verification of Collision-Free Control of Vehicles.* In: *21st ICECCS*, pp. 33–42, doi:10.1109/ICECCS.2016.014.