Carl von Ossietzky Universität Oldenburg
Fakultät II – Informatik, Wirtschafts- und Rechtswissenschaften
Department für Informatik

Distributed Controllers for Provably Safe, Live and Fair Autonomous Car Manoeuvres in Urban Traffic

Von der Fakultät für Informatik, Wirtschafts- und Rechtswissenschaften
der Carl von Ossietzky Universität Oldenburg
zur Erlangung des Grades und Titels eines

Doktors der Naturwissenschaften

angenommene Dissertation

von Maike Schwammberger

geboren am 10.03.1986 in Osterholz-Scharmbeck

Gutachter:	Prof. Dr. Ernst-Rüdiger Olderog Prof. Dr. Kim Guldstrand Larsen
Datum der Einreichung:	30. November 2019
Datum der Verteidigung:	20. März 2020

Abstract

During the last years, automated driving techniques are increasingly capturing the market. Therefore, it is particularly important to consider vital functional properties of these systems. Amongst others, *safety* in the context of collision freedom is to be guaranteed at all times.

With *Multi-lane Spatial Logic (MLSL)*, a formal approach was introduced to logically reason about the safety of traffic situations on multi-lane motorways and country roads. Safety (collision freedom) of an informally specified lane change controller for highways and of an overtaking protocol for country roads was proven.

The main focus of this thesis is to develop an extension of MLSL to deal with urban traffic scenarios, thereby focusing on safety aspects of *crossing manoeuvres at intersections*. To this end, we introduce a generic *topology* of urban traffic networks and a crossing controller using formulae of our extension *Urban Multi-lane Spatial Logic (UMLSL)* for turn manoeuvres at intersections. We show that even at intersections we can use purely spatial reasoning, detached from the underlying car dynamics, to prove safety of the crossing controller.

While all existing MLSL approaches focus solely on guaranteeing safety, we also examine *liveness* and *fairness* of the controllers. Here, liveness means that something good, e.g. a lane change manoeuvre, finally happens. Further on, fairness means that no car has to wait unreasonably long before starting a planned manoeuvre. We verify these system properties with the help of UPPAAL, a model-checker for (extended) timed automata.

Furthermore, we introduce a case study, where we adapt the MLSL approach to a *hazard warning communication protocol*. Again with the assistance of UPPAAL, we show that with our protocol, a hazard warning message is delivered timely.

Zusammenfassung

Während der letzten Jahre erobern autonome Fahrsysteme mehr und mehr die Märkte. Deshalb ist es insbesondere wichtig, zentrale Funktionen dieser Systeme sicherzustellen. Beispielsweise muss die *Sicherheit* im Sinne von Kollisionsfreiheit stets gewährleistet werden.

Die *Multi-lane Spatial Logic (MLSL)* erlaubt es, logische Schlussfolgerungen über die Sicherheit von Verkehrssituationen sowohl auf mehrspurigen Autobahnen als auch auf Landstraßen zu führen. Die Sicherheit eines informell skizzierten Controllers für Fahrspurwechsel auf Autobahnen und eines Überhol-Protokolls für Landstraßen wurde damit bewiesen.

Das Hauptanliegen dieser Doktorarbeit ist es, die Logik MLSL so zu erweitern, dass auch Stadtverkehr betrachtet werden kann. Konkret bedeutet dieses, dass die Sicherheit von *Abbiege-Manövern an Kreuzungen* untersucht wird. Hierfür wird eine *Graph-Struktur* vorgestellt, um Stadtverkehr-Netzwerke zu formalisieren, und die Logik MLSL zur *Urban Multi-lane Spatial Logic (UMLSL)* erweitert. Weiterhin wird ein Kreuzungs-Controller vorgestellt, der UMLSL-Formeln für Abbiege-Manöver an Kreuzungen benutzt. Wir zeigen, dass wir sogar an Kreuzungen formale räumliche Schlussfolgerungen nutzen können, um die Sicherheit des Kreuzungs-Controllers zu beweisen.

Während sich die bisher existierenden Ansätze zu MLSL lediglich mit Sicherheits-Aspekten befassen, untersuchen wir weiterhin die *Lebendigkeit* und *Fairness* sowohl des Fahrspurwechsels- als auch des Kreuzungs-Controllers. In diesem Fall bedeutet Lebendigkeit, dass ein wünschenswertes Ziel, beispielsweise ein Fahrspurwechsel, schließlich passiert. Weiterhin bedeutet Fairness, dass kein Auto unangemessen lange warten muss, bevor es ein geplantes Fahr-Manöver starten kann. Wir zeigen diese System-Eigenschaften mit Hilfe des Model-Checking Tools *UPPAAL*.

Als Fallstudie stellen wir zudem eine Erweiterung des MLSL-Ansatzes zu einem *Kommunikations-Protokoll* zum Warnen vor Hindernissen vor. Wieder mit Hilfe von UPPAAL beweisen wir, dass Unfall-Warn-Nachrichten zeitig genug übermittelt werden.

Acknowledgements

Looking back, it feels like I have not just sat in my office writing this thesis but sometimes visited rather dark places, where I basically wished to throw things at other things, but also quite joyful places, where I have been proud of an achievement or simply pleased by a particularly neat formulated sentence that I pondered on for apparent hours. I thank all those different people that abode me when I have been in the darker places and that helped me visit the joyful places more often.

I thank Ernst-Rüdiger Olderog for being such a kind and supporting supervisor and for giving me a lot of freedom both in research and in teaching. I feel that I have become an independent researcher with your help. I also thank you for the always very detailed feedback, without which this thesis could not have become what it is now. For being a second mentor throughout my PhD and always being willing to lend an ear, I thank Martin Georg Fränzle. For suffering my presence when I have been in those places where I wanted to throw things, I thank Christopher Bischopink.

While I wrote this thesis, Almuth Meyer, Christoph Peuser, Christopher Bischopink, Heinrich Ody, Nick Würdemann and Nils Worzyk gave me their valuable time and feedback, for which I thank you all. I thank Martin Hilscher and Sven Linker for inspiring my topic. My gratitude goes also to the board of examiners: Ernst-Rüdiger Olderog, the external examiner Kim Guldstrand Larsen, chairman Martin Georg Fränzle and Eike Möhlmann. I especially thank you for supporting me and seeing my disputation through during and despite the memorable chaos of the corona situation in March 2020.

For the friendly environment and nice lunch or coffee chats about anything and everything, I thank my former colleagues from the different groups working on theoretical computer science in Oldenburg: Björn Engelmann, Christian Sandmann, Christoph Peuser, Christopher Bischopink, Elke Wilkeit, Evgeny Erofeev, Hans Fleischhack, Heinrich Ody, Hendrik Radke, Mani Swaminathan, Manuel Gieseking, Martin Hilscher, Nick Würdemann, Nils-Erik Flick, Okan Özkan, Paul Hannibal, Stephanie Kemper, Sven Linker, Tim Strazny, Uli Schlachter and Valentin Spreckels. Special thanks goes to Annegret Habel for the nice mentoring pub evenings we had with our students and to Andrea Göken, Ira Wempe, Jörg Lehners, Marion Bramkamp, Mark Kettner, Nicolai Degen and Patrick Uven for your help with administrative and technical issues. Further thanks goes to my colleagues from the graduate school "Scare" for the stimulating discussions in our weekly meetings and for the inspiring annual retreats.

Last, but not least, I thank my friends and family for always supporting me and helping me throughout the tough times of the last years. I owe you much. Thank you.

Contents

List of Figures

List of Tables

1 Introduction

In a more and more mobile and autonomous world, traffic safety of autonomously driving cars is a topic of the utmost importance. To increase road safety in Europe, in 2011, the European Commission released a white paper stating their "Vision zero", which includes the goal for moving close to zero fatalities in road transport by 2050 and halving road casualties by 2020 [Com11]. However, the Annual Road Safety Performance Index (PIN) Report from the European Transport Safety Council (ETSC) from 2018 shows that the number of deathly road accidents in the EU has only fallen by 20% since 2010 [Cou18]. While 20% is already a notable progress, it was far from enough to reach the goal of halving road deaths in the EU by 2020. Thus, in 2018, the commission adapted their vision zero to now halve road deaths only by 2030 [Cou19].

However worthily such a Vision Zero is, experts in the field of autonomous driving state that official institutions like the European Commission pay only little attention to road fatalities caused by "fully autonomous cars" [SS16], meaning cars at Society of Automotive Engineers (SAE) driving level 5 [Int18]. But for those (partly) automated cars already driving on the worlds' roads, system failures are not uncommon and even human fatalities have been reported every now and then [Har, DB16]. Nonetheless, driving assistance systems and fully autonomously driving cars are increasingly capturing the market and gaining visibility for consumers.

All the more, *traffic safety* of these autonomous systems needs to be thoroughly investigated. In the context of road traffic, safety means collision freedom and thus reasoning about car dynamics and spatial properties. An example for such a spatial property could be the information that two cars are positioned one behind the other, while an example for a dynamic property is the exact position of a car after some time elapsed, in general calculated as an integral of its speed. A lot of research approaches in the field of autonomous driving use hybrid automata to handle such dynamic aspects. But these complex hybrid models are hard to reason about and to verify. Thus, two widely used approaches are to either use modular verification, meaning to verify the system component by component [MC81, KFS13], or to use abstraction techniques, meaning to separate out some parts of concern [CGL94].

An approach to separate the car dynamics from the spatial considerations and thereby to simplify reasoning, was introduced in [HLOR11] with the Multi-lane Spatial Logic (MLSL) for expressing spatial properties on multi-lane motorways with one driving direction for all cars. This logic and its dedicated abstract model was extended with length measurement in [HLO13] for country roads to reason about the distance to oncoming

traffic. The authors informally introduce respective controllers for lane change manoeuvres on motorways and country roads. These controllers use formulae of MLSL to reason about traffic situations and to decide if a car can safely change lanes.

Besides highway traffic and country roads, considerations about urban traffic scenarios are of high importance. As urban land structures are dense and individual travel routes are short, there is a rapid expansion of Mobility-on-Demand (MoD) Services, like car-sharing. Together with more and more promising research results on autonomous vehicles, a new urban traffic transportation type arises: *Autonomous Mobility-on-Demand*, where a fleet of connected autonomous cars provide travel services on-demand [IRW+18]. Also foresight institutions claim that the city-of-tomorrow will be a hypermobile city [Hei16]. In these urban traffic scenarios, managing lane intersections are especially critical as several cars may enter them from various directions and fatal intersection related accidents account for more than 20% during the last decade [ERSO12].

Thus, we focus on safety at such urban traffic lane intersections in this thesis, whereby a major part of this thesis is to develop an extension of MLSL to deal with urban traffic scenarios, focusing on desirable properties of controllers for crossing manoeuvres at intersections. We give a detailed overview of our contribution in Sect. 1.2. Before that, we start with an introduction to the existing related work for the MLSL approach in the next section.

1.1 Overview of the Research Field of MLSL

The approach of logical reasoning about autonomous car manoeuvres with MLSL originates from the group of E.-R. Olderog from University of Oldenburg. Table 1.1 gives an overview of the different MLSL contributions, sorted by type of traffic that is considered and the level of knowledge the autonomous cars and their respective controllers have, e.g. about their surroundings. For reasons of clarity and comprehensibility, only topics directly related to the original MLSL from [HLOR11] are included into Table 1.1. Nevertheless, other significant approaches using MLSL in some way, we also cover at the end of this section. Besides the works covered in the following paragraphs, several masters and bachelors theses related to MLSL exist, which we do not mention in this section, but in the respective chapters and sections they relate to.

A recap about the different approaches with (extensions of) MLSL is also provided by Olderog in [Old18]. We summarise the different approaches in the following and refer to the respective works for more details.

As mentioned before, [HLOR11] is the paper, where the approach with the spatial traffic logic MLSL for multi-lane highway traffic manoeuvres is introduced first by Hilscher, Linker, Olderog and Ravn. Besides the logic MLSL itself, both a controller for a concept of perfect knowledge and a controller with less knowledge are introduced. While the

	Basic Cases	Extensions	Implementations
Highway Traffic	[HLOR11] [Sch14]	[FHO15], [Lin15] [Ody15,17,19]	[Lin17a,17b]
Country Roads	[HLO13]		

Table 1.1: Overview of the existing MLSL approaches.

controllers were introduced informally in [HLOR11], we provided them with a semantic structure in our master's thesis [Sch14].

While in [HLOR11] only one-way highway traffic is considered, in [HLO13] the authors extend their approach to reason about overtaking manoeuvres on country roads with oncoming traffic. As overtaking cars need to measure the distance to an oncoming car on their overtaking lane, the core contribution of [HLO13] is to extend MLSL by distance measurement to formalise an overtaking protocol.

In [Lin15], Linker shows that the spatial fragment of the logic MLSL is *undecidable* and in [Ody15], Ody proves that even *robust* satisfiability of MLSL is undecidable. In this case robust means that values are only known approximately. Fortunately, in [FHO15], Fränzle, Hansen and Ody prove that MLSL is *decidable*, when considering only a bounded number of 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. In [Ody17, Ody19], the author considers *monitoring* of traffic situations, where also the abstract model and bounded version of MLSL from [FHO15] is used. For single sequences of traffic snapshots it is automatically checked if an MLSL formula holds globally throughout the sequence, which is again a statement into the direction of robustness of the logic MLSL.

The first computer-based approach for reasoning with a new hybrid extension of MLSL (*HMLSL*) was introduced by Linker [Lin17b]. In this case, *hybrid* means that concepts of *Hybrid Logic* [Brä11] and universal modalities are introduced to MLSL. The modality @$_c$ from hybrid logic is used within Hybrid MLSL to be able to switch the car from whose perspective a traffic situation is considered. The author successfully investigates safety constraints for the motorway traffic scenarios from [HLOR11] with Isabelle/HOL [NPW02]. Details for the logic HMLSL and the implementation embedded into Isabelle/HOL can be found in [Lin17a].

1.2 Contribution of this Thesis

We extend the existing model and logic MLSL to urban traffic scenarios with complex *multi-lane intersections* and construct distributed *crossing controllers* with different *types of knowledge* which we prove to operate safely. Besides safety, we also examine

other *controller properties* (i.e. (bounded) liveness and fairness) and conduct a hazard warning case study to emphasise the versatility of our approach.

Our approach for urban traffic focuses on intersections. Since the purely spatial reasoning with MLSL is very convenient for verification, we reuse the approaches of [HLOR11] and [HLO13] by extending them to our urban traffic manoeuvres. However, MLSL is defined on infinite and parallel highway or country road lanes, whereas in urban traffic, lanes intersect. We thus introduce a complex urban road network, consisting of intersections connected by several finite lane segments. To enable reasoning with our new urban extension of MLSL over such a road network, we introduce a *procedure for flattening* finite parts of the urban road network.

Only after that, we introduce an extended timed automata syntax and semantics for MLSL traffic controllers and construct crossing controllers for turn manoeuvres at intersections. We first introduce a controller with a concept of *perfect knowledge*, but after that *weaken this assumption* and introduce now *communicating controllers*, capable of committing crossing manoeuvres safely with less information.

We mathematically prove the *safety* of the crossing controllers. Besides safety, other properties are desirable for autonomous cars. We examine the *(bounded) liveness* behaviour of the lane change controller from [HLOR11] and our crossing controllers with an implementation in UPPAAL [BDL04]. Here, *liveness* means that something good, i.e. a planned lane change or crossing manoeuvre, can finally be achieved under certain assumptions. *Bounded liveness* restricts the time within the good thing has to happen. We also examine *fairness* properties of the crossing controller, by introducing a fair extension of the crossing controller which ensures that no car has to wait unreasonably long in front of an intersection. We further on weaken an assumption about having completely reliable communication by introducing *probabilities for failures*.

Finally, we apply our approach in a *case study* by introducing and implementing a *hazard warning communication protocol* with our logic and controllers. We prove the correctness of our hazard warning protocol with a proof assisted by UPPAAL.

Consider Table 1.2, which extends Table 1.1 from the previous section by a new row for our urban traffic contributions. Our UPPAAL implementation of the lane change controller and the hazard warning case study are added and highlighted in the first row. We describe the depicted publication abbreviations later in Sect. 1.5.

	Basic Cases	Extensions	Implementations
Highway Traffic	[HLOR11] [Sch14]	[FHO15], [Lin15] [Ody15,17,19], [OS17]	[Lin17a,17b] [Sch18b]
Country Roads	[HLO13]		
Urban Traffic	[HS16, Sch18a]	[Sch17]	[BS19]

Table 1.2: Weaving in own contribution into the MLSL approaches.

1.3 Related Work

In this section, we give an overview of related work regarding the overall content of the thesis. We give dedicated related work for the specific content of the following chapters exactly there in the respective chapters and sections. We split-up this related work section into two parts:

- Related formalisms for urban traffic scenarios at intersections, including works on intelligent transportation systems,
- Related approaches for non-urban automotive traffic scenarios and also related approaches for other autonomous domains (e.g. aeroplanes) as results of other domains can often be applied to the automotive domain,

Related formalisms for autonomous urban traffic at intersections.

Several promising approaches to safety of autonomous cars in urban traffic have been brought forwards in the course of the DARPA Urban Challenge. However, a great weakness of even the most successful DARPA entries is that most of them use algorithms solely applicable for the specific road map of the DARPA Urban Challenge, which is not applicable for real world settings.

To overcome this weakness, several follow-up projects arose from successful DARPA Challenge candidates. E.g. the team behind second placed 'Junior' [Mea08] in DARPA Grand Challenge 2007 summarise their recent research towards safe and robust autonomous manoeuvres in more realistic traffic situations in [LAB+11]. Instead of just using the DARPA map, Junior now generates high resolution maps of its surroundings using an automatically calibrating lidar sensor. Improved perception and image recognition algorithms allow Junior to recognise the type of other traffic participants (e.g. cyclists, pedestrians, other cars) and to dynamically plan its route. However, in unexpected events, manual control is needed to preserve safety.

A notable German follow-up project from DARPA Grand Challenge 2007 finalist 'Caroline' [WF08] is TU Braunschweig's *Stadtpilot* project [Bra]. After the DARPA Grand Challenge, the team around Stadtpilot further improved their approach by developing the autonomous vehicle 'Leonie' [NHO+11]. According to the authors, 'Leonie is one of the first [vehicles] worldwide to show the ability of driving autonomously in real urban traffic scenarios [at the time the paper was published in 2011]'. In a public demonstration in 2010, Leonie drove in mixed traffic without errors for 40km within the 4 hours of the demonstration. A sound safety concept is presented. However, Leonie is built solely to drive on Braunschweig's inner city ring road with again a fixed road map. Nonetheless the result is impressive, as it is one of the few approaches on autonomous cars that already seem fit for driving on (a limited part of) a real road in 2010.

Beside the DARPA candidates, at the present day, several other labels exist under whose names autonomous cars (successfully) drive far more miles on real roads [WB18]. In

Germany, research centres of both Mercedes-Benz and BMW have been testing automated cars on real roads in challenging driving situations (e.g. morning traffic) close to Stuttgart and Munich by 2018. Besides approaches in Europe, far more autonomous cars are launched on the roads in the USA, especially in California due to less legal restrictions. To name only some of them, [WB18] reports that Google's Waymo has already run over 5 million road miles in 25 cities (e.g. in San Francisco) and even reaches maximum velocity on highways. Waymo is based on results of the previous project 'Google Driverless Car' [DSB16], which already launched some (semi-) autonomous self-driving cars onto the streets in some US states several years before. Similarly, Tesla, General Motors and several other companies can report successful tests with autonomous cars on real roads in the USA for many thousand miles [DB16, WB18]. However, collisions, and from time to time also fatalities, have been reported for several of the approaches, meaning safety is not guaranteed [DB16]. Further on, other common (non-lethal) system failures (e.g. uncomfortable speed change, failures to detect lanes) have been reported as of [DB16].

The previously described approaches concern mixed traffic, where besides autonomous cars other traffic participants are considered. As motivated in the introduction, we consider fully autonomous cars, where *all* our cars drive *autonomously*. With this, we can ensure complete safety. However, mixed traffic with both autonomous and human-driven cars is a widely researched field and closer to market introduction than fully autonomous vehicles, even though in these scenarios safety can only be approximated and the previously described system failures are quite common [DB16, LAB+11].

Mixed traffic includes cases where people or other non-autonomous objects (e.g. human-driven cars) may invade the safety envelope of autonomous cars. For this, we refer to the group of Althoff from Munich [AD14, AM16]. In [AD14], the authors verify safety of cars online with respect to uncertain inputs (e.g. sensor noise, disturbances). Additional to that, in [AM16], the authors compute an over-approximation of possible occupancies of traffic participants over time. With this, one can formally prove, whether an automated car can possibly collide with other traffic participants in the near future.

An approach for coordinating vehicles in urban traffic that avoid collisions while moving towards their driving goals is presented in [BTK07]. The core of the work are planning methods for a dynamic communication network with real-time assumptions. The authors prove and simulate safety of their system. However, the trajectories chosen by their dynamically moving cars do not strictly follow predefined lanes and often use large detours to avoid trajectories of other cars.

A wide range of *intelligent transportation systems* (ITS) [FJM+01] were introduced to increase safety, security and comfort of autonomous driving during the last years. In these systems often centralised scheduling mechanisms are employed to improve vehicular safety and efficiency. These systems are especially often applied in urban traffic, as the density of road-side units, acting as schedulers, is far higher in urban environments than in rural areas. We present some selected approaches from the ITS community in

the following paragraphs. The huge difference of all these approaches is that our self-organising MLSL controllers decentrally plan and negotiate traffic manoeuvres and that no central scheduler is used.

Further on, in [LP11] an approach for simple intersections of single lanes with one car on each lane is proposed. The authors use traffic lights as a central control mechanism, where a car is not permitted to enter an intersection when the light is red. Though limited to single lanes with only one car on each lane, the strong point of this work is that the authors verify the safety of their hybrid systems with the tool KeYmaera.

A different approach to intelligent traffic lights as a centralised scheduler is presented in [EHK+17]. There, the authors propose to use the tool UPPAAL Stratego [DJL+15], which combines machine learning and the model checking techniques of basic UPPAAL, to minimise waiting times, energy consumption and CO_2 emissions at intersections. In comparison to the approach with MLSL, the latter approach focuses on traffic flow optimisation at intersections and not on safety aspects of self-organised traffic manoeuvres.

Related work for other autonomous traffic scenarios and other domains.

Amongst the vast amount of different approaches for autonomous traffic manoeuvres, one of the approaches possibly soonest to be released for market introduction is the *platooning* concept, in general applied to highway traffic scenarios. Two international projects solely focusing on platooning are the California PATH project [LGS98] and the European project SARTRE [LUS12]. In the platooning approach one leader vehicle exists for each platoon, which is in general not-autonomous. Numerous cars can autonomously follow this steady driving leader with minimal space between the vehicles, reducing both space and fuel waste on highways. As much advanced as platooning might be, thorough certification of correct functionality and safety of platooning is still an issue.

A concept of platooning may also be introduced to urban traffic as shown in [NB04, LW06]. With that, cars may simultaneously pass an intersection. In [NB04], cars build a *virtual platoon* via communication. Safety is guaranteed and aspects like efficiency, environment protection and comfort are taken into account. Possible conflicts between these objectives are identified and an optimal compromise is suggested. In [LW06], a *Virtual Traffic Light (VTL)* concept is introduced, where one of the vehicles approaching an intersection is cooperatively selected to be the *VTL leader*. This VTL leader is then responsible for the trajectory planning of a virtual platoon through the intersection.

In [DMPR18], the authors introduce a formal semantics for *Traffic Sequence Charts* (TSCs), which provide a visual specification language for describing first-order logic predicates for traffic situations. As TSCs are based on *Life sequence charts* (LSCs) [DH01], they are designed to specify the dynamic evolution of traffic situations. An *oracle* is used to take the future evolution of traffic into account for movement decisions. They also include dynamic models of other traffic participants and present cars as non-linear hybrid automata. For now, perfect knowledge is assumed and cooperating cars are not

considered. For future work, simulation runs are planned to check TSCs for consistency and completeness. Amongst others, a strength of TSCs are their self-explaining visuals. However, TCSs are a young and new topic and hitherto the authors abstract from several aspects.

In the MLSL approaches, we do not consider *how* an autonomous car plans its route and e.g. selects a fitting gap on a neighbouring lane to change into. Instead, our controllers randomly pick a lane to change to and also do not accelerate or brake to find a fitting gap, which would surely optimise the waiting times for changing lanes for our cars, but would again mean reasoning on a dynamic level. However, in [CSB+17], the authors introduce and evaluate an approach for gap selection with small inter-vehicle gaps in dense traffic situations, to which we refer to. There the authors calculate possible longitudinal and lateral trajectories for changing lanes on a highway into smaller gaps on adjacent lanes. The authors demonstrate safety and robustness of their controller by simulation.

Focusing on a different transport domain, in [CRT09] the authors extend linear-time temporal logic with regular expressions and hybrid aspects for formalisation and validation of specifications of the European train control system (ETCS) [Gro02]. There e.g. timely braking manoeuvres are safety-critical. Additionally, in [MFHR08], duration calculus safety specifications for the ETCS system are verified.

Furthermore, an approach on the verification of cooperating traffic agents is introduced in [DHO06]. This approach is applicable to avionics (cf. traffic alert and collision avoidance system (TCAS) [LLL00]), train applications (cf. ETCS) or even automotive applications (cf. platooning [LGS98, LUS12]). However, the difference of these approaches from our urban traffic scenario is the intersection problem: In the mentioned systems the traffic is considered one-dimensional. In this context, "one-dimensional" means that intersecting travelling routes are not considered. For instance, in avionics, planes use distinct flight tunnels and probable flight adjustments due to collision risks are not negotiated spontaneously but planned far ahead in time. In train applications, only the distance to a train or obstacle in front is of interest, as trains cannot evade sideways and railway crossings are centrally controlled.

1.4 Structure of this Thesis

As first part of this thesis, in Chapter 2, we give the preliminaries for our work. We introduce the MLSL for highway traffic and briefly recapitulate knowledge about the well-known concept of timed automata. We further on give an overview of the lane change controllers for highway traffic and country roads from [HLOR11] and [HLO13].

In Chapter 3, we introduce our abstract model for the urban traffic networks of intersections and explain how real traffic situations are abstractly captured in our model. We also introduce our flattening process of the complex graph topology and introduce how to reason with our logic Urban Multi-lane Spatial Logic in this topology.

The following Chapter 4 introduces our automotive-controlling timed automata (ACTA) we use for our various car controllers. Besides syntax and semantics of these ACTA, we also introduce the synchronisation procedure of these automata via broadcast communication channels with data types and introduce networks of ACTA.

Subsequently in Chapter 5, we explain the interplay of different types of discrete and dynamic controllers in one autonomous car and construct our crossing controllers, which we prove to be safe, live and fair in Chapter 6. We introduce both a crossing controller for a concept of perfect knowledge and one with less knowledge that needs to communicate to cope with the imperfect knowledge in Chapter 5.

In Chapter 7 we introduce our case study on a hazard warning communication protocol. There we first explain the adaptations of our MLSL approach to cope with hazards, construct our hazard warning controllers and finally prove their correct behaviour with a proof assisted by UPPAAL.

We finish this thesis with a short summary of our work in Chapter 8, followed by an informal evaluation of our approach, a brief overview of recent work and an outlook to possible future work directions.

1.5 Sources

The very heart of this thesis goes back to the content about an abstract model for crossing manoeuvres introduced in [Sch18a], which is an extension and adaptation of the conference paper [HS16]. Besides the crossing controller and its safety proof, the abstract model for complex urban networks and the process of gaining virtual views to reason about traffic situations is the central part of [Sch18a] (cf. Chapter 3 and Sects. 5.3 and 6.3). Whereas in [HS16] the abstract model for urban traffic was defined hard-coded specifically for 2-by-2 intersections, in [Sch18a] we generalise that approach to cope with intersections where any numbers of lanes meet. As the latter approach is more powerful and universal, we omit details for the hard-coded case throughout the thesis, but refer to [HS16] for details on that.

Mainly focusing on the controllers, [Sch17] introduces a new communicating extension of the controller concept from [Sch18a], capable of crossing manoeuvres with less knowledge (cf. Sect. 5.4). Additionally, in [Sch18b] and [BS19] implementation approaches using UPPAAL are provided to examine certain desirable system properties of the controllers: Safety, (bounded) liveness and fairness (cf. Chapter 6).

Finally, in [OS17] the case study is introduced, where we apply the MLSL approach to a hazard warning communication protocol (cf. Chapter 7). The semantics of the controllers used in all previously mentioned publications bases on the work in [Sch14] (cf. Chapter 4). Only as an outlook on recent work, we consider explainability of cyber-physical systems in [BGG$^+$19] (cf. Sect. 8.3).

2 Preliminaries

In this chapter, we introduce the preliminaries for our work. For this, we start in Sect. 2.1 with a brief introduction to the Z notation from [WD96] which we use throughout this thesis to formalise concepts. In Sect. 2.2, we give an overview of the abstract model and logic MLSL for highway traffic from [HLOR11]. In Sect. 2.3, we briefly discuss extended timed automata, which serve as a base for the automotive-controlling timed automata that we introduce later in this thesis. In Sect. 2.4, we introduce the lane change controller for highway traffic from [HLOR11] and we also informally introduce the overtaking protocol for country roads from [HLO13].

2.1 Z Specification Language

The Z notation is a formal specification language which is based on set theory and mathematical logic. It is thoroughly introduced and examined in [WD96]. As a reference manual we also recommend [Spi89]. In this section, we only briefly introduce those Z concepts that we actually use within this thesis and refer to the named books for details.

First-order logic, sets and general notation. Using Z, the notation of expressions and formulae is slightly different than commonly used. E.g. if we translate the first-order logic formula

$$\exists n \in \mathbb{N},\, i \in \{1, \ldots, n\} :\; n > 10 \land n > i$$

into Z notation, we write

$$\exists n \colon \mathbb{N};\; i \colon \{1 \,..\, n\} \bullet n > 10 \land n > i.$$

For abbreviations we write *symbol* == *term*, where *symbol* is a new name for the expression *term* that is defined elsewhere. For a *power set* of some set X, we write $\mathbb{P}\, X$ in Z notation instead of $\mathcal{P}(X)$.

A well-known mathematical construct for the separation of specific parts of a set M is

$$\{x \colon M \mid F\},$$

where those elements x are separated, for which the predicate logic formula F holds.

Z offers a unique additional method to separate specific elements from one set:

$$\{x : M \mid F \bullet exp\}.$$

Here, first those elements x are selected, for which F holds. Then, the given expression *exp* transforms the elements x into other elements. For examples for this notation, we refer to the next paragraph, where it is used to denote specific operations on relations.

Relations and functions. A relation R between the sets X and Y is denoted by $R : X \leftrightarrow Y$. For $x \in X$ and $y \in Y$, we may write $x \mapsto y$ if $(x, y) \in R$. The *domain* of such a relation $R : X \leftrightarrow Y$ is the set of elements in X related to some elements in Y:

$$\mathsf{dom}\, R = \{x : X; y : Y \mid x \mapsto y \in R \bullet x\}.$$

On the other hand, the *range* of R is the set of elements of Y to which some element of X is related:

$$\mathsf{ran}\, R = \{x : X; y : Y \mid x \mapsto y \in R \bullet y\}.$$

Sometimes, it is desirable to consider only a part of a function. For this, both the domain and the range may be *restricted* or *subtracted*. For a subset A of the set X, we denote the *domain restriction* of R to A by $A \lhd R$ which is defined by

$$A \lhd R = \{x : X; y : Y \mid x \mapsto y \in R \land x \in A \bullet x \mapsto y\}.$$

The concepts of *range restriction* as well as *domain* and *range subtraction* are defined analogously but left out here as we do not use them within this thesis.

For defining a *partial finite function* f from a set X to a set Y we write $f : X \pfun Y$. Such a function f maps each element of a finite subset of X to at most one element of Y. Further on, for a *total bijective function* g from X to Y, we write $g : X \bij Y$. The function g maps each element of X exactly to one unique element of Y, meaning g is both injective and surjective.

Further on, we use the *overriding operator* $\oplus$, where for a function $f : X \to Y$ with an expression $f \oplus \{x_1 \mapsto y_1\}$ the value of x_1 is updated to y_1. All other parts of the function remain the same.

Sequences. A *sequence* s is an ordered collection of objects and is formally defined as a partial finite function defined on the natural numbers and ranging over some set X. The set of all finite sequences of elements from a given set X is denoted by:

$$\mathsf{seq}\, X == \{s : \mathbb{N} \pfun X \mid \exists n : \mathbb{N} \bullet \mathsf{dom}\, s = 1 \,..\, n\}.$$

A sequence s consisting of elements a, b, c is written as $s = \langle a, b, c \rangle$. It stands for the function $s = \{1 \mapsto a, 2 \mapsto b, 3 \mapsto c\}$ from indices $1, 2, 3$ to elements a, b, c. Thus the i-th element of s is obtained by the function application $s(i)$, e.g., $s(2) = b$. The *length* of s is denoted by $\#s$, here $\#s = 3$. For the empty sequence $\langle\rangle$ the length is 0.

Sequences may be composed using the *sequence concatenation* operator $\frown$. For instance, we can write $\langle a, b, c\rangle \frown \langle c, d\rangle = \langle a, b, c, c, d\rangle$. As and abbreviation, we use $\langle d_1\rangle = d_1$ for a sequence with only one element.

Further on, the function $\mathsf{head}\ s$ returns the first element of a non-empty sequence s while the function $\mathsf{tail}\ s$ returns the part that follows the first element of s, such that $s = \langle \mathsf{head}\ s\rangle \frown \mathsf{tail}\ s$. To this notation we add the function $\mathsf{second}\ s$, which returns the second element of s for $|s| > 1$. With $s' == \mathsf{tail}\ s$ and $\mathsf{second}\ s = \mathsf{head}\ s'$, this leads to $s = \langle \mathsf{head}\ s\rangle \frown \langle \mathsf{second}\ s\rangle \frown \mathsf{tail}\ s'$.

It can be of interest to compress a finite function defined upon the natural numbers to a sequence. For instance, this can be useful, if a range restriction applied to a sequence has left empty spaces within that sequence. For this, Z offers the operator squash which compacts the domain of a finite function defined upon $\mathbb{N}$ to remove any empty spaces, while preserving the order of the remaining elements. For instance, we have

$$\mathsf{squash}\,\{1 \mapsto a, 3 \mapsto c, 5 \mapsto e\} = \{1 \mapsto a, 2 \mapsto c, 3 \mapsto e\} = \langle a, c, e\rangle.$$

We use sequences defined over $\mathbb{Z}$ starting from Sect. 3.2 and also define and use an adapted version of squash over $\mathbb{Z}$ directly there (cf. Def. 15, p. 40).

2.2 Model and Logic MLSL for Highway Traffic

In general, when speaking of 'the MLSL approach', we consider the following five central concepts, which all exist in different versions in the highway traffic approach from [HLOR11], in the country roads approach from [HLO13], as well as in the urban traffic approach described in the main part of this thesis:

Abstract model: An abstracted version of real-world traffic situations. It consists e.g. of abstracted versions of driving lanes, intersections and cars.

Traffic snapshot: The 'global picture' of all traffic. One distinct traffic snapshot captures e.g. the current positions of *all* cars in the abstract model.

View: A cut-out of a traffic snapshot. This is used to consider only the local traffic around a specific car to simplify reasoning.

Logic: The spatial traffic logic MLSL (or one of its extensions). It is used to reason about the specific traffic situation captured in a view.

Controller: An automaton-style controller, implementing the lane change or turn procedures. It uses formulae of (extended) MLSL to determine whether a lane change or turn manoeuvre can be safely conducted or not.

Note that while all our three traffic approaches use these five concepts, the respective instances sometimes differ greatly. For instance, while the abstract model for highway traffic consists only of a number of infinite adjacent lanes with the same driving direction, for urban traffic, we have an abstract model consisting of a complex network of different intersections connected to some other intersections by finite lane segments.

Common to all MLSL approaches, we assume that *all* cars on the road are driving autonomously and thus do not model human drivers. This assumption is done because we do not cope with cars driving completely unpredictable and possibly even violating traffic rules. Approaches with mixed traffic in general need to include some type of probabilities, as the behaviour of unpredictable traffic participants obviously can only be guessed, not known. With this, a proof of absolute safety is no longer possible. Some basic ideas on how to introduce human drivers into the MLSL approach can be found in [Bis18]. There, the author extends our traffic controllers with probabilities for communication failures.

We now briefly introduce the above concepts up to the controller part for the first an easiest case: highway traffic (cf. [HLOR11]). For reasons of clarity, before introducing the lane change controller in Sect. 2.4, we first give an overview of extended timed automata in next Sect. 2.3 as the controller uses concepts of timed automata. We start with the abstract model for highway traffic and refer to [HLOR11] for more details.

Abstract Model for Highway Traffic

The *abstract model for highway traffic* consists of a set $\mathbb{L} = \{0, \ldots, N\}$ of infinite neighbouring *lanes* of continuous space, heading in the same direction, for some fixed $N \geq 1$. Typical elements from $\mathbb{L}$ are l, m, n. Every car has a unique *car identifier* from the set $\mathbb{I}$ with typical elements $A, B, \ldots$ 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. 2.1. 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. 2.1, we assume that E is our ego car and thus have the valuation $\nu(\text{ego}) = E$.

In the abstract model, the space that 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 car E in Fig. 2.1, showing the desire of E to change to lane 2). Thus, a claim represents the space a car plans to drive on in the future.

The car rectangles depicted in Fig. 2.1 comprise the *safety envelopes* of the cars, for now including both the *physical size* and the *braking distance* of the cars. To indicate that cars vary in size and that their braking distance also varies in size due to different speed values, the rectangles also have different sizes. By considering the braking distance within the safety envelope, each car may perform an emergency brake and still remain safe.

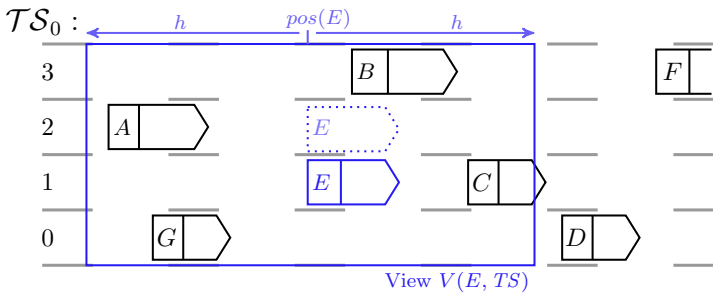

Figure 2.1: Abstract model with adjacent lanes 0 to 3 and cars with unique identifiers. The ego car E plans to change to lane 2, indicated by its dotted claim on lane 2. Cars D and F are too far away from car E to be considered in E's standard view $V(E, TS)$.

Traffic Snapshot

Static information about cars like their positions and their reserved or claimed lanes is captured in a *traffic snapshot* $\mathcal{TS}$. One distinct traffic snapshot $\mathcal{TS}$ describes the traffic on the freeway at a given point in time.

Definition 1 (Traffic snapshot [HLOR11])**.** *A global* traffic snapshot $\mathcal{TS}$ *is a structure* $\mathcal{TS} = (res, clm, pos, spd, acc)$*, where* res, clm, pos, spd *and* acc *are functions*

- $res : \mathbb{I} \to \mathbb{P}(\mathbb{L})$ *such that* $res(C)$ *is the set of lanes that car* C *reserves,*
- $clm : \mathbb{I} \to \mathbb{P}(\mathbb{L})$ *such that* $clm(C)$ *is the set of lanes that car* C *claims,*
- $pos : \mathbb{I} \to \mathbb{R}$ *such that* $pos(C)$ *is the position of the rear of car* C *on its lane,*
- $spd : \mathbb{I} \to \mathbb{R}$ *such that* $spd(C)$ *is the current speed of car* C*,*
- $acc : \mathbb{I} \to \mathbb{R}$ *such that* $acc(C)$ *is the current acceleration of car* C*.*

Example 1 (Traffic snapshot). Consider the depicted traffic situation in Fig. 2.1 as an excerpt of a traffic snapshot $\mathcal{TS}_0$. We give the respective sets for reservations and claims of the cars and their position values in $\mathcal{TS}_0$ in the following table:

Note that in Table 2.1 the position, speed and acceleration values are exemplary and possibly do not scale exactly to Fig. 2.1. An intuition behind the chosen values is that car D currently brakes (negative acceleration), while cars F and G accelerate (positive acceleration). All other cars drive constantly at their indicated speeds. △

Def. 1 does not restrict the structure of a traffic snapshot, thus including completely senseless ones. E.g., it allows for a car C to claim, and thus change to, a lane which is not neighbouring to its currently occupied lane. To exclude such insane traffic snapshots. Linker [Lin15] introduces the following notion of *sanity conditions*.

Car	A	B	C	D	E	F	G
Reservation res	$\{2\}$	$\{3\}$	$\{1\}$	$\{0\}$	$\{1\}$	$\{3\}$	$\{0\}$
Claim clm	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{2\}$	$\emptyset$	$\emptyset$
Position pos	20	83	110	135	70	165	35
Speed spd	130	135	90	95	110	150	80
Acceleration acc	0	0	0	-15	0	30	25

Table 2.1: Respective sets of claimed and reserved lanes and possible real position, speed and acceleration values for the cars visible in the excerpt of traffic snapshot $\mathcal{TS}_0$ depicted in Fig. 2.1.

Definition 2 (Sanity conditions [Lin15])**.** *A traffic snapshot $\mathcal{TS}$ is* sane*, if the following conditions hold for all cars $C \in \mathbb{I}$.*

1. $res(C) \cap clm(C) = \emptyset$
2. $0 \leq \#clm(C) \leq 1$
3. $1 \leq \#res(C) \leq 2$
4. $1 \leq \#res(C) + \#clm(C) \leq 2$
5. $\#res(C) = 2$ *implies* $\exists n : \mathbb{L} \bullet res(C) = \{n, n+1\}$
6. $clm(C) \neq \emptyset$ *implies* $\exists n : \mathbb{L} \bullet res(C) \cup clm(C) = \{n, n+1\}$.

Let $\mathbb{TS}$ denote the set of all sane traffic snapshots.

With conditions 1 and 2 and 6, it is ensured, that a car can only claim one lane at once, which has to be neighbouring to the currently reserved lane. Likewise, condition 5 states that if a car can only change to a neighbouring lane. Further on, condition 4 forbids that a car claims a lane while changing lanes and thus $\#res(C) = 2$. Finally, condition 3 ensures that there always remains a reservation for each car and thus cars do not completely disappear from the highway. This is necessary, as the lanes in the abstract model from [HLOR11] are infinite and an option to leave the highway is not provided.

To describe changes that may occur to an arbitrary traffic snapshot $\mathcal{TS}$, the following *traffic snapshot transitions* are introduced. We use the overriding notation $\oplus$ of Z for function updates. With the update $clm'(C) = clm \oplus \{C \mapsto \{n\}\}$, we e.g. update the set of claimed lane segments for car C to $\{n\}$.

Definition 3 (Traffic snapshot transitions [HLOR11])**.** *The following* transitions *describe the changes that may occur at a traffic snapshot $\mathcal{TS} = (res, clm, pos, spd, acc)$. We use the overriding notation $\oplus$ of Z for function updates (cf. Sect. 2.1).*

$$\mathcal{TS} \xrightarrow{t} \mathcal{TS}' \quad \Leftrightarrow \quad \mathcal{TS}' = (res, clm, pos', spd', acc)$$

$$\begin{aligned} & \wedge \forall C \in \mathbb{I}\colon pos'(C) = pos(C) + spd(C) \cdot t + \tfrac{1}{2} acc(C) \cdot t^2 \\ & \wedge \forall C \in \mathbb{I}\colon spd'(C) = spd(C) + acc(C) \cdot t \end{aligned} \tag{2.1}$$

$$\begin{aligned} \mathcal{TS} \xrightarrow{\mathsf{acc}(C,a)} \mathcal{TS}' \quad \Leftrightarrow \quad & \mathcal{TS}' = (res, clm, pos, spd, acc') \\ & \wedge\, acc' = acc \oplus \{C \mapsto a\} \end{aligned} \tag{2.2}$$

$$\begin{aligned} \mathcal{TS} \xrightarrow{\mathsf{c}(C,n)} \mathcal{TS}' \quad \Leftrightarrow \quad & \mathcal{TS}' = (res, clm', pos, spd, acc) \\ & \wedge\, |clm(C)| = 0 \wedge\ |res(C)| = 1 \\ & \wedge\, \{n+1, n-1\} \cap res(C) \neq \emptyset \\ & \wedge\, clm' = clm \oplus \{C \mapsto \{n\}\} \end{aligned} \tag{2.3}$$

$$\begin{aligned} \mathcal{TS} \xrightarrow{\mathsf{wd\ c}(C)} \mathcal{TS}' \quad \Leftrightarrow \quad & \mathcal{TS}' = (res, clm', pos, spd, acc) \\ & \wedge\, clm' = clm \oplus \{C \mapsto \emptyset\} \end{aligned} \tag{2.4}$$

$$\begin{aligned} \mathcal{TS} \xrightarrow{\mathsf{r}(C)} \mathcal{TS}' \quad \Leftrightarrow \quad & \mathcal{TS}' = (res', clm', pos, spd, acc) \\ & \wedge\, clm' = clm \oplus \{C \mapsto \emptyset\} \\ & \wedge\, res' = res \oplus \{C \mapsto res(C) \cup clm(C)\} \end{aligned} \tag{2.5}$$

$$\begin{aligned} \mathcal{TS} \xrightarrow{\mathsf{wd\ r}(C,n)} \mathcal{TS}' \quad \Leftrightarrow \quad & \mathcal{TS}' = (res', clm, pos, spd, acc) \\ & \wedge\, res' = res \oplus \{C \mapsto \{n\}\} \\ & \wedge\, n \in res(C) \wedge\ |res(C)| = 2 \end{aligned} \tag{2.6}$$

We start with describing transition (2.3) for claiming a lane n for car C. The side conditions for this transition e.g. state that car C is not allowed to have already claimed other lanes and further on it is checked whether the lane n is actually neighbouring to the lane car C is already driving on. The transition (2.4) for withdrawing a claim is much easier, as it only empties the set of claimed lanes $clm(C)$ of car C.

For reserving a lane the transition (2.5) is used. Here the claimed lane is inserted into the set of reserved lanes $res(C)$ and again, the set of claimed lanes $clm(C)$ of car C is emptied. For withdrawing a reservation with transition (2.6), the car C has to have two reservations. This ensures that after withdrawing a reservation, there still exists one reservation for car C.

With the time transition (2.1), the movement of cars according to their respective speed and acceleration when some time t passes is formalised. Formula (2.2) defines how a car may change its acceleration. Note that a very abstract formalisation is used here for the car dynamics, as we abstract from them (also cf. explanation in Sect. (5.1)). We still give a definition for them here, to have a defined continuous behaviour of the cars, if time passes.

At this point, we like to follow the idea of Linker, who introduced the notion of *evolution transitions* in [Lin15] by combining a finite number of time transitions (2.1) and acceleration transitions (2.2). We denote such an evolution transition by $\mathcal{TS} \rightsquigarrow \mathcal{TS}'$ and refer to [Lin15] for the formal definition of evolutions.

Example 2 (Traffic snapshot transitions). As the initial traffic snapshot for the following transition sequence, consider again the traffic snapshot $\mathcal{TS}_0$ from Fig. 2.1, where car E drives on lane 1 and has already claimed a space on lane 2. In the following, we give a transition sequence, where $\mathcal{TS}_0$ evolves to a future traffic snapshot $\mathcal{TS}_5$. The traffic snapshot $\mathcal{TS}_0$ up to $\mathcal{TS}_5$ are depicted in Fig. 2.2.

We first consider some time t_1 passes and all cars move according to their speed and acceleration. Secondly, car E reserves its previously claimed space on lane 2 and after some time t_2 passes withdraws its reservation to lane 2 solely, as it finished its lane change manoeuvre. Finally, we consider that an evolution transition occurs, where car E accelerates on its new lane 2 and after some time passes starts overtaking car C.

$$\mathcal{TS}_0 \xrightarrow{t_1} \mathcal{TS}_1 \xrightarrow{r(E)} \mathcal{TS}_2 \xrightarrow{t_2} \mathcal{TS}_3 \xrightarrow{wd\ r(E,2)} \mathcal{TS}_4 \rightsquigarrow \mathcal{TS}_5$$

We have $\mathcal{TS}_0 \Rightarrow \mathcal{TS}_5$. △

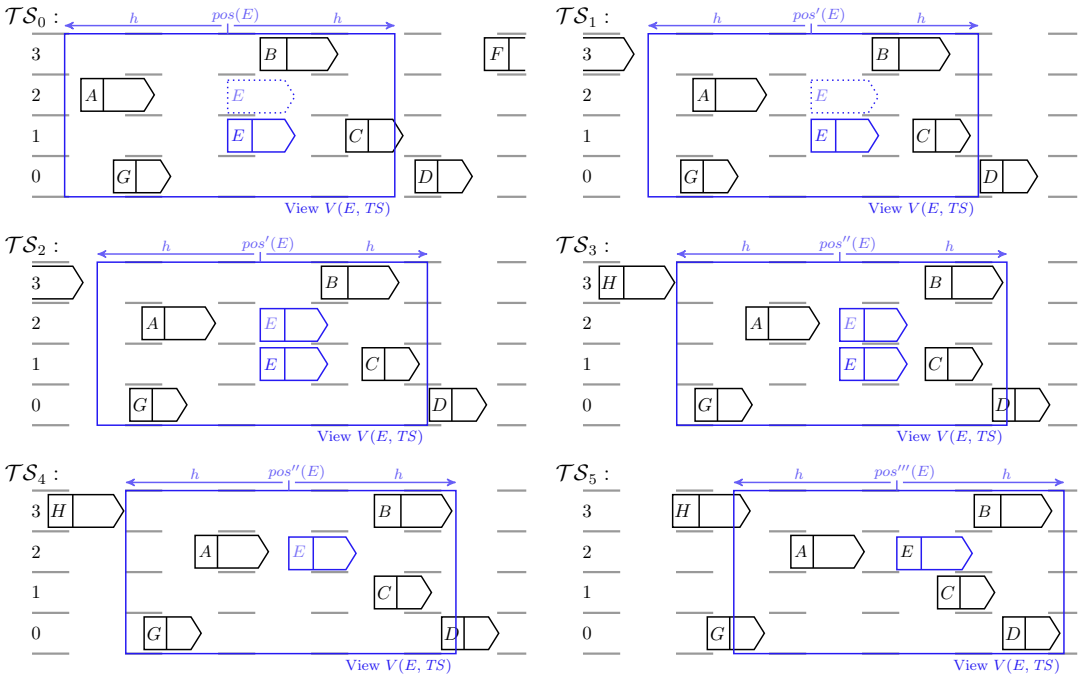

Figure 2.2: Stepwise evolution of the initial traffic snapshot $\mathcal{TS}_0$ from Fig. 2.1 to a traffic snapshot $\mathcal{TS}_5$.

The traffic snapshot transitions are well-defined, meaning that when we start from a sane traffic snapshot $\mathcal{TS}$, only sane traffic snapshots are reachable via the transitions from Def. 3. A proof for this property can be found in [Lin15].

As lanes are of infinite 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.

View

We use a local *view* for logical reasoning about traffic situations with MLSL. If a certain area around ego car E is considered in a view, we name it by the *standard view* of E. For example, in Fig. 2.1, the standard view of car E is indicated by the blue rectangle.

Definition 4 (View and standard view)**.** *For an arbitrary traffic snapshot $\mathcal{TS}$, 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 $V(E, \mathcal{TS}) = (\mathbb{L}, [pos(E) - h, pos(E) + h], E)$, where h is a sufficiently large* horizon *for looking forwards resp. backwards from the position $pos(E)$, as given in the traffic snapshot $\mathcal{TS}$.*

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 standard view $V(E, \mathcal{TS})$. In the example in Fig. 2.1, car D is not considered in V, as it is to far away from E. We use a car dependent *sensor function* $\Omega_E \colon \mathbb{I} \times \mathbb{TS} \to \mathbb{R}_+$ which, given an identifier $C \in \mathbb{I}$ and a traffic snapshot $\mathcal{TS} \in \mathbb{TS}$, provides the *size* $\Omega_E(C, \mathcal{TS})$ of C as perceived by E's sensors.

We assume a concept of *perfect knowledge* in the following: 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 perform an emergency braking manoeuvre at every moment, without causing a collision. In [HLOR11], the authors also introduce the more realistic concept of *imperfect knowledge*, where each car only knows the physical size of other cars. We do not describe the approach with imperfect knowledge for highway traffic here for reasons of brevity. However, we describe this concept of imperfect knowledge later more detailed in our assumptions for our crossing controllers in Sect. 5.2, as we introduce crossing controllers both for perfect and imperfect knowledge in the respective Sects. 5.3 and 5.4.

Abbreviation 1 (Visible elements in a view). For a view $V = (L, X, E)$ and a traffic snapshot $\mathcal{TS} = (res, clm, pos, spd, acc)$, we introduce the following abbreviations, used for the semantics definition of the logic MLSL in the next section:

$$res_V \colon \mathbb{I} \to \mathbb{P}\, L \text{ with } res_V(C) = res(C) \cap L \tag{2.7}$$

$$clm_V \colon \mathbb{I} \to \mathbb{P}\, L \text{ with } clm_V(C) = clm(C) \cap L \tag{2.8}$$

$$len_V \colon \mathbb{I} \to \mathbb{P}\, L \text{ with } len_V(C) = [pos(C), pos(C) + \Omega_E(C, \mathcal{TS})] \cap X \tag{2.9}$$

△

The functions (2.7) and (2.8) restrict their counterparts $res(C)$ and $clm(C)$ from $\mathcal{TS}$ to the set of lanes considered in V. Function (2.9) defines the part of car C that E perceives with its sensors in the extension X of the considered view V.

Multi-lane Spatial Logic

Multi-lane Spatial Logic (MLSL) is inspired by *Interval Temporal Logic* (ITL) [Mos85] and *Duration Calculus* (DC) [CHR91], which both allow for one-dimensional reasoning by specification of temporal intervals. With these intervals, one can e.g. specify system states happening one after the other. Further on, *Shape Calculus* [Sch05] extends Duration Calculus by more dimensions as it allows an arbitrary amount of spatial and temporal dimensions.

Taking up this idea of a multi-dimensional logic, MLSL and its derivatives, including the urban logic proposed in this thesis, extend ITL and DC by a second dimension, whilst considering continuous (positions on lanes) and discrete components (the number of a lane). With these two-dimensional features we can, for instance, express that a car is occupying a certain space on a lane.

However, all versions of MLSL are not comparable to Shape Calculus, as they are created for a specific field of application: motorway traffic, country roads, or urban traffic. To this end, MLSL allows for quantification over cars, which is not implementable in Shape Calculus. Therefore, a reduction of MLSL to a decidable subset of Shape Calculus is not possible.

Syntax of MLSL. With MLSL, we can reason about traffic situations in our local view $V = (L, X, E)$. We distinguish *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 ego $\in$ CVar and the set of all variables Var = CVar $\cup$ LVar has typical elements u, v. We now introduce the *valuation function* ν for car and lane variables.

Definition 5 (Valuation of variables)**.** *A* valuation ν *is a function* $\nu\colon \text{Var} \to \mathbb{I} \cup \mathbb{L}$*, where* Var = CVar $\cup$ LVar *and* $\nu\colon \text{CVar} \to \mathbb{I}$ *and* $\nu\colon \text{LVar} \to \mathbb{L}$*.*

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 $u, v, \in$ Var of the same type.

We utilise concepts of interval temporal logic (ITL) [Mos85] for the spatial interval-like arrangement of MLSL formulae. In ITL, the term $\sigma \mapsto (P = 0)$ states that within the discrete time interval σ the signal P has the value 0. ITL allows for length specification of such a temporal interval, where e.g. the formula $[len > n \wedge (P = 0)]$ states that P has the value 0 in an interval of time greater than n. One can also divide time intervals into adjacent subintervals in ITL. Here, $[(P = 0); skip; (P = 1)]$ evaluates to true, iff there exists an interval, where P is 0 for a while and then jumps to 1.

We use this ITL chop operator for MLSL, by defining a *horizontal chop operator*, denoted by $\frown$ instead of ; for interval temporal logic, and a *vertical chop operator* given by the

vertical arrangement of formulae. Please note that the operator $\frown$ as used in MLSL formulae, has a different meaning as the Z sequence concatenation operator $\frown$ (cf. p. 13), which we frequently use later for our urban traffic in Sects. 3.2 and 3.4. 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 the formula 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 $i = 1, 2$.

Definition 6 (Syntax of MLSL)**.** *The syntax of a* Multi-lane Spatial Logic *formula ϕ_M is defined by*

$$\phi_M ::= \mathit{true} \mid u = v \mid \mathit{free} \mid \mathit{re}(c) \mid \mathit{cl}(c) \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \exists c\colon \phi_1 \mid \phi_1 \frown \phi_2 \mid \begin{smallmatrix}\phi_2\\ \phi_1\end{smallmatrix},$$

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

Note that in MLSL the type of the variable c in $\exists c\colon \phi_1$ is fixed: c ranges over the set $\mathbb{I}$ of car identifiers.

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

Definition 7 (Semantics of MLSL)**.** *The* satisfaction *of MLSL formulae φ with respect to a traffic snapshot $\mathcal{TS}$, 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:*

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

In this thesis we often use the abbreviation $\langle\phi\rangle$ to state that a formula ϕ holds *somewhere* in the considered view of car E. This somewhere modality is used to abstract from exact positions in MLSL formulae. Note that in MLSL formulae, $\langle$ and $\rangle$ are not to be confused with the Z sequence delimiters $\langle$ and $\rangle$ we use later in Sects. 3.2 and 3.4.

Abbreviation 2 (Somewhere modality). For a formula $\phi \in \Phi_{\mathbb{M}}$ we write

$$\langle\phi\rangle \equiv \mathit{true} \frown \begin{pmatrix} \mathit{true} \\ \phi \\ \mathit{true} \end{pmatrix} \frown \mathit{true},$$

where $\langle\,\phi\,\rangle$ distributes over disjunction by Def. 7:

$$\langle\phi_1 \vee \phi_2\rangle \equiv \langle\phi_1\rangle \vee \langle\phi_2\rangle.$$

△

Example 3 (MLSL formulae and somewhere modality). Consider Fig. 2.1 and assume a valuation of variables $\nu(\text{ego}) = E$, $\nu(a) = A$, $\nu(c) = C$, $\nu(d) = D$ and $\nu(g) = G$. Consider the following MLSL formulae:

$$\varphi_1 \equiv \langle \mathit{re}(\text{ego}) \frown \mathit{free}\rangle$$
$$\varphi_2 \equiv \langle \begin{pmatrix} \mathit{cl}(\mathit{ego}) \\ \mathit{re}(\mathit{ego}) \end{pmatrix} \frown \mathit{free} \frown \mathit{re}(c)\rangle$$
$$\varphi_3 \equiv \langle \mathit{cl}(g) \frown \mathit{free} \frown \mathit{re}(d)\rangle$$

In view $V(E, \mathcal{TS})$ the formula φ_1 holds, as there is free space in front of car E, meaning $\mathcal{TS}, V(E, \mathcal{TS}), \nu \models \varphi_1$. However, φ_2 does not hold, as even while there is free space both in front of the claim and the reservation of car E, the reservation of car C is only in front of E's reservation, not in front of its claim. Thus $\mathcal{TS}, V(E, \mathcal{TS}), \nu \not\models \varphi_2$. Likewise, $\mathcal{TS}, V(E, \mathcal{TS}), \nu \not\models \varphi_3$, as car D is not part of view $V(E, \mathcal{TS})$. △

Before introducing the lane change controller for highway traffic from [HLOR11], we introduce extended timed automata which may serve as its semantic basis.

2.3 Extended Timed Automata

As a formal semantics for all controllers introduced within this thesis, we introduce automotive-controlling timed automata (ACTA) later in Chapter 4. ACTA are an extension of the well-known concept of *timed automata* from [AD94], which we introduce in this section. We also use some concepts of the *extended UPPAAL timed automata* [BDL04] in Chapter 4 and thus also briefly introduce them here. As a worthwhile introduction

to timed automata and further formalisms from the topic of real-time systems, we also recommend the book [OD08], which inspired some of the definitions in this section.

Notation. Before giving more details on (extended) timed automata, we delimit our meaning of the following three terms that are frequently used throughout our thesis and that are confusion-prone:

Location: The specific (syntactical) location an (extended) timed automaton is in.

State: The (semantical) system state of an (extended) timed automaton, meaning one specific configuration of the automaton during a run.

Phase: A phase a traffic controller is in, e.g. the 'crossing ahead phase'.

A main advantage of timed automata is that they are capable of representing timing behaviour. This is done with *clock variables* x, y from the set of all clock variables $\mathbb{X}$ which range over a continuous *time dimension Time*. For these clock variables, we define *clock constraints* $\varphi_{\mathbb{X}}$.

Definition 8 (Clock constraints)**.** *For clock variables* $x, y \in \mathbb{X}$*, the set* $\Phi_{\mathbb{X}}$ *of all* clock constraints $\varphi_{\mathbb{X}}$ *is defined by*

$$\varphi_{\mathbb{X}} ::= x \sim c \mid x - y \sim c \mid \neg\varphi_{\mathbb{X}} \mid \varphi_{\mathbb{X},1} \wedge \varphi_{\mathbb{X},2},$$

where $c \in \mathbb{Q}_{\geq 0}$ *and* $\sim \in \{<, >, \leq, \geq\}$*. Constraints of the type* $x - y \sim c$ *are named clock differences.*

An example for a clock constraint $\varphi_{\mathbb{X}}$ is $x > 10$, where the clock x is compared with the value 10. While clock constraints are used in *guards* and *invariants* of timed automata, *clock resets* like $x := 0$ are used to reset a clock variable x to 0.

Additional to the clock variables from [AD94], UPPAAL introduces *data variables* with respective *data constraints* and *data updates*, which we also use for our ACTA. As these UPPAAL data constraints and updates are defined analogously to the previous clock constraints and resets, we omit a definition for them at this point and introduce our own usage of data variables and data constraints for ACTA directly later in Chapter 4.

For the *synchronisation of automata*, there exist two widely used concepts. The first one is the so-called *handshake communication*, where automata communicate in a one-to-one fashion directly with each other. The sender has to synchronise with the receiving automaton and thus has to wait until the receiving automaton can synchronise with it. Such a handshake synchronisation concept is e.g. described by Milner in [Mil82] for his *Calculus of Communicating Systems*. The second one is the *broadcast communication* UPPAAL implements that we use later for our automotive-controlling timed automata in Chapter 4. With broadcast communication, one sending automaton can synchronise with none or an arbitrary number of receiving automata. Broadcast communication is an extended version of handshake communication. Thus, we decide to only define the

simpler handshake communication as of [Mil82] in this preliminary section as an easier introduction to the concept of synchronisation of automata via channels. We introduce the broadcast communication concept later in the respective Sect. 4.3 on communication of ACTA in Chapter 4.

Definition 9 (Channels, actions and alphabets)**.** *For a channel a from the set of all channels $Chan$ there exist two visible actions:*

- $a?$ *is an* input action *and*
- $a!$ *is its respective* output action*,*

where $a?, a! \notin Chan$. Further on there exists the internal action τ*, with $\tau \notin Chan$. The set* Act *of all actions α, β is defined by*

$$Act = \{a? \mid a \in Chan\} \cup \{a! \mid a \in Chan\} \cup \{\tau\}. \tag{2.10}$$

An alphabet B *is a set of channels with $B \subseteq Chan$. To each alphabet B there exists a set of actions $B_{?!}$, where*

$$B_{?!} = \{a? \mid a \in B\} \cup \{a! \mid a \in B\} \cup \{\tau\}. \tag{2.11}$$

For $B_{?!}$ we observe $B_{?!} \subseteq Act = Chan_{?!}$.

With this, we now define extended timed automata from [AD94], merged with the broadcast communication concept of UPPAAL timed automata [BDL04].

Definition 10 (Extended timed automaton)**.** *An* extended timed automaton $\mathcal{A}$ *is a tuple $\mathcal{A} = (L, B, \mathbb{X}, \mathcal{I}, E, l_{ini})$, where*

- L *is a finite set of locations l,*
- $B \subseteq Chan$ *is a finite set of channels,*
- $\mathbb{X}$ *is a finite set of clock variables,*
- $\mathcal{I} : L \to \Phi(\mathbb{X})$ *is an invariant, assigning a clock constraint to each location,*
- $E \subseteq L \times B_{?!} \times \Phi(\mathbb{X}) \times \mathcal{P}(\mathbb{X}) \times L$ *is the set of all directed edges and*
- $l_{ini} \in L$ *is the initial location.*

An element $(l, \alpha, \varphi, Y, l') \in E$ describes a transition from location l to l', labelled with an action α, a guard φ and a set $Y \in \mathbb{X}$ of clock variables. All clocks in Y are reset when taking this transition.

Example 4 (Traffic lights). In Fig. 2.3, we present an example for an extended timed automaton $\mathcal{A}_P$ describing the possible behaviour a pedestrian traffic light. Initially, the traffic light is in location l_0 showing green for pedestrians for 10 time units. After that, the automaton changes to location l_1 showing red for pedestrians, while informing whoever may listen via broadcast channel *stop* about this change. After 15 time units, the automaton changes back to the initial location l_0, only if a respective message was received via channel *go*. △

$\mathcal{A}_P$:

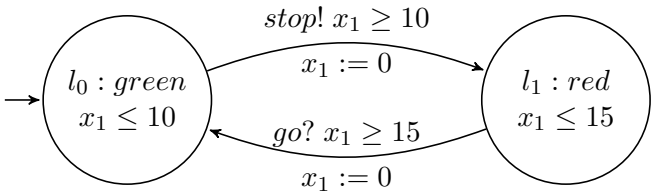

Figure 2.3: Extended timed automaton $\mathcal{A}_P$ for a pedestrian light using channels *stop* and *go*.

For defining the semantics of extended timed automata, we first introduce the following *valuation function for clock variables*.

Definition 11 (Clock valuation)**.** *Consider a continuous time dimension Time. A* valuation ν *of clock variables* $x \in \mathbb{X}$ *is a mapping*

$$\nu : \mathbb{X} \rightarrow Time.$$

For a clock constraint $\varphi_{\mathbb{X}}$ *(cf. Def. 8), we have*

$$\nu \models \varphi_{\mathbb{X}},$$

where this satisfaction of clock constraints $\varphi_{\mathbb{X}}$ *is inductively defined by*

$$\begin{aligned} \nu \models x \sim c \quad & \textit{iff} \quad \nu(x) \sim c, \\ \nu \models x - y \sim c \quad & \textit{iff} \quad \nu(x) - \nu(y) \sim c, \\ \nu \models \varphi_{\mathbb{X}1} \wedge \varphi_{\mathbb{X}2} \quad & \textit{iff} \quad \nu \models \varphi_{\mathbb{X}1} \textit{ and } \nu \models \varphi_{\mathbb{X}2}, \end{aligned}$$

with $\sim \in \{=, \neq, <, >, \leq, \geq\}$.

Further on, we observe two possible operations for changing the value of clock variables:

Time shift: The valuation ν is increased by some $t \in Time$, denoted by $\nu + t$, as follows:

$$(\nu + t)(x) = \nu(x) + t$$

for all clock variables $x \in \mathbb{X}$.

Modification: The valuation ν of a subset of clock variables $x \in Y$ with $Y \subseteq \mathbb{X}$ is set to a value $t \in Time$, denoted by $\nu[Y := t]$, which is defined as follows:

$$\nu[Y := t](x) = \begin{cases} t & \text{, if } x \in Y, \\ \nu(x) & \text{, else} \end{cases}$$

for all clock variables $x \in \mathbb{X}$.

Note that in timed automata, clock variables are generally reset to 0 and other variables are not existent in the timed automata as of Def. 10. We give the operational semantics of extended timed automata in the following.

Definition 12 (Operational semantics of an extended timed automaton)**.** *The operational semantics of an extended timed automaton $\mathcal{A} = (L, B, \mathbb{X}, \mathcal{I}, E, l_{ini})$ is defined by a labelled transition system*

$$\mathcal{T}(\mathcal{A}) = (Conf(\mathcal{A}), B_{?!} \cup \mathbb{T}, \{\xrightarrow{\lambda} \mid \lambda \in B_{?!} \cup Time\}, C_{ini}), \tag{2.12}$$

where

- *the set of all* configurations $Conf(\mathcal{A})$ *is defined by*

$$Conf(\mathcal{A}) = \{\langle l, \nu \rangle \mid l \in L \wedge \nu : \mathbb{X} \to Time \wedge \nu \models \mathcal{I}(l)\},$$

- $C_{ini} = \{\langle l_{ini}, \nu_{ini} \rangle\} \cap Conf(\mathcal{A})$ *is the initial configuration with $\nu_{ini}(x) = 0$ for all clock variables $x \in \mathbb{X}$,*
- *the set $B_{?!} \cup Time$ contains all labels possible at transitions,*
- *to each label $\lambda \in B_{?!} \cup Time$ the transition relation $\xrightarrow{\lambda}$ is of one of the following types:*
  - Time transition: *Time passes, but the location remains unchanged. We have*

$$\langle l, \nu \rangle \xrightarrow{t} \langle l, \nu + t \rangle,$$

    where $\nu + t' \models \mathcal{I}(l)$ for all $t' \in [0, t]$.
  - Discrete transition: *An action $\alpha \in B_{?!}$ is done and clock variables can be reset, but no time passes. We have*

$$\langle l, \nu \rangle \xrightarrow{\alpha} \langle l', \nu' \rangle,$$

    iff a transition $(l, \alpha, \varphi, Y, l') \in E$ exists with $\nu \models \varphi$ and $\nu' = \nu[Y := 0]$, where $\nu' \models \mathcal{I}(l')$.

In the above definition, the set intersection of the initial configuration with the set of all configurations ensures that the initial configuration is contained in the set of all allowed configurations.

A location is *reachable*, if a configuration of the form $\langle l, \nu \rangle$ is reachable by a *transition sequence*

$$\langle l_0, \nu_0 \rangle \xrightarrow{\lambda_1} \langle l_0, \nu_0 \rangle \xrightarrow{\lambda_2} \langle l_0, \nu_0 \rangle \xrightarrow{\lambda_3} \dots$$

This concept of *reachability* (resp. *unreachability*) of good (resp. bad) configurations is often used to show that desirable properties hold in a system (resp. to show the absence of undesirable properties) (cf. [BK08] and Chapter 6). To remember how much time passed since the start of a transition sequence, time stamped configurations $\langle l, \nu \rangle, t$ are used. In contrast to the values of clocks variables in normal configurations, the time stamp t is never reset and thus comprises the global time. Time-stamped transition sequences are referred to as *computation paths* or *runs*, depending on what time stamps are allowed.

Before introducing formal details, we motivate the synchronisation of two extended timed automata via communication channels by continuing the previous example.

Example 5 (Traffic lights, cont.). In the previous Example 4, in Fig. 2.3, we introduced an automaton $\mathcal{A}_P$, controlling a pedestrian traffic light. Fig. 2.4 depicts the respective automaton $\mathcal{A}_C$ controlling the traffic light for the cars. Initially, this controller is in location m_0 showing a red light for the cars. Only when receiving a stop message from the pedestrian light controller $\mathcal{A}_P$, it switches to green light for the cars and after 15 time units it switches to red light for the cars while informing the pedestrian light via *go* about the change. △

$\mathcal{A}_C$:

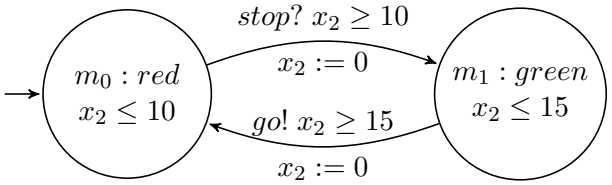

Figure 2.4: Extended timed automaton $\mathcal{A}_C$ for a traffic light for cars capable of communication with $\mathcal{A}_P$ from Fig. 2.3 via channels *stop* and *go*.

For the *parallel composition*, two automata either synchronise via handshake communication or not. The latter case is called *interleaving*, where one automaton takes a transition, while the configuration of the other automaton remains unchanged.

Definition 13 (Parallel composition of extended timed automata)**.** *The parallel composition* $\mathcal{A}_1 \parallel \mathcal{A}_2$ *of two extended timed automata*

$$\mathcal{A}_1 = (L_1, B_1, \mathbb{X}_1, \mathcal{I}_1, E_1, l_{ini,1}) \text{ and } \mathcal{A}_2 = (L_2, B_2, \mathbb{X}_2, \mathcal{I}_2, E_2, l_{ini,2})$$

with disjoint sets of clock variables $\mathbb{X}_1$ *and* $\mathbb{X}_2$ *yields the extended timed automaton*

$$\mathcal{A}_1 \parallel \mathcal{A}_2 \;=\; (L_1 \times L_2, B_1 \cup B_2, \mathbb{X}_1 \cup \mathbb{X}_2, \mathcal{I}, E, (l_{ini,1}, l_{ini,2})).$$

Here the invariants for locations $l_1 \in L_1$ *and* $l_2 \in L_2$ *are conjugated:*

$$\mathcal{I}(l_1, l_2) \overset{def}{\Longleftrightarrow} \mathcal{I}(l_1) \wedge \mathcal{I}(l_2) \tag{2.13}$$

The transition relation E *is constructed according to the following rules:*

- Handshake synchronisation: *An output* $a!$ *is synchronised with an input* $a?$*, yielding an internal action* τ*. For respective transitions* $(l_1, a!, \varphi_1, Y_1, {l_1}') \in E_1$ *and* $(l_2, a?, \varphi_2, Y_2, {l_2}') \in E_2$*, we have*

$$((l_1, l_2), \tau, \varphi_1 \wedge \varphi_2, Y_1 \cup Y_2, ({l_1}', {l_2}')) \in E.$$

- Interleaving: *Here we observe two different cases:*
  - *If for* $l_1 \in L_1$ *there exists a transition* $(l_1, \alpha, \varphi_1, Y_1, {l_1}') \in E_1$*, then for all locations* $l_2 \in L_2$ *there exists the transition* $((l_1, l_2), \alpha, \varphi_1, Y_1, ({l_1}', l_2)) \in E$*.*
  - *If for* $l_2 \in L_2$ *we have a transition* $(l_2, \alpha, \varphi_2, Y_2, {l_2}') \in E_2$*, then for all locations* $l_1 \in L_1$ *there exists the transition* $((l_1, l_2), \alpha, \varphi_2, Y_2, (l_1, {l_2}')) \in E$*.*

Example 6 (Parallel composition). Parts of the parallel composition $\mathcal{A}_P \parallel \mathcal{A}_C$ of the two traffic light controllers from Figs. 2.3 and 2.4 are depicted in Fig. 2.5. Note that we only show the transitions where the automata synchronise and leave out the interleaving transitions for reasons of brevity. Here, the output action *stop*! in $\mathcal{A}_P$ synchronised with the respective input action *stop*? in the automaton $\mathcal{A}_C$, leads to an internal action τ. $\triangle$

$\mathcal{A}_P \parallel \mathcal{A}_C$:

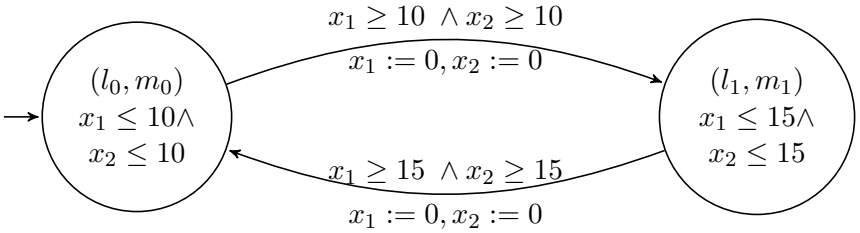

Figure 2.5: In their parallel composition $\mathcal{A}_P \parallel \mathcal{A}_C$, the respective input and output actions of $\mathcal{A}_P$ and $\mathcal{A}_C$ synchronised to a the internal invisible action τ.

For the operational semantics of networks of timed automata, a configuration now contains a vector $\overrightarrow{l} = (l_1, \ldots, l_n)$ with the locations of the networks' components, together with the variable valuation ν for the variables of all components. For the transition relation between configurations, three different types exist:

- Local transition: Only one component takes a discrete transition. For the other components neither location nor variable valuation is affected by this.
- Synchronisation transition: Two automata synchronise their output and input actions.
- Delay transition: Time passes and clock variables are updated.

For reasons of brevity, we refer to [OD08] for details and the formal definiton of the operational semantics of networks of timed automata.

2.4 Controller for Highway Traffic and Country Roads

Our thesis bases on the previously introduced models and controllers for lane change manoeuvres in highway traffic [HLOR11] and overtaking manoeuvres on country-roads [HLO13], which we briefly explain in this section.

Lane Change Controller for Highway Traffic

In this section, we recall the *lane change controller* $\mathcal{A}_{lc}$ from [HLOR11]. The crossing controller we introduce in Chapter 5 for urban crossing manoeuvres is an adaptation of this controller. We also present an implementation of this highway lane change controller with UPPAAL in Sect. 6.2, where we examine some desirable properties of this controller.

While the authors of [HLOR11] did not provide a formalism for their lane change controller, their protocol can be expressed as an automotive-controlling timed automaton (ACTA), which we introduce in Chapter 4. In this section, we describe the lane change controller only informally, but refer to Chapter 4 for a formal semantics of the controller $\mathcal{A}_{lc}$ depicted in Fig. 2.6.

The overall goal of the lane change controller is to undertake *lane change manoeuvres* safely 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

$$Safe(ego) \;\equiv\; \neg\exists c\colon c \neq ego \wedge \langle re(ego) \wedge re(c)\rangle\,. \tag{2.14}$$

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. 2.1) but is applicable to all cars as ego can be substituted by an arbitrary car variable $c \in \text{CVar}$.

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

$$col(ego) \;\equiv\; \exists c\colon c \neq \text{ego} \wedge \langle re(\text{ego}) \wedge re(c)\rangle. \tag{2.15}$$

Formula (2.15) is evaluated to true iff nowhere there exists a car different from the ego car whose reservation overlaps with the actors reservation. We assume $\neg col$ 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*

$$pc(c) \;\equiv\; c \neq \text{ego} \wedge \langle cl(\text{ego}) \wedge (re(c) \vee cl(c))\rangle. \tag{2.16}$$

Formula (2.16) 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 (2.14). However, if a potential collision is detected, the car must withdraw its claim immediately.

If $\exists c\colon pc(c)$ does not hold, the car 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 a successful reservation, t_{lc} time passes before the lane change is finished and the reservation of car E is reduced to the new lane.

Model and Overtaking Protocol for Country Roads

In this section we briefly introduce the abstract model and *overtaking protocol* for country roads with oncoming traffic and refer to [HLO13] for formal details. We use the controller introduced here later in Chapter 5 for a road controller, which deals with overtaking manoeuvres on roads between two intersections.

Abstract model for country roads. The *abstract model for country roads* differs only slightly from the abstract model for highway traffic we introduced in Sect. 2.2. We still consider infinite neighbouring lanes of continuous space from the set of all lanes $\mathbb{L} = \{0, \ldots, N\}$ (cf. p. 14). The difference is that now we introduce a border $b \in \mathbb{L}$, where traffic on lanes $l \leq b$ drives in the direction of increasing position values of $\mathbb{R}$ and traffic on lanes $l > b$ drives in the direction of decreasing values of $\mathbb{R}$. We give an

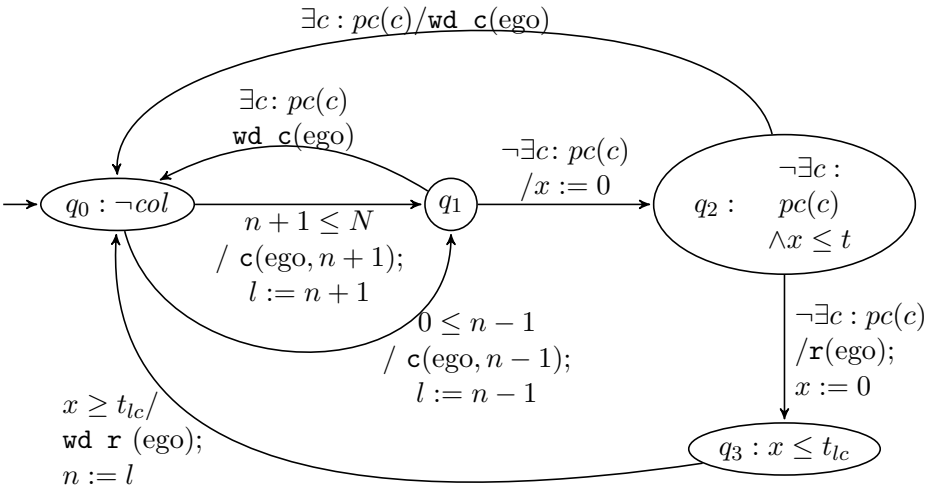

Figure 2.6: Lane change controller $\mathcal{A}_{lc}$ for highway traffic from [HLOR11].

example for a country road model with Fig. 2.7. We use this example to visualise the idea of the overtaking protocol instead of giving all formal details from [HLO13].

Considering a border $b = 1$ in Fig. 2.7, cars on lanes 0 and 1 drive from 'left to right' and cars on lanes 2 and 3 drive from 'right to left'. However, in general, on the border lanes b and $b+1$, cars are allowed to temporarily drive in opposing direction for an *overtaking manoeuvre*. This can be observed in Fig. 2.7, where the claim of car E on the lane 2 with opposing direction indicates its plan to overtake car C.

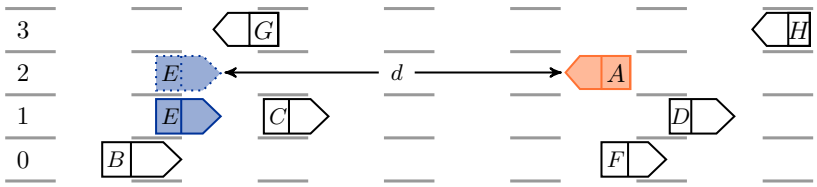

Figure 2.7: Abstract model with adjacent lanes 0 to 3, where lanes 0 and 1 have the direction from "left to right" and lanes 2 and 3 are from the set of lanes with opposing direction. For its overtaking manoeuvre, car E has to take the distance d to the oncoming car A under consideration.

Multi-lane Spatial Logic with length measurement. To enable the overtaking controller to measure distances to other cars, we introduce *length measurement* into the logic MLSL from Def. 6 in Sect. 2.2. This extension of MLSL by length measurement consists of two small features which we describe subsequently in the following two paragraphs.

Consider the atom $u = v$ for $u, v \in \text{Var}$, from basic MLSL. For *Extended Multi-lane Spatial Logic EMLSL*, we extend the set Var to $\text{Var} = \text{CVar} \cup \text{RVar}$, where RVar is a set

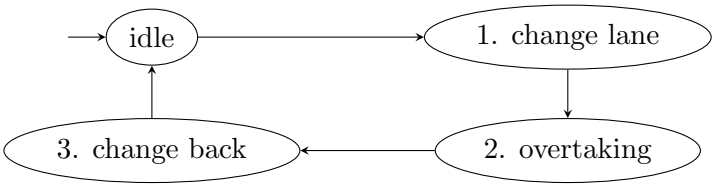

Figure 2.8: Protocol for overtaking manoeuvres from [HLO13].

of variables ranging over values from $\mathbb{R}$. Now, e.g. with an EMLSL formula $c_1 = 2.5$, we can check the size of a constant c_1.

The second extension is to add the atom $\ell = \theta$ to MLSL, where the letter ℓ stands for *length* and θ is a *real-valued term*. Such a term can e.g. involve a function. As an example, consider the function $d_{ot} \colon \mathbb{I} \times \mathbb{I} \to \mathbb{R}_+$, returning the distance needed for one car to overtake another car. With the MLSL formula $\mathit{free}^{d_{ot}(E,C)}$, the car E from our example could e.g. determine, whether there is enough free space on lane 2 for successfully overtaking car C. Note that the authors of [HLO13] use far more complex distance functions for their overtaking protocol, which we do not discuss here in detail.

Overtaking protocol and controllers. Consider again Fig. 2.7. To avoid a collision with car A, car E is only then allowed to transform its claim on lane 2 into a reservation, if the distance d to car A driving in opposing direction is big enough to complete the following phases of the overtaking protocol:

1. Change lanes onto the lane with opposing traffic,
2. Overtake the car on the original lane and
3. Change back onto the original lane in front of the overtaken car.

An overview of these phases is depicted in Fig. 2.8.

For each of these steps one distinct controller exists. The first controller is an adaptation of the lane change controller for highway traffic we introduced in Sect. 2.2. Additionally to the known lane change concept, this controller checks whether the free space needed for the overtaking manoeuvre is big enough. The second controller is a simple controller that accelerates the ego car to a higher speed to pass the car ahead. The third controller is a simplified version of the first controller as less distances need to be calculated for changing back onto the original driving lane in front of the overtaken car.

For the third step, enough space needs to be available in front of the car that is supposed to be passed. To ensure this, a *helper controller* is introduced. We again illustrate the idea of this helper concept with our running example. As soon as car E starts the overtaking manoeuvre, it sends a message to inform the overtaken car C about the manoeuvre. During the whole manoeuvre of E, the duty of *helper car* C is to ensure that the free space in front of it remains big enough for car E to change into it. Any requests of other cars to change into this space are declined immediately by car C.

Chapter summary. In this preliminary chapter, we first gave an overview of the Z notation that we use throughout this thesis and then introduced the approaches for safe car manoeuvres on highways and country roads our thesis bases on. Additionally, extended timed automata were introduced.

3 A Model for Urban Multi-lane Intersections

In this chapter, we introduce our abstract model for urban traffic scenarios. As one of our basic goals is to extend the original MLSL from [HLOR11], we re-use as many concepts as possible, however adapting and extending them wherever needed to cope with our urban traffic requirements. Thus, while some concepts are known from Sect. 2.2, we introduce how they are adapted and used now. We also introduce additional new concepts. For all definitions in this Chapter, we use the Z specification language [WD96] (cf. Sect. 2.1). This chapter is based on our work published in [Sch18a].

From Highway Traffic MLSL to Urban Traffic with Intersections

First of all, we like to outline the key changes we have to face when adapting the basic highway traffic or country roads MLSL approach from to urban traffic scenarios.

The existing abstract models from [HLOR11] and [HLO13] consist of adjacent lanes of infinite length, where cars move along the lanes whenever time passes. This is no longer sufficient for urban traffic, as we deal with intersecting lanes, where the intersections are critical parts of the model as traffic participants coming from different directions wish to enter them. We therefore now deal with finite lanes starting and ending at intersections. For this, we consider finite lanes and introduce special segments for modelling the intersection. To formalise the connections of the finite lanes with intersections, we introduce a generic graph topology, comparable with an abstraction of a street map, capable of representing intersections of any numbers of lanes meeting there.

For statements about the safety of one distinct car, we do not consider traffic situations throughout the whole topology, but focus on the local surroundings of that car by considering a dedicated *view* as known from the highway traffic model. However, for urban traffic with arbitrary sized intersections, we introduce flattened virtual views to cope with turning at intersections. We then extend MLSL by an atom for expressing intersection segments with our *Urban Multi-lane Spatial Logic* (UMLSL). With this logic, we can reason about traffic situations in a view. In later chapters, we introduce syntax and semantics of *automotive-controlling timed automata* (ACTA) to construct a crossing controller which uses UMLSL formulae to determine if an intersection can safely be passed. Over the semantics of UMLSL and ACTA, we prove safety of our crossing controller.

An Informal Introduction to the Abstract Model

We now give a brief overview of our abstract model for urban traffic scenarios and give formal details in the respective following subsections (cf. forward references in the following paragraphs). An example for our abstract model and urban traffic approach is given in Fig. 3.1, which we use throughout the dissertation to explain concepts. We now explain the concepts visible in Fig. 3.1.

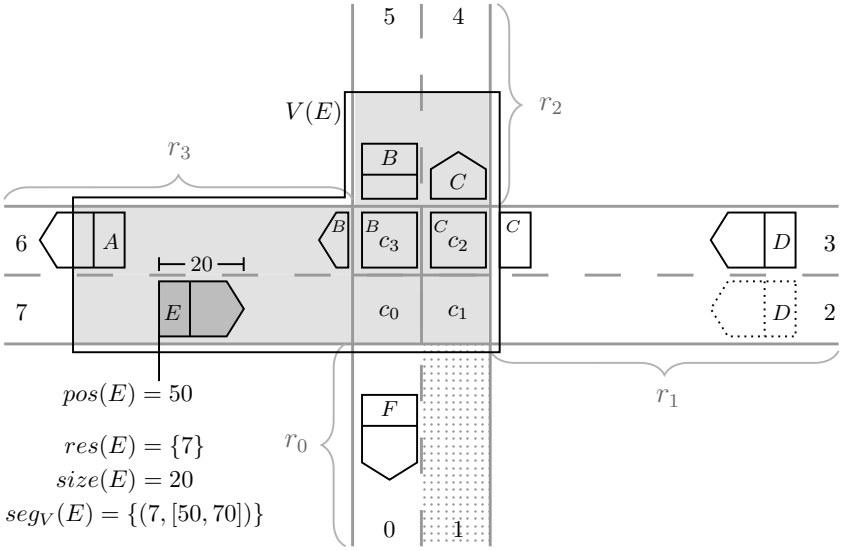

Figure 3.1: In its view $V(E)$, indicated by the grey shading, car E sees car A driving on lane segment 6 and cars B and C, which are currently both turning right at the intersection. $V(E)$ does not cover the road segments r_0 and r_1 and thus not the cars F and D. Car D is currently changing lanes to lane segment 2, indicated by the dotted part of D.

Our *abstract model for urban traffic* focuses on modelling traffic situations at intersections and contains a set $\mathbb{CS}$ of *crossing segments* $c_0, c_1, \ldots$ and a set $\mathbb{L}$ of *lane segments* $0, 1, \ldots$ connecting crossings. Each crossing segment and each lane segment has a *finite length*. See e.g. the grey dotted lane segment 1 in Fig. 3.1, ending where crossing segment c_1 begins.

All adjacent lane segments are bundled to *road segments* with typical representatives $r, r_0, r_1, r_2, \ldots \in \mathbb{RS}$. We assign the type $\mathbb{RS} \subseteq \mathbb{PL}$ and considering again Fig. 3.1, the depicted road segments comprise $r_0 := \{0, 1\}$, $r_1 := \{2, 3\}$, $r_2 := \{4, 5\}$ and $r_3 := \{6, 7\}$. Similarly, all adjacent crossing segments are grouped to *intersections/ crossings* $cr, cr_0, cr_1 \ldots \in \mathbb{CR}$, in the example we have the intersection $cr := \{c_0, c_1, c_2, c_3\}$. Similarly to the case for road segments, we have $\mathbb{CR} \subseteq \mathbb{PCS}$ as the type for crossings. The connections of lane and crossing segments are defined by an underlying graph topology called *urban road network* $\mathcal{N}$ (cf. Sect. 3.2).

As in [HLOR11], every car has a unique *car identifier* $A, B, \ldots$ from the set $\mathbb{I}$ of all car identifiers and a real value for the *position pos* of its rear on a lane or crossing segment. We again use car E as the car under consideration with valuation $\nu(ego) = E$ to refer to this car. When we are talking about an arbitrary car, we use the identifier C.

While the *reservation* $res(E)$ (resp. $cres(E)$) contains all lane segments (resp. crossing segments) car E is actually occupying, a *claim* $clm(E)$ (resp. $cclm(E)$) of a lane segment (resp. of crossing segments) is akin to setting the direction indicator (cf. dotted part of car D in Fig. 3.1, showing the desire of car D to change its lane segment from 3 to 2). Thus, a claim represents the segment a car plans to drive on in the future. To allow for uncertain sensors, both reservations and claims might be seen as over-approximations of the actual space occupied by the cars. The static information about cars like the road network $\mathcal{N}$, positions, reservations and claims of all cars is captured in a traffic snapshot $\mathcal{TS}$ (cf. Sect. 3.3). Reserved and claimed spaces of a car have the extension of its *safety envelope*, which includes the car's physical size and its braking distance. For now, we use a concept of *perfect knowledge*, which means we assume that every car perceives the full safety envelopes of all other cars.

To simplify reasoning, only local parts of traffic are considered as every car has its own local *view* (cf. Sect. 3.4), like view $V(E)$ of car E in Fig. 3.1. Using the size of a car C as perceived by E's sensors, we calculate the visible segments $seg_V(C)$ of the car in the considered view V. If no crossing is within some given *horizon*, the *standard view* $V(E)$ of car E only contains a bounded extension of all adjacent lane segments. If an intersection is within the horizon of car E, its standard view covers a bounded part of the road segment it is driving on, the intersection itself, and a bounded extension of the road segment it is about to drive on after passing the crossing. We refer to this as a *bended view* as the car E may turn left or right at the crossing. We straighten this bended view to *virtual lanes*, to allow for purely spatial reasoning around the corner with our logic *Urban Multi-lane Spatial Logic* (UMLSL) (cf. Sect. 3.5). UMLSL extends the logics introduced in [HLOR11] and [HLO13] by atoms to formalise traffic situations on crossing segments and has a continuous (real positions on lane segments) and a discrete dimension (number of lane and crossing segments).

3.1 Assumptions for the Model

The overall goal of our approach is to enable and undertake formal logical reasoning about traffic situations and to prove correct functionality of autonomous driving manoeuvres. Thus, we now collect assumptions we postulate for our abstract model of real-world urban traffic situations. Note that we wish to only list the assumptions informally at this point of the thesis, and introduce formal details for them (if needed) in the respective sections, where they are introduced and used (cf. forward references). Also note that this section only covers assumptions for the abstract model, while assumptions for our controllers and the actual traffic manoeuvres are provided later in Sect. 5.2.

Driving direction and movement on lane segments. We distinguish between the movement of cars on lane segments and crossing segments. We allow for two-way traffic on lane segments of continuous space and finite length, assuming every lane segment has one *driving direction*, although cars may temporarily drive in the opposite direction to perform an overtaking manoeuvre (cf. Sect. 3.2).

Discretisation of crossing segments. As a car's direction will change while turning on an intersection, in contrast to lane segments, we cannot assign one specific driving direction to a crossing segment. Therefore, we consider crossing segments as discrete elements which are either fully occupied by a car or empty. When a car is about to drive onto a discrete crossing segment and time elapses, the car's safety envelope will stretch to the whole crossing segment, while disappearing continuously on the lane segment it drove on. Consider e.g. car B in Fig. 3.1, leaving lane segment 5 and entering lane segment 6 continuously, while it occupies the whole discrete crossing segment c_3.

Minimal distance between two intersections. We assume two different intersections to be at least a fixed minimal distance apart from each other, to allow for our view construction method in the following sections. With this, we exclude that one view contains two intersections. This assumption is reasonable, as for the safety of crossing manoeuvres only the intersection where the manoeuvre is planned on is interesting. However, this assumption could be softened in future work, see Sect. 8.4 for details on this.

Sensor function. We abstract from specifying a concrete procedure for perceiving the surroundings of cars through sensors. However, we give details on our abstract version of a sensor function in Sect. 3.4.3 on p. 59.

3.2 Topology

We allow for road segments between intersections with arbitrarily many parallel lane segments and do not restrict the number of lane segments for each direction. With this, we allow for common 2-by-2 intersections (cf. example in Fig. 3.1), but we also allow for modelling bigger intersections (cf. Fig. 3.2) and one-way streets in our topology. The graph topology we introduce in this section is capable of modelling any type of intersections of any number of lanes and roads meeting there, e.g. also n-by-m-by-o-by-p-by-q intersections are possible. However, we name these bigger intersections by *n-by-m intersections* throughout this thesis, whenever we do not mean a particular intersection.

While giving a lot of freedom in the construction of our topology, we have to restrict it partially to prevent deadlocks and senseless topologies. We e.g. demand that each lane or crossing segment has a predecessor and a successor. Note that with this assumption we can still model blind alleys, as we can consider the end of a blind alley as a u-turn-only intersection. Later in this section in Def. 16, we formalise these restrictions as *topological sanity conditions*.

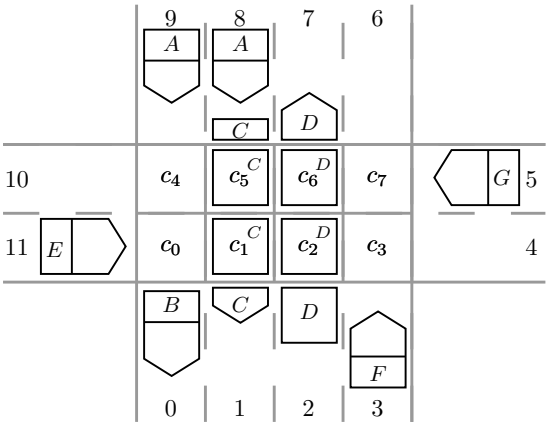

Figure 3.2: Abstract model for a 2-by-4 intersection, depicting the reservations of cars A to G, including their braking distances. Note that car A is currently changing lanes and thus reserves both lane segments 8 and 9.

3.2.1 Urban Road Networks

We describe the connections between lane segments from the set $\mathbb{L} \subseteq \mathbb{N}$ and crossing segments from the set $\mathbb{CS}$ by an *urban road network* $\mathcal{N}$, whose *nodes* are elements from $\mathbb{L}$ and $\mathbb{CS}$ with $\mathbb{L} \cap \mathbb{CS} = \emptyset$. Additionally, as we are dealing with traffic that is evolving over time, we need to capture the size of lane and crossing segments in our graph. In our model, a car moves gradually and continuously in a lane segment, but for crossing segments we assume that a car is either outside of it or reserves it fully. Thus, crossing segments are discrete elements in our network.

We later use the information given by the urban road network $\mathcal{N}$ to determine the parts $seg_V(C)$ of lane and crossing segments the so-called safety envelope of an arbitrary car C occupies and to construct virtual lanes for our bended view in Sect. 3.4.

Definition 14 (Urban road network $\mathcal{N}$)**.** *An* urban road network *is defined as a graph* $\mathcal{N} = (\mathcal{V}, E_u, E_d, \omega)$*, where*

- $\mathcal{V} = \mathbb{L} \cup \mathbb{CS}$ *is a finite set of nodes,*
- $E_u \subseteq \mathbb{L} \times \mathbb{L}$ *is the set of undirected edges, i.e., E_u is a symmetric relation on $\mathbb{L}$,*
- $E_d \subseteq (\mathcal{V} \times \mathcal{V}) \setminus (\mathbb{L} \times \mathbb{L})$ *is the set of directed edges, and*
- $\omega \colon \mathcal{V} \to \mathbb{R}_+$ *is a mapping that assigns a positive, real-valued size to each node in $\mathcal{V}$.*

The set of all edges is defined by $E == E_u \cup E_d$. We forbid self-loops, i.e. $\forall v : \mathcal{V} \bullet (v, v) \notin E$.

Two elements $v_1, v_2 \in \mathbb{L}$ with $(v_1, v_2) \in E_u$ constitute two neighbouring lane segments, where the undirected edge allows for bidirectional lane change manoeuvres between these two lane segments. A part of the road network $\mathcal{N}$ corresponding to Fig. 3.1 is depicted on the left-hand side of Fig. 3.3.

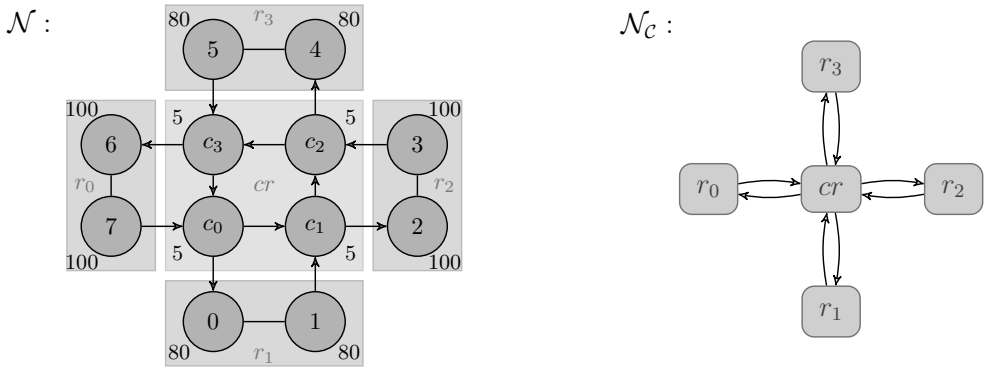

Figure 3.3: Urban road network $\mathcal{N}$ corresponding to Fig. 3.1 on the left-hand side and coarser version $\mathcal{N}_\mathcal{C}$, only depicting the strongly connected components and their relations with each other, at the right-hand side.

3.2.2 Infinite Paths and Finite Sequences

We define *infinite paths pth* in the urban road network $\mathcal{N}$ which represent possible travelling routes through $\mathcal{N}$. We also introduce *finite path sequences* $\overrightarrow{\pi}$ by defining a restricted version $\mathsf{seq}_E\,\mathcal{V}$ of the Z data type $\mathsf{seq}\,\mathcal{V}$ describing finite sequences of objects from the set $\mathcal{V}$. In Sect. 3.4, we also use *inverted* path sequences $\overleftarrow{\pi}$ *backwards* through the topology resembling lanes with opposing traffic. For defining these, we use the relational inverse $E^\sim$ of E.

Definition 15 (Infinite paths *pth* and finite path sequences)**.** *In the network $\mathcal{N}$ with the set E of edges the set $pth_\mathcal{N}$ of* infinite paths *pth is defined by*

$$pth_\mathcal{N} = \{pth : \mathbb{Z} \to \mathcal{V} \mid \forall i : \mathbb{Z} \bullet (pth(i), pth(i+1)) \in E\}.$$

We define the set $\mathsf{seq}_E\,\mathcal{V}$ of finite path sequences $\overrightarrow{\pi}$ *over E as follows:*

$$\mathsf{seq}_E\,\mathcal{V} = \{\overrightarrow{\pi} : \mathsf{seq}\,\mathcal{V} \mid \forall i : 1\,..\,\#\overrightarrow{\pi} - 1 \bullet (\overrightarrow{\pi}(i), \overrightarrow{\pi}(i+1)) \in E\}.$$

We denote the set of all inverted path sequences $\overleftarrow{\pi}$ *by $\mathsf{seq}_{E^\sim}\,\mathcal{V}$, where $E^\sim$ is the relational inverse to E. In Sect. 3.4 we consider also $\mathsf{seq}_{E_d}\,\mathcal{V}$ for the set E_d of directed edges instead of all edges E. We denote that a finite path sequence $\overrightarrow{\pi} \in \mathsf{seq}_E\,\mathcal{V}$ is a* cut-out of a path $pth \in pth_\mathcal{N}$ *by $\overrightarrow{\pi} \sqsubset pth$, with*

$$\overrightarrow{\pi} \sqsubset pth \Leftrightarrow \exists p : \mathbb{Z} \rightarrowtail\!\!\!\!\!\!\rightarrow \mathcal{V} \bullet p \subseteq pth \wedge \overrightarrow{\pi} = \mathsf{squash}_\mathbb{Z}(p),$$

where $\mathsf{squash}_{\mathbb{Z}}$ *is an adaptation of the* Z *operation* squash *(cf. Sect. 2.1) for functions with domain over* $\mathbb{N}$ *to cope with functions* f *with* $\mathsf{dom}\ f \subseteq \mathbb{Z}$*:*

$$\begin{aligned}\mathsf{squash}_{\mathbb{Z}}(f) \;=\; (\mu\, g\colon 1\,..\,\#f \to \mathcal{V} \mid \exists \beta\colon 1\,..\,\#f \rightarrowtail\!\!\!\!\twoheadrightarrow \mathsf{dom}\ f \bullet \\ (\forall i, j\colon 1\,..\,\#f \bullet (i < j \Leftrightarrow \beta(i) < \beta(j)) \land g = \beta \fatsemi f)).\end{aligned}$$

Informally, $\mathsf{squash}_{\mathbb{Z}}(f)$ squeezes the possibly scattered elements of $\mathsf{dom}\, f$ in an order-preserving way to a cohesive interval in $\mathbb{N}$, leading to a sequence from $\mathsf{seq}\,\mathcal{V}$. The restriction of $\mathsf{seq}\,\mathcal{V}$ to $\mathsf{seq}_E\,\mathcal{V}$ guarantees that all finite path sequences $\overrightarrow{\pi} \in \mathsf{seq}_E\,\mathcal{V}$ are consistent with the underlying graph topology $\mathcal{N}$, whereas the set $\mathsf{seq}\,\mathcal{V}$ also contains sequences $\langle v, v' \rangle$ of nodes $v, v' \in \mathcal{V}$ with $(v, v') \notin E$. Note that in Z, the arrow $\rightarrowtail\!\!\!\!\twoheadrightarrow$ is used for partial, finite functions and the arrow $\rightarrowtail\!\!\!\!\twoheadrightarrow$ denotes a total, bijective function.

We use the Z sequence notation with brackets $\langle\ \rangle$ not only for finite sequences, but informally also for infinite paths *pth*, however indicating their infinity by ....

Example 7 (Infinite paths and finite sequences). For the traffic situation depicted in Fig. 3.1 and the related road network $\mathcal{N}$ in Fig. 3.3, the path $pth = \langle \ldots, 7, c_0, c_1, c_2, 4, \ldots \rangle$ is a suitable travelling path for car E, when it plans to turn left. A finite cut-out $\overrightarrow{\pi} \in \mathsf{seq}_E\,\mathcal{V}$ with $\overrightarrow{\pi} \sqsubset pth$ is $\overrightarrow{\pi} = \langle 7, c_0, c_1 \rangle$ with its inversion $\overleftarrow{\pi} = \langle c_1, c_0, 7 \rangle$ from $\mathsf{seq}_{E^\sim}\,\mathcal{V}$, which denotes moving backwards through the intersection from crossing segment c_1 to lane segment 7. △

Inspired by the sanity conditions for traffic snapshots from [Lin15] (cf. Def. 2, p. 16), we introduce *topological sanity conditions* to exclude unrealistic urban road networks $\mathcal{N}$; e.g. crossing segments without an outgoing edge, which are useless for infinite paths and would only serve to trap cars in them.

Definition 16 (Topological sanity conditions)**.** *An urban road network* $\mathcal{N}$ *with the set of edges* E *is* sane*, if the following conditions hold.*

$$\forall v\colon \mathcal{V} \bullet \exists v', v''\colon \mathcal{V} \bullet v \neq v' \land v \neq v'' \land (v', v) \in E_d \land (v, v'') \in E_d, \tag{3.1}$$

$$\begin{aligned}\forall v\colon \mathbb{CS} \bullet \exists s, s'\colon \mathsf{seq}_{E_d}\,\mathbb{CS}, \exists l, l'\colon \mathbb{L} \bullet \\ \langle v \rangle \frown s \frown \langle l \rangle \in \mathsf{seq}_{E_d}\mathcal{V} \land \langle l' \rangle \frown s' \frown \langle v \rangle \in \mathsf{seq}_{E_d}\mathcal{V}.\end{aligned} \tag{3.2}$$

Sanity condition (3.1) ensures that every node in $\mathcal{N}$ has at least one incoming and at least one outgoing edge to prevent deadlocks in the topology. Condition (3.2) ensures that somewhere there exists a lane segment both in the prefix and postfix path of any crossing segment. This prevents a topology that only consists of a crossing. For our current approach, e.g. for the virtual view construction in Sect. 3.4, these restrictions to $\mathcal{N}$are sufficient. However, for scenarios not covered here, other sanity conditions might be required.

3.2.3 Coarser Networks and Paths

For the view construction in Sect. 3.4, we also consider a *coarse-grained* version $\mathcal{N}_\mathcal{C}$ of $\mathcal{N}$. While $\mathcal{N}$ contains crossing segments $cs \in \mathbb{CS}$ and lane segments $l \in \mathbb{L}$ as nodes, $\mathcal{N}_\mathcal{C}$ contains abstracted versions of these as nodes. Following [CGP99], we define *strongly connected components* $r \in \mathbb{RS}$ (road segments) resp. $cr \in \mathbb{CR}$ (intersections/ crossings) in our graph topology $\mathcal{N}$ as sets of *maximal subgraphs* of nodes solely from $\mathbb{L}$ (resp. $\mathbb{CS}$). These maximal subgraphs can be formalised as *equivalence classes* of lane segments (resp. crossing segments).

Definition 17 (Strongly connected components)**.** *Consider a road network $\mathcal{N}$ with nodes $\mathcal{V} = \mathbb{L} \cup \mathbb{CS}$ and edges E. For an arbitrary lane segment $l \in \mathbb{L}$ we define the equivalence class $f_\mathbb{L}(l)$, where $f_\mathbb{L} \colon \mathbb{L} \to \mathbb{P}(\mathbb{L})$ with*

$$f_\mathbb{L}(l) = \{l' \colon \mathbb{L} \mid l \; E_u^* \; l'\}. \tag{3.3}$$

Similarly, we define for an arbitrary crossing segment $cs \in \mathbb{CS}$ the equivalence class $f_\mathbb{CS}(cs)$, where $f_\mathbb{CS} \colon \mathbb{CS} \to \mathbb{P}(\mathbb{CS})$ with

$$f_\mathbb{CS}(cs) = \{cs' \colon \mathbb{CS} \mid cs \; ((E_d \cup E_d^\sim) \cap (\mathbb{CS} \times \mathbb{CS}))^* \; cs'\}. \tag{3.4}$$

With $f_\mathbb{L}$ and $f_\mathbb{CS}$, we derive the respective sets $\mathbb{CR}$ and $\mathbb{RS}$ of all strongly connected components *in $\mathcal{N}$ as follows:*

$$\begin{aligned} \mathbb{RS} &= \{X \colon \mathbb{P}(\mathbb{L}) \mid \exists l \colon \mathbb{L} \bullet X = f_\mathbb{L}(l)\}, \\ \mathbb{CR} &= \{Y \colon \mathbb{P}(\mathbb{CS}) \mid \exists cs \colon \mathbb{CS} \bullet Y = f_\mathbb{CS}(cs)\}. \end{aligned}$$

Formula (3.3) yields for an arbitrary lane segment $l \in \mathbb{L}$ its equivalence class $f_\mathbb{L}(l)$ by including only those lane segments $l' \in \mathbb{L}$ connected with l solely by edges from E_u. Formula (3.4) analogously defines for an arbitrary crossing segment $cs \in \mathbb{CS}$ the equivalence class $f_\mathbb{CS}(cs)$, containing solely crossing segments $cs' \in \mathbb{CS}$ strongly connected to cs. The sets $\mathbb{RS}$ resp. $\mathbb{CR}$ contain *all* respective strongly connected components defined by the equivalence classes from (3.3) and (3.4).

We now construct the *coarse urban road network* $\mathcal{N}_\mathcal{C}$ starting with set of nodes $\mathcal{V}_\mathcal{C} = \mathbb{CR} \cup \mathbb{RS}$. By Def. 14, edges from the sets $\mathbb{L} \times \mathbb{CS}$ and $\mathbb{CS} \times \mathbb{L}$ are directed, whereby entry and exit points to an intersection $cr \in \mathbb{CR}$ and to a road segment $r \in \mathbb{RS}$ are defined unambiguously. Here a road segment r is connected with a crossing cr with a directed edge (r, cr) resp. (cr, r) iff there exists a corresponding directed edge in the underlying graph $\mathcal{N}$. Thus, $\mathcal{N}_\mathcal{C}$ is an existential abstraction (over-approximation) of $\mathcal{N}$.

Definition 18 (Coarse-grained urban road network $\mathcal{N}_\mathcal{C}$)**.** *To an urban road network $\mathcal{N} = (\mathcal{V}, E_u, E_d, \omega)$ with connected components $\mathbb{CR}$ and $\mathbb{RS}$ from Def. 17, the corresponding* coarse-grained urban road network *is defined by the graph $\mathcal{N}_\mathcal{C} = (\mathcal{V}_\mathcal{C}, E_\mathcal{C})$, where*

- *$\mathcal{V}_\mathcal{C} = \mathbb{CR} \cup \mathbb{RS}$ is the set of nodes and*

- $E_{\mathcal{C}} = \{e_c : \mathcal{V}_{\mathcal{C}} \times \mathcal{V}_{\mathcal{C}} \mid \exists (v, v') : E_d \bullet e_c = (f_{\mathbb{L}}(v), f_{\mathbb{CS}}(v')) \vee e_c = (f_{\mathbb{CS}}(v), f_{\mathbb{L}}(v'))\}$ *is the set of directed edges.*

The coarser version $\mathcal{N}_{\mathcal{C}}$ of the urban road network $\mathcal{N}$ for Fig. 3.1 is depicted on the right-hand side of Fig. 3.3. For the coarse network $\mathcal{N}_{\mathcal{C}}$, we define *coarse-grained paths* $pth_{\mathcal{C}}$ and finite *coarse-path sequences* $\overrightarrow{\pi_{\mathcal{C}}}$ both analogously to the finer paths from Def. 15.

Definition 19 (Coarse path $pth_{\mathcal{C}}$ and coarse sequence)**.** *In a coarse grained urban road network* $\mathcal{N}_{\mathcal{C}}$ *with the set of edges* $E_{\mathcal{C}}$ *an* infinite coarse-grained path *is defined by* $pth_{\mathcal{C}} : \mathbb{Z} \to \mathcal{V}_{\mathcal{C}}$*, where*

$$\forall i : \mathbb{Z} \bullet (pth_{\mathcal{C}}(i), pth_{\mathcal{C}}(i+1)) \in E_{\mathcal{C}}.$$

The set of all possible paths in $\mathcal{N}_{\mathcal{C}}$ *is defined by* $pth_{\mathcal{N}_{\mathcal{C}}}$*. We derive* finite coarse sequences $\overrightarrow{\pi_{\mathcal{C}}}$ *from the set* $\mathsf{seq}_{E_{\mathcal{C}}} \mathcal{V}_{\mathcal{C}}$ *analogously to Def. 15.*

Example 8 (Coarse-grained path). Consider again Example 7 for the traffic situation depicted in Fig. 3.1 and the path $pth = \langle \ldots, 7, c_0, c_1, c_2, 4, \ldots \rangle$. A coarser version of pth is $pth_{\mathcal{C}} = \langle \ldots, r_0, cr, r_3, \ldots \rangle$. Here the lane segments 7 and 4 are mapped to their respective road segments $r_0 = f_{\mathbb{L}}(7)$ resp. $r_3 = f_{\mathbb{L}}(4)$, and the crossing segments c_0, c_1 and c_2 are condensed to the intersection $cr = f_{\mathbb{CS}}(c_0) = f_{\mathbb{CS}}(c_1) = f_{\mathbb{CS}}(c_2)$. △

Note that while a path $pth \in pth_{\mathcal{N}}$ can be mapped to exactly one coarser path $pth_{\mathcal{C}} \in pth_{\mathcal{N}_{\mathcal{C}}}$ due to the unambiguously defined functions $f_{\mathbb{CS}}$ and $f_{\mathbb{L}}$ (cf. Def. 17), the reverse direction is not unambiguous; A coarse path $pth_{\mathcal{C}}$ is generally associated with more than one fine-grained path pth.

Example 9 (Several finer paths relating to one coarse path). Consider the bigger intersection from Fig. 3.2 and the respective corresponding fine and coarse topologies $\mathcal{N}_2$ and $\mathcal{N}_{2,\mathcal{C}}$ depicted in Fig. 3.4. For turning left from road segment $r_{left} = \{10, 11\}$ to $r_{above} = \{6, 7, 8, 9\}$, we have the coarse path sequence $\overrightarrow{\pi_{\mathcal{C}}} == \langle r_{left}, cr, r_{above} \rangle$. For $\overrightarrow{\pi_{\mathcal{C}}}$, there exist the four fine path sequences as indicated by the different shading. △

We show a general procedure for retrieving all fine path sequences relating to one arbitrary coarse path sequence through an intersection later in Sect. 3.4.

3.3 Traffic Snapshot

In Sect. 2.2, we introduced a traffic snapshot for the highway traffic approach, e.g. containing information about current positions and reserved lanes of cars. As of previous Sect. 3.2, the topology of our abstract model for urban traffic at intersections is a lot more complex compared to the highway traffic case. We thus also extend the traffic snapshot, e.g. by information about reserved crossing segments and the path of each car.

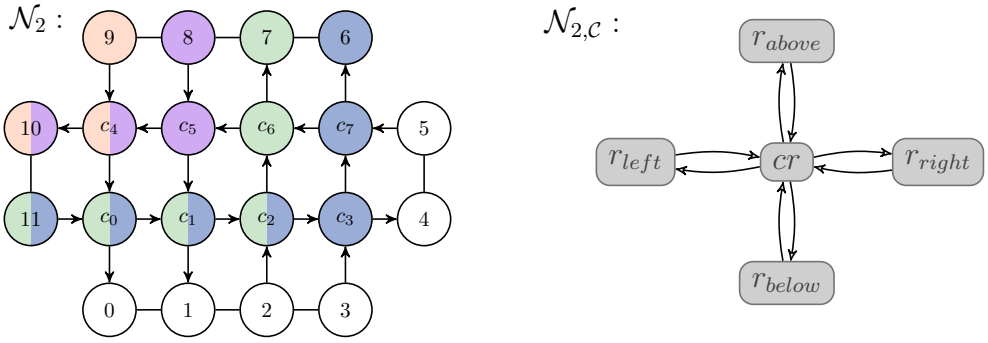

Figure 3.4: Fine and coarse topologies for 2-by-4 intersection from Fig. 3.2 with highlighted finer paths through the intersection.

3.3.1 Traffic Snapshot: The Global Picture

Compared to the traffic snapshot from preliminary Sect. 2.2, the global *traffic snapshot for urban traffic* $\mathcal{TS}$ now captures the traffic on an urban road network $\mathcal{N}$ at a given arbitrary moment. We recall definitions from the preliminary section, extending them where needed.

To formalise two-way traffic on road segments, we assume that every lane segment has one direction and cars normally drive on a lane segment in the direction of increasing real values. Cars may only temporarily drive in the opposite direction of a lane segment to perform an overtaking manoeuvre. The first main extension is that we introduce claims and reservations for crossing segments, where we now differentiate between claims and reservations on lane segments (clm, res) and on crossing segments ($cclm$, $cres$). Remember that a reservation is the space a car is currently occupying and a claim is the space the car plans on driving in the future (cf. setting the turn signal).

The second extension concerns the position of a car, as it is no longer sufficient to consider only a real position $pos(C)$ for each car C in a topology like our urban road network. Additionally to the position pos of each car, we assign an infinite path $pth(C) \in pth_{\mathcal{N}}$ to every car C (cf. Def. 15, p. 40). Onwards, we denote the index of the path segment from $pth(C)$ car C is currently driving on by $curr(C)$. The real-valued position $pos(C)$ now defines the position of the rear of car C on the current segment $pth(C)(curr(C))$.

Definition 20 (Traffic snapshot)**.** *Given a road network $\mathcal{N}$, a* traffic snapshot $\mathcal{TS}$ *is a structure $\mathcal{TS} = (\mathcal{N}, res, clm, cres, cclm, pth, curr, pos, spd, acc)$ with*

- $res : \mathbb{I} \to \mathbb{P}(\mathbb{L})$ *such that $res(C)$ is the set of lane segments which are reserved by a car C,*
- $clm : \mathbb{I} \to \mathbb{P}(\mathbb{L})$ *such that $clm(C)$ is the set of lane segments which are claimed by car C,*

- $cres : \mathbb{I} \to \mathbb{P}(\mathbb{CS})$ *such that* $cres(C)$ *is the set of crossing segments which are reserved by a car* C*,*
- $cclm : \mathbb{I} \to \mathbb{P}(\mathbb{CS})$ *such that* $cclm(C)$ *is the set of crossing segments that are claimed by a car* C*,*
- $pth : \mathbb{I} \to pth_{\mathcal{N}}$ *such that* $pth(C)$ *is the path pursued by car* C*,*
- $curr : \mathbb{I} \to \mathbb{Z}$ *such that* $curr(C)$ *is the index of the path element of* $pth(C)$ *currently occupied by the rear of* C*,*
- $pos : \mathbb{I} \to \mathbb{R}$ *such that* $pos(C)$ *is the position of the rear of car* C *in* $pth(C)(curr(C))$*,*
- $spd : \mathbb{I} \to \mathbb{R}_+$ *such that* $spd(C)$ *is the current speed of car* C *and*
- $acc : \mathbb{I} \to \mathbb{R}$ *such that* $acc(C)$ *is the current acceleration of car* C*.*

Let $\mathbb{TS}$ *denote the set of all traffic snapshots.*

Example 10 (Traffic snapshot). Consider Fig. 3.1 as a cut-out of a traffic snapshot $\mathcal{TS}_1$ with related road network $\mathcal{N}$ from Fig. 3.3. We exemplarily list the entries in $\mathcal{TS}_1$ for car E. We have $res(E) = \{7\}$ for the reservation of E on lane segment 7 but $clm(E) = cclm(E) = cres(E) = \emptyset$, as car E neither claims a lane segment nor claims or reserves a crossing segment in $\mathcal{TS}_1$. Consider $pth(E) = \langle \ldots, 7, c_0, c_1, c_2, 4, \ldots \rangle$, already informally used as a path for E in examples in Sect. 3.2. As index of the current path element 7 in $pth(E)$, let us assume $curr(E) = -2$ and let $pos(E) = 50$ for the position of E in segment 7, as indicated in Fig. 3.1. Consider $spd(E) = 40$ and $acc(E) = 0$. △

Analogously to the sanity conditions for traffic snapshots we introduced in Def. 2 for highway traffic, we introduce respective sanity conditions for the urban traffic case to exclude senseless traffic snapshots. E.g., assuming that intersections are a specific distance apart from each other (cf. Sect. 3.1), it should not be allowed that a car has reserved crossing segments on two incoherent intersections.

Note that we keep all sanity conditions from Def. 2 for claims or reservations on lane segments between intersections, with the exception of conditions 5 and 6 which state that a second reservation (resp. a claim) on a lane segment must be on a neighbouring lane segment to the currently occupied one. The reason is that at intersections we have to claim and later reserve some space on the lane segment *after* the intersection. E.g. in our example from Fig. 3.1, car E currently reserves space on lane segment 7 and needs to reserve some space on lane segment 4 in the future to be able to leave the intersection after its turn left manoeuvre. Thus at some point we will have $res(E) = \{4, 7\}$ in our traffic snapshot, whereas easily observable, lanes 4 and 7 are not neighbouring.

Definition 21 (Urban traffic snapshot sanity sonditions)**.** *We recall sanity conditions 1-4 from Def. 2 for highway traffic. An urban traffic snapshot* $\mathcal{TS}$ *is sane, if the following conditions hold for all cars* $C \in \mathbb{I}$*.*

1. $\#cres(C) \cup \#res(C) > 0$
2. $cres(C) \neq \emptyset$ *equivalent* $cclm(C) = \emptyset$
3. $cclm(C) \neq \emptyset$ *equivalent* $clm(C) = \emptyset$
4. $cclm(C) \neq \emptyset$ *implies* $\forall n \colon res(C), \exists cs \colon cclm(C) \bullet (n, cs) \in E_d$
5. $\#cres(C) > 0$ *implies* $\forall cs, cs' \colon cres(C) \bullet cs \neq cs' \rightarrow f_{\mathbb{CS}}(cs) = f_{\mathbb{CS}}(cs')$
6. $\#res(C) = 2$ *and* $cres(C) = \emptyset$ *implies* $\exists n \colon \mathbb{L} \bullet res(C) = \{n, n+1\}$
7. $\#res(C) = 2$ *and* $cres(C) \neq \emptyset$ *implies*
   $\forall n \in res(C)\, \exists cs \colon cres(C) \bullet (n, cs) \in E_d \vee (cs, n) \in E_d$
8. $\#clm(C) = 1$ *implies* $(\#cres(C) > 0 \wedge \exists cs \colon cres(C), n \colon clm(C) \bullet (cs, n) \in E_d)$
   or $(\#res(C) = 1 \wedge \exists n \colon res(C), l \colon clm(C) \bullet (n, l) \in E_u)$

Let $\mathbb{TS}$ *denote the set of all sane traffic snapshots.*

Condition 1 states that there always must exist some reservation for each car C, let it be a reservation on a lane or a crossing segment. The next condition 2 ensures that no car C can reserve (resp. claim) crossing segments while already involved in an active crossing manoeuvre with a claim (resp. reservation). With condition 3, we forbid to simultaneously start a lane change and a crossing manoeuvre. Condition 4 states that a crossing claim is only possible if a reservation on the lane segment *before* the intersection exists and condition 5 states that no car may reserve crossing segments of two incoherent intersections. Condition 6 is the adapted version of old sanity condition 5 with the addition that two reservations on lane segments are only then neighbouring, when no crossing reservation exists for car C (i.e. during a lane change manoeuvre). Additionally to that, condition 7 states that with an active crossing reservation of car C two reservations on lane segments may not be *not* neighbouring, but one is before and the other one after the intersection. Finally, condition 8 adapts the old condition 6 by stating that a claim is either on a neighbouring lane segment or after an intersection for which a crossing reservation exists.

3.3.2 Traffic Snapshot Evolution

As in highway traffic snapshots from Sect. 2.2, we allow for transitions between traffic snapshots to model the behaviour of cars in urban traffic. We introduce new transitions for crossing claims and crossing reservations and adapt all of the existing transitions introduced in Sect. 2.2, Def. 3. This is due to new pre- and postconditions for the traffic snapshot transitions in urban traffic. In Sect. 5.3, we define that our controllers can only commit one manoeuvre at a moment, thus for now we exclude simultaneous transitions. For a more thorough discussion of concurrent traffic snapshot transitions, see [BHLO17].

Before giving the formal definitions of the new transitions, we discuss the change on a traffic snapshot, when time elapses. In the previous approaches on highway traffic and country roads, cars moved forwards on their currently reserved infinite lanes according to their speed and acceleration when time passes. While the MLSL approach abstracts from the dynamics, this movement was nonetheless expressed with the formula $pos'(C) = pos(C) + spd(C) \cdot t + \frac{1}{2} acc(C) \cdot t^2$. As we now consider *finite* lane segments in urban traffic that start resp. end at intersections, cars now move along their paths in $\mathcal{N}$ as follows.

When some time t elapses, every car C with a positive speed $spd(C)$ changes its position within path $pth(C)$. With the following algorithm, we calculate the index $next(C)$ of the node in $pth(C)$ that is reached after time t. We also calculate the new position $pos'(C)$ of car C on $pth(C)(next(C))$. This node can either be a crossing or lane segment. Note that $curr(C) = next(C)$ iff C did not move far enough to leave its current node. Recall from Def. 14 that ω is a mapping, assigning a size to each node $v \in \mathcal{V}$.

Algorithm 1 (Reached node and position after t time units)**.** *Consider a traffic snapshot $\mathcal{TS} \in \mathbb{TS}$, $t \in Time$ and $C \in \mathbb{I}$.*

newPos$_{\mathcal{TS}}$ (C, t) {
  $i := 0$;
  $dist_0 := spd(C) \cdot t + \frac{1}{2} acc(C) \cdot t^2$;
  $n := curr(C)$;
  $p_0 := pos(C)$;
  while $(dist_i > 0)$
    $dist_{i+1} := dist_i - (\omega(pth(C)(n)) - p_i)$;
    $p_{i+1} := p_i + dist_i$;
    if $(p_{i+1} > \omega(pth(C)(n)))$
      $p_{i+1} := 0$;
      $n := n + 1$
    fi;
  $i := i + 1$
  end while;
  return $(n,\ p_i)$;
}

We assign the return values to $next(C) == first(\mathbf{newPos}_{\mathcal{TS}}(C, t))$ *and* $pos'(C) == second(\mathbf{newPos}_{\mathcal{TS}}(C, t))$.

With $dist_0$, we calculate the distance car C moves within t time. Starting at the segment $pth(C)(curr(C))$ and $pos(C)$, we calculate the new position on this current segment by

$pos(C) + dist_0$. If the new position exceeds the size $\omega(pth(C)(curr(C)))$ of the current segment, we increase the index i by one and calculate the new distance $dist_1$ yet to go from the new segment. We subsequently repeat this procedure until $dist_i \leq 0$ and we have the position with p_i and the index of the next segment with n.

Note that while the node $pth(C)(next(C))$ is only one distinct segment, the safety envelope of car C might stretch over more than this single segment. E.g. if $pth(C)(next(C))$ is a lane segment l, but the position of car C on l is close to the next intersection. However, we consider the size of safety envelopes from local viewpoints of distinct cars and do not include them in the central traffic snapshot. We calculate the occupied segments of the safety envelopes of cars on lane and crossing segments in Sect. 3.4 with Alg. 2.

In Defs. 22 to 27, we now give the definitions of the traffic snapshot transitions, starting with the previously motivated *time transition* in Def. 22. We again use the overriding notation $\oplus$ of Z for function updates. With the update $clm'(C) = clm \oplus \{C \mapsto \{n\}\}$, we e.g. update the set of claimed lane segments for car C to $\{n\}$. For all transitions we assume a current traffic snapshot $\mathcal{TS} = (\mathcal{N}, res, clm, cres, cclm, pth, curr, pos, spd, acc)$ as origin.

For turn manoeuvres, we require a car C to reserve *all* needed crossing segments at once to prevent deadlocks. We derive the set of these crossing segments by the function $nextCr(C, next(C))$ which is included in following Def. 22.

Definition 22 (*Time* transition)**.** *For all $C \in \mathbb{I}$ and for a distinct $t \in Time$, we use $next(C)$ and $pos'(C)$ as calculated with* $\mathbf{newPos}_{\mathcal{TS}}(C, t)$ *in Alg. 1 and obtain*

$$\begin{aligned}\mathcal{TS} \xrightarrow{t} \mathcal{TS}' \quad \Leftrightarrow \quad & \mathcal{TS}' = (\mathcal{N}, res', clm, cres', cclm, pth, curr', pos', spd', acc) \\ & \wedge \forall C : \mathbb{I} \bullet (res' = res \oplus \{C \mapsto \{pth(C)(next(C))\}\} \\ & \qquad \wedge\ cres' = cres \oplus \{C \mapsto \{nextCr(C, next(C))\}\} \\ & \qquad \wedge\ curr'(C) = next(C) \\ & \qquad \wedge\ 0 \leq pos'(C) \leq \omega(pth(C)(next(C))) \\ & \qquad \wedge\ spd'(C) = spd(C) + acc(C) \cdot t),\end{aligned}$$

where $nextCr : \mathbb{I} \times \mathbb{Z} \to \mathbb{P}(\mathbb{CS})$ is defined by

$$nextCr(C, n) = \begin{cases} \emptyset & , \text{ if } pth(C)(n) \notin \mathbb{CS} \\ \mathsf{ran}(\{k{+}1, \ldots, m{-}1\} \lhd pth(C)) & , \text{ else,} \end{cases} \tag{3.5}$$

where the indices k and m restricting the domain of $pth(C)$ are defined by

$$\begin{aligned} k &== max\{i : \mathbb{Z} \mid i < n \wedge pth(C)(i) \in \mathbb{L}\} \\ m &== min\{j : \mathbb{Z} \mid j > n \wedge pth(C)(j) \in \mathbb{L}\}. \end{aligned}$$

In formula (3.5), with the Z operator $\lhd$, we restrict the domain of $pth(C)$ to the sequence of crossing segments through the intersection. The Z range operator ran is only used to ensure the result is a set of segments, not a sequence.

As a result of the time transition, the position $pos'(C)$ of car C can either be on the same segment as before, or on a new lane or crossing segment, as determined with Alg. 1. If $curr(C) \neq curr'(C)$, i.e. if the current path segment of C changed, the set $res(C)$ for reservations on lane segments and the set $cres(C)$ for reservations on crossing segments is updated. Note that $res'(C)$ is updated with the empty set, iff $pth(C)(next(C)) \in \mathbb{CS}$. The index $curr(C)$ is changed to the calculated new index. The speed of all cars is updated according to their current acceleration.

In Defs. 23 and 24, we adapt the transitions from [HLOR11] for creating (resp. removing) a claim or reservation for a neighbouring lane segment or for a lane segment following an intersection. These transitions are only allowed on road segments between two intersections as they only apply for lane segments. We start with the *claim transition* and its dedicated *withdraw claim transition.*

Definition 23 (*Claim* and *withdraw claim* transition)**.** *For all $C \in \mathbb{I}$ and for $n \in \mathbb{L}$ the following transitions hold:*

$$
\begin{aligned}
\mathcal{TS} \xrightarrow{\mathsf{c}(C,n)} \mathcal{TS}' \quad \Leftrightarrow \quad & \mathcal{TS}' = (\mathcal{N}, res, clm', cres, cclm, pth, curr, pos, spd, acc) \\
& \wedge \#clm(C) = \#cclm(C) = 0 \\
& \wedge (\#res(C) = 1 \vee \#cres(C) > 0) \\
& \wedge (\exists l \colon res(C) \bullet (l,n) \in E_u \vee (n,l) \in E_u \vee \\
& \quad \exists cs \colon cres(C) \bullet (cs,n) \in E_d) \\
& \wedge clm' = clm \oplus \{C \mapsto \{n\}\} \\
\mathcal{TS} \xrightarrow{\mathsf{wd}\ \mathsf{c}(C)} \mathcal{TS}' \quad \Leftrightarrow \quad & \mathcal{TS}' = (\mathcal{N}, res, clm', cres, cclm, pth, curr, pos, spd, acc) \\
& \wedge clm' = clm \oplus \{C \mapsto \emptyset\}
\end{aligned}
$$

The application condition for setting a new claim for an arbitrary car C is that neither a claim on a lane segment $clm(C)$ nor a claim on a crossing segment $cclm(C)$ exists. In our urban traffic scenario, we observe two possible, but not compulsory mutually exclusive cases for claims on lane segments:

1. A claim on a neighbouring lane segment n, while $\#res(C) = 1$, i.e. while driving on a lane segment between two intersections. This is a claim to perform a lane change for an overtaking manoeuvre, as ensured by an existing undirected edge $(l,n) \in E_u$ or $(n,l) \in E_u$ in the road network.

2. A claim on a lane segment n, while $\#cres(C) > 0$, i.e. while being on a crossing. This is a claim to leave an intersection and enter a lane segment which is connected to the intersection, as ensured by an existing directed edge $(cs,n) \in E_d$.

With $\mathsf{c}(C,n)$ the set of claimed lane segments for car C is updated to $\{n\}$ while the withdrawal transition empties the set of claimed lane segments.

Definition 24 (*Reservation* and *withdraw reservation* transitions)**.** *For all $C \in \mathbb{I}$ and for $n \in \mathbb{L}$ the following transitions hold:*

$$\begin{aligned}
\mathcal{TS} \xrightarrow{\mathsf{r}(C)} \mathcal{TS}' \quad &\Leftrightarrow \quad \mathcal{TS}' = (\mathcal{N}, res', clm', cres, cclm, pth, curr, pos, spd, acc) \\
&\quad \wedge \#clm(C) > 0 \wedge clm' = clm \oplus \{C \mapsto \emptyset\} \\
&\quad \wedge res' = res \oplus \{C \mapsto res(C) \cup clm(C)\} \\
\mathcal{TS} \xrightarrow{\mathsf{wd\ r}(C,n)} \mathcal{TS}' \quad &\Leftrightarrow \quad \mathcal{TS}' = (\mathcal{N}, res', clm, cres, cclm, pth, curr, pos, spd, acc) \\
&\quad \wedge res' = res \oplus \{C \mapsto \{n\}\} \wedge n \in res(C) \\
&\quad \wedge (\#res(C) = 2 \vee \#cres(C) > 0)
\end{aligned}$$

With the reservation transition $\mathsf{r}(C)$ an existing claim for car C is transformed into a reservation. With the conditions for $\mathsf{wd\ r}(C, n)$, car C is only then allowed to withdraw a reservation of a lane segment, if it is about to finish a lane change manoeuvre and thus $\#res(C) = 2$ or if it reserves some crossing segments and thus $\#cres(C) > 0$. Note that n is not the lane segment, for which the reservation is withdrawn, but the lane segment car C reserves after $\mathsf{wd\ r}(C, n)$.

With the following Definitions 25 and 26 we add four transitions which create (resp. remove) claims and reservations for crossing segments to pass through an intersection. For crossing claims and reservations, the traffic snapshot claims or reserves crossing segments for a car $C \in \mathbb{I}$ according to its path $pth(C)$. Thus, we do not have a parameter for crossing segments at the transition $\mathsf{cc}(C)$ in Def. 25.

Definition 25 (*Crossing claim* and *withdraw crossing claim* transition)**.** *We reuse the function nextCr from Def. 22, now with $curr(C) + 1$ as next index. With this, for all $C \in \mathbb{I}$ the following transitions hold:*

$$\begin{aligned}
\mathcal{TS} \xrightarrow{\mathsf{cc}(C)} \mathcal{TS}' \quad &\Leftrightarrow \quad \mathcal{TS}' = (\mathcal{N}, res, clm, cres, cclm', pth, curr, pos, spd, acc) \\
&\quad \wedge \#res(C) = 1 \\
&\quad \wedge \#clm(C) = \#cclm(C) = \#cres(C) = 0 \\
&\quad \wedge cclm' = cclm \oplus \{C \mapsto \{nextCr(C, curr(C) + 1)\}\} \\
\mathcal{TS} \xrightarrow{\mathsf{wd\ cc}(C)} \mathcal{TS}' \quad &\Leftrightarrow \quad \mathcal{TS}' = (\mathcal{N}, res, clm, cres, cclm', pth, curr, pos, spd, acc) \\
&\quad \wedge cclm' = cclm \oplus \{C \mapsto \emptyset\}
\end{aligned}$$

Transition $\mathsf{cc}(C)$ only allows for a crossing claim, if $\#res(C) = 1$, i.e., if there is no active lane change manoeuvre for car C. Further on, with $\#clm(C) = 0$, we forbid a crossing claim with an active claim on a lane segment. As the next segment in $pth(C)$ has to be a crossing segment to commit a crossing claim, we derive all needed crossing segments with $nextCr(C, curr(C) + 1)$.

Note that as before in Def. 23 for setting a claim on a lane segment, we explicitly forbid to simultaneously commit a claim on a lane segment and a claim on a crossing segment,

because we only allow a car to either perform an overtaking manoeuvre on a road segment or a turn manoeuvre at an intersection.

Definition 26 (*Crossing reservation* and *withdraw crossing reservation* transition)**.** *For all $C \in \mathbb{I}$ the following transitions hold:*

$$\begin{aligned}
\mathcal{TS} \xrightarrow{\mathsf{rc}(C)} \mathcal{TS}' \quad &\Leftrightarrow \quad \mathcal{TS}' = (\mathcal{N}, res, clm, cres', cclm', pth, curr, pos, spd, acc) \\
&\quad \wedge \#res(C) = 1 \wedge \#cclm(C) > 0 \wedge \#clm(C) = 0 \\
&\quad \wedge cclm' = cclm \oplus \{C \mapsto \emptyset\} \\
&\quad \wedge cres' = cres \oplus \{C \mapsto cclm(C)\} \\
\mathcal{TS} \xrightarrow{\mathsf{wd}\ \mathsf{rc}(C)} \mathcal{TS}' \quad &\Leftrightarrow \quad \mathcal{TS}' = (\mathcal{N}, res, clm, cres', cclm, pth, curr, pos, spd, acc) \\
&\quad \wedge \#res(C) = 1 \wedge \#cres(C) > 0 \\
&\quad \wedge cres' = cres \oplus \{C \mapsto \emptyset\}
\end{aligned}$$

A crossing reservation for car C requires a reservation on a lane segment (i.e. the car is approaching an intersection) and a crossing claim. Then the crossing claim is transformed into a crossing reservation. The withdrawal transition requires the car to have a reservation on a lane segment to ensure that there still exists a reservation for car C. The last possible transition is the acceleration transition, where with $\mathsf{acc}(C, a)$ the value of $acc(C)$ is updated to a.

Definition 27 (*Acceleration* transition)**.** *For all $C \in \mathbb{I}$ and $a \in \mathbb{R}$ the following transition holds:*

$$\begin{aligned}
\mathcal{TS} \xrightarrow{\mathsf{acc}(C,a)} \mathcal{TS}' \quad &\Leftrightarrow \quad \mathcal{TS}' = (\mathcal{N}, res, clm, cres, cclm, pth, curr, pos, spd, acc') \\
&\quad \wedge acc' = acc \oplus \{C \mapsto a\}
\end{aligned}$$

Example 11 (Traffic snapshot evolution)*.* In the following we give an example for the evolution of the traffic snapshot $\mathcal{TS}_1$ from Example 10 to a future traffic snapshot $\mathcal{TS}_7$ using the transitions defined in Defs. 22 to 27.

$$\mathcal{TS}_1 \xrightarrow{\mathsf{t}_1} \mathcal{TS}_2 \xrightarrow{\mathsf{wd}\ \mathsf{rc}(C)} \mathcal{TS}_3 \xrightarrow{\mathsf{t}_2} \mathcal{TS}_4 \xrightarrow{\mathsf{cc}(E)} \mathcal{TS}_5 \xrightarrow{\mathsf{t}_3} \mathcal{TS}_6 \xrightarrow{\mathsf{rc}(E)} \mathcal{TS}_7.$$

When some time t_i passes for $i \in \{1, 2, 3\}$, all cars move according to their paths and speed (cf. Alg. 1). Let us focus on cars C and E, ignoring that after t_i time, car D might arrive at the intersection or that B possibly leaves the intersection.

First, some time t_1 passes, where all cars move according to their paths and speed. After t_1 time, we assume car C finished the crossing manoeuvre depicted in Fig. 3.1 and thus withdraws its crossing reservation with $\mathsf{wd}\ \mathsf{rc}(C)$. Again some time t_2 passes, after which we assume that car E arrived at the intersection and thus with $\mathsf{cc}(E)$ claims the upcoming crossing segments c_0, c_1 and c_2 according to $pth(E)$. As car C already

left the intersection, there is no potential for a collision of E with another car on the claimed crossing segments and thus after waiting t_3 time units, car E finally reserves the previously claimed crossing segments with $\mathsf{rc}(E)$ and enters the intersection. For reasons of brevity we omitted the process, where car C eliminates its reservation on lane segment 3. △

3.4 Virtual View

For examining functional properties like safety and liveness of our car manoeuvres, we restrict ourselves to finite parts of the traffic snapshot $\mathcal{TS}$ from the previous section. The intuition is that the functionality of a car depends only on its immediate surroundings. We formalise this by introducing a structure called *view* $V(E)$, which only contains the parts of lanes and crossing segments that lie within a certain range around the considered car E.

In the remainder of this section, we describe the challenges and our solution to derive a view for complex multi-lane intersections to reason in the logic we introduce in the next Sect. 3.5. We start with an informal introduction to concepts and give formal details for them later.

3.4.1 An Intuition on why and how to Straighten Views

In previous work [HLOR11, HLO13] 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 urban traffic, since the relation between lane segments is defined through the network $\mathcal{N}$ with the equivalence classes from Def. 17. If an intersection is within the considered range around car E, we deal with a bended view as cars are allowed to turn in any possible direction at the crossing (cf. $V(E)$ in Fig. 3.1). Original two-dimensional MLSL was created for logical reasoning on straight parallel lanes. Equally, our urban extension UMLSL, which we introduce in Sect. 3.5, cannot reason about lanes that go around the corner. As a first step, in [HS16], we construct a straight *virtual view* corresponding to the bended view from the urban road network $\mathcal{N}$ and the path $pth(E)$ of car E solely for 2-by-2 intersections. The goal of this straightened view is to reuse concepts from the country roads approach, where the set of lanes is split up into lanes in driving direction from "left to right" and lanes in the opposing driving direction. Consider Fig. 3.5, where the depicted virtual view corresponds to bended view $V(E)$ from Fig. 3.1.

This virtual view consists of one *virtual lane* from the path $pth(E)$ and one virtual lane from the opposing driving direction. These two virtual lanes are parallel and allow for reasoning with UMLSL. For this explicit view construction tailored solely for 2-by-2 intersections, we refer to [HS16].

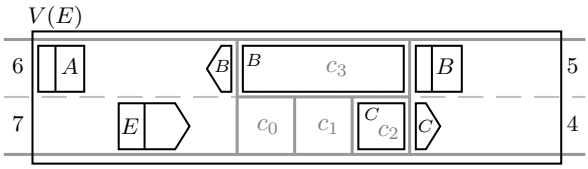

Figure 3.5: Virtual view with cars for view $V(E)$ from Fig. 3.1.

In this thesis, we consider the more general case of n-by-m intersections. This is because in Sect. 3.2, we already allow for an urban road network $\mathcal{N}$ with n-by-m intersections where arbitrary many lane segments n resp. m meet at an intersection (cf. [Sch18a]). For these complex intersections, our virtual view construction changes dramatically, compared with [HS16].

In general, we formally construct a view from the urban road network $\mathcal{N}$ (cf. Sect. 3.2), the current traffic snapshot $\mathcal{TS}$ (cf. Sect. 3.3), a real-valued interval $X = [a, b]$ for the horizontal extension and a considered car E, which we call *owner of the view*. We adapt the view definition from [HLOR11] as follows.

Definition 28 (View)**.** *For $\mathcal{TS} \in \mathbb{TS}$ defined over an urban road network $\mathcal{N}$, the* view *$V(E) = (L, X, E)$ owned by car $E \in \mathbb{I}$ contains*

- *a finite sequence $L \in \mathsf{seq}\,(\mathsf{seq}_{E_d}\,\mathcal{V} \cup \mathsf{seq}_{E_d^{\sim}}\,\mathcal{V})$,*
- *the space interval along the lanes $X = [a, b] \subseteq \mathbb{R}$ visible in $V(E)$, and*
- *the car identifier E itself.*

The remainder of this section covers how to retrieve the finite sequence L, which we call *virtual lanes*. Whereas in Fig. 3.5, the depicted view can be figuratively compared to unbending the view from the original traffic situation in Fig. 3.1, this procedure is more difficult e.g. for the traffic situation from Fig. 3.2. Consider Fig. 3.6, where an attempt is conducted to straighten the traffic situation from Fig. 3.2 analogously to the (informal) straightening process depicted in Fig. 3.5. How should the respective road segments $r_0 == \{10, 11\}$ on the left and $r_1 == \{6, 7, 8, 9\}$ on the right be connected using crossing segments?

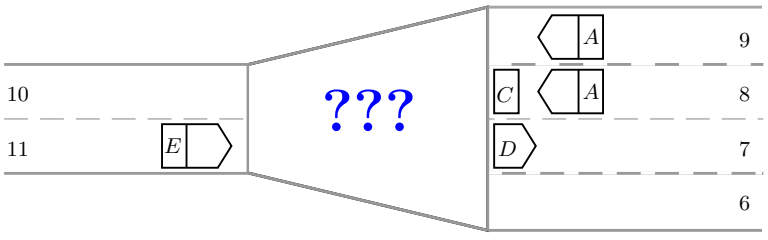

Figure 3.6: Unknown paths through the intersection for the traffic situation from Fig. 3.2.

Basically, the answer to this question is to consider all possible paths that cars can take for driving through the intersection from r_0 to r_1. For the example we observe four possible paths for turning left at the intersection from Fig. 3.2 which are highlighted in the topology depicted on the right-hand side of Fig. 3.4.

First, we briefly motivate this new construction method for straightening arbitrary complex intersections by retrieving all paths through an intersection. To cope with different numbers n and m of lane segments meeting at an intersection, we build several virtual views $V_i = (L_i, X, E)$ (cf. Def. 28) for one intersection. For this, we undertake the following steps, which are formally defined in Defs. 29 to 32 on the following pages:

1. As a cut-out of the path $pth(E)$, construct the finite *path through the intersection* $\pi_E \sqsubset pth(E)$ and abstract it to the *coarse path sequence* $\overrightarrow{\pi_C} \in \mathsf{seq}_{E_C} \mathcal{V}_C$ (cf. Defs. 19 and 29).

2. As refinements of $\overrightarrow{\pi_C}$ from step 1, build *all* finite *virtual lanes* through the intersection $\overrightarrow{\pi_E} \in \mathsf{seq}_{E_d} \mathcal{V}$ in driving direction of $pth(E)$ and virtual lanes $\overleftarrow{\pi_E} \in \mathsf{seq}_{E_d^\sim} \mathcal{V}$ in opposite driving direction (cf. Def. 30).

3. Bundle virtual lanes from step 2 to sequences L_i we call *parallel virtual lanes* (cf. Def. 31).

4. Finally build one *virtual view* $V_i = (L_i, X, E)$ for each L_i from step 3 with the same horizontal extension $X = [a, b]$ along the lanes. The set of all V_i is named *multi-view* $V_M(E, \mathcal{TS})$ (cf. Def. 32).

3.4.2 Virtual Lanes and Virtual View

We introduce the construction method for the coarse excerpt $\overrightarrow{\pi_C}$ from step 1 and the virtual lanes $\overrightarrow{\pi_E}$ and $\overleftarrow{\pi_E}$ from step 2 with the following example.

Example 12 (Coarse path and virtual lanes.). Consider the 2-by-4 intersection from Fig. 3.2 on p. 39. We assume, car E plans to turn left, with *path through the intersection* $\pi_E = \langle 11, c_0, c_1, c_2, c_3, c_7, 6\rangle$. The corresponding *coarse path excerpt* is $\overrightarrow{\pi_C} = \langle r_{left}, cr, r_{above}\rangle$, where r_{left} and r_{above} are the respective road segments left and above of the intersection cr. With the connections from the topology on the right-hand side of Fig. 3.2, we retrieve four possible finite *virtual lanes* from $\overrightarrow{\pi_C}$, two in E's driving direction ($\overrightarrow{\Pi}_E$) and two in the opposing driving direction ($\overleftarrow{\Pi}_E$):

$$\overrightarrow{\Pi}_E = \{\langle 11, c_0, c_1, c_2, c_3, c_7, 6\rangle, \langle 11, c_0, c_1, c_2, c_6, 7\rangle\}$$
$$\overleftarrow{\Pi}_E = \{\langle 10, c_4, c_5, 8\rangle, \langle 10, c_4, 9\rangle\}.$$

We exclude unnecessarily long virtual lanes, e.g. virtual lanes with loops in it, and thus retrieve no further virtual lanes. △

Following the previous example, we start from the *finite path through the intersection* $\pi_E \sqsubset pth(E)$ for car E. Besides the intersection itself, π_E also contains the respective lane segments directly before and after the intersection. We abstract π_E to $\overrightarrow{\pi_\mathcal{C}} \in \mathsf{seq}_{E_\mathcal{C}}\, \mathcal{V}_\mathcal{C}$ using the equivalence classes of elements from π_E, whereas $\overrightarrow{\pi_\mathcal{C}}$ consists of three elements: The current road segment ($\overrightarrow{\pi_\mathcal{C}}(1)$), the next crossing ($\overrightarrow{\pi_\mathcal{C}}(2)$) and the road segment car E plans to drive on in the future ($\overrightarrow{\pi_\mathcal{C}}(3)$).

Definition 29 ((Coarse) path through the intersection)**.** *Consider the owner of the view* $E \in \mathbb{I}$ *with* $pth(E)$ *and current path element* $pth(E)(curr(E)) \in \mathbb{L}$ *and* $pth(E)(curr(E)+1) \in \mathbb{CS}$*. Let*

$$k == min\{i : \mathbb{Z} \mid i > curr(E) \wedge pth(E)(i) \in \mathbb{L}\}. \tag{3.6}$$

Then $E's$ path through the intersection $\pi_E \sqsubset pth(E)$ *is defined by*

$$\pi_E = \{curr(E), \ldots, k\} \lhd pth(E). \tag{3.7}$$

We derive the coarse path through the intersection $\overrightarrow{\pi_\mathcal{C}} \in \mathsf{seq}_{E_\mathcal{C}}\, \mathcal{V}_\mathcal{C}$ *by*

$$\overrightarrow{\pi_\mathcal{C}} = \langle f_\mathbb{L}(\pi_E(1)), f_\mathbb{CS}(\pi_E(2)), f_\mathbb{L}(\pi_E(\#\pi_E))\rangle. \tag{3.8}$$

With formula (3.6), we get the index k of the first lane segment after the intersection and with formula (3.7), we restrict the domain of $pth(E)$ to indices from $curr(E)$ to k. For part (3.8), we only abstract the first crossing segment $\pi_E(2)$ from π_E to the corresponding intersection $f_\mathbb{CS}(\pi_E(2))$, as the other crossing segments in π_E relate to the same connected component and repeated occurrences of elements are not allowed in $\overrightarrow{\pi_\mathcal{C}}$ by definition of $\mathsf{seq}_{E_\mathcal{C}}\, \mathcal{V}_\mathcal{C}$.

As suggested in Example 12, for the virtual lanes we have to find *all* possible paths through the intersection starting from the first coarse segment $\overrightarrow{\pi_\mathcal{C}}(1)$ in $\overrightarrow{\pi_\mathcal{C}}$ and ending at its last element $\overrightarrow{\pi_\mathcal{C}}(3)$. As one of the sanity conditions from Def. 16 demands that every lane segment is connected with a crossing segment, we retrieve *for each* lane segment from the two road segments in $\overrightarrow{\pi_\mathcal{C}}$ *at least one* respective virtual lane.

We distinguish between virtual lanes $\overrightarrow{\pi_E} \in \mathsf{seq}_{E_d}\, \mathcal{V}$ in driving direction of $pth(E)$ leading *forwards* from $\overrightarrow{\pi_\mathcal{C}}(1)$ to $\overrightarrow{\pi_\mathcal{C}}(3)$ and virtual lanes $\overleftarrow{\pi_E} \in \mathsf{seq}_{E_d^\sim}\, \mathcal{V}$ in the opposite driving direction, *backwards* from $\overrightarrow{\pi_\mathcal{C}}(1)$ to $\overrightarrow{\pi_\mathcal{C}}(3)$.

Definition 30 (Virtual lanes)**.** *For* $E \in \mathbb{I}$ *with* $\overrightarrow{\pi_\mathcal{C}}$ *from Def. 29, we derive the set of* virtual lanes $\overrightarrow{\Pi}_E \subseteq \mathsf{seq}_{E_d}\, \mathcal{V}$ *in driving direction of* $pth(E)$ *with*

$$\begin{aligned}\overrightarrow{\Pi}_E = \{\overrightarrow{\pi_E} : \mathsf{seq}_{E_d}\, \mathcal{V} \mid\ & f_\mathbb{L}(\overrightarrow{\pi_E}(1)) = \overrightarrow{\pi_\mathcal{C}}(1) \wedge f_\mathbb{L}(\overrightarrow{\pi_E}(\#\overrightarrow{\pi_E})) = \overrightarrow{\pi_\mathcal{C}}(3) && (3.9)\\ & \wedge \exists s : \mathsf{seq}_{E_d}\mathbb{CS} \bullet \forall cs : s \bullet f_\mathbb{CS}(cs) = \overrightarrow{\pi_\mathcal{C}}(2) && (3.10)\\ & \wedge \langle \overrightarrow{\pi_E}(1), s(1)\rangle \subset \overrightarrow{\pi_E} \wedge \langle s(\#s), \overrightarrow{\pi_E}(\#\overrightarrow{\pi_E})\rangle \subset \overrightarrow{\pi_E}\}. && (3.11)\end{aligned}$$

We define the set of inverted virtual lanes $\overleftarrow{\Pi}_E \subseteq \mathsf{seq}_{E_d^\sim}\,\mathcal{V}$ *in the opposing driving direction symmetrically to the previous definition with*

$$\begin{aligned}\overleftarrow{\Pi}_E = \{\overleftarrow{\pi_E} : \mathsf{seq}_{E_d^\sim}\,\mathcal{V} \mid\ & f_{\mathbb{L}}(\overleftarrow{\pi_E}(1)) = \overrightarrow{\pi_{\mathcal{C}}}(1) \wedge f_{\mathbb{L}}(\overleftarrow{\pi_E}(\#\overleftarrow{\pi_E})) = \overrightarrow{\pi_{\mathcal{C}}}(3) \\ & \wedge \exists s : \mathsf{seq}_E^{-1}\mathbb{CS} \bullet \forall cs : s \bullet f_{\mathbb{CS}}(cs) = \overrightarrow{\pi_{\mathcal{C}}}(2) \\ & \wedge \langle \overleftarrow{\pi_E}(1), s(1)\rangle \subset \overleftarrow{\pi_E} \wedge \langle s(\#s), \overleftarrow{\pi_E}(\#\overleftarrow{\pi_E})\rangle \subset \overleftarrow{\pi_E}\}.\end{aligned} \tag{3.12}$$

The set of all virtual lanes $\overleftrightarrow{\pi_E}$ *is defined by* $\overleftrightarrow{\Pi}_E == \overrightarrow{\Pi}_E \cup \overleftarrow{\Pi}_E$.

With formula (3.9) we ensure that the first (resp. last) element of each virtual lane $\overrightarrow{\pi_E}$ is a lane segment from $\overrightarrow{\pi_{\mathcal{C}}}(1)$ (resp. $\overrightarrow{\pi_{\mathcal{C}}}(3)$). Formula (3.10) and (3.11) demand that there exists a sequence solely of crossing segments $\mathbb{CS}$ connecting the two lane segments $\overrightarrow{\pi_E}(1)$ and $\overrightarrow{\pi_E}(\#\overrightarrow{\pi_E})$. Formula (3.12) analogously builds the virtual lanes $\overleftarrow{\pi_E}$ backwards through the intersection.

We consider the special case of an intersection cr with a one-way road segment r_{one} meeting a two-way road segment r_{two}. For r_{one} we observe *either* an edge $(r_{one}, cr) \in E_{\mathcal{C}}$ *or* $(cr, r_{one}) \in E_{\mathcal{C}}$ in the underlying topology $\mathcal{N}_{\mathcal{C}}$. We thus cannot connect every lane segment from r_{two} with a lane segment from r_{one} with a virtual lane from $\overleftrightarrow{\Pi}_E$ from Def. 30. However, since we cannot just ignore these unconnected lane segments from r_{two} for safety reasons, we extend $\overleftrightarrow{\Pi}_E$ from Def. 30 by using additional *placeholder lane segments* $l_p \in \mathbb{Z}\backslash\mathbb{L}$. With considering these separately from actual lane segments from $\mathbb{L}$, we ensure that l_p is not part of any path from $pth_{\mathcal{N}}$. Thus no car can ever enter l_p and it is used *solely* as a placeholder for building virtual lanes.

Remark 1 (Special case: one-way roads)**.** *Consider an intersection of a one-way road segment* r_{one} *with a two-way road segment* r_{two}*. For all elements* $\pi \in r_{two}$ *not contained in any virtual lane* $\overleftarrow{\pi_E} \in \overleftrightarrow{\Pi}_E$*, we add a virtual lane* $\overleftrightarrow{\pi_p} == \langle l_p\rangle \frown s_{\mathbb{CS}} \frown \langle \pi\rangle$ *to* $\overleftarrow{\Pi}_E$*, where* $l_p \in \mathbb{Z}\backslash\mathbb{L}$ *is a* placeholder lane segment *and* $s_{\mathbb{CS}} \in \mathsf{seq}_{E_d^\sim}\,\mathbb{CS}$*. The case for intersections of a two-way road segment with a one-way road segment is handled symmetrically.*

Remember that we aim to build a virtual view V_i containing a set of *parallel* and *neighbouring* lanes L_i. However, we cannot simply align the virtual lanes from Def. 30 to parallel neighbouring lanes to create a view like the one for the 2-by-2 intersection in Fig. 3.5. With n-by-m intersections, there exists not only one refined path per direction through the crossing, but possibly several (cf. virtual lanes from example 12). Thus, it is possible that one lane segment l is contained in several virtual lanes from $\overleftrightarrow{\Pi}_E$.

Example 13 (Parallel virtual lanes). Consider again Example 12 relating to Fig. 3.7, where lane segments 10 and 11 are each contained twice throughout all four virtual lanes. Including lane segment 11 twice in our virtual view would result in a duplication of the

reservation of E. This would wrongly indicate a lane change for E. We thus build the sequences of parallel virtual lanes

$$
\begin{aligned}
L_1 &= \langle\langle 11, c_0, c_1, c_2, c_3, c_7, 6\rangle, \langle 10, c_4, 9\rangle\rangle \ , \\
L_2 &= \langle\langle 11, c_0, c_1, c_2, c_6, 7\rangle, \langle 10, c_4, c_5, 8\rangle\rangle \ , \\
L_3 &= \langle\langle 11, c_0, c_1, c_2, c_3, c_7, 6\rangle, \langle 10, c_4, c_5, 8\rangle\rangle \ , \\
L_4 &= \langle\langle 11, c_0, c_1, c_2, c_6, 7\rangle, \langle 10, c_4, 9\rangle\rangle
\end{aligned}
$$

each containing one different virtual lane from $\overrightarrow{\Pi}_E$ and $\overleftarrow{\Pi}_E$. The first two of these parallelised virtual lanes are depicted in Fig. 3.7. △

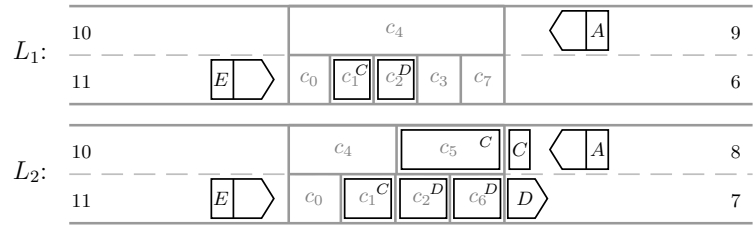

Figure 3.7: Parallel virtual lanes L_1 and L_2 for a left-turn view for car E in Fig. 3.2.

In each of the parallel lanes L_i defined in the following Def. 31 we maintain the spatial structure in road segment r_{curr} (in the example road segment $\{10, 11\}$), but build up the remaining parts with the virtual lanes $\overrightarrow{\pi_E}$ resp. $\overleftarrow{\pi_E}$ from Def. 30. While maintaining the spatial structure in r_{curr}, we lose information about vertical spatial aspects on the intersection and in the road segment after the intersection. Consider for example car A in Fig. 3.7. We still perceive two reservations in total for car A, one in each set of parallel lanes L_1 resp. L_2, but we lost the property that those reservations were neighbouring (cf. Fig. 3.2). This is no problem for the later safety considerations with our crossing controller (cf. Sect. 5.3). For collision freedom, we only need the information, *that* there is a car occupying a specific space or approaching an intersection, not *which* car exactly it is. It is also sufficient, to know *that* a car occupies a crossing segment. We still intend to generalise this view construction for future considerations (cf. Sect. 8.4).

For the following Def. 31, recall that two lane segments from the same road segment are neighbouring iff they are connected with an undirected edge in the set E_u in the road network $\mathcal{N}$.

Definition 31 (Parallel virtual lanes)**.** *Consider a network $\mathcal{N}$ with undirected edges E_u, car identifier $E \in \mathbb{I}$, $\overrightarrow{\pi_C}$ from Def. 29 and $\overleftrightarrow{\Pi}_E$ from Def.30 and Remark 1. We define the set* $\mathbf{L}$ *of* sequences of parallel virtual lanes L *by*

$$
\begin{aligned}
\mathbf{L} := \{ & L \in \mathsf{seq}\ \overleftrightarrow{\Pi}_E \mid \forall l \colon \overrightarrow{\pi_C}(1) \bullet \exists_1 i \colon 1, .., \#L \bullet l = L(i)(1) && (3.13)\\
& \wedge \forall j \colon 1..\#L - 1 \bullet (L(j)(1), (L(j+1)(1)) \in E_u && (3.14)\\
& \wedge \exists m \colon 0..\#L \bullet (\forall j \colon 1..m; k \colon m{+}1..\#L \bullet L(j) \in \overrightarrow{\Pi}_E \wedge L(k) \in \overleftarrow{\Pi}_E)\}. && (3.15)
\end{aligned}
$$

For formulae (3.13) and (3.14), we recall from Def. 30 that the first element $\overleftrightarrow{\pi}_E(1)$ in all virtual lanes $\overleftrightarrow{\pi}_E \in \overleftrightarrow{\Pi}_E$ is a lane segment from the current road segment (represented by $\overrightarrow{\pi_{\mathcal{C}}}(1)$). Thus, (3.13) states that each lane segment from that road segment is represented *exactly once* in each $L \in \mathbf{L}$. With formula (3.14), we define the previously described property of maintaining the structure of $\overrightarrow{\pi_{\mathcal{C}}}(1)$; for any two neighbouring parallel lanes $L(j), L(j{+}1)$, we require an undirected edge from the set E_u between their respective first elements. With (3.15), we finally define that there exists a value $m \in \mathbb{N}$ with $m \leq \#L$, which splits up the sequence L in lanes from $\overrightarrow{\Pi}_E$ with the same driving direction as $pth(E)$ and in lanes from $\overleftarrow{\Pi}_E$ in the opposite direction.

The parallel lanes L obtained through Def. 31 are finite, as they consist of finite virtual lanes from Def. 30. However, virtual lanes can still be arbitrarily long. We thus restrict the horizontal extension of the parallel lanes L by defining the size of the extension $X = [a, b]$ of a view along the virtual lanes. It is consistent with reality to assume that there exists a maximum velocity v_{max} for all cars, where no car may drive faster than v_{max}. We use a sufficiently large horizon h, such that any car driving at v_{max} can come to a complete standstill within this distance. We demand that h is big enough, to include a car approaching the intersection that could already have a claim or reservation for the intersection. We consider the same extension $X = [pos(E) - h, pos(E) + h]$ for each sequence of lanes L_i from Def. 31.

Definition 32 (Virtual multi-view)**.** *For a car identifier $E \in \mathbb{I}$, $\mathcal{TS} \in \mathbb{TS}$, parallel virtual lanes $L_i \in \mathbf{L}$ from Def. 31, the* horizon $h \in \mathbb{R}_+$ *and the interval $X = [pos(E){-}h, pos(E){+}h]$, one respective* virtual view V_i *of E is defined by $V_i(E, \mathcal{TS}) = (L_i, X, E)$. All virtual views are collected in the* multi-view $V_M(E, \mathcal{TS}) = (V_1(E, \mathcal{TS}), \ldots, V_n(E, \mathcal{TS}))$*, where $n = \#\mathbf{L}$.*

Example 14 (Virtual multi-view). For $i \in \{1, 2, 3, 4\}$, for each of the sequences of virtual lanes L_i from Example 13 (cf. Fig. 3.7), with $X = [pos(E) - h, pos(E) + h]$, we retrieve a respective virtual view $V_i(E, \mathcal{TS}) = (L_i, X, E)$. With this, we obtain the multi-view $V_M(E, \mathcal{TS}) = (V_1(E, \mathcal{TS}), V_2(E, \mathcal{TS}), V_3(E, \mathcal{TS}), V_4(E, \mathcal{TS}))$. △

We adapt the *chopping operations of views* from [LH15] to the new notion of views in the following definition. While horizontal chopping $\oplus$ is analogous to ITL [Mos85], vertical chopping $\ominus$ is defined for arbitrarily many lanes.

Definition 33 (Chop operations on views)**.** *Let $V = (L, X, E)$ be a view of traffic snapshot $\mathcal{TS} \in \mathbb{TS}$. Then we have $V = V_1 \ominus V_2$ for views V_1 and V_2, iff $V_1 = (L_1, X, E)$ and $V_2 = (L_2, X, E)$ with $L = L_1 \frown L_2$ and $L_1 \cap L_2 = \emptyset$, where L_1 and L_2 both are sequences of parallel lanes as of Def. 31. Furthermore, $V = V_1 \oplus V_2$, iff $V = (L, [r, t], E)$ and $\exists s \in [r, t]$ such that $V_1 = (L, [r, s], E)$ and $V_2 = (L, (s, t], E)$.*

3.4.3 Perception of Cars through Sensors

So far in this section, we specified how we internally represent a cut-out of an urban road network as a virtual view, consisting of parallelised virtual lanes. Before introducing our logic Urban Multi-lane Spatial Logic to reason about traffic situations in such a view, we need to specify *what* a car perceives in its virtual view. As stated in Sect. 3.1 on p. 37, we use two different concepts of knowledge about *what* a car perceives in its virtual view $V_M(E, \mathcal{TS})$. However, we now address *how* the car E retrieves this knowledge. Recall that the physical size of a car together with its braking distance is called safety envelope.

To estimate the size of (parts of) other cars the car E perceives in its view, we model sensor capabilities by introducing a car dependent abstract *sensor function* Ω_E. This function returns the *size of a car* as perceived by the sensors of E. For now, let this *size* comprise the whole safety envelope of cars, meaning their physical size together with their braking distances. We explain different concepts of what is contained in the safety envelope of a car later in Sect. 5.2, before we introduce our controllers.

Definition 34 (Sensor function)**.** *For $E \in \mathbb{I}$ with virtual view $V_i(E, \mathcal{TS})$ and an arbitrary car $C \in \mathbb{I}$, the sensor function $\Omega_E : \mathbb{I} \times \mathbb{TS} \to \mathbb{R}_+$ yields the size of C as perceived in view $V_i(E, \mathcal{TS})$ by E's sensors.*

For urban traffic, we consider *crossing manoeuvres* at intersections and build virtual views only if the ego car E approaches an intersection. As indicated earlier, road segments between intersections are comparable to country roads, where for *lane change manoeuvres* on these we refer to [HLO13]. Such a lane change manoeuvre on roads with opposing traffic means that a car may temporarily drive against driving direction on *lane segments*. As it would go beyond the scope of this thesis, we do not introduce how cars that drive against driving direction are perceived by car E, but refer to [HLO13] for this. However, it is planned for future work to formally integrate the country roads approach from [HLO13] and the urban traffic approach from this thesis (cf. Sect. 8.4).

In our urban traffic scenario, it is not sufficient to solely know the size $\Omega_E(C)$ of a car C as given through Def. 34, as the visible parts of car C can be distributed over several lane and crossing segments. In Alg. 2, we thus calculate the *set of visible segments* $seg_V(C)$ occupied by a car C, where one element $(s_i, X_i) \in seg_V(C)$ contains a lane or crossing segment $s_i \in \mathcal{V}$ and the interval of space X_i car C occupies on it. We only consider those parts of C, which are actually visible in one virtual view V_i. Remember that in Sect. 3.2 in Def. 14 for the road network $\mathcal{N}$, we consider a real size $\omega(v)$ for each node $v \in \mathcal{V}$. Using the size of car C as given by Def. 34 and starting at $pth(C)(curr(C))$, we introduce Alg. 2, which subsequently calculates the real intervals of space that car C occupies on lane or crossing segments along $pth(C)$. Note that Alg. 2 uses a comparable procedure as Alg. 1 from Sect. 3.3, calculating the reached position of a car C after t time units. However, the results in Alg. 1 differ greatly from those in the following Alg. 2.

Algorithm 2 (Visible segments of cars in a view)**.** *Consider $E \in \mathbb{I}$ with virtual view $V_i(E, \mathcal{TS}) = (L_i, X, E)$ and an arbitrary car $C \in \mathbb{I}$ with path $pth(C)$, index $curr(C)$ and position $pos(C)$.*

visSeg$_{V_i(E,\mathcal{TS})}$ (C) {
  $i := 0$;
  $seg_V(C) := \emptyset$;
  $s_0 := pth(C)_{curr(C)}$;
  $a_0 := pos(C)$;
  $size_0 := \Omega_E(C, \mathcal{TS})$;
  while (**size$_\mathbf{i}$** $> \mathbf{0}$)
    if $(s_i \in \mathbb{CS})$
      $b_i := \omega(s_i)$
    else;
      $b_i := min(a_i + size_i,\ \omega(s_i))$
    fi;
    $X_i := [a_i,\ b_i]$;
    $size_{i+1} = size_i - (b_i - a_i)$;
    if $(size_{i+1} > 0)$
      $a_{i+1} := 0$;
      $s_{i+1} := pth(C)_{curr(C)+i+1}$
    fi;
    if $(\exists \overleftrightarrow{\pi} : \overleftrightarrow{\Pi} \bullet s_i \in \overleftrightarrow{\pi})$
      $seg_V(C) := seg_V(C) \cup \{(s_i, X_i)\}$
    fi;
    $i := i + 1$
  end while;
  return $(seg_V(C))$; }

The case where C drives against the direction of the considered lane segment, i.e. during a lane change manoeuvre, is handled symmetrically; instead of adding the size, we have to subtract it.

In the first part of the algorithm, we calculate the value of b_i to estimate the interval $X_i = [a_i,\ b_i]$ in which car C is visible on segment s_i. If $size_i$ does not exceed the size of the current segment s_i, we have $b_i := a_i + size_i$. If the size of s_i is exceeded, or if $s_i \in \mathbb{CS}$, we set $b_i := \omega(s_i)$. We then calculate the remaining size $size_{i+1}$ of C's safety envelope and set a_{i+1} to 0, as on next segment s_{i+1}, the visible part of car C will again start at 0. We add only those segments s_i to our set of visible segments $seg_V(C)$ that are part of one of the virtual lanes of car E. The algorithm ends, if $size_i \leq 0$. As in every iteration we

id	$pth(id)$	$res(id)$	$pos(id)$	$\Omega_E(id)$
A	$\langle \ldots, c_3, 6, c_7, \ldots \rangle$	$\{6\}$	60	20
B	$\langle \ldots, 5, c_3, 6, \ldots \rangle$	$\{5\}$	70	20
C	$\langle \ldots, 3, c_2, 4, \ldots \rangle$	$\{3\}$	95	20

Table 3.1: Exemplary traffic snapshot and sensor function values for the traffic situation from Fig. 3.1.

i	$size_i$	s_i	a_i	b_i	$seg_V(B)$
0	$\Omega_E(B) = 20$	5	70	$min(90, 80) = 80$	$\{(5, [70, 80])\}$
1	$20 - (10) = 10$	c_3	0	$\omega(c_3) = 5$	$\{(5, [70, 80]), (c_3, [0, 5])\}$
2	$10 - (10) = 0$	6	0	$min(10, 100) = 10$	$\{(5, [70, 80]), (c_3, [0, 5]), (6, [0, 10])\}$

Table 3.2: Calculation of visible segments for car E using Alg. 2 and the values from Fig. 3.1.

have $b_i \geq a_i$ by definition of b_i, we observe that with $size_{i+1} = size_i - (b_i - a_i)$ the value of $size_i$ indeed decreases with every iteration and the algorithm actually terminates.

Example 15 (Visible segments of cars). Consider again the traffic situation from Fig. 3.1 with exemplary traffic snapshot and sensor function values for cars A, B and C as given in Table 3.1. We calculate the visible segments $seg_V(B)$ of car B from the viewpoint of car E with Alg. 2. For the size $\omega(s_i)$ of lane and crossing segments s_i, consider the corresponding topology in Fig. 3.3. The stepwise calculation of $seg_V(B)$ is captured in Table 3.2. The result of Alg. 2 is $seg_V(B) = \{(5, [70, 80]), (c_3, [0, 5]), (6, [0, 10])\}$. △

3.5 Urban Multi-lane Spatial Logic

In the previous section, we constructed our virtual standard view $V_M(E)$ for car E and estimated the parts of other cars that E perceives in its view. For logical reasoning about traffic situations in $V_M(E)$, we introduce *Urban Multi-lane Spatial Logic* (UMLSL) in this section. Recall that we introduced the original MLSL from [HLOR11] and its extension by length measurement for country roads from [HLO13] in Sects. 2.2 and 2.4. With UMLSL, we can e.g. examine if a space on a neighbouring (virtual) lane is free, or if a car approaches an intersection.

In the following paragraphs, we specify the syntactic extension of the logics from [HLOR11, HLO13] to UMLSL. After that, we define the evaluation of UMLSL formulae over a traffic snapshot $\mathcal{TS}$ and a (virtual) view V. Note, that the intention of our UMLSL approach is to re-use as many concepts as possible from the original MLSL, only extending or modifying it where absolutely necessary.

Syntax

For the syntax extension to UMLSL, we use the basic MLSL from Def. 6, p. 21. We also use a part of the extension by length measurement introduced for country roads; however, where the there described MLSL extension includes real-valued terms for the length measurement, for our purposes real-valued constants $r \in \mathbb{R}$ are sufficient. With this, we allow for a real-valued length atom denoted by $\ell = r$ in UMLSL. For real variables we again use the set RVar from Sect. 2.4.

We do not need specific variables for crossing segments and also do not introduce a specific set of lane variables, as $\mathbb{L} \subset \mathbb{N} \subset \mathbb{R}$ and thus RVar also ranges over $\mathbb{L}$. We again use the set CVar for car variables and assume $\mathrm{RVar} \cap \mathrm{CVar} = \emptyset$ and now consider $\mathrm{Var} == \mathrm{RVar} \cup \mathrm{CVar}$.

Besides length measurement, UMLSL also extends MLSL by the spatial atom *cs* to represent crossing segments. Hereby, we can, e.g., state that car E occupies a crossing segment ($cs \wedge re(\mathrm{ego})$) or that a crossing segment is free ($cs \wedge free$). We can also distinguish if a reservation is on a lane with $re(\mathrm{ego}) \wedge \neg cs$.

Definition 35 (Syntax of UMLSL)**.** *Consider* $c \in \mathrm{CVar}$, $r \in \mathrm{RVar}$ *and* $u, v \in \mathrm{Var}$. *The syntax of the* Urban Multi-lane Spatial Logic *extends Def. 6 by the atomic formulae cs and* $\ell = r$ *with* $r \in \mathrm{RVar}$, *such that atomic* UMLSL formulae *are defined by*

$$\mathsf{a} ::= cs \mid true \mid u = v \mid \ell = r \mid free \mid re(c) \mid cl(c).$$

With that, an arbitrary UMLSL *formula* ϕ_U *is formalised by*

$$\phi_U ::= \mathsf{a} \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \exists v\colon \phi_1 \mid \phi_1 \frown \phi_2 \mid \begin{smallmatrix}\phi_2\\ \phi_1\end{smallmatrix}.$$

We denote the set of all UMLSL formulae by $\Phi_{\mathbb{U}}$.

Note that our use of existential quantification is automatically bound to the type of car variables from CVar and thus we do not specify any type in the UMLSL formula itself.

Before we define the semantics of UMLSL formulae, we adapt the *valuation function* from preliminary Sect. 2.2 ν for variables from the now extended set Var.

Definition 36 (Valuation and modification)**.** *Consider the set of variables* $\mathrm{Var} = \mathrm{CVar} \cup \mathrm{RVar}$. *A* valuation *is a function* $\nu\colon \mathrm{Var} \to \mathbb{I} \cup \mathbb{R}$ *respecting the types of variables. We* modify *the valuation of a variable* $v \in \mathrm{Var}$ *with* $\nu \oplus \{v \mapsto \alpha\}$, *where the value of* v *is modified to* α.

Example 16 (UMLSL formula and valuation). Consider the view $V(E)$ of car E in Fig. 3.1 and its corresponding straight virtual view from Fig. 3.5, together with the valuation of variables $\nu(\mathrm{ego}) = E$ and $\nu(c) = C$. In a part of this view, the UMLSL formula $\phi \equiv re(\mathrm{ego}) \frown free \frown (cs \wedge free) \frown (cs \wedge re(c))$ holds. Here, $re(\mathrm{ego})$ is the space

car E *reserves* on lane segment 7, the atom *free* represents the free space in front of car E, the part $cs \wedge free$ stands for the unoccupied space on crossing segments c_0 and c_1 and with $cs \wedge re(c)$ the part car C reserves on segment c_2 is formalised. △

Semantics

While the syntax of UMLSL is not very different from previous versions of MLSL [HLOR11, HLO13], the *semantics* of the spatial atoms change dramatically, to accommodate for virtual views (cf. Sect. 3.4). Further on, we now use the atoms $cl(c)$ and $re(c)$ for both, lane and crossing claims and reservations. The following semantics for UMLSL formulae is defined over a traffic snapshot $\mathcal{TS}$, a view V and a valuation of variables ν. Remember that we calculate the set $seg_V(C)$ of segments and resp. intervals of space a car C occupies in a view V with Alg. 2. We denote the length of a real interval $X \subseteq \mathbb{R}$ by $|X|$. While the following definition holds for a view $V = (L, X, E)$ as of Def. 28, we define the semantics for a multi-view $V_M(E, \mathcal{TS}) = (V_1(E, \mathcal{TS}), \ldots, V_n(E, \mathcal{TS}))$ as of Def. 32 in the next Def. 38.

Definition 37 (Semantics of UMLSL)**.** *The* satisfaction *of UMLSL formulae ϕ with respect to a traffic snapshot $\mathcal{TS}$, a virtual view $V = (L, X, E)$ and a valuation of variables ν is defined inductively as follows:*

$$
\begin{aligned}
&\mathcal{TS}, V, \nu \models \mathit{true} && \text{for all } \mathcal{TS}, V, \nu \\
&\mathcal{TS}, V, \nu \models \neg\phi && \Leftrightarrow \text{not } \mathcal{TS}, V, \nu \models \phi \\
&\mathcal{TS}, V, \nu \models \phi_1 \wedge \phi_2 && \Leftrightarrow \mathcal{TS}, V, \nu \models \phi_1 \text{ and } \mathcal{TS}, V, \nu \models \phi_2 \\
&\mathcal{TS}, V, \nu \models u = v && \Leftrightarrow \nu(u) = \nu(v) \\
&\mathcal{TS}, V, \nu \models \mathit{free} && \Leftrightarrow \#L = 1 \text{ and } |X| > 0 \text{ and} \\
& && \quad \forall c\colon \mathrm{CVar} \bullet seg_V(\nu(c)) = \emptyset \\
&\mathcal{TS}, V, \nu \models cs && \Leftrightarrow \#L = 1 \text{ and } |X| > 0 \text{ and} \\
& && \quad \forall \pi\colon L(1) \bullet \pi \in \mathbb{CS} \\
&\mathcal{TS}, V, \nu \models re(c) && \Leftrightarrow \#L = 1 \text{ and } |X| > 0 \text{ and} \\
& && \quad \forall \pi_i\colon L(1); \exists X_i \subseteq X \bullet \pi_i \in cres(\nu(c)) \cup res(\nu(c)) \\
& && \quad \text{and } (\pi_i, X_i) \in seg_V(\nu(c)) \text{ and } X \subseteq \bigcup_{i=1}^{\#L(1)} X_i \\
&\mathcal{TS}, V, \nu \models cl(c) && \Leftrightarrow \#L = 1 \text{ and } |X| > 0 \text{ and} \\
& && \quad (\forall \pi\colon L(1) \bullet \pi \in \mathbb{CS} \text{ and } \pi \in cclm(\nu(c)) \text{ or} \\
& && \quad (\#L(1) = 1 \text{ and } \exists \pi\colon \mathbb{L}; \exists X' \subseteq \mathbb{R} \bullet \\
& && \quad (\pi, X') \in seg_V(\nu(c)) \text{ and } \pi = L(1)(1) \text{ and } X \subseteq X') \\
&\mathcal{TS}, V, \nu \models \ell = r && \Leftrightarrow |X| = \nu(r) \\
&\mathcal{TS}, V, \nu \models \exists v\colon \phi_1 && \Leftrightarrow \exists \alpha \in \mathbb{I} \cup \mathbb{R} \bullet \mathcal{TS}, V, \nu \oplus \{v \mapsto \alpha\} \models \phi_1
\end{aligned}
$$

$$\begin{aligned}
\mathcal{TS}, V, \nu \models \phi_1 \frown \phi_2 \quad &\Leftrightarrow \exists V_1, V_2 \bullet V = V_1 \oplus V_2 \text{ and} \\
&\quad \mathcal{TS}, V_1, \nu \models \phi_1 \text{ and } \mathcal{TS}, V_2, \nu \models \phi_2 \\
\mathcal{TS}, V, \nu \models \begin{smallmatrix}\phi_2\\ \phi_1\end{smallmatrix} \quad &\Leftrightarrow \exists V_1, V_2 \bullet V = V_1 \ominus V_2 \text{ and} \\
&\quad \mathcal{TS}, V_1, \nu \models \phi_1 \text{ and } \mathcal{TS}, V_2, \nu \models \phi_2
\end{aligned}$$

To satisfy the atomic formulae *cs*, *free*, $cl(c)$ and $re(c)$, the view V has to be occupied completely by the respective element. This is the case if the view consists of only one virtual lane ($\#L = 1$) and has a positive extension ($|X| > 0$). For *free*, we demand that no part of any car is contained in the considered view V. For $re(c)$, where we first check, if for all segments π_i in our virtual lane $L(1)$ we have a (crossing) reservation in $\mathcal{TS}$ and a related element $(\pi_i X_i)$ in $seg_V(\nu(c))$. If all of these conditions are satisfied, we finally have to check that all segments in V are completely occupied by $\nu(c)$. For $cl(c)$, we distinguish between a crossing claim solely of elements from $\mathbb{CS}$ or a lane claim where the considered virtual lane $L(1)$ consists of only that lane. For the semantics of the chop operators, we refer to Def. 33 for chop operations on views. In case of $\ell = r$, the interval X is of length $\nu(r)$.

In case of a single (virtual) view $V(E)$ of car E, the semantics of UMLSL formulae is evaluated as in Def. 37. In case of a standard multi-view V_M containing several views V_i (cf. Def. 32), we define the following satisfaction of a formula ϕ over V_M.

Definition 38 (Multi-view semantics of UMLSL formulae)**.** *For a multi-view* $V_M = \{V_0, \ldots, V_n\}$*, a traffic snapshot* $\mathcal{TS}$ *and a valuation* ν *the* satisfaction *of a UMLSL formula* $\phi \in \Phi_{\mathbb{U}}$ *is defined by the existential satisfaction*

$$\mathcal{TS}, V_M, \nu \models \phi \;\Leftrightarrow\; \exists V_i \colon V_M \bullet \mathcal{TS}, V_i, \nu \models \phi.$$

With Def. 38, it is e.g. sufficient, if our controllers observe a collision in *one* virtual view.

Twisted views and the evaluation of UMLSL formulae. For highway traffic and country roads [HLOR11, HLO13], 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. E.g. consider the virtual views $V_1(E)$ and $V_2(E)$ from Example 14 with sequences of virtual lanes L_1 and L_2 as depicted in Fig. 3.7. In both views, the formula $\phi \equiv \langle re(E) \frown free \frown cs \rangle$ holds. Now consider the respective view $V(A)$, comprising the same road segments as $V_1(E)$ and $V_2(E)$, but build up from the sight of car A. This view is built up from virtual lanes $L == L_1 \cup L_2$, twisted around by 180 degrees. In $V(A)$, 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.

Last but not least, as UMLSL is an extension of MLSL, the (un-) decidability results from related work, as described in the beginning in Sect. 1.1, apply for UMLSL too.

3.6 Overview of More Complex Intersections and Special Cases

In this section, we first give a brief overview over some special cases for topologies, which can also be modelled with all concepts we hitherto introduced in this chapter. Additionally, as by default we consider a rather coarse segmentation for our crossing segments, we take a glance on how to generate finer-grained intersections.

Special Cases for Intersections

Note that with the definition of our topology and the sanity conditions, we not only allow for n-by-m intersections with arbitrary $n, m \in \mathbb{N}$ (cf. Fig. 3.2), but also for even more complex intersections. Imagine we would erase lane nodes 6 and 9 and all their outgoing or incoming edges from the topology on the right-hand side of Fig. 3.2. This would result in a 2-*by*-2-*by*-2-*by*-4 *intersection* where three road segments contain two lane segments, and only one road segment (the one below the intersection) contains four lane segments. Still all driving possibilities are clearly defined by the topology as the directed edges are defined unambiguously.

Now consider a *roundabout*, as depicted in Fig. 3.8. Again a fixed driving direction determines the driving direction from a lane segment onto a roundabout segment (hitherto "crossing segment"), as indicated by the arrows in the figure. For the depicted example, we can simply reuse the topology from Fig. 3.3, for a standard 2-by-2 intersection.

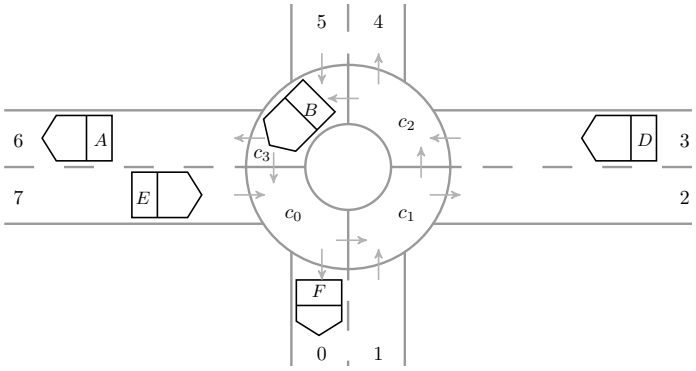

Figure 3.8: Abstract model for a roundabout. The arrows indicate the directed edges from the underlying graph topology (cf. Fig. 3.3).

To cope with *dead-ends*, we simply model the dead-end as an intersection, where a car turns to again leave the dead-end (cf. right-hand side of Fig. 3.9). Of course, this requires that there exists a lane segment leading away from the dead-end, which is always fulfilled as of sanity condition (3.2) from Def. 16, p. 41.

Additionally, to the left-hand side of Fig. 3.9, an example for an abstract model for a *parking spot* is depicted, where the parking spot itself is formalised by a segment p_0.

Again, as long a we can present a valid topology for such an abstract model, we can build a virtual view to reason in with our logic UMLSL. However, note that the depicted virtual lane ends on the left-hand side, as the parking spot from the example also has no second exit or entry.

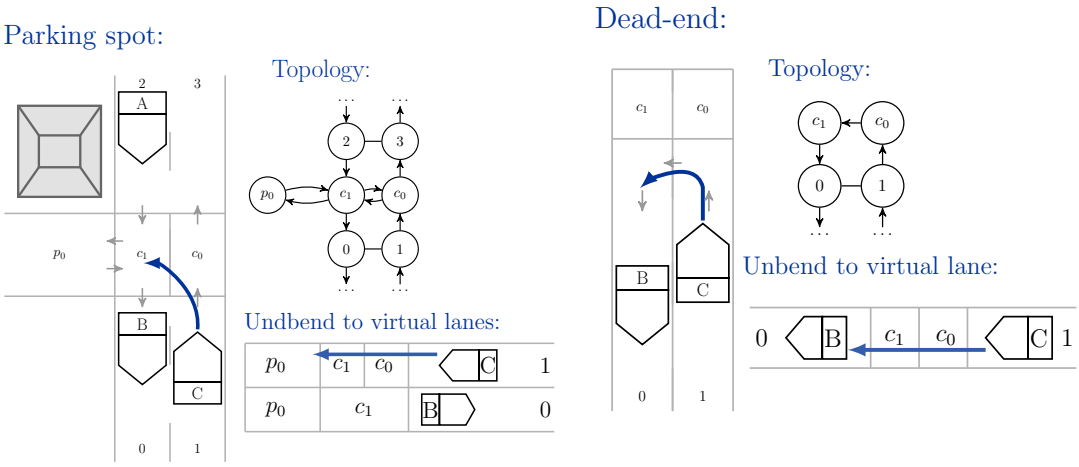

Figure 3.9: A parking spot, formalised by a segment p_0 and a dead-end road with respective relating topology and virtual view.

In all mentioned cases, we can reuse the definitions from this chapter for our topology and our virtual view without any redefinition. Also, we can use the crossing controller concept that we introduce later in this thesis. However, for parking spots a slight adaptation of our controller is needed.

Generating Topologies and Finer Segmentations

Note that we are *not* providing an explicit generic procedure for actually generating urban road networks for different traffic situations, but instead only give restrictions for what networks we can use with our approach with Def. 16 in Sect. 3.2.

Also, we give no limitation in how fine an intersection is segmented. Our running example from Fig. 3.1 is a 2-by-2 intersection with four crossing segments. Of course a finer segmentation of intersections (e.g. 16 crossing segments for the running example) is possible to e.g. improve traffic flow. All our concepts in this thesis actually function for arbitrarily fine segmentations.

However, as the controllers from Chapter 5 always reserve *all* crossing segments they need for their turn manoeuvre *at once*, a finer segmentation would currently not improve anything. We reserve all segments at once, to prevent deadlocks on the intersection. Using a step-wise reservation of crossing segments would require implementing a currently non-existent deadlock-prevention strategy.

3.7 Related Work

The notions of a *claim* and a *reservation* as introduced in [HLOR11] are not completely novel to the MLSL approach, as similar notions are e.g. used in [VMDS08]. Further on, we use a directed graph topology for modelling urban traffic networks, which is also not unique to our approach. For instance, in [ASS11], the authors introduce a discretised network, where each car fully occupies exactly one node and each node is either fully occupied by one car or empty. Cars have no right to access intersection nodes by themselves, as those are controlled by a centralised control mechanism called *Autonomous Intersection Management (AIM)*.

Such a discretised road network, as described for AIM in [ASS11], is also introduced under the term *space-grid model* in [XL16, XL17]. There, the authors attempt to broaden the spatial approach of MLSL for highway traffic and country roads to intersection scenarios, quite comparable to our goal in this thesis. In their approach, single discrete grids may belong to horizontal lanes or vertical lanes or to no lane at all, e.g. because they are blocked or because they simply do not belong to a road. Horizontal lanes (resp. vertical lanes) are labelled with even integers (resp. odd integers), and the driving dimension of an arbitrary car C is $dim(C) = 0$ (resp. $dim(C) = 1$) on horizontal lanes (resp. vertical lanes). However, the authors only apply their results to T-junctions and construct a timed automaton controller for this special case. Moreover, for changes in traffic situations, only a discrete time dimension is used instead of continuous time. The authors extend their approach in [XL17] by an estimation structure to predict future driving decisions of other cars. Further on, a case study including amongst others a multi-lane roundabout is introduced in [XLGD19] (see also Sect. 6.5, p. 150).

As a coarser version of such a grid model, *traffic cellular automata (TCA)* [MM05] can be mentioned for modelling urban traffic networks. However, TCA are rather used for a correct description of macroscopic traffic behaviour, while the description of detailed microscopic interactions (i.e. single cars) is minimised. For instance, in [MM05], the authors describe an application of TCA to optimise traffic flow from one cell to the next.

In contrast to all of the previously mentioned approaches, we allow for continuous traffic on lanes between intersections, motivated by reality and to enable a smoother movement on lane segments. Also, continuous lanes allow for re-using an adapted version of the lane change controller for country roads from [HLO13] for overtaking manoeuvres between intersections. We give more details on the adaptations needed to re-use the overtaking controller from [HLO13] for our urban traffic purposes later in Sect. 5.3, where we introduce our urban traffic controllers.

Lastly, while the authors of [XL16, XL17] limit their model to only two possible driving dimensions (horizontal and vertical), our topology allows for arbitrary driving dimensions. This is because in our case, driving directions are determined by directed edges in our topologies. For instance, if five roads meet at an intersection, it is not possible

to only consider horizontal and vertical lanes as this only allows for at most four roads meeting at an intersection.

We restrict our graph topology with sanity conditions to forbid senseless topologies. Related work on building such a correct traffic network as an assembly of smaller provably correct components like (different types of) intersections can be found in [MMP15]. The authors also verify their large networks component-based with support from their tool SAFE-T.

The principle of considering only a certain part of a dynamically changing, possibly infinitely large system of systems, similar to our concept of a view, is introduced as *Spotlight Principle* in [WW07]. The idea is to partition the systems processes into a *spotlight* and a *shade*. The processes inside the spotlight are considered, while the systems in the shade are summarised to one component and considered as unknown. In [TWR10] the authors extend their approach to systems that vary dynamically in size and topology which directly applies for the considered surroundings of our ego car. Over time new cars around ego will come closer, while others move further away from ego due to different values for speed and acceleration. As an example for this, consider again the traffic snapshot evolution example from Sect. 2.2 depicted in Fig. 2.2 on page 18. There, in traffic snapshot $\mathcal{TS}_0$, car D is not contained in the view $V(E, \mathcal{TS}_0)$ of ego car E. But after some time passes, car D is contained in the view $V(E, \mathcal{TS}_5)$ of the reached traffic snapshot $\mathcal{TS}_5$. This is due to the assumption that car E drives faster than D (cf. Table 2.1). An optimised solution on how systems can be moved better from the shade to the spotlight and vice versa, is considered in [Tim14].

In our thesis, we leave the process of retrieving what a car perceives about other cars abstract (cf. sensor function Ω_E from Sect. 3.4.3. In general, information about an autonomous cars' surroundings, like the size of (safety envelopes of) other cars is retrieved by an interplay of various different sensors. Widely used are e.g. radar sensors, lidar sensors, ultrasonic sensors and optical cameras [WHLS15]. The different and possibly impaired sensor information is processed by a perception unit in the car, where with *sensor fusion* a preferably accurate image of the surroundings is generated [Elm01, Kle99]. An old, but still powerful and state-of-the-art sensor fusion algorithm, is the *Kalman Filter* [Kal60].

Chapter summary. In this chapter, we introduced an urban road network $\mathcal{N}$ and a traffic snapshot $\mathcal{TS}$ for formalising abstract models of urban traffic situations. Further on, we described a procedure for generating flattened virtual views $V_M(E)$ to reason in with our urban logic UMLSL. Finally, we gave an outlook on some special cases for abstract models we can handle with our approach.

4 Automotive-Controlling Timed Automata

In Chapter 3, we introduced our abstract model for urban traffic. Particularly in Sect. 3.3, we formalised the evolution transitions for changes in a traffic snapshot $\mathcal{TS}$. Apart from the uncontrollable time transition, occurring whenever time elapses, the changes in a traffic snapshot $\mathcal{TS}$ are actively triggered by the traffic participants; In our case the autonomous cars. For this, we introduced lane change controllers for highway traffic and country roads from related work in Sect. 2.4. For crossing manoeuvres, we introduce a crossing controller in Chapter 5 and variations of it as well as other traffic controllers in the following Chapters 6 and 7.

For all these traffic controllers, we now introduce an extension of timed automata [AD94] as underlying automaton type. Besides the original timed automata from [AD94], we use some of the extensions UPPAAL[BDL04] introduces, where for details on these extended timed automata we refer to the preliminary Sect. 2.3. We name our automata *automotive-controlling timed automata* (ACTA). These ACTA use special controller actions to undertake traffic manoeuvres and trigger traffic snapshot changes with those actions. Further on, our ACTA use the logic UMLSL from previous Sect. 3.5 to reason about traffic situations.

We define syntax and semantics of ACTA without communication in the respective Sects. 4.1 and 4.2. We start with an ACTA definition without communication as it is easier to understand and as several of our controllers do not use communication. For instance, the first crossing controller we introduce in Chapter 5 and also the basic highway traffic lane change controller from Sect. 2.4 do not need to communicate. However, we also introduce several controllers using communication to cope with an imperfect knowledge of their surroundings or to optimise their performance. Thus, we devote Sect. 4.3 to a broadcast communication concept for ACTA, followed by an introduction into synchronisation and networks of ACTA in Sect. 4.4.

We recall and extend definitions for basic concepts for extended timed automata as provided in preliminary Sect. 2.3, wherever needed.

4.1 Syntax

For the *syntax* of our ACTA, we start with introducing the sets of possible variables. As for timed automata, we use clock variables x from the set $\mathbb{X}$ of all clock variables ranging over the continuous time domain $Time = \mathbb{R}_{\geq 0}$.

As indicated in Sect. 2.3, we follow UPPAAL by using *data variables* d from the set of all data variables $\mathbb{D}$ additionally to clock variables. For instance, UPPAAL allows for bounded integers, Boolean variables and arrays of these. To the range of data variables that UPPAAL allows, we add three types of data variables, motivated by our automotive application. We now briefly introduce them together with the sets of values over which they range:

- *Lane variables* $l, n, \ldots \in \mathbb{D}_{\mathbb{L}}$: Variables l ranging over the set of lane segments $\mathbb{L}$.
- *Crossing segment variables* $cs, cs_A, cs_B, \ldots \in \mathbb{D}_{\mathbb{CS}}$: Variables cs ranging over the power set of crossing segments from $\mathbb{CS}$. The intuition to have sets instead of single crossing segments is that our crossing controller (cf. Chapter 5) generally reserves a set of crossing segments to traverse an intersection. However, we allow for the abbreviation $c_1 == \{c_1\}$ in case of a singleton with $c_1 \in \mathbb{CS}$.
- *Car variables* $c, d, \overrightarrow{c} \ldots \in \mathbb{D}_{\mathbb{I}}$: Variables c ranging over the set of sequences of car identifiers from $\mathbb{I}$. For this, we use the Z notation [WD96] of sequences, as introduced in Sect. 2.1. We use sequences, as sometimes we wish to reason about specific orders of cars, e.g. for the hazard warning communication protocol we introduce in Chapter 7. As before, we allow for the abbreviation $C == \langle C \rangle$ in case of a singleton with $C \in \mathbb{I}$.

The set of all data variables $\mathbb{D}$ is given by $\mathbb{D} == \mathbb{D}_{\mathbb{L}} \cup \mathbb{D}_{\mathbb{CS}} \cup \mathbb{D}_{\mathbb{I}}$. According to the above informal description, we now define the *valuation* function ν for our data variables from $\mathbb{D}$. For the valuation of clock variables from $\mathbb{X}$, we refer to Def. 11 (p. 25).

Definition 39 (Data valuation)**.** *Consider data variables from* $\mathbb{D} = \mathbb{D}_{\mathbb{L}} \cup \mathbb{D}_{\mathbb{CS}} \cup \mathbb{D}_{\mathbb{I}}$*. Then the* valuation ν *of a variable* $d \in \mathbb{D}$ *is defined by*

$$\nu(d) = \begin{cases} m \in \mathbb{L} & , \text{ if } d \in \mathbb{D}_{\mathbb{L}} \\ p_{cs} \in \mathbb{PCS} & , \text{ if } d \in \mathbb{D}_{\mathbb{CS}} \\ s \in \mathsf{seq}\,\mathbb{I} & , \text{ if } d \in \mathbb{D}_{\mathbb{I}} \end{cases} .$$

For clock variables from $\mathbb{X}$, we introduced clock constraints $\varphi_{\mathbb{X}}$ from the set $\Phi_{\mathbb{X}}$ in Def. 8 on p. 23. Analogously to clock constraints, we introduce *data constraints* $\varphi_{\mathbb{D}}$ for our data variables. For defining data constraints, we first introduce *data terms* comprising possible mathematical operations on data variables and on the values they map to. As we use three types of data variables, we respectively have data terms on:

- Lane variables: Simple arithmetic operations with variables from the set $\mathbb{D}_{\mathbb{L}}$ and values from $\mathbb{N}$. With this, one can e.g. calculate the neighbouring lane $l + 1$ to an arbitrary lane $l \in \mathbb{D}_{\mathbb{L}}$.
- Crossing segment variables: Set operations for variables from $\mathbb{D}_{\mathbb{CS}}$ or sets of variables from $\mathbb{CS}$. With this, one can e.g. check whether two sets of reserved crossing segments $cs_E = \{c_0, c_1, c_2\}$ and $cs_C = \{c_2\}$ for two cars E and C intersect with a data term $cs_E \cap cs_C$.

- Car variables: Atomic expressions with sequences of variables from $\mathbb{D}_\mathbb{I}$ or sequences of values from $\mathbb{I}$. We also allow for functions on sequences here as introduced for Z sequences in Sect. 2.1. For instance, we recall the function $head\,\langle B, C, A\rangle$, which with $\langle B\rangle$ returns the first element B of the given sequence. The motivation for using functions in our data types is to enable further modification of data variables.

The intuition of introducing three different kinds of data terms is that a car variable $c \in \mathbb{D}_\mathbb{I}$ needs to be handled differently than a lane variable $l \in \mathbb{D}_\mathbb{L}$. E.g. it is type coherent to calculate $l + 1$ as $\mathbb{D}_\mathbb{L}$ ranges over $\mathbb{L} \subseteq \mathbb{N}$. On the other hand, a term like $c+1$ is completely senseless, as $\mathbb{D}_\mathbb{I}$ ranges over (sequences of) car identifiers $A, B, \ldots \in \mathbb{I}$, which do not allow for conventional arithmetic integer calculations.

Definition 40 (Data terms)**.** *Consider data variables $l \in \mathbb{D}_\mathbb{L}$, $\overrightarrow{c} \in \mathbb{D}_\mathbb{I}$, $cs \in \mathbb{D}_{\mathbb{CS}}$ and values $k \in \mathbb{N}$, $s \in \mathsf{seq}\,\mathbb{I}$ and $p_{cs} \in \mathbb{PCS}$. Further on, consider functions $f : \mathbb{N} \to \mathbb{N}$ and $g : \mathsf{seq}\,\mathbb{I} \to \mathsf{seq}\,\mathbb{I}$. The sets of* data terms $\Psi_{\mathbb{D}_i}$ *for the respective sets of data variables $\mathbb{D}_i$ with $i \in \{\mathbb{L}, \mathbb{CS}, \mathbb{I}\}$ are then defined by*

$$\begin{aligned}
\psi_{\mathbb{D}_\mathbb{L}} &::= l \mid k \mid \psi_{\mathbb{D}_\mathbb{L}1} + \psi_{\mathbb{D}_\mathbb{L}2} \mid \psi_{\mathbb{D}_\mathbb{L}1} - \psi_{\mathbb{D}_\mathbb{L}2} \mid f\,\psi_{\mathbb{D}_\mathbb{L}},\\
\psi_{\mathbb{D}_{\mathbb{CS}}} &::= cs \mid p_{cs} \mid \psi_{\mathbb{D}_{\mathbb{CS}}1} \cap \psi_{\mathbb{D}_{\mathbb{CS}}2} \mid \psi_{\mathbb{D}_{\mathbb{CS}}1} \cup \psi_{\mathbb{D}_{\mathbb{CS}}2} \quad \textit{and}\\
\psi_{\mathbb{D}_\mathbb{I}} &::= \overrightarrow{c} \mid s \mid g\,\psi_{\mathbb{D}_\mathbb{I}}.
\end{aligned}$$

We collect all possible data terms in the set $\Psi_\mathbb{D} == \Psi_{\mathbb{D}_\mathbb{L}} \cup \Psi_{\mathbb{D}_\mathbb{I}} \cup \Psi_{\mathbb{D}_{\mathbb{CS}}}$.

Note that the types of data variables we currently use in our approach in the set $\mathbb{D}$ could be even more extended for future considerations. E.g. one could broaden the usage of data term $\psi_{\mathbb{D}_\mathbb{L}}$ from natural numbers to reals and also add further arithmetic operations to this data term. With this, real-valued distance measures could be compared in a guard. However, for our current use case the data terms as defined in Def. 40 are sufficient.

Using these data terms, we now define data constraints to be used as guards and invariants in our ACTA.

Definition 41 (Data constraints)**.** *Consider a data variable $d \in \mathbb{D}$ and data terms as defined in Def. 40 and values $k \in \mathbb{N}$, $s \in \mathsf{seq}\,\mathbb{I}$ and $p_{cs} \in \mathbb{PCS}$. The set of all* data constraints $\Phi_\mathbb{D}$ *is inductively defined by*

$$\varphi_\mathbb{D} ::= \psi_{\mathbb{D}_\mathbb{L}} \sim k \mid \psi_{\mathbb{D}_\mathbb{L}} \sim k \mid \psi_{\mathbb{D}_{\mathbb{CS}}} = p_{cs} \mid \psi_{\mathbb{D}_\mathbb{I}} = s \mid \exists d\colon \varphi_\mathbb{D} \mid \neg\varphi_\mathbb{D} \mid \varphi_{\mathbb{D}_1} \wedge \varphi_{\mathbb{D}_2}\,,$$

where $\sim \in \{=, <, >, \leq, \geq\}$.

The satisfaction $\nu \models \varphi_\mathbb{D}$ of data constraints, follows similar rules as the satisfaction of clock constraints as defined in Def. 11, whereas we refer to that definition for this. For the existential quantifier we refer to the satisfaction of the respective UMLSL formula which is defined in Def. 37.

Example 17 (Data constraints). With $c, d \in \mathbb{D}_{\mathbb{I}}$, a simple example for a data constraint is $c = d$ (abbreviation for $\langle c \rangle = \langle d \rangle$), where two car identifiers are compared with each other. Further on, consider two sets of claimed crossing segments cs_a and cs_b for two different cars. With a data constraint $cs_a \cap cs_b = \emptyset$ it can be compared whether the claimed segments are disjoint. △

With this definition of data constraints we can define which sorts of formulae may occur as guards φ of transitions and in invariants $\mathcal{I}$ in locations q of ACTA. We build up our *guards* and *invariants* from the previously defined clock and data constraints and UMLSL formulae.

Definition 42 (Guards and invariants)**.** *With data constraints $\varphi_{\mathbb{D}} \in \Phi_{\mathbb{D}}$ (cf. Def. 41), clock constraints $\varphi_{\mathbb{X}} \in \Phi_{\mathbb{X}}$ (cf. Def. 8 ,p. 23) and UMLSL formulae $\varphi_{\mathbb{U}} \in \Phi_{\mathbb{U}}$ (cf. Def. 35, p. 62), a* guard *or* invariant *φ is defined by*

$$\varphi ::= \varphi_{\mathbb{D}} \mid \varphi_{\mathbb{X}} \mid \varphi_{\mathbb{U}} \mid \varphi_1 \wedge \varphi_2 \mid true.$$

The set of all guards and invariants is denoted by Φ.

Example 18 (Guards and invariants). Consider the guard (resp. invariant) $\varphi \equiv \exists c, d : \langle re(c) \wedge re(d) \rangle \vee x > t$ with a clock variable $x \in \mathbb{X}$ and a constant $t \in \mathbb{R}_{\geq 0}$. Guard φ consists of the UMLSL formula $\exists c, d : \langle re(c) \wedge re(d) \rangle$ and the clock constraint $x > t$. It states that somewhere exists a collision between cars $\nu(c) = C$ and $\nu(d) = D$ or the clock x exceeds t. △

Using the data terms as given in Definition 40, we now introduce *variable modifications* by extending the definition for resets of clock variables described in [AD94] to modifications of data and clock variables. As we do not yet use modifications of data variables ranging over $\mathbb{CS}$, we omit a definition for those here.

Definition 43 (Variable modifications)**.** *Consider a clock variable $x \in \mathbb{X}$, a car variable $c \in \mathbb{D}_{\mathbb{I}}$, a lane variable $l \in \mathbb{D}_{\mathbb{L}}$ and a crossing segment variable $cs \in \mathbb{D}_{\mathbb{CS}}$. A* variable modification *ν_{act} is defined by*

$$\nu_{act} ::= l := \psi_{\mathbb{D}_{\mathbb{L}}} \mid c := \psi_{\mathbb{D}_{\mathbb{I}}} \mid cs := \psi_{\mathbb{D}_{\mathbb{CS}}} \mid x := 0 \mid \nu_{act,1}; \nu_{act,2} \mid \epsilon,$$

where $\psi_{\mathbb{D}_i} \in \Psi_{\mathbb{D}_i}$ is a data term for each set of data variables $\mathbb{D}_i$ with $i \in \{\mathbb{L}, \mathbb{CS}, \mathbb{I}\}$ as of Def. 41. The set of all possible variable modifications is denoted by $\mathcal{V}_{Act}$.

Example 19 (Variable modifications). An example for a variable modification with $l \in \mathbb{D}_{\mathbb{L}}$ is $l := l + 1$, where a lane variable l is set to the value of its neighbouring lane. △

Note that Def. 43 allows for the occurrence of multiple variable modifications $\nu_{act,i}$, distinguished from each other by the symbol ';'. To avoid variable modifications being in conflict with each other, we assume that the variable modification is executed 'from left to right'. For instance, with a variable modification $\nu_{act,1} == l := l + 1; l := l - 1$, where $l \in \mathbb{D}_{\mathbb{L}}$ as before, the variable l is first incremented by 1 and after that this new incremented value for l is again decremented by 1. Thus, after completion of $\nu_{act,1}$, the value of l is not changed.

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 a (crossing) claim or a (crossing) reservation. Table 4.1 presents an informal description of all available controller actions.

Definition 44 (Controller actions)**.** *With $c \in \mathbb{D}_{\mathbb{I}}$, a* controller action c_{act} *is defined by*

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

where $\psi_{\mathbb{D}_{\mathbb{L}}}$ as in Def. 40 and τ is a null action. The set of all controller actions is defined by $Ctrl_{Act}$.

With τ, we allow for transitions in an ACTA without any controller action. Note that in contrast to $\mathtt{c}(c, \psi_{\mathbb{D}})$, the action $\mathtt{cc}(c)$ does not need a second parameter, because the path through the crossing is automatically claimed by the traffic snapshot (cf. Sect. 3.3). The case for $\mathtt{wd\,rc}(c)$ is analogous. We omit a controller action for acceleration, as we do not implement a speed or distance controller and thus do not yet use acceleration for our controllers. However, we refer to our traffic snapshot definition Def. 27, where we define an update for a cars acceleration in the traffic snapshot. We keep that definition there to allow for an easier introduction of acceleration for future considerations.

Def. 44 also does not allow for concatenation of controller actions, as we only allow for one controller action per transition in our ACTA for reasons of simplicity.

We are considering at least three controllers (distance controller, lane change controller, crossing controller) in one single car. Thus, we need a way to identify the car in which an ACTA is located, as their different data variables might refer to the same car. For this purpose we introduce an *identifying tuple* $I \in \mathbb{D}_{\mathbb{I}} \cup \{\text{ego}\} \times \mathbb{I}$ whereby for a controller action $\mathtt{r}(c)$ with $I = (c, C)$ the traffic snapshot will recognise a reservation for car C. We identify the car under consideration E with ego and the tuple $I = (\text{ego}, E)$.

Additionally we use *committed locations*, as introduced for UPPAAL. Whenever an ACTA is in a committed location, time may not pass and the committed location has to be left with the next possible transition. Thus, no other automaton is allowed to take a transition until an automaton in a committed location has left this location. Later, we also use urgent locations but do not include them in the following definition for reasons of brevity (cf. Sect. 5.4, p. 105).

c_{act}	Informal semantics of command (for $\nu(c) = C$)
$\texttt{c}(c, n)$	Claim lane segment n for car C.
$\texttt{wd c}(c)$	Withdraw currently claimed lane segment for car C.
$\texttt{cc}(c)$	Claim needed crossing segments of next crossing for car C.
$\texttt{wd cc}(c)$	Withdraw all currently claimed crossing segments for C.
$\texttt{r}(c)$	Reserve all previously claimed lane segments for C.
$\texttt{wd r}(c, n)$	Withdraw lane reservations for C, except for the one on n.
$\texttt{rc}(c)$	Reserve C's previously claimed crossing segments.
$\texttt{wd rc}(c)$	Withdraw reservation for all crossing segments of C.
τ	Null action.

Table 4.1: Possible controller actions (cf. Def. 44) for arbitrary car $\nu(c) = C$ and their informal meaning.

Definition 45 (Syntax of an automotive-controlling timed automaton)**.** *An ACTA is defined by a tuple* $\mathcal{A} = (Q, C, \mathbb{X}, \mathbb{D}, \mathcal{I}, T, q_{ini}, I)$*, where*

- Q *is a finite set of locations* $q_0, q_1, q_2, \ldots$,
- $C \subseteq Q$ *is a finite set of committed locations* $q_0, q_1, q_2, \ldots$
- $\mathbb{X}$ *is the set of clocks* $x, y, z, \ldots$,
- $\mathbb{D} = \mathbb{D}_{\mathbb{L}} \cup \mathbb{D}_{\mathbb{I}} \cup \mathbb{D}_{\mathbb{CS}}$ *is the set of data variables* $d_1, d_2, d_3, \ldots$,
- $\mathcal{I} : Q \to \Phi$ *assigns an invariant* $\mathcal{I}(q)$ *to every location* q,
- $T \subseteq Q \times \Phi \times Ctrl_{Act} \times \mathcal{V}_{Act} \times Q$ *is the set of all directed edges, where an element* $(q, \varphi, c_{act}, \nu_{act}, q') \in T$ *is an edge from locations* q *to* q' *labelled with a guard* φ*, a controller action* c_{act} *and a set of variable modifications* ν_{act}*,*
- $q_{ini} \in Q$ *is the initial locations, and*
- $I \in \mathbb{D}_{\mathbb{I}} \cup \{\text{ego}\} \times \mathbb{I}$ *identifies the car for which the controller works.*

Notation. We denote an element of T as follows:

$$q \xrightarrow{\varphi \,/\, c_{act}; \nu_{act}} q'$$

Informally, this means that for elements before '/' the truth value of φ is checked, while elements after '/' resemble executions: controller actions c_{act} are executed or variables are set to specific values with ν_{act}.

Note that while Def. 44 only allows for one controller action per transition, we allow for arbitrarily many variable modifications ν_{act} with Def. 43. As mentioned before, the lane change controller $\mathcal{A}_{lc}$ informally introduced in Fig. 2.6, on p. 31 is one example for an ACTA. However, we introduce a simpler example in the following. We will come back

to the lane change controller $\mathcal{A}_{lc}$ in an example for the semantics of ACTA later on in Sect. 4.2.

Example 20 (ACTA Syntax). Consider the ACTA in Fig. 4.1, which is not intended to be a sane or safe controller, but is only a basic example. The depicted automaton with identifying tuple (c, C) starts in initial location q_0 with the invariant $\langle re(c) \rangle$, stating that somewhere exists a reservation for the related car C. If somewhere exists a free space, and thus the guard $\langle free \rangle$ holds, the automaton changes to location q_1 while setting a reservation for car C and resetting the clock variable x to 0. After 10 time units, the car withdraws its reservation and is again in its initial location. △

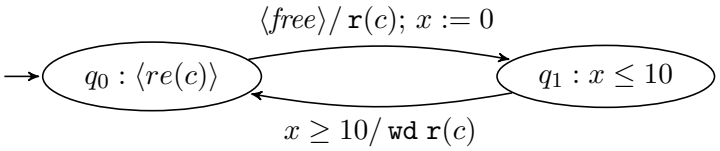

Figure 4.1: A basic example for an ACTA.

We introduce the parallel composition of ACTA later in Sect. 4.4, there already for ACTA with the communication concept that we introduce in Sect. 4.3. We now first focus on the semantics of an ACTA in the following section.

4.2 Semantics

In Sect. 3.3, specifically in Defs. 22 to 27, we introduced traffic snapshot transitions to describe the changes that may occur to a traffic snapshot $\mathcal{TS}$. As the *semantics* of an ACTA is linked to the traffic snapshot, we observe corresponding changes to the possible *system states* of an ACTA $\mathcal{A}$. These system states are called *configurations* $\mathcal{C}$ and consist of a traffic snapshot $\mathcal{TS}$, a valuation ν for all clock and data variables, and a location q the automaton $\mathcal{A}$ is in in that particular configuration. The formal semantics of $\mathcal{A}$ is then built from the set of all possible configurations $Conf(\mathcal{A})$ and the transitions between these configurations.

We define the configuration of an ACTA, where we recall Def. 39 for the variable valuation ν from the previous section.

Definition 46 (Configuration of an ACTA)**.** *Let $\mathcal{A} = (Q, \mathbb{X}, \mathbb{D}, \mathcal{I}, T, q_{ini}, I)$, $\mathcal{TS} \in \mathbb{TS}$, a valuation $\nu(v)$ for all $v \in \mathbb{X} \cup \mathbb{D}$ and a location $q \in Q$. The* configuration $\mathcal{C}$ *of $\mathcal{A}$ at location q is then defined by*

$$\mathcal{C} = \langle \mathcal{TS}, \nu, q \rangle,$$

where the initial configuration $\mathcal{C}_{ini} \in Conf(\text{ACTA})$ *is given by*

$$\mathcal{C}_{ini} = \langle \mathcal{TS}_{ini}, \nu_{ini}, q_{ini} \rangle,$$

with initial traffic snapshot $\mathcal{TS}_{ini} \in \mathbb{TS}$, $\forall x \in \mathbb{X}\colon \nu_{ini}(x) = 0$ *and* $\nu_{ini} \models \mathcal{I}(q_{ini})$. *The set of all configurations is given by*

$$Conf(\mathcal{A}) = \{\langle \mathcal{TS}, \nu, q \rangle \mid q \in Q \wedge (\mathcal{TS}, V_M(E, \mathcal{TS}), \nu \models \mathcal{I}(q))\}.$$

As indicated before, both, controller actions and the elapse of time, change not only the configuration of the controller $\mathcal{A}$ but also the traffic snapshot $\mathcal{TS}$. Basically, the possible changes can be sorted into three different types of transitions:

- *Controller action:* A controller action occurs and the respective change in the configuration of $\mathcal{A}$ triggers a change in the current traffic snapshot $\mathcal{TS}$. Note that in the following Def. 47, we distinguish the two cases of controller actions with two parameters (e.g. $\mathtt{c}(c, n)$) and those with only one parameter (e.g. $\mathtt{r}(c)$).
- *Internal transition:* An internal transition occurs in $\mathcal{A}$ and while the configuration of $\mathcal{A}$ changes, there is no corresponding change in the traffic snapshot $\mathcal{TS}$.
- *Time transition:* Time passes and there is no change in the configuration of $\mathcal{A}$ observable, apart from new values for all clock variables. Also, the current traffic snapshot changes: all cars $C \in \mathbb{I}$ move according to their velocity and their path $pth(C)$ with new values for position and speed (cf. Def. 22).

With these preliminary considerations, we define the semantics of an ACTA $\mathcal{A}$, given by its transition system $\mathcal{T}(\mathcal{A})$.

Definition 47 (Semantics of an ACTA)**.** *For an ACTA* $\mathcal{A}$, *with* $Conf(\mathcal{A})$ *and* $\mathcal{C}_{ini}$ *as in Def. 46, the* semantics *of* $\mathcal{A}$ *is defined by the transition system*

$$\mathcal{T}(\mathcal{A}) = (Conf(\mathcal{A}), Ctrl_{Act} \cup Time, \{\xrightarrow{\lambda} \mid \lambda \in Ctrl_{Act} \cup Time\}, \mathcal{C}_{ini}).$$

The set $Ctrl_{Act} \cup Time$ *contains all labels that may appear at transitions. We examine the types of the transition relation* $\xrightarrow{\lambda} \subseteq Conf(\mathcal{A}) \times Conf(\mathcal{A})$*:*

Controller actions (with one parameter): *Consider an arbitrary car variable* $c \in \mathbb{D}_{\mathbb{I}}$ *with* $\nu(c) = C$ *and controller actions* $\sim \in \{\mathtt{cc}, \mathtt{wd\ c}, \mathtt{wd\ cc}, \mathtt{r}, \mathtt{rc}, \mathtt{wd\ rc}\}$. *The transition* $\langle \mathcal{TS}, \nu, q \rangle \xrightarrow{\sim(c)} \langle \mathcal{TS}', \nu', q' \rangle$ *exists iff there are corresponding transitions* $\mathcal{TS} \xrightarrow{\sim(C)} \mathcal{TS}'$ *and* $q \xrightarrow{\varphi/\sim(c);\nu_{act}} q'$, *where* $\mathcal{TS}, V_M(E, \mathcal{TS}), \nu \oplus I \models \varphi$ *and* $\mathcal{TS}', V_M(E, \mathcal{TS}'), \nu' \oplus I \models \mathcal{I}(q')$ *hold.*

Controller actions (with two parameters): *With controller actions* $\sim \in \{\texttt{c}, \texttt{wd r}\}$ *and a lane segment* $n \in \mathbb{L}$*, we have a transition* $\langle \mathcal{TS}, \nu, q\rangle \xrightarrow{\sim(c,n)} \langle \mathcal{TS}', \nu', q'\rangle$ *iff there exist respective transitions* $\mathcal{TS} \xrightarrow{\sim(C,n)} \mathcal{TS}'$ *and* $q \xrightarrow{\varphi/\sim(c,l);\nu_{act}} q'$*, where* $\mathcal{TS}, V_M(E, \mathcal{TS}), \nu \oplus I \models \varphi$ *holds and for* $l \in \mathbb{D}_{\mathbb{L}}$ *with* $\nu'(l) = n$ *also the expression* $\mathcal{TS}', V_M(E, \mathcal{TS}'), \nu' \oplus I \models \mathcal{I}(q')$ *holds.*

Internal transition: *The internal transition* $\langle \mathcal{TS}, \nu, q\rangle \xrightarrow{\tau} \langle \mathcal{TS}, \nu', q'\rangle$ *exists iff there exists* $q \xrightarrow{\varphi/\nu_{act}} q'$ *where* $\mathcal{TS}, V_M(E, \mathcal{TS}), \nu \oplus I \models \varphi$ *and* $\mathcal{TS}, V_M(E, \mathcal{TS}), \nu' \oplus I \models \mathcal{I}(q')$ *hold.*

Time passes: *If* $\lambda \in$ *Time, we have a transition* $\langle \mathcal{TS}, \nu, q\rangle \xrightarrow{t} \langle \mathcal{TS}', \nu', q\rangle$ *iff* $q \notin C$ *and a transition* $\mathcal{TS} \xrightarrow{t} \mathcal{TS}'$ *exists and for all points in time* $t' \in [0, t]$ *the valuation update* $\nu' = \nu \oplus \{x \mapsto \nu(x) + t'\}$ *with* $x \in \mathbb{X}$ *implies* $\mathcal{TS}', V_M(E, \mathcal{TS}'), \nu' \models \mathcal{I}(q)$.

Note that the time transition requires the invariant $\mathcal{I}(q)$ to hold not only after t time but also during all time units from the interval $[0, t]$.

Similar as for extended timed automata, a location q in an ACTA is *reachable*, if there exists a *transition sequence* starting from an initial configuration $\langle \mathcal{TS}_0, \nu_0, q_0\rangle$ leading to a configuration $\langle \mathcal{TS}, \nu, q\rangle$ (cf. Sect. 2.3, p. 27). Also recall from Sect. 2.3 that a time-stamped transition sequence is named a *computation path* or *run*.

Example 21 (ACTA semantics). Consider again the preliminary Example 2, p. 18, from Sect. 2.2, where we show traffic snapshot transitions from an initial traffic snapshot $\mathcal{TS}_0$ to a traffic snapshot $\mathcal{TS}_5$. There, parts of a lane change manoeuvre for a car E are examined and depicted in Fig. 2.2. We further on consider the lane change controller $\mathcal{A}_{lc}$ from preliminary Sect. 2.4, depicted in Fig. 2.6, p. 31, to be our ACTA in this example.

Let us now focus on the first four steps of Example 2 and the configuration of the controller $\mathcal{A}_{lc}$ of car E. We do not discuss the respective traffic snapshot changes here, as those are shown in Example 2. We list the contents of the configurations $\mathcal{C}_i$ for $i \in \{1, \ldots, 4\}$ for $\mathcal{A}_{lc}$ in Table 4.2.

As our car E already has a claim in $\mathcal{TS}_0$, we derive from the lane change controller that our controller is already in its location q_2. We further on conclude the following valuation of variables from Example 2: $\nu_0(l) = 2$, $\nu_0(n) = 1$ and $\nu_0(x) = 1$. With that, we start from the initial configuration $\mathcal{C}_0 = \langle \mathcal{TS}_0, \nu_0, q_2\rangle$. Further on, we set the two time constants t_1 and t_2 from Example 2 to $t_1 := 1 < t$ and $t_2 := t_{lc} = 2$. With this, we have

$$\langle \mathcal{TS}_0, \nu_0, q_2\rangle \xrightarrow{1} \langle \mathcal{TS}_1, \nu_1, q_2\rangle \xrightarrow{\texttt{r}(E)} \langle \mathcal{TS}_2, \nu_2, q_3\rangle \xrightarrow{2} \langle \mathcal{TS}_3, \nu_3, q_3\rangle \xrightarrow{\texttt{wd r}(E,2)} \langle \mathcal{TS}_4, \nu_4, q_0\rangle.$$

i	$\mathcal{TS}_i$	ν_i	q_i	$\mathcal{C}_i$
0	$\mathcal{TS}_0$	$\nu_0(l) = 2$, $\nu_0(n) = 1$, $\nu_0(x) = 1$	q_2	$\mathcal{C}_0 = \langle \mathcal{TS}_0, \nu_0, q_2 \rangle$
1	$\mathcal{TS}_1$	$\nu_1(l) = 2$, $\nu_1(n) = 1$, $\nu_1(x) = 2$	q_2	$\mathcal{C}_1 = \langle \mathcal{TS}_1, \nu_1, q_2 \rangle$
2	$\mathcal{TS}_2$	$\nu_2(l) = 2$, $\nu_2(n) = 1$, $\nu_2(x) = 0$	q_3	$\mathcal{C}_2 = \langle \mathcal{TS}_2, \nu_2, q_3 \rangle$
3	$\mathcal{TS}_3$	$\nu_3(l) = 2$, $\nu_3(n) = 1$, $\nu_3(x) = t_{lc}$	q_3	$\mathcal{C}_3 = \langle \mathcal{TS}_3, \nu_3, q_3 \rangle$
4	$\mathcal{TS}_4$	$\nu_4(l) = 2$, $\nu_4(n) = 2$, $\nu_4(x) = t_{lc}$	q_0	$\mathcal{C}_4 = \langle \mathcal{TS}_4, \nu_4, q_0 \rangle$

Table 4.2: Exemplary configurations $\mathcal{C}_i = \langle \mathcal{TS}_i, \nu_i, q_i \rangle$ for $i \in \{0, .., 4\}$ for car E for the respective transitions from Example 21, relating to the traffic snapshot changes from Example 2 and to the lane change controller $\mathcal{A}_{lc}$ from Fig. 2.6.

For the two time transitions to be valid, we require the respective invariants of q_2 and q_3 to hold. The time invariants $x \leq t$ (resp. $x \leq t_{lc}$) hold as we set the constants t_1 and t_2 appropriately. Further on, we can recognise from Fig. 2.2, that the second part of the invariant of q_2, namely the potential collision check $\neg\exists pc(c)$, holds, as no intersections of the claim of car E with any other car exists.

For the reservation transition from location q_2 to q_3 to be valid, the guard $\neg\exists pc(c)$ has to hold, which we again can conclude from Fig. 2.2. There, the clock x is reset, as the reader can observe in Table 4.2. For the last transition from location q_3 to q_0, the guard $x \geq t_{lc}$ needs to hold. On the previous transition, t_{lc} time units passed, whereas it is $x = t_{lc}$ and the guard subsequently holds. On this transition, the lane variable n is updated to l. △

Note that, with our definition of ACTA, only controllers for lane change or crossing manoeuvres are possible. However, further controller actions are of course imaginable, as long as the underlying concept is also integrated into our traffic snapshot from Sect. 3.3. For this we also recall Sect. 3.6, where we already hinted at possible other traffic situations besides normal intersections and highway lanes running in parallel. E.g. a *parking manoeuvre* as in Fig. 3.9 could be conducted by a parking controller using the previously presented controller actions `cc` and `rc` (resp. `wd cc` and `wd rc`).

While the semantics as described in this section is defined for independent ACTA, let us now consider a communication concept, which allows for synchronisation of ACTA via broadcast messages.

4.3 Broadcast Communication with Data Constraints

As indicated in the introduction of this chapter, there are several cases where *communication* between ACTA is required. Possible cases we examine in this thesis are the following:

- Lack of knowledge (cf. imperfect knowledge, Sect. 5.4),
- Priorities that shall be communicated and negotiated between cars (cf. Sect. 6.4.3),
- Warning messages that should be propagated between cars (cf. Chapter 7).

To this end, we introduce a concept of *guarded broadcast communication* with data constraints for our ACTA in this section. As an important inspiration on the communication between distributed systems, we refer to the approach from Tony Hoare on *Communicating Sequential Processes* [Hoa78].

In any kind of traffic situation, autonomous cars can be understood as dynamic nodes in a *Vehicular ad-hoc network* (VANET)[SD14], without a fixed wireless infrastructure and in our case without taking roadside units into account. We use broadcast communication, meaning that when one controller sends a message via an output action, all cars that are in a location with a possible relating input action are required to synchronise with the sender.

In Sect. 4.1, we introduced data variables and data constraints to be used in guards, invariants and variable updates of our ACTA, comparable to the use of data variables in the extended timed automata from [BDL04] for UPPAAL. We now broaden this use of data variables and data constraints in timed automata even further by sending data via our *broadcast channels*. On receiving a message, the data is analysed with a special *communication guard* for its relevance for the receiving ACTA.

Our type of *broadcast communication* is inspired by the *Calculus for Attribute-based Communication* [ADNL$^+$15], which itself is inspired by the *calculus of mobile processes* of Milner [MPW92]. The authors of [ADNL$^+$15] 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 synchronise with an output $(u)@\Pi$ if they have an input $\Pi(x)$ and their local attributes a, together with the received message x, satisfy the predicate Π.

In our case the attributes to be sent are sequences $\overrightarrow{d}$ of data variables from our set $\mathbb{D}$. Similarly to the handshake communication introduced in Sect. 2.3, Def. 13, we have *output actions* $a!$ for sending messages on a broadcast channel a and input actions $a?$ for receiving messages. We add the communication guard φ to *input actions*, where the received data is analysed for its relevance for the respective receiving controller. The type of a communication guard is the set of all guards and invariants Φ as of Def. 42.

Definition 48 (Input and output actions)**.** *For a finite sequence of data variables* $\overrightarrow{d} \in \mathsf{seq}\,\mathbb{D}$ *and a* communication guard $\varphi \in \Phi$ *we define an* output action out *on a broadcast channel* $a \in Chan$ *by* $\mathsf{out} := a!\overrightarrow{d}$ *and a related* input action in *by* $\mathsf{in} := a?\overrightarrow{d} : \varphi$*. We allow for the empty input action* $\mathsf{in} := true$ *and the empty output action* $\mathsf{out} := \tau$*. The respective sets of all input and output actions are IN and OUT.*

We now extend Def. 45 to a definition for the syntax of ACTA with communication. By construction, we forbid more than one communication action occurring on an edge of an ACTA with communication, meaning either an input action or an output action may occur, but not both on the same edge.

Definition 49 (Syntax of an ACTA with communication)**.** *An* ACTA with communication *is defined by a tuple* $\mathcal{A} = (Q, C, B, \mathbb{X}, \mathbb{D}, \mathcal{I}, T, q_{ini}, I)$*, where*

- B *is a set of broadcast channels with* $B \subseteq Chan$ *and*
- $T \subseteq Q \times \Phi \times IN \times OUT \times Ctrl_{Act} \times \mathcal{V}_{Act} \times Q$ *is the set of all directed edges, where an element* $(q, \varphi, \mathsf{in}, \mathsf{out}, c_{act}, \nu_{act}, q') \in T$ *is an edge from location* q *to* q' *labelled with a guard* φ*, a input action* in *on a channel* $b \in B$*, an output action* out *on a channel* $b \in B$*, a controller action* c_{act} *and a set of variable modifications* ν_{act}*.*

All other elements of the tuple $\mathcal{A}$ *remain as defined in Def. 45. For each element of* T *we require that if* $\mathsf{in} \neq true$ *we have* $\mathsf{out} = \tau$ *and vice versa.*

Notation. With the extension of broadcast communication, the action language of T now is of the following syntax:

$$q \xrightarrow{\varphi \wedge \mathsf{in} \,/\, \mathsf{out}; c_{act}; \nu_{act}} q'$$

We motivate ACTA with communication and their synchronisation with the following example. As we only introduce two ACTA with communication in the example, for now the actual synchronisation is not much different to the synchronisation via handshake communication we defined for the parallel composition in Def. 13, except for the treatment of data. Thus note that the example is solely meant to illustrate how data is sent and how guarded synchronisation with communication guards is done. We define the actual synchronisation process after the example and motivate networks of ACTA in the following Sect. 4.4.

Example 22 (Input and output actions). Consider cars A and E, a variable valuation $\nu(\text{ego}) = E$ and $\nu(a) = A$ and respective sets of crossing segments for the next intersection cs_E for car E and cs_A for car A. Now consider the two simple controllers $\mathcal{A}_E$ and $\mathcal{A}_A$ as depicted in Fig. 4.2.

After some waiting time greater than 1 a request of car E for its claimed crossing segments cs_E is sent via broadcast channel *cross* with the output action $cross!\langle \text{ego}, cs_E \rangle$. Consider the corresponding input action in $\mathcal{A}_A$ on the edge from p_0 to p_1. The controller checks whether the request is sent from a car different than its own car with $a \neq d_1$ and it checks disjointedness of the requested crossing segments with its own claimed or reserved segments with $cs_A \cap cs_1 = \emptyset$. Only if these two communication guards hold, the controller synchronise on this transition. Note that on synchronisation we have

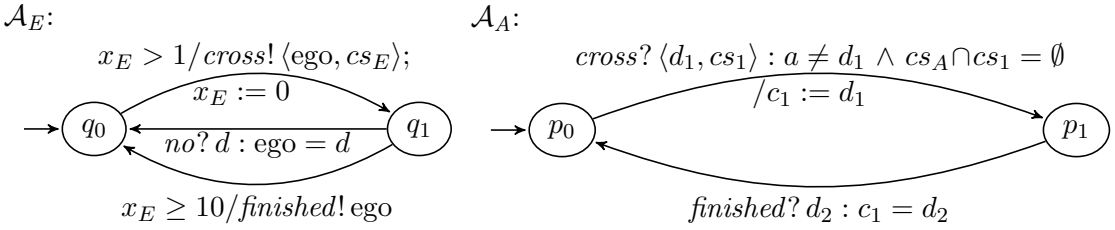

Figure 4.2: ACTA $\mathcal{A}_E$ with output actions on broadcast channels *cross* and *finished* on the left and ACTA $\mathcal{A}_A$ with compatible input actions on the right.

$\nu(d_1) = \nu(\text{ego}) = E$ and $\nu(cs_1) = \nu(cs_E)$. Secondly, with $c_1 := d_1$ the controller stores the received data in a local variable c_1 and we have $\nu(c_1) = \nu(d_1) = E$.

Likewise, after 10 time units, the respective input and output actions for channel *finished* synchronise, as $\nu(c_1) = E$ from before and after synchronisation also $\nu(d_2) = E$ holds. Note that it is not possible for $\mathcal{A}_E$ to execute the transition labelled with $no?\, d : \text{ego} = d$, as the controller has to wait for a suitable output action on channel *no* for that. △

4.4 Synchronisation and Networks of ACTA

The example from the previous section emphasises that in contrast to the classical broadcast communication, which e.g. UPPAAL uses, a peculiarity of our *guarded* broadcast communication is that the receiving automata do *not* compulsorily synchronise with the sender. Thus, on synchronisation, we have to keep some information about the communication guards and we do not solely get a simple τ-transition as observed for the unguarded communication from Def. 13. As formalised in the following definition, the result of our guarded broadcast synchronisation is a conjunction of (negations of) the respective (communication) guards of the receiving automata.

Definition 50 (Synchronisation of input and output actions)**.** *Consider an ACTA $\mathcal{A}$ with an output action* $\mathsf{out} == a!\,\overrightarrow{d}$ *in OUT on a broadcast channel $a \in Chan$, where $\overrightarrow{d} \in \mathsf{seq}\,\mathbb{D}$ is a finite sequence of data variables. Further on consider ACTA $\mathcal{A}_i$ with respective guards $\varphi_{i,1} \in \Phi$, each conjugated with a respective input action* $\mathsf{in}_i == a?\overrightarrow{c_i} : \varphi_{i,2}$ *in IN, where $\varphi_{i,2} \in \Phi$ is a communication guard and $i \in \{1, \ldots, n\}$.*

Now assume that m of the input actions in_i *synchronise with* out*, with $0 \leq m \leq n$ and that the remaining $n - m$ input actions* in_i *do not synchronise with* out*, either because $\varphi_{i,1}$ or* in_i *does not hold. After the synchronisation process we derive the following* synchronisation expression syn*, with a variable valuation $\sigma == \{\overrightarrow{c_i} \mapsto \nu(\overrightarrow{d})\}$:*

$$\mathsf{syn} := \bigwedge_{i=1}^{m} (\varphi_{i,1} \wedge \varphi_{i,2}(\sigma)) \wedge \bigwedge_{i=m+1}^{n} \neg(\varphi_{i,1} \wedge \varphi_{i,2}(\sigma)). \tag{4.1}$$

On synchronisation, besides the expression syn*, the variable modification* ν_{act} *and the controller action* c_{act} *of the sender* $\mathcal{A}$*, together with all variable modifications* $\nu_{act,i}$ *and controller actions* $c_{act,i}$ *for automata* $\mathcal{A}_i$*,* $i \in \{1, \ldots, m\}$*, are kept but updated with the valuation* σ*, yielding the expressions*

$$c_{act,\mathsf{syn}} := c_{act}; c_{act,1}; \ldots; c_{act,m}\,(\sigma), \tag{4.2}$$

$$\nu_{act,\mathsf{syn}} := \nu_{act}; \nu_{act,1}; \ldots; \nu_{act,m}\,(\sigma). \tag{4.3}$$

In formula syn (4.1), with the updated variable valuation σ, the value of $\overrightarrow{c}$ is set to the sent data $\overrightarrow{d}$. With that, the sent data is used in the respective communication guards $\varphi_{i,2}$, which match the type of a normal guard $\varphi \in \Phi$, as does the entire expression syn. Thus, while more complex in appearance than τ, syn nevertheless leads to an internal transition, as no details about the actual communication part are kept.

Note that only ACTA $\mathcal{A}_i$ for $i \in \{1, \ldots, m\}$ actually synchronise with $\mathcal{A}$, while the information that automata did not synchronise is kept with the negated guards in the right part of the expression syn. Equally, only variable modifications and controller actions of the sender $\mathcal{A}$ and ACTA $\mathcal{A}_i$ for $i \in \{1, \ldots, m\}$ are kept, as formalised in expressions (4.2) and (4.3). It is necessary to apply σ to both controller actions and variable modifications, as it is possible to update internal variables with sent data or to use sent data in controller actions.

In contrast to Defs. 49 resp. 45 for single ACTA (with communication), we allow for the occurrence of multiple controller actions in the expression $c_{act,\mathsf{syn}}$ (4.2). Analogously to the case for multiple variable modifications as explained after Def. 43, we assume multiple controller actions are executed from 'left to right'. For instance, for an expression $c_{act,\mathsf{syn}} == c(\text{ego}, n); c(\text{ego}, m)$, the claim for ego on lane n is overwritten by its claim on lane m.

For building a *network* of several ACTA using broadcast communication, we observe two design choices:

- We build the network in one step,
- We build the network consecutively by adding one ACTA after the other.

In both cases, if we wish to enable adding another ACTA to the network, e.g. whenever a car enters a view, we would have to keep information about the output action to allow for synchronisation with a new automaton. Note that, even for standard timed automata, it is generally not possible to remove one timed automaton from a network of timed automata, as information about each single timed automaton gets lost during the process of building a network. Thus, as we could not remove an ACTA whenever a car leaves a view, it seems inconsistent to allow for adding an ACTA when a car enters the view. Due to this, and as it seems the natural choice for automata with broadcast communication, where several automata may synchronise with one sender, we decide to take the first choice and build a network of ACTA in one step. With this, we assume

that we already know all cars we need to consider for our crossing manoeuvre on building the network.

Example 23 (Networks of ACTA). To illustrate our approach of building a network, we continue the previous Example 22 with the two ACTA $\mathcal{A}_E$ and $\mathcal{A}_A$ from Fig. 4.2 together with a third automaton $\mathcal{A}_{A,help}$, which is depicted in Fig. 4.3. We assume that, same as for $\mathcal{A}_A$, the ACTA $\mathcal{A}_{A,help}$ is located in car A and we thus assign the identifying tuple (a, A) to $\mathcal{A}_{A,help}$. With that, this new ACTA also uses the car variable a and the same set of crossing segments cs_A.

The intuition for $\mathcal{A}_{A,help}$ is to prevent the ego car E from starting a crossing manoeuvre with an output action on channel no, if there is an overlap of A and E's crossing segments. To send the message $no!c_2$ without any delay, location r_1 is committed as indicated with the **C**. Similarly to the case in Example 22, the three controllers synchronise on channel *cross*, if the respective (communication) guards hold.

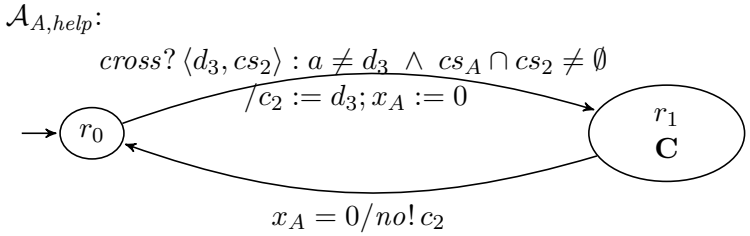

Figure 4.3: A third ACTA $\mathcal{A}_{A,help}$ with compatible input and output actions to $\mathcal{A}_E$. The intuition of $\mathcal{A}_{A,help}$ is to prevent the ego car to do a crossing manoeuvre if its crossing segments overlap with the segments of car A.

The network $\mathcal{A}_E \parallel \mathcal{A}_A \parallel \mathcal{A}_{A,help}$ is depicted in Fig. 4.4. On sending $cross!\,\langle \text{ego}, cs_E\rangle$, only one of the automata $\mathcal{A}_A$ and $\mathcal{A}_{A,help}$ can synchronise with the sender $\mathcal{A}_E$, as the respective communication guards on the transitions from p_0 to p_1 (resp. r_0 to r_1) contradict each other. Thus, semantically, the successor locations (q_1, p_1, r_1) and (q_1, p_0, r_0) are unreachable, but do exist syntactically. However for reasons of simplicity, these two unreachable locations are not depicted in Fig. 4.4.

Let us consider more detailed how the edge to location (q_1, p_1, r_0) in Fig. 4.4 was constructed. For that edge, it is assumed that $\mathcal{A}_A$ did synchronise with $\mathcal{A}_E$, while $\mathcal{A}_{A,help}$ did not. Applying Def. 50 thus yields the synchronisation expression

$$\begin{aligned}&(a \neq d_1 \ \wedge \ cs_A \cap cs_1) = \emptyset)\,(\sigma_1) \wedge \\ &(\neg(a \neq d_3 \ \wedge \ cs_A \cap cs_2 \neq \emptyset))\,(\sigma_2),\end{aligned} \tag{4.4}$$

with $\sigma_1 == \{d_1 \mapsto \text{ego}, cs_1 \mapsto cs_E\}$ and $\sigma_2 == \{d_3 \mapsto \text{ego}, cs_2 \mapsto cs_E\}$. The first part of expression (4.4) corresponds to the communication guard of $\mathcal{A}_A$ and the second part

to the communication guard of $\mathcal{A}_{A,help}$. The result where σ_1 and σ_2 are already applied to (4.4) is depicted as guards on the considered edge in the network in Fig. 4.4. Equally the variable modification $c_1 := d_1$ from the corresponding edge in $\mathcal{A}_A$ is updated to $c_1 := d_1(\sigma_1)$. Note that both the guard $x_E > 1$ and the clock update $x_E := 0$ from ACTA $\mathcal{A}_E$ are kept for the considered transition.

The edge to location (q_1, p_0, r_1) is constructed analogously, representing the case where $\mathcal{A}_{A,help}$ did synchronise with $\mathcal{A}_E$, while $\mathcal{A}_A$ did not. Note that the joint location (q_1, p_0, r_1) is committed as the included location r_1 is committed. △

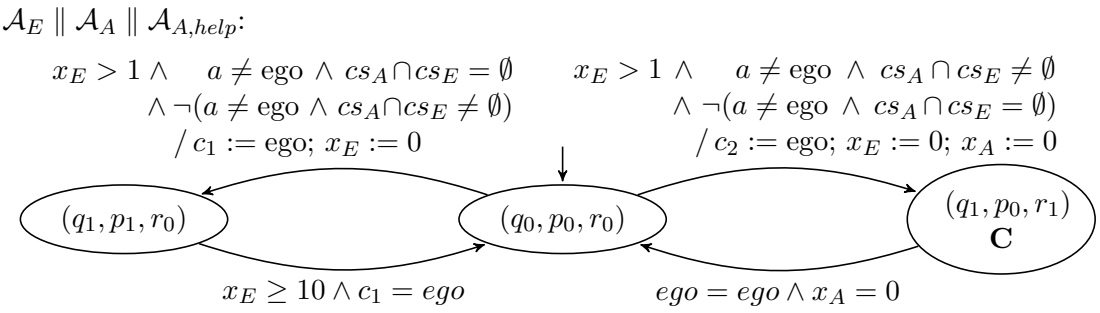

Figure 4.4: Parallel composition $\mathcal{A}_E \parallel \mathcal{A}_A \parallel \mathcal{A}_{A,help}$ without unreachable locations.

With this, we now give the formal definition for the *parallel composition* of several ACTA $\mathcal{A}_i$. Remember that by construction, we forbid the occurrence of multiple synchronisation processes at one transition (only one communication action is allowed on one edge of an ACTA, cf. Def. 49).

Definition 51 (Parallel composition of ACTA with communication)**.** *For $N == \{1, .., n\}$ with $n \geq 2$ and $i \in N$ consider several ACTA*

$$\mathcal{A}_i = (Q_i, C_i, B_i, \mathbb{X}_i, \mathbb{D}_i, \mathcal{I}_i, T_i, q_{ini,i}, I_i),$$

where the sets $\mathbb{X}_i$ and $\mathbb{D}_i$ are each disjoint. The parallel composition $||\mathcal{A}_i$ *of all ACTA $\mathcal{A}_i$ is defined by the network*

$$\begin{aligned}\Big|\Big|\mathcal{A}_i = &(Q_1 \times .. \times Q_n, C_1 \cup .. \cup C_n, B_1 \cup .. \cup B_n, \mathbb{X}_1 \cup .. \cup \mathbb{X}_n, \mathbb{D}_1 \cup .. \cup \mathbb{D}_n, \mathcal{I}, Ctrl_{Act},\\ &T, (q_{ini,1}, .., q_{ini,n}), I_1 \cup .. \cup I_n),\end{aligned}$$

where the respective invariants $\mathcal{I}_i$ are conjugated as follows:

$$\mathcal{I}(q_1, .., q_n) \overset{def}{\Longleftrightarrow} \bigwedge\nolimits_{i=1}^{n} \mathcal{I}(q_i).$$

For the construction of the transition relation T we observe the following two possibilities:

Interleaving: *If for an arbitrary ACTA $\mathcal{A}_i$ with $i\colon N$ with $q_i \in Q_i$, without any communication, i.e. $\mathsf{in}_i = true$ and $\mathsf{out}_i = \tau$, there exists a transition $(q_i, \varphi_i, true, \tau, c_{act,i}, \nu_{act,i}, {q_i}') \in T_i$, then for all $j\colon N\backslash\{i\}$ for all locations $q_j \in Q_j$ there exists the transition $((q_1, .., q_n), \varphi_i, true, \tau, c_{act,i}, \nu_{act,i}, (q_1, .., q_n)\{q_i \mapsto {q_i}'\}) \in T$. If $q_k \in C_k$ for some $k \in \{1, .., n\}$, then $q_i \in C_i$.*

Guarded communication via a broadcast channel: *In an arbitrary ACTA $\mathcal{A}_i$ for a channel $a\colon B_i$ and a data sequence $\overrightarrow{d} : \mathsf{seq}\,\mathbb{D}$, we have an output action $\mathsf{out}_i = a!\vec{d}$ and the transition $(q_i, \varphi_i, true, \mathsf{out}_i, c_{act,i}, \nu_{act,i}, {q_i}') \in T_i$. The other ACTA $\mathcal{A}_j$ with $j\colon N\backslash\{i\}$ and transitions $(q_j, \varphi_j, \mathsf{in}_j, \tau, c_{act,j}, \nu_{act,j}, {q_j}') \in T_j$ synchronise with the sender $\mathcal{A}_i$ as defined in Def. 50 iff their respective input actions in_j are defined on the channel a. Assuming that m ACTA $\mathcal{A}_j$ did synchronise (cf. Def. 50), we derive a transition $((q_1, .., q_n), \mathsf{syn}, \tau, c_{act,\mathsf{syn}}, \nu_{act,\mathsf{syn}}, (q_1, .., q_n)\{q_l \mapsto {q_l}'\}) \in T$, where $l \in \{i\} \cup \{1, \ldots, m\}$. If $q_k \in C_k$ for some $k \in \{1, .., n\}$, then $q_i \in C_i$ or $q_j \in C_j$, where $j \in \{1, \ldots, m\}$ and thus $\mathcal{A}_j$ is an ACTA that did synchronise with $\mathcal{A}_i$.*

Note that Def. 51 includes that whenever there exists an ACTA $\mathcal{A}_i$ in a committed location $q_i \in C_i$, then the transition relation T must include that $\mathcal{A}_i$ leaves its committed location.

Comparable to the semantics of one single ACTA as defined in Def. 47, the operational semantics of a network $||\mathcal{A}_i$ of ACTA $\mathcal{A}_i$ is defined by a transition system $\mathcal{T}(||\mathcal{A}_i)$. One specific configuration of $\mathcal{T}(||\mathcal{A}_i)$ now is defined by

$$\mathcal{C}_{||} = \langle \mathcal{TS}, \nu, \overrightarrow{q} \rangle,$$

where ν is the valuation $\nu(v)$ for all variables $v \in \mathbb{X} \cup \mathbb{X}_1 \cup .. \cup \mathbb{X}_n \cup \mathbb{D}_1 \cup .. \cup \mathbb{D}_n$ and $\overrightarrow{q} \in Q_1 \times .. \times Q_n$. However, in contrast to networks of pure timed automata without the extension of committed locations, for the semantics of networks of ACTA we can only partially re-use the semantics of a single ACTA $\mathcal{A}$ from Def. 47. The reason for this is that we have to consider the meaning of committed locations in the context of the entire network $||\mathcal{A}_i$. Thus, we informally describe the three possible cases for the transition relation of $\mathcal{T}(||\mathcal{A}_i)$:

Internal transition: The i-th ACTA $\mathcal{A}_i$ executes a transition, thus the respective guards and invariants must be satisfied as defined in Def. 47 for one single ACTA. The locations and variable valuation of all other ACTA remain unchanged.

Synchronisation transition: A synchronisation transition exists and the variable valuation satisfies the synchronisation expression syn as of Def. 50. For the satisfaction of syn, we refer to the satisfaction of guards and invariants (cf. p. 72). If there exist some ACTA $\mathcal{A}_i$ in a committed location, they have to be involved in the synchronisation process and leave their committed location in its course.

Delay transition: Time passes and for all ACTA $\mathcal{A}_i$ for all clock variables the variable valuation is changed as defined in Def. 47 for one single ACTA. For a delay transition, no ACTA $\mathcal{A}_i$ may be in a committed location.

Chapter summary. In this chapter, we presented syntax and semantics of automotive-controlling timed automata, which are used in the following chapters to construct controllers for traffic manoeuvres. Our ACTA are capable of committing controller actions for car manoeuvres, use formulae of UMLSL as guards and invariants and can use and analyse different data variables. Also a guarded broadcast communication concept for networks of ACTA was presented, with which controllers only synchronise, if a possible communication guard holds.

5 Controllers for Safe Crossing Manoeuvres

In Chapter 3, we introduced our abstract model for urban traffic and formalised the evolution transitions for changes in a traffic snapshot $\mathcal{TS}$ in Sect. 3.3. Apart from the passive time transition, occurring whenever time elapses, the changes in a traffic snapshot $\mathcal{TS}$ are actively triggered by the traffic participants. For this, we now introduce *controllers*, which use our logic UMLSL from Sect. 3.5 to reason about traffic situations and perform traffic manoeuvres. The underlying automaton type of our traffic manoeuvre controllers are the ACTA we introduced in the previous Chapter 4. One of the main requirements for the controllers is to preserve a *safety property* which can be formalised by the UMLSL formula

$$Safe \;\equiv\; \forall c, d : c \neq d \rightarrow \neg\langle re(c) \wedge re(d)\rangle. \tag{5.1}$$

Formula (5.1) states that for all possible pairs of different cars there does nowhere exist an overlap of their reserved spaces. We prove that the controllers which we introduce in this Chapter fulfil safety requirement (5.1) and other desirable system properties in the following Chapter 6.

For urban traffic manoeuvres, we introduce three types of controllers in this chapter: A *road controller* to cover lane change manoeuvres on road segments, a *crossing controller* to handle crossing manoeuvres and a *distance controller* that maintains the safety distance to a car in front or an occupied crossing. For this, we first assume a concept of *perfect knowledge*, where we assume a car has certain knowledge about other cars. Later in this chapter, we drop the assumption of perfect knowledge by introducing a crossing controller for *imperfect knowledge*, which has less information about other cars' behaviours and needs to communicate to cope for the lack of knowledge.

The structure of this chapter is as follows; in Sect. 5.1, we start with an overview over what different types of controllers we consider in our approach and also delimit our discrete control from dynamics components in an autonomous car. After that, we give a list of all assumptions for our controllers in Sect. 5.2, similarly as we did in Sect. 3.1 for our abstract model. The heart of this chapter is the actual construction of our crossing controller that we explain in Sect. 5.3, followed by a discourse about an approach to weaken some of our assumptions from Sect. 5.2 to make our approach more realistic in Sect. 5.4. For this, we introduce a communicating variant of our crossing controller that performs crossing manoeuvres with the help of other cars.

5.1 Interplay of Controllers: One Car, Several Controllers

In this section, we classify in which layer of an autonomous car the different controllers we construct in the following sections can be found. For this, we give an overview over the different layers in an autonomous system with Fig. 5.1, which is inspired by a respective figure from [KRS+12].

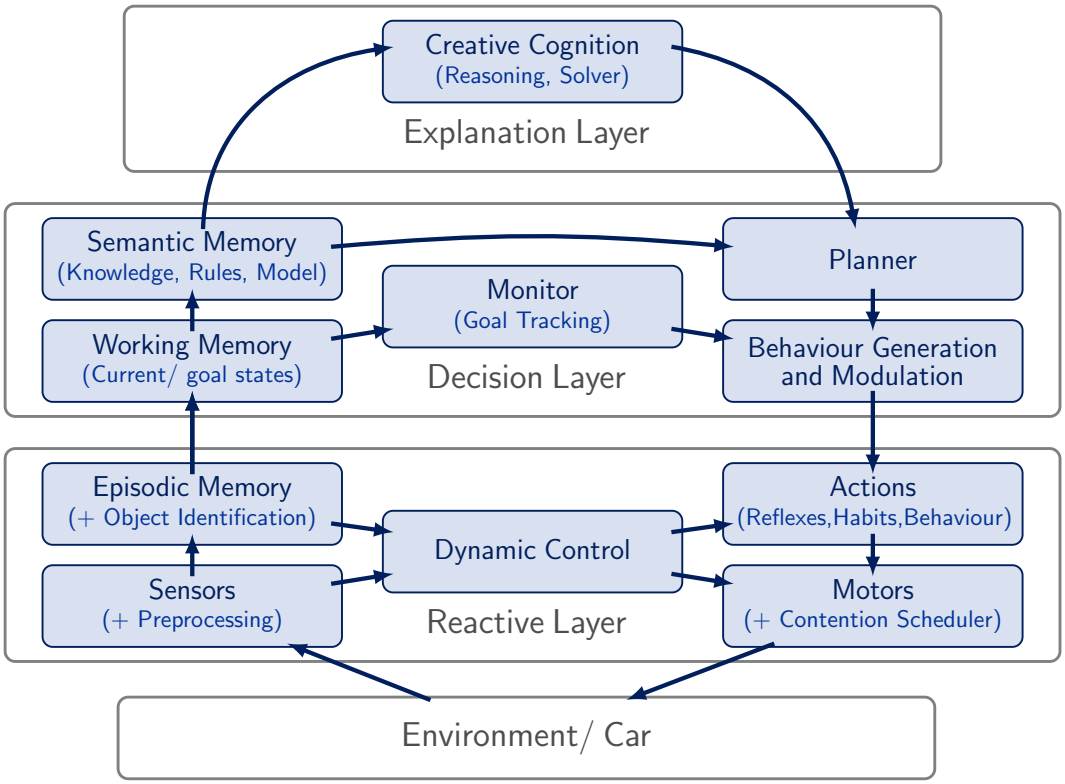

Figure 5.1: Overview over the interplay of discrete control and dynamics in an autonomous system.

In our approach, we explicitly separate our controllers from the reactive layer and focus on the decision making level. Thus, our MLSL controllers can be found in the decision layer, while dynamic control, e.g. an emergency braking manoeuvre, is planned and conducted directly in the reactive layer. That is, our controllers, e.g., decide how and whether a lane change or a crossing manoeuvre is conducted, but do not steer the motors for actually committing the manoeuvre. This approach allows for a purely spatial reasoning with our logic UMLSL.

We now map the concepts depicted in Fig. 5.1 to our approach, starting with the decision layer, where our MLSL controllers can be found.

Controllers on the Decision Layer

Considering Fig. 5.1, the *decision layer* gets information about its surroundings via object identification units from the reactive layer. Note that the decision layer does not process sensor data itself. These information about surroundings are an input of the *working memory* of an MLSL controller $\mathcal{A}$, which is the state the automaton is in, i.e. its current configuration $Conf(\mathcal{A})$ (cf. Sect. 4.2). This information received from the reactive layer thus might change the system state, e.g. because some space that was free before now is occupied by a car, as evaluated by an MLSL formula (cf. Sect. 3.5).

Respective *goal states* in our case are desirable system states of the controller, e.g. a state where a UMLSL formula formalising an *on-crossing check*

$$oc(\text{ego}) \equiv \langle re(\text{ego}) \wedge cs \rangle$$

holds invariantly in a location q_i. The on-crossing check states that somewhere there exists a reservation of car E on at least one crossing segment, meaning our car successfully entered an intersection. The *semantic memory* of an MLSL controller is the static knowledge it has about the road, e.g. about connections of segments as formalised by our urban road network $\mathcal{N}$ (cf. Sect. 3.2) and the existing preconditions for changes of the traffic snapshot $\mathcal{TS}$ (cf. Sect. 3.3).

Depending on the current state of the controller, together with possible (and desirable) goal states and knowledge about the semantic memory, respective decisions are *planned* by the controller (e.g. to reserve some crossing segments, because there is enough free space). This *reasoning* about traffic situations is done via *creative cognition*, which in our case is the crossing controller protocol we introduce in Sect. 5.3. As a result of the reasoning process, some needed *behaviour* is *generated* to conduct the previously planned manoeuvre (e.g. a controller action `rc`(ego) is committed, cf. Sect. 4.1).

Note that our controllers in the decision layer do not control any motors directly. For instance, if the crossing controller which we introduce later successfully reserves some space on the intersection with the action `rc`(ego), only this generated behaviour is forwarded to the reactive layer. Thus, our crossing controller does not control the motors' speed or the angle at which the car drives onto the intersection.

We introduce our crossing controller and a road controller for roads between intersections which are located on the decision layer later in Sect. 5.3.

Controllers on the Reactive Layer

Referring to Fig. 5.1, in this section we focus on the part of *dynamic control*. Thus, for details on *(preprocessing of) sensor information*, *object identification*, as well as *action and motor control* we refer to relevant related work (e.g. [WHLS15]). For dynamic control, we briefly introduce the field of duty of a *distance controller* and a *velocity controller*.

Distance Controller. In the beginning of this chapter we claimed that our controllers are designed to preserve, amongst other properties, the safety property (5.1) in the sense of disjointedness of reservations under all time and action transitions. Note that by demanding the disjointedness of (the speed-dependent) reserved spaces, the formula indirectly requires that any car C lowers its speed (to shorten its reserved space) when a car ahead of it starts breaking. Thus, to maintain safety property (5.1) under time transitions, each car is equipped with a *distance controller* $\mathcal{A}_{dc}$.

Furthermore, for urban traffic we demand that the distance controller keeps a distance ≥ 0 to an intersection, if the car does not get permission to enter the intersection fast enough. This means in the worst case the car comes to a standstill in front of the crossing until it successfully reserves the needed crossing segments for its turn manoeuvre.

We do not explicitly construct a distance controller in this thesis, but refer to a formally verified distance controller that can be found in [DHO06] (cf. Sect. 1.3 (related work), p. 8). This distance controller is applicable for avionics (cf. TCAS), train applications (cf. ETCS) or automotive applications. Further on, the authors provide proof rules for verifying safety of their distance controller. Additionally, in [RIA16], reasonable safety distances for autonomous vehicles are considered.

Distance keeping is also the major goal of *Adaptive Cruise Control (ACC)* approaches, where e.g. in [LMT15] Larsen, Mikučionis and Taankvist present a distance controller which is synthesised with the UPPAAL extension *Stratego* [DJL+15]. As the authors base their work on the spatial model of MLSL, this approach is of high interest for our work. 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. 5.2). Their idea is that the ego car keeps *always* track of its distance to the *front* car. Additionally, their goal is to minimise the distance between ego and *front*. For this, one UPPAAL automaton each for ego and *front* is used, additionally to a system controller.

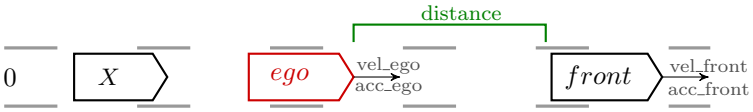

Figure 5.2: One-lane scenario with distance keeping from [LMT15].

For an extension of the approach in [LMT15] to a multi-lane scenario as needed for the highway traffic lane change controller, consider a traffic situation as depicted in Fig. 5.3. 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 the state space explosion, as the number of considered parallel timed automata for UPPAAL increases significantly when using the approach from [LMT15] directly. A second problem is the discretisation of space in

their approach. Further on, the extension to our urban traffic scenarios would be even more demanding.

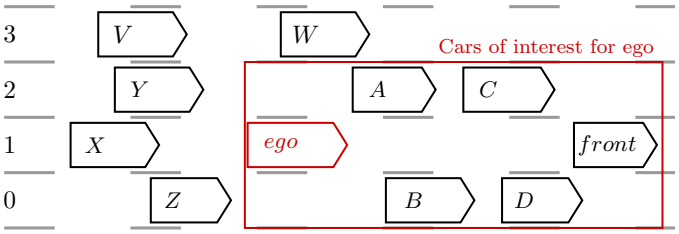

Figure 5.3: Cars of interest for ego car for distance keeping in multi-lane highway scenario [HLOR11].

Hence, in the implementations of our approach in UPPAAL we present later in Sect. 6.2 for highway traffic and in Sect. 6.4 for urban traffic, we do not implement a distance controller. We give details on why this is reasonable for the implementations directly in the respective sections.

Velocity and Steering Controller. For any acceleration, speed keeping and braking manoeuvres, we assume that each car is equipped with a *velocity controller* $\mathcal{A}_{vc}$ which sets inputs for the motors and actuators in the reactive layer. Such a controller is also responsible for e.g. the acceleration to enter an intersection after a car came to a standstill in front of an intersection. Being a variant of a velocity controller, we assume our velocity controller includes a *steering controller*, e.g. to enable lateral movement for our car when changing lanes or turning at intersections.

An example for a case study on a component-based velocity and steering controller on the reactive layer is given in [DMR14]. There, depending on certain system assumptions, certain guarantees are ensured to hold for the specified hybrid system. Further on, in [NBCF17], an approach for estimation of optimal trajectories for changing lanes is proposed. Particularly interesting in this approach is that in [CSB+17], the authors optimise their steering controller so that now it accelerates resp. brakes appropriately to fit into especially small traffic gaps on a neighbouring lane.

5.2 Assumptions for the Controllers

As in Sect. 3.1 on p. 37 where we postulate the assumptions for our abstract model for urban traffic situations, we have assumptions for our traffic manoeuvre controllers. Some of these assumptions are deduced from the previous Sect. 5.1. We explain the assumptions briefly here and refer forwards for formal details, whenever applicable.

All are cars autonomous. Later in Chapter 6 for analysing and proving the validity of our system properties, we assume that all cars follow specific rules. An example is that each car first needs to claim its crossing segments before entering an intersection. Moreover,

in our approach we do not consider a model for a human driver. This is necessary for proving absolute safety of our approach, as when e.g. considering a human driver falling asleep and losing control of his car somewhere on the road, we could not warrant 100% safety of our autonomous cars anymore. For an outlook on how to broaden our approach to mixed traffic situations see Sect. 8.4 on future work.

Perfect and imperfect knowledge. We use two different concepts of the degree of knowledge a car has about its surroundings. In Sect. 5.3 we first introduce a crossing controller using a concept of *perfect knowledge*, where every car perceives not only the physical sizes but also the braking distances of all other cars in its view. We again name the physical size of a car, together with its braking distance, *safety envelope*. Later in Sect. 5.4, we weaken this strong assumption of perfect knowledge to a more realistic approach with less knowledge. In the so-called concept of *imperfect knowledge*, we assume a car knows its own braking distance, but perceives only the physical size of other cars in its view.

Same technology. We demand that all our autonomous cars are equipped with the same technology, e.g. the same sensors. With this, we ensure that all cars are capable of interacting at a sufficient level with their surroundings.

Controllers. We demand that all cars are equipped with the controllers as specified in the previous and following sections of this chapter:

Longitudinal controllers: Distance controller for distance keeping and velocity controllers for speed keeping and braking/ acceleration (cf. Sect. 5.1)

Road controller: Controller for overtaking manoeuvres between intersections (cf. Sect. 5.3)

Crossing controller: Controller for turn manoeuvres at intersections (cf. Sect. 5.3)

With this, we ensure that other cars behave expectably and we can exclude unknown behaviour.

Communication. Each car is equipped with the same communication mechanism. With this, we know that all cars are capable to answer, whenever a car enquires something. For an example for related work on reliability and security of wireless mobile networks we refer to [PT02]. Note that we briefly present an approach on weakening the assumption about having 100% reliable communication later in Sect. 6.4.4, where we present an implementation of the crossing controller in UPPAAL Stratego.

Lane keeping. We abstractly assume that a car occupies the whole lane width, not taking swerving back and forth of a car within the lane into account in our approach. However, there exists a lot of related work on *lane keeping assistance systems* (LKAS) and we refer to [BDH+19] or [DMR14] for research on this topic.

5.3 Controller Construction

Let us now construct our controllers in the decision layer of Fig. 5.1. We first introduce the *crossing controller* $\mathcal{A}_{cc}$ for turning manoeuvres at crossings in Sect. 5.3.1. In Sect. 5.3.2, we adapt the lane change controller for two-way traffic from [HLO13] to a *road controller* $\mathcal{A}_{rc}$ for manoeuvres on road segments between intersections and for leaving intersections. Note that we prove desirable system properties, i.e. safety and liveness of $\mathcal{A}_{cc}$ and an adaptation to a fair crossing controller, later in Chapter 6.

5.3.1 Crossing Controller

Our *crossing controller* $\mathcal{A}_{cc}$ is based on the idea of the lane change controller from [HLOR11]. Therefore, we first *claim* an area we want to enter and *reserve* it only if no potential collision is detected. We assume a *crossing manoeuvre* to take at most t_{cr} time to be completed. The term 'crossing manoeuvre' e.g. includes manoeuvres like turning left and driving straight ahead. The crossing controller is constructed for the car under consideration E (also referred to as 'actor' or 'ego car' in the following) with $\nu(\text{ego}) = E$ but scales to all cars as ego can simply be substituted by an arbitrary car variable $c \in \mathbb{D}_{\mathbb{I}}$. We give an overview over the three possible *crossing controller phases* of $\mathcal{A}_{cc}$ in Fig. 5.4 and start with a description of the respective phases in the next paragraph. After that we introduce the detailed crossing controller which is depicted in Fig. 5.5. In Fig. 5.4 it is indicated, which locations q_i of the detailed controller map to which of the three phases. Note that the controller introduced in this section should be considered as a basic crossing controller protocol, which we show to be safe in Sect. 6.3. However, later in Sects. 6.4.2 and 6.4.3, we introduce respective extensions of $\mathcal{A}_{cc}$, each displaying a new desirable and provable feature beyond safety.

Crossing controller phases (cf. Fig. 5.4). The locations of the crossing controller can be summarised to three basic *phases*:

1. Away from the intersection (initial phase),
2. In the crossing ahead phase, where a distance d_c to the intersection is crossed over and
3. On the crossing, meaning the car now occupies some crossing segments.

We assume that initially in phase 1 no intersection is close but as soon as a crossing comes ahead within a *distance* d_c the car changes to the crossing ahead phase. We assume d_c is a constant, with a length of at least the size of the safety envelope of the fastest car with the weakest brakes. With that, any car can brake fast enough in front of the intersection if it does not get access to it soon enough. Note that this assumed size of the distance d_c includes that a car at maximum speed might have to conduct an emergency braking manoeuvre in front of an intersection. While this might lower the driving comfort, the system remains safe, which is the overall goal of our approach. Certainly, the size of

d_c might be adapted to increase driving comfort for future considerations. For this, we again refer to [RIA16], where reasonable safety distances for autonomous vehicles are considered.

Recall from Sect. 3.1 the assumptions we stated for our model for urban traffic. For instance, we stated that intersections are far enough apart from each other. With this, it is reasonable to assume that there exists a phase where a car is *away* from the intersection and thus the distance to the intersection is $> d_c$.

Phase 2 is the most complex phase, as several details have to be considered to ensure that no collision is caused on entering phase 3 and thus driving onto the intersection. Thus, to prepare the crossing manoeuvre, in phase 2 needed crossing segments are claimed and potential collisions are examined. If no potential collision is detected, phase 3 is entered and the crossing manoeuvre begins (i.e. the car is on the crossing now). After t_{cr} time we assume the crossing manoeuvre to be finished and phase 3 is left. Now the crossing controller is prepared for the next crossing manoeuvre as soon as another intersection comes ahead.

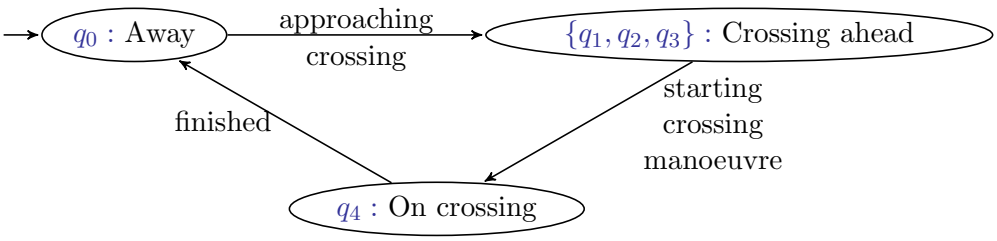

Figure 5.4: Overview over the crossing controller phases.

Details (cf. Fig. 5.5). We explain the construction of the crossing controller $\mathcal{A}_{cc}$ starting with the initial location. As our main goal, we want to prevent different reservations from overlapping, an thus recall the *collision check* formula for the actor E from p. 30 formalised by

$$col(\text{ego}) \;\equiv\; \exists c : c \neq \text{ego} \wedge \langle re(\text{ego}) \wedge re(c) \rangle. \tag{5.2}$$

Formula (5.2) is evaluated to true iff somewhere there exists a car different from E whose reservation overlaps with the actor's reservation. As this would be an unsafe situation, we assume $\neg col(\text{ego})$ to hold in the initial location q_0 of $\mathcal{A}_{cc}$. Next we need to detect whether E approaches an intersection. To that end, we formalise the *crossing ahead check* for car E by the formula

$$ca(\text{ego}) \;\equiv\; \langle re(\text{ego}) \frown (free^{<d_c} \wedge \neg\langle cs \rangle) \frown cs \rangle, \tag{5.3}$$

where the subformula $free^{<d_c}$ states that there is an interval of free space with a size less than d_c. With that, formula (5.3) states that in front of E's reservation there is

an intersection within less than d_c distance to E. Thus, car E has not yet entered the intersection, but is close to it within a distance in the interval $[0, d_c)$ and there is no other car between E and the intersection. We add a preferably small time bound t to the locations q_1 and q_2 to ensure progress of $\mathcal{A}_{cc}$. We also add the guard $x \geq t$ to the outgoing edges of q_1 and q_2, together with resetting x, whereas the time t may be considered as a reaction time for the controller, as it is unrealistic to consider immediate reactions. For a more thorough explanation for these time bounds, see Sect. 6.4.3, where liveness and fairness of $\mathcal{A}_{cc}$ is examined.

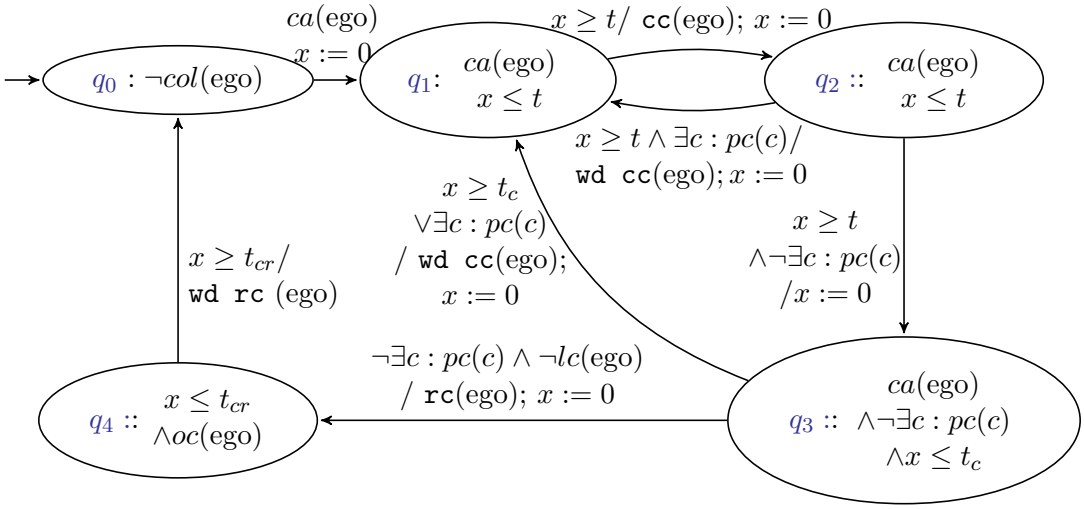

Figure 5.5: Crossing controller $\mathcal{A}_{cc}$.

Similarly to the case for changing lanes in Sect. 2.4, but now in order to enter a crossing, a car first needs to claim a path through the crossing (cc(ego)) and check if there are any overlaps of other cars' claims or reservations, for which we recall the *potential collision check* from p. 30

$$pc(c) \;\equiv\; c \neq \text{ego} \wedge \langle cl(\text{ego}) \wedge (re(c) \vee cl(c)) \rangle. \tag{5.4}$$

Formula (5.4) evaluates to true iff somewhere exists a car different from E whose claim or reservation on a lane or crossing segment overlaps with E's own claim. Unlike for the collision check (5.2), a (temporary) potential collision is allowed, because it does not endanger the safety property (5.1). However, if a potential collision is detected, the car must withdraw its claim immediately (wd cc(ego)).

Further on, we want to exclude that the actor is entering an intersection while changing lanes. Therefore we introduce the *lane change check*

$$lc(\text{ego}) \;\equiv\; \left\langle \begin{array}{c} re(\text{ego}) \\ re(\text{ego}) \end{array} \right\rangle, \tag{5.5}$$

which states that somewhere exist two neighbouring reservations for car E and therefore E currently changes lanes. When lc(ego) does not hold, the actor reserves the claimed path and starts the crossing manoeuvre. Again to ensure progress, we set a time bound t_c for the time that may pass between claiming and reserving crossing segments. Only if no potential collision is detected and there exists no ongoing lane change manoeuvre, the car may reserve the previously claimed crossing segments (`rc`(ego)). If car E is driving on the intersection, the *on-crossing check*

$$oc(\text{ego}) \equiv \langle re(\text{ego}) \wedge cs \rangle \tag{5.6}$$

we already introduced on p. 89 holds, meaning the ego car has entered some crossing segments. When the actor has left the last crossing segment and is driving on a normal lane segment, the crossing manoeuvre is finished. The reservation of actor E is then reduced to the lane segment that is the next segment in $pth(E)$ (`wd rc`(ego)).

Note that our crossing controller does not explicitly have the functionality for 'leaving an intersection', as the construction goal of the controller is to always enter an intersection *safely* and *timely* (liveness). Both of these properties are fulfilled, as we show in the next Chapter 6. Thus for now, let us assume that the part behind the intersection is implicitly reserved on reserving the crossing segments, e.g. by considering it as an additional crossing segment. For now, this results in several reserved segments at once for one crossing manoeuvre.

5.3.2 Road Controller

The *road controller* $\mathcal{A}_{rc}$ is responsible for overtaking manoeuvres on road segments between intersections. These road segments are structurally comparable to country roads, as in both cases neighbouring lane segments with (possibly) two different driving directions and thus oncoming traffic are considered. Thus, we refer to [HLO13], where an overtaking controller for these types of roads was presented (cf. Sect. 2.4). However, while in [HLO13] infinite lanes are considered, we now consider finite lane segments.

We thus modify the overtaking controller from [HLO13] to ensure that, as soon as the car approaches an intersection and ca(ego) holds, any claim for a lane segment must be withdrawn immediately and no new claim or reservation on a lane may be created until the crossing is entered. Additionally, the controller may only start an overtaking manoeuvre, if it can be finished before the car reaches an intersection. That is, it must be possible to finish the phase 'change back' from the protocol introduced in Sect. 2.4 before the intersection is reached (cf. Fig. 2.8).

However, the car may finish an already begun overtaking manoeuvre. Therefore we make sure that the distance d_c used in ca(ego) is big enough to do so. We depict a simplified version of the adapted overtaking protocol including its *phases* in Fig. 5.6. In the phase 'Prepare overtaking', the car has only claimed a lane and must withdraw the claim if ca(ego) holds, while in the phase 'Overtaking manoeuvre', the actual overtaking protocol

as of Fig. 2.8 is conducted, which may be finished even if $ca(\text{ego})$ holds. We add the invariant $lc(\text{ego})$ indicating the overtaking manoeuvre and $\neg oc(\text{ego})$ to ensure that the intersection is not entered before the overtaking manoeuvre is finished.

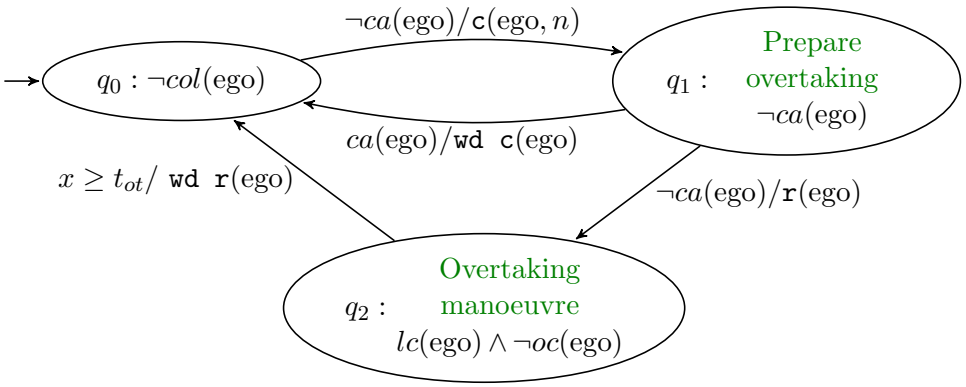

Figure 5.6: Simplified version of road controller $\mathcal{A}_{rc}$, based on the overtaking controller from [HLO13], adapted for changing lanes between intersections.

5.4 A more Realistic Approach: Manoeuvres with Imperfect Knowledge

Until now, we considered the concept of perfect knowledge for our cars, meaning that all cars know the braking distances of the other cars. As this is a strong assumption, let us now adapt the controller for perfect knowledge from the previous section 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 (cf. Sect. 5.2). To cope with this limitation, we extend the crossing controller by using the guarded broadcast communication with data constraints as introduced in Sect 4.3. Remember that there we consider output actions OUT which can synchronise with appropriate input actions IN in another controller. As a communication counterpart, we introduce helper cars, which roughly follows the helper approach for imperfect knowledge for highway traffic from [HLOR11].

This section bases on our work from [Sch17]. However, in [Sch17], we consider the 2-by-2 intersections as they were introduced in [HS16]. Thus, in this section, we introduce an adaptation of the approach from [Sch17] to the more complex intersections introduced in [Sch18a] (cf. Chapter 3).

For this, in Sect. 5.4.1, we describe the changes needed in the abstract model, where we also introduce a *communication view* covering all roads that meet at an intersection to enable communication with all cars approaching an intersection. Next, in Sect. 5.4.2, we first introduce our new controller concept, including a motivation for needed helper

cars to communicate with. We then subsequently introduce the adapted communicating crossing controller and the related helper controller.

5.4.1 Communication Multi-View

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. 5.7), but not the braking distance (cf. dashed parts). Only the 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. Remember that 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.

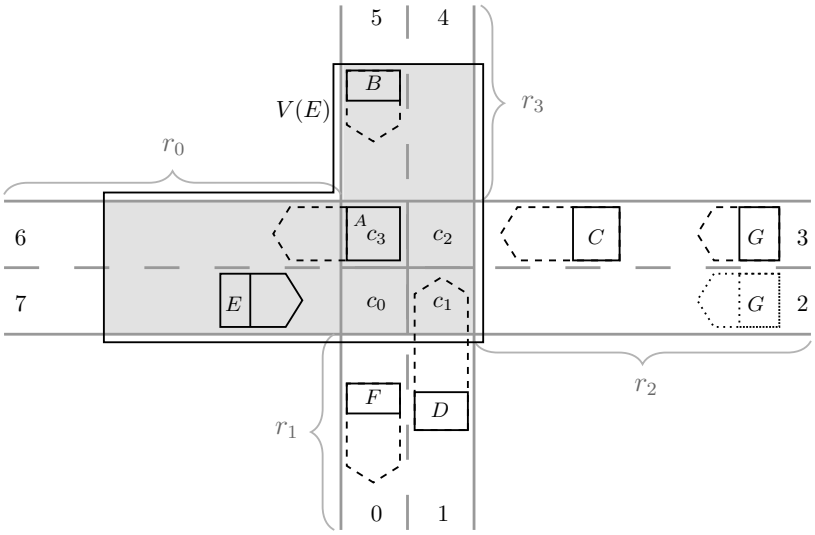

Figure 5.7: Car E perceives the physical size of other cars in its view $V_1(E)$. The dashed braking distances of other cars are invisible for E.

For perfect knowledge, it was sufficient to consider only that multi-view $V_M(E)$ for the actor E which corresponds to its path $pth(E)$ (cf. Sect. 3.4). In the example from Fig. 5.7, this is the view $V_{M_1}(E) = \{V(E)\}$, where E plans on turning left. However, with imperfect knowledge, E cannot perceive whether the safety envelope of a car that is not (yet) physically driving on the crossing already stretches to some crossing segments.

For this, consider that in Fig. 5.7 car E cannot perceive the braking distance of car D which already stretches to the intersection. Thus, 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 multi-view $V_{M_1}(E)$ and introduce the concept of a *communication view*

$V_C(E)$. The communication view $V_C(E) = \{V_{M_1}(E), V_{M_2}(E), V_{M_3}(E), V_{M_4}(E)\}$ for the example consisting of four multi-views $V_{M_i}(E)$ is depicted in Fig. 5.8. This view covers the already introduced view $V_{M_1}(E)$ ('turning left'), as well as the view $V_{M_2}(E)$ covering road segment r_0, the intersection and segment r_2 ('driving straight ahead'), the view $V_{M_3}(E)$, covering r_0, the intersection and r_1 ('turning right') and the view $V_{M_4}(E)$, covering r_0, the intersection and r_0 again ('u-turn'). Thus, a communication view is built from several multi-views. Note that we need to consider the u-turn direction as there can also be cars different from the ego car approaching from that direction, e.g. if there are more than one lane leading to the intersection from ego's road segment.

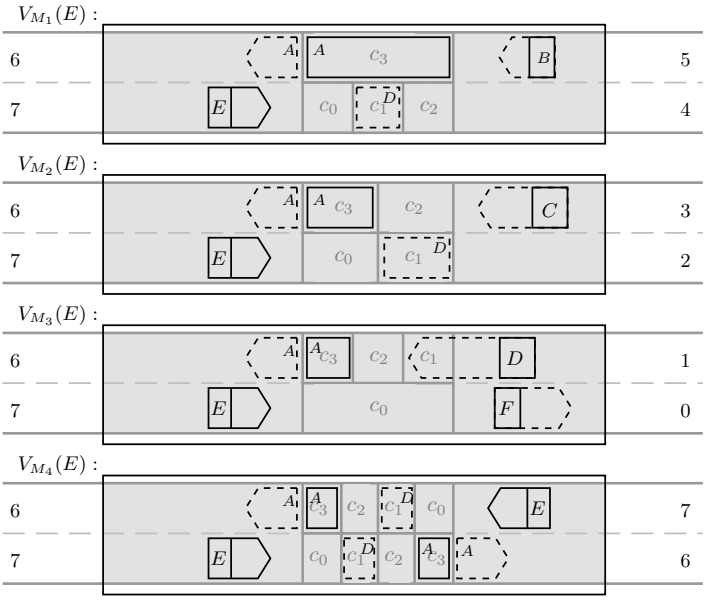

Figure 5.8: The communication view $V_C(E) = \{V_{M_1}(E), V_{M_2}(E), V_{M_3}(E), V_{M_4}(E)\}$ of car E covers the road segment E is driving on, the intersection and all other road segments linked to the intersection from Fig. 5.7.

To formally build the communication view, we first identify the road segment E is currently driving on with the underlying graph topology (cf. Sect. 3.2) and the data from the traffic snapshot (cf. Sect. 3.3). The current path segment E is driving on is defined by $pth(E)(curr(E))$ in the traffic snapshot (cf. Def. 20, p. 44) and the related road segment r is given through the strongly connected component $I_l(pth(E)(curr(E)))$ (cf. Def. 17, p. 42). In the example we have $pth(E)(curr(E)) = 7$ and $I_l(7) = r_0$. When a crossing is ahead, the first crossing segment E will drive on when entering the intersection is given by $pth(E)(curr(E) + 1)$ and therefore the whole intersection cr is obtained through the connected component $I_{cs}(pth(E)(curr(E) + 1))$. In the example we have $pth(E)(curr(E) + 1) = c_0$ and $I_{cs}(c_0) = cr$.

Next we identify all road segments r' which are connected to the intersection cr with a directed edge (r', cr) in the coarser graph $\mathcal{N}_\mathcal{C}$ (cf. Def. 18, p. 18). This way, we derive

all road segments r' from which cars can enter the junction and derive respective virtual views $V_{M_i}(E)$ as defined in Defs. 30 to 32 in Sect. 3.4. Finally, we collect all of these virtual views $V_{M_i}(E)$ in our communication view $V_C(E)$. This sketched procedure is formalised in the following definition.

Definition 52 (Communication view)**.** *Consider a car E and a traffic snapshot $\mathcal{TS}$ with $pth(E)(curr(E))$ as current path segment. As a precondition assume $pth(E)(curr(E)) \in \mathbb{L}$ and $pth(E)(curr(E)+1) \in \mathbb{CS}$. We derive a set $\overrightarrow{\Pi}_C$ of coarse path sequences $\overrightarrow{\pi_\mathcal{C}} \in \mathsf{seq}_{E_\mathcal{C}}\, \mathcal{V}_\mathcal{C}$ through the intersection (cf. Def. 29) using the coarse network $\mathcal{N}_\mathcal{C}$:*

$$\overrightarrow{\Pi}_C == \{\overrightarrow{\pi_\mathcal{C}} : \mathsf{seq}_{E_\mathcal{C}}\, \mathcal{V}_\mathcal{C} \mid \exists r, cr, r' : \mathcal{V}_\mathcal{C} \bullet \overrightarrow{\pi_\mathcal{C}} = \langle r, cr, r' \rangle \wedge r = I_l(pth(E)(curr(E))) \wedge cr = I_{cs}(pth(E)(curr(E)+1)) \wedge (r', cr) \in E_\mathcal{C}\}. \quad (5.7)$$

For each $\overrightarrow{\pi_\mathcal{C}} \in \overrightarrow{\Pi}_C$ and with $i \in \{1, .., \#\overrightarrow{\Pi}_C\}$, we build a respective virtual view $V_{M,\overrightarrow{\pi_\mathcal{C}}}(E, \mathcal{TS})$ as of Defs. 30 to 32 from Sect. 3.4 and derive our communication view *V_C with*

$$V_C(E) == \{V_{M,\overrightarrow{\pi_\mathcal{C}}}(E, \mathcal{TS}) \mid \overrightarrow{\pi_\mathcal{C}} \in \overrightarrow{\Pi}_C\}. \quad (5.8)$$

With formula (5.7), we determine all finite path sequences $\overrightarrow{\pi_\mathcal{C}} \in \mathsf{seq}_{E_\mathcal{C}}\, \mathcal{V}_\mathcal{C}$ through the intersection that have the structure $\overrightarrow{\pi_\mathcal{C}} = \langle r, cr, r' \rangle$. Here, the road segment r we are currently driving on and the next intersection cr are retrieved through their respective strongly connected components I_l and I_{cs}. With the subformula $(r', cr) \in E_\mathcal{C}$, we ensure that it is possible to enter the intersection from road segment r'. Finally, the views $V_{M,\overrightarrow{\pi_\mathcal{C}}}$ we collect in formula (5.8) are the respective virtual multi-views relating to a coarse path sequence $\overrightarrow{\pi_\mathcal{C}} \in \overrightarrow{\Pi}_C$ from formula (5.7).

With this notion of a communication view, we now extend our crossing controller protocol from Sect. 5.3 by communication parts.

5.4.2 Communicating Crossing and Helper Controller

Similarly to the crossing controller $\mathcal{A}_{cc}$ from Sect. 5.3, the ego car must withdraw its claim, whenever a potential collision is detected with the potential collision check $pc(c)$. 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, it has to communicate with cars that might cause a potential collision. Following [HLOR11], we call those cars *helper cars*. We motivate how to determine these helper cars in the following.

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 first case where a car is driving on a crossing segment is formalised by the already known on crossing check $oc(c)$ from formula (5.6), p. 96.

The second case, meaning a car $C \in \mathbb{I}$ different from the ego car E approaching the intersection from another direction, is a little bit more complicated. 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 an 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 extended distance $d'_c = d_c + se\_max(C)$. We formalise that a car approaches an intersection from an opposite side of the intersection within the distance d'_c with the *opposing car approaching the crossing check*

$$ocac(c) \equiv \langle re(\text{ego}) \rangle \frown \langle cs \frown \neg cs \wedge free^{<d'_c} \frown re(c) \rangle \wedge dir(c). \tag{5.9}$$

The atom $dir(c)$ in formula (5.9) returns whether a car drives in the direction *to* the intersection, which the ego car is able to perceive with its sensors. This atom is needed to exclude cars driving away from the intersection. A car that is driving away from the intersection is not of interest, as its own braking distance cannot stretch to the intersection and as it might leave the view of ego car E soon anyway.

Remember that we generally forbid a car entering an intersection while changing lanes as the directed edges in our topology do not allow this (cf. Sect. 3.2). Therefore, we re-use the lane change check $lc(\text{ego})$ from formula (5.5), p. 95.

With formulae (5.6), (5.9) and (5.5), the ego car identifies all described suitable helper cars apart from itself with the *potential helper check*

$$ph(c) \equiv c \neq \text{ego} \wedge (oc(c) \vee ocac(c)) \wedge \neg lc(c). \tag{5.10}$$

Crossing Controller with Communication

We now construct the adapted crossing controller $\mathcal{A}'_{cc}$ with imperfect knowledge, based on the crossing controller $\mathcal{A}_{cc}$ from Sect. 5.3 (cf. Fig. 5.5). Again, the overall goal of the crossing controller is to perform turn manoeuvres at intersections while always maintaining the safety property (5.1). We keep several concepts of the controller $\mathcal{A}_{cc}$ from Sect. 5.3, e.g. the basic idea of claiming and only after that reserving crossing segments. We start with explaining the new communication concept using a coarser version of $\mathcal{A}'_{cc}$ in Fig. 5.9. After that, we introduce details concerning the detailed crossing controller $\mathcal{A}'_{cc}$ which is depicted in Fig. 5.10. Note that the location names q_i denoted in Fig. 5.9 refer to the respective location in the detailed controller in Fig. 5.10.

Overview over the crossing controller phases (cf. Fig. 5.9). We again assume the initial location of the controller to be safe, i.e. no collision exists (location q_0). When a crossing is ahead (locations q_1 and q_2), the car may enter the intersection by itself iff no helper car exists (e.g. the multi-view is empty except for the ego car). This case resembles the crossing procedure of the original crossing controller $\mathcal{A}_{cc}$ without communication from Fig. 5.5, as $\mathcal{A}_{cc}$ also commits crossing manoeuvres without help.

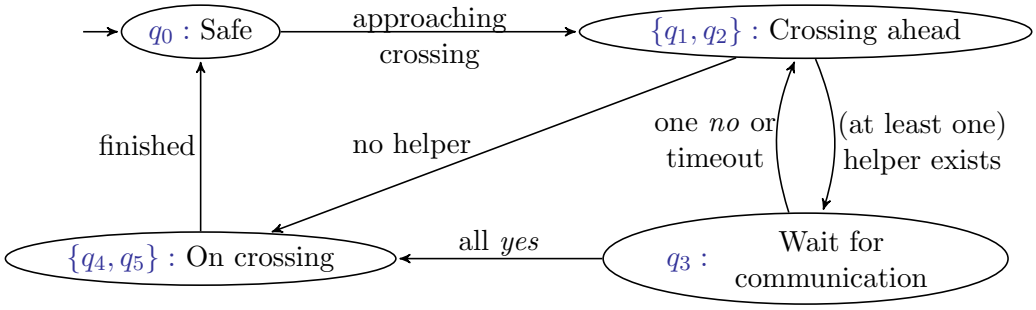

Figure 5.9: Overview over crossing controller protocol $\mathcal{A}'_{cc}$ with communication.

If at least one potential helper exists, the actor needs to communicate with the helpers. For this, after sending a broadcast request, $\mathcal{A}_{cc}$ waits for answers (location q_3). If one helper sends a *no*-message or one helper does not answer timely, the actor withdraws the crossing claim and may try to enter the intersection later again (eventually the conflicting other car will have left the intersection). If and only if *all* helpers send a *yes*-message, the ego car may safely enter the intersection and finish the crossing manoeuvre.

Details (cf. Fig. 5.10). If the formula $ca(\text{ego})$ (5.3) holds, the crossing controller becomes active and *claims* the crossing segments needed for the turn manoeuvre with the controller action $\texttt{cc}(\text{ego})$ on the transition from location q_1 to q_2. If no potential collision is detected, the controller evaluates whether a helper for the manoeuvre is available. This is done with the potential helper check $ph(c)$ (5.10). For $ph(c)$ 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 (transition above of q_2). Similarly as for $\mathcal{A}_{cc}$, if the lane change check $lc(\text{ego})$ and the potential collision check $pc(c)$ do not hold, the actor reserves the claimed crossing segments and starts the crossing manoeuvre moving to location q_5. We again assume the crossing manoeuvre to be finished after t_{cr} time. The reservation of the ego car is then reduced to the next segment after the intersection in $pth(E)$.

In the second case, where helper cars are available, the crossing controller needs to communicate because of the missing information about the braking distances of the helpers (transition below of q_2). E sends the message $cross!\langle \text{ego}, cs_{\text{ego}} \rangle$, where cs_{ego} is the set of crossing segments the ego car claims according to $pth(E)$. We attach the set cs_{ego} to the broadcast message, as with imperfect knowledge the other cars also do not now the full safety envelope of the ego car. With this, cars can compare their respective own sets of claimed or reserved crossing segments with cs_{ego} and with this detect a potential collision.

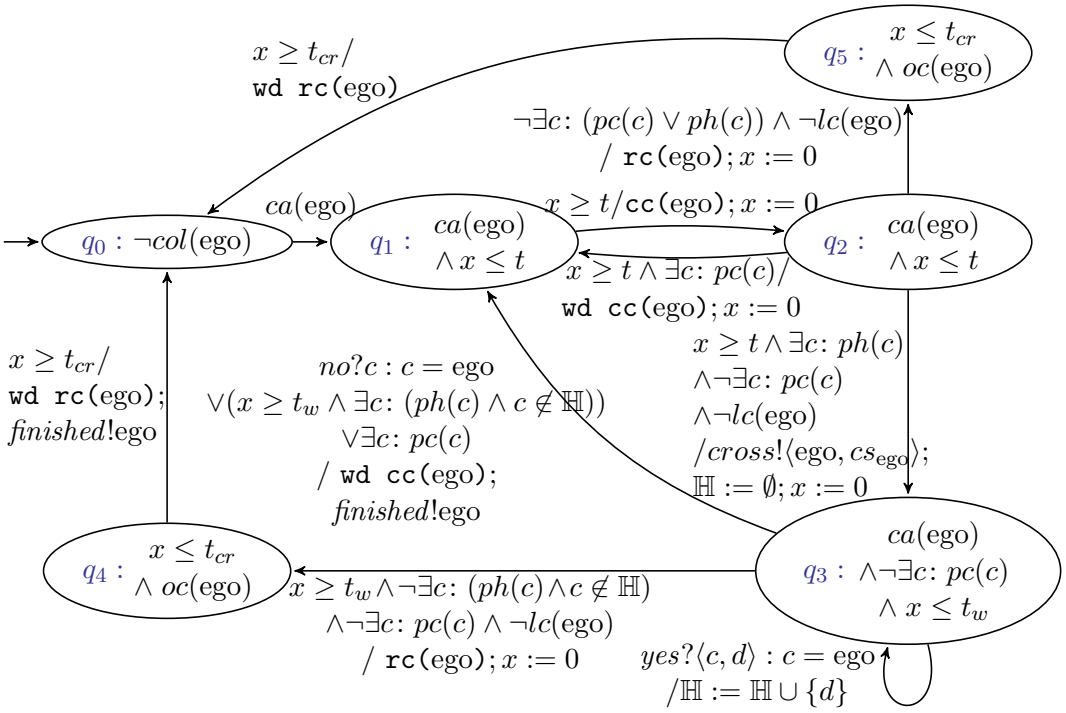

Figure 5.10: Communicating crossing controller $\mathcal{A}'_{cc}$.

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 to enter the intersection. 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 refer to the work of Asplund et al. [ANT12].

On waiting in location q_3, the controller $\mathcal{A}'_{cc}$ 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 $\neg\exists c\colon (ph(c) \wedge c \notin \mathbb{H})$. This guard holds if there does not exist a car not collected in $\mathbb{H}$ which is a potential helper and thus did not answer timely.

Depending on whether all potential helpers answered with *yes* timely, the controller either reserves the claimed crossing segments with `rc`(ego) or withdraws the claim with `wd cc`(ego), if at least one potential helper did not answer. Once the crossing controller entered location 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*. With this the helper cars are informed that they do no longer need to help the ego car.

Helper Cars and Helper Controller

As introduced in the beginning of this section, a helper car is either driving on the crossing or approaching it from a different lane segment than the ego car. An arbitrary car is allowed to be helper for more than one requesting car. This is needed e.g. if four cars turn simultaneously right at an intersection. We therefore assume that every car owns several clones of the *helper controller* $\mathcal{A}_{hc}$, but only one of the helper controllers assists one specific car at once. As done in the previous paragraph for the $\mathcal{A}_{cc}$, we first briefly explain an overview over $\mathcal{A}_{hc}$ which is depicted in Fig. 5.11. The detailed helper controller $\mathcal{A}_{hc}$ is depicted in Fig. 5.12 and explained in the remainder of this section. Note that we call a car searching for a helper *enquirer* from now on.

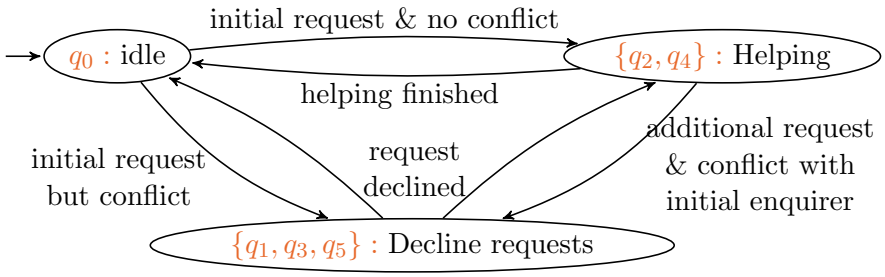

Figure 5.11: Overview over helper controller protocol.

Overview over the helper controller phases (cf. Fig. 5.11). Whenever an idle helper controller receives a crossing request it checks if it meets the helper requirements (on crossing or approaching crossing) and whether there exist a potential collision of the request with its own crossing claim or reservation. Depending on this, 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. 5.12). In the helper controller $\mathcal{A}_{hc}$ we use the unique variable a to identify the helper controller. Furthermore, we use the set $cs_a == cclm(a) \cup cres(a)$ to denote the claimed and reserved crossing segments of the helper car itself. If a car receives a broadcast request $cross!\langle c, cs\rangle$, its helper controller $\mathcal{A}_{hc}$ first checks if it is a potential helper for c with the *inverse potential helper check*

$$ph^{-1}(c, cs) \;\equiv\; a \neq c \wedge (oc(a) \vee ca(a)) \wedge \neg lc(a) \wedge (cs_a \cap cs = \emptyset). \tag{5.11}$$

With the first part of the formula, the potential helper checks if its position is suitable (i.e. approaching or on crossing) and whether it is currently changing lanes. With the latter part of the formula the potential helper checks for disjointedness of its own crossing segments cs_a and the received crossing segments cs. Note that this check resembles the potential collision check for lanes. However, we cannot simply use the potential collision check here, as with imperfect knowledge the helper car itself also does not perceive the

safety envelope of the enquirer. If $cs_a \cap cs = \emptyset$ does not hold and thus the helper controller detects a potential collision, it changes to the *urgent location* q_1. If a controller is in an urgent location, not time may pass before the urgent location was left. Thus, after entering q_1, $\mathcal{A}_{hc}$ immediately sends a *no*-message to the enquiring car (cf. transitions between locations q_0 and q_1).

Note that for timed automata, the concept 'urgency' is weaker than the concept 'commitment' (cf. Sect.2.3, p. 74). This is because urgency allows that other automata take transitions while another automaton is in an urgent location. Due to this, committed locations are generally more deadlock-prone than urgent locations. Also in our case, we would allow a deadlock to occur, if we would use committed instead of urgent locations for $\mathcal{A}_{hc}$. For instance, consider two cars A, B that both have intersecting crossing segments with an enquirer E. Due to that, the related helper controllers of A and B would both change to their respective location q_1, if E sends a request $cross!\langle \text{ego}, cs_e \rangle$. This would cause a deadlock, if q_1 would be committed.

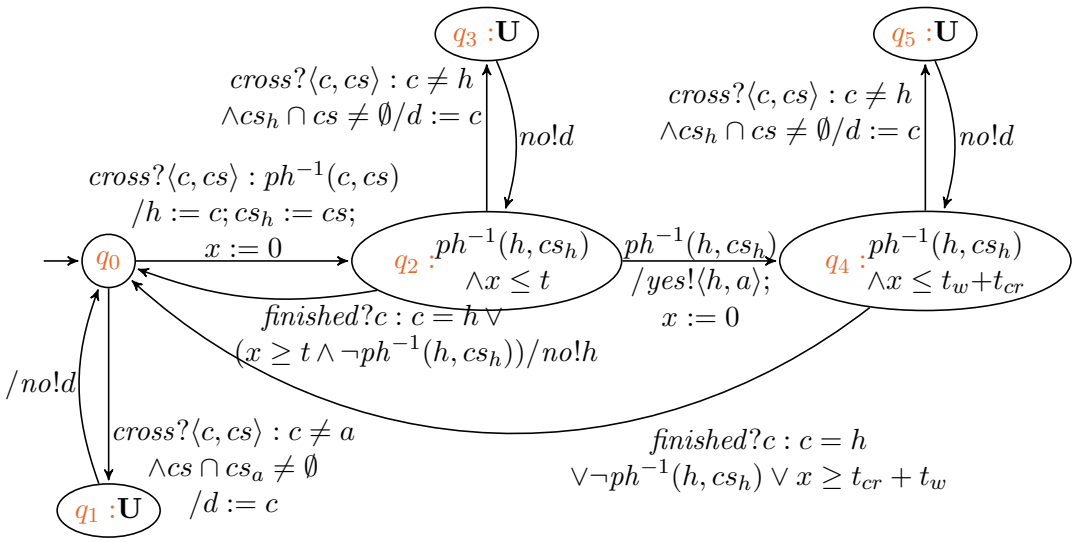

Figure 5.12: Helper controller $\mathcal{A}_{hc}$.

If a car is a potential helper (including that no potential collision with the enquirer exists), $\mathcal{A}_{hc}$ changes to location q_2 and stores the received car identifier of the car it is helping in an internal variable h. Further on, it stores the received set of crossing segments of the car h relates to in the internal variable cs_h.

Once in location q_2, $\mathcal{A}_{hc}$ sends a *yes*-message to the enquiring car in up to t time units. We assume $t < t_w$, where t_w is the time bound the crossing controller waits for the answers of the helpers (cf. Fig. 5.10). With that, it is ensured that the *yes* answers

are sent timely enough. We do not require the *yes* messages to be sent urgently, as we assume the communication to take some time.

While helping, $\mathcal{A}_{hc}$ additionally declines crossing requests from other cars whose requests overlap with the crossing segments of the car the helper already assists (cf. transitions upwards from q_2 and q_4). Here, we use the internal variable d to store the identifier of additionally enquiring car whose set of crossing segments overlaps with cs_h in. We again require the locations q_3 and q_5 to be urgent to ensure that the *no*! messages are sent without any elapse of time to the enquiring car.

The helping process is finished and $\mathcal{A}_{hc}$ returns to its initial location for two possible cases. In the first case, the car $\mathcal{A}_{hc}$ is located in left the intersection and with that no longer is a potential helper, meaning the inverted potential helper check $ph^{-1}(h, cs_h)$ does not hold anymore. The second case is that the crossing manoeuvre of the enquirer is finished, whereas $\mathcal{A}_{hc}$ receives the message *finished*?$c : c = h$ from the originally enquiring car h.

5.5 Related Work

Our approach is to *construct* controllers that behave correct w.r.t. certain desired system behaviour and properties. Another approach is to *synthesise* controllers from given properties. This was investigated for basic MLSL for highway traffic in [BHLO17]. Also, in [DPRW13], the authors propose a design and verification approach for cooperative driver assistance systems, where controller strategies are synthesised. However, in [BHLO17], the synthesised controllers abstract from a continuous time dimension and in [DPRW13], only driver assistance systems are considered instead of fully autonomous cars. Moreover, it is far more difficult to synthesise controllers for the more complex urban traffic case. Nonetheless, controller synthesis is a direction worthwhile to investigate in future work (cf. Sect. 8.4).

The idea pursued in this thesis, to separate dynamics from control laws, follows the work by Raisch et al. [MRO02] and van Schuppen et al. [HCvS06]. However, it is still of great interest to relate the purely spatial reasoning to car dynamics and thus link the approaches with MLSL to hybrid systems. In [ORW17], the authors propose a concrete dynamic model suitable for the abstract model for MLSL. There the authors refine the abstract spatial atoms of MLSL to concrete distance measures.

Also related, in [KDM$^+$17] the authors distinguish between dynamic behaviour and higher-level planning components in an approach concerning vehicle platooning. A combined verification approach is used, where UPPAAL is used to verify timing behaviour and the model-checker AJPF [DFWB12] is used to evaluate autonomous decisions. Particularly interesting, in [KLF19] an approach was presented to combine the approach of [KDM$^+$17] with the MLSL highway traffic controller from [HLOR11]. Additionally to

the results already achieved within [KDM+17], the authors can now also guarantee correctness of their spatial model by using the results from [HLOR11]. On the other hand, the authors of [HLOR11] now have a sound extension of their lane change controller to a platooning system.

Further on, in [LP11] an approach only fitted for simple intersections of single lanes with one car on each lane is proposed. The authors use traffic lights as a central control mechanism, where a car is not permitted to enter an intersection when the light is red. Though limited to single lanes with only one car on each lane, the strong point of this work is that the authors verify the safety of their hybrid systems with the tool KeYmaera.

In Sect. 3.7, we briefly introduced the approach from [ASS11] where a discretised traffic network is used as a model. In that approach, cars have no right to access intersection nodes by themselves, as those are controlled by a centralised control mechanism called *Autonomous Intersection Management (AIM)*. Such centralised control mechanisms for intersections are often collected under the name *cooperative intersection management* (CIM), where a road-side unit, e.g. a traffic light, acts as a centralised scheduler (e.g. also [LP11]). However, these approaches are limited to signalised intersections. In contrast to that, we use decentralised controllers that independently negotiate access to all parts of the network. The advantage of this is that no roadside-units are required. Furthermore, our controllers accomplish their manoeuvres as independently as possible, and thus use as few as possible communication actions (cf. crossing controller $\mathcal{A}_{cc}$ from Sect. 5.3). This has the advantage of minimising faulty behaviour that is caused by insecure or unreliable communication mechanisms. Note that we nevertheless take uncertain communication into account for our approach later in Sect. 6.4.4.

Certainly, several other approaches on decentralised traffic control without road-side units exist. We briefly introduce an approach on a *virtual traffic light* (VTL) for urban traffic platooning in Sect. 1.3. In [FFCa+10], a VTL concept is used for scheduling at intersections. There, one of the vehicles approaching an intersection is cooperatively selected to be the *VTL leader*. This leader then again acts as a central scheduler for all cars approaching the crossing. However, [FVP+13] points out that a problem with VTL is the leader selection, as this a) takes some time and b) possible communication failure during the negotiation phase may lead to a disagreement in the leader selection.

For more literature concerning both decentralised and centralised CIM, we also refer to the overview paper [CE16].

Chapter summary. In this chapter, we distinguished controllers working on different layers in an autonomous car and briefly introduced the assumptions we have about controllers on the reactive layer, e.g. a distance controller. We then introduced our crossing controller for the case of perfect knowledge, followed by an extension to a communicating crossing controller committing crossing manoeuvres at intersections with the help of other cars.

6 Desirable System Properties for Autonomous Cars

In [HLOR11] and [HLO13], as well as initially in this thesis, the main goal for the introduced traffic controllers was to fulfil a *safety* property. For this, the safety formula (5.1) was given in the beginning of Chapter 5. While safety is a crucial system property for autonomous cars, other system properties are also desirable. Consider e.g. a controller with which no autonomous car ever moves. This controller obviously fulfils the safety property, but *liveness* is non-existent. The desirable system properties we examine for the (U)MLSL controllers are the following:

- Safety,
- Liveness,
- Bounded liveness, and
- Fairness.

We give details of our approach for showing these system properties, together with our interpretation of their (formal) meaning in Sect. 6.1. Afterwards, we start with examining the system properties for the lane change controller for highway traffic with an UPPAAL[BDL04] implementation in Sect. 6.2. In Sect. 6.3, we prove safety of the crossing controller for urban traffic for both perfect and imperfect knowledge with a mathematical proof by hand. We then examine (bounded) liveness and fairness of our crossing controller from Sect. 5.3 in Sect. 6.4, again with UPPAAL.

6.1 Approach and Meaning of the System Properties

UMLSL formulae are used to specify purely spatial aspects in *one single* traffic snapshot $\mathcal{TS}$. Now, for proving system properties, we show the validity of the property throughout possible traffic snapshots reachable from one initial traffic snapshot $\mathcal{TS}_0$. For this, we analyse the property throughout the *system's runs* (cf. Sect. 4.2). In our case, the *system* consists of controllers for all cars inside the specified view around the ego car (cf. Sect. 3.4), all executed in parallel. Recall from Sect. 4.4 that one specific *system state* (or configuration) $\langle \mathcal{TS}, \nu, \overrightarrow{q} \rangle$ of such a network of ACTA contains the location q_i of each automaton, a variable valuation ν and the current traffic snapshot $\mathcal{TS}$. A run of such a system contains one specific evolution of an initial configuration $\mathcal{C}_{ini}$ to a

future configuration $\mathcal{C}_k$, also including the evolution of an initial traffic snapshot $\mathcal{TS}_0$ to a future traffic snapshot $\mathcal{TS}_k$. As an example for proving a system property, it could be shown that in all possible system runs, there does not exist a configuration, where the considered property does not hold. Alternatively, it might be of interest to show that a specific property holds at least once during each run. These system properties are expressible as formulae of *temporal logic* [Pnu77]. In temporal logic, the expression $\Box\,\varphi$ formalises that φ holds *globally* and with $\Diamond\,\varphi$ it is formalised that φ holds *finally*.

Besides a mathematical proof provided in Sect. 6.3, we use respective UPPAAL [BDL04] implementations for the highway traffic lane change controller from [HLOR11] in Sect. 6.2 and for the urban traffic crossing controller in Sect. 6.4 for showing our properties. With UPPAAL, we can conveniently analyse all possible runs of our controllers and hereby detect and correct unwanted behaviour. As UPPAAL implements *model-checking* for (extended) timed automata [ACD90, HNSY94, LPY95], it starts from one specific traffic snapshot $\mathcal{TS}_0$ as its *model* and then simulates all possible system runs starting from $\mathcal{TS}_0$ to verify a query. If a property is not satisfied, it is possible to analyse a counterexample trace.

As specification language for their *verification queries*, UPPAAL uses a subset of *timed computation tree logic* (*TCTL*) [ACD90, HNSY94]. In addition to the modal operators $\Box$ and $\Diamond$ from temporal logic, traditional CTL [CE82, QS82] introduces quantification over paths of the system. Here, **A** means 'along all paths', while **E** means 'along at least one path'. The main difference between TCTL and traditional CTL is that TCTL allows clocks. The specific subset of TCTL UPPAAL uses is TCTL with only one path quantifier.

The version of UPPAAL we use is the UPPAAL extension *Stratego* [DJL$^+$15] (version 4.1.20-stratego-4). UPPAAL Stratego includes *UPPAAL SMC* [DLL$^+$15], which allows for statistical model checking. With that, we can estimate the probability with which a property holds among random runs of the system within some time bound, if that property does not hold generally in our system. Note that we never actually use the extensions Stratego itself offers (strategies for stochastic priced timed games) in this thesis, but still use UPPAAL Stratego, as to our best knowledge this includes the latest version of UPPAAL SMC. For reasons of brevity note that whenever the name 'UPPAAL' is written in the remainder of this chapter, we mean the previously mentioned version of UPPAAL Stratego, resp. UPPAAL SMC.

One after the other, we motivate our system properties in the next paragraphs. We also give interpretations of our properties as CTL-style formulae. Note that we do not formally extend our UMLSL from Sect. 3.5 by temporal operators. Thus, the CTL-style formulae given in the following should only be considered as a motivating instrument for the following sections. One benefit of giving these CTL-style formalisations here is that they may be mapped conveniently to the respective UPPAAL queries we introduce later in Sects. 6.2 and 6.4. Further on, with the CTL-style formalisations, in particular with using the quantifier **A**, we can express that a property is supposed to hold throughout *all* reachable traffic snapshots.

Safety

As introduced in the beginning of Chapter 5, *safety* in our case means collision freedom, expressed by the UMLSL formula

$$Safe \;\equiv\; \forall c, d : c \neq d \rightarrow \neg\langle re(c) \wedge re(d)\rangle. \tag{6.1}$$

We require formula 6.1 to hold *globally* throughout *all* reachable traffic snapshots, starting from an arbitrary initial safe traffic snapshot $\mathcal{TS}_0$. We formalise this with the CTL-style formula

$$\mathbf{A}\Box\ Safe. \tag{6.2}$$

We show safety of our crossing controller with a mathematical proof by induction over *all* reachable configurations $\langle \mathcal{TS}, \nu, q\rangle$ in Sect. 6.3. More precisely, we prove that independent of the location q and the valuation ν, *all* traffic snapshots $\mathcal{TS}$ which are reachable from an *arbitrary* safe initial traffic snapshot $\mathcal{TS}_0$ remain safe.

Additionally, we subsequently confirm the mathematically proven safety result of the lane change controller from [HLOR11] and our crossing controller from Sect. 5.3 with each a UPPAAL implementation that we present in the respective Sects. 6.2 and 6.4. In contrast to our mathematical proof by induction, UPPAAL can only use a specific initial model for model checking. Thus, only one specific initial traffic snapshot $\mathcal{TS}_0$ is used as starting point of the verification. However, we use representative models $\mathcal{TS}_0$, allowing for a variety of possible future traffic snapshot $\mathcal{TS}_k$.

Liveness

According to [Lam77], *liveness* means that something good *finally* happens. 'Something good' in our case is a desired lane change manoeuvre for the highway traffic controller respectively a crossing manoeuvre for our crossing controller, as expressed with the following MLSL formulae for an arbitrary car C with $\nu(c) = C$:

$$Good_H(c) \;\equiv\; \left\langle \begin{array}{c} re(c) \\ re(c) \end{array} \right\rangle \qquad\qquad Good_U(c) \;\equiv\; \langle cs \wedge re(c)\rangle. \tag{6.3}$$

Here $Good_H(c)$ implements the lane change check given with formula (5.5), p. 95 and $Good_U(c)$ is the on-crossing check $oc(c)$ from formula (5.6), p. 96. After a claim was set the respective formulae (6.3) should *finally* hold in some traffic snapshot $\mathcal{TS}$. This can be expressed by the CTL-style formula

$$Live_i \;\equiv\; \mathbf{A}\Box\,\forall c\colon (\langle cl(c)\rangle \rightarrow \mathbf{A}\Diamond\ Good_i(c)), \tag{6.4}$$

where $i \in \{H, U\}$. With formula (6.4) we state that for all cars, whenever a claim resp. crossing claim was set, finally the car changes lanes respectively enters an intersection.

Note that this type of liveness formula (6.4) may be also interpreted as a 'leads-to' property, here $\langle cl(c)\rangle \rightsquigarrow \mathit{Good\_i}(c)$.

A simplified liveness formula is

$$\mathit{Live}'_i \equiv \forall c\colon \mathbf{A}\Diamond\, \mathit{Good}_i(c), \tag{6.5}$$

which we frequently use later in Sects. 6.2.2 and 6.4.2. It is sufficient to use this simplified version, as in our implementation we implicitly force our controllers to non-deterministically claim a lane or some crossing segments every now and then with a time-bound invariant in the initial location. Thus a controller that never wishes to claim a lane or crossing segment does nor occur and property (6.5) implies that before the reservation a claim must have been set anyway. Also, with UPPAAL SMC it is not possible to estimate the probability for a leads-to query of the form $\phi_1 \rightsquigarrow \phi_2$ or a 'globally' query $\mathbf{A}\Box\phi$, like formula (6.4). However, it is possible to estimate the probability that a query of the type $\mathbf{A}\Diamond\phi$, like formula (6.5), holds.

We analyse liveness for the lane change controller from [HLOR11] with our UPPAAL implementation in Sect. 6.2.2. The lane change controller shows both a variant of *zeno behaviour* and *livelocks*, leading to *starvation* of (parts of) the system. We give details on this in Sect. 6.2.2 and also present an adaptation to a controller that is live, with the exception of one special case.

The crossing controller we presented in Sect. 5.3 also shows a high degree of liveness but may also have a livelock for the same special case. We give details on this faulty behaviour and an implementation to correct it in Sect. 6.4.2. We use UPPAAL to show that both the adapted lane change controller and the adapted crossing controller indeed have a high probability of being live, by examining *bounded liveness*, as we explain in the following paragraph.

Bounded Liveness

Liveness in our case only states that *at some point in the future* something good happens. Thus, a lot of time is allowed to pass before the desired behaviour happens, which is unrealistic for real-life implementations. As a first step we adapt the liveness property from formula (6.4) to the *bounded liveness* property

$$\mathit{Bd\_Live}_i \equiv \mathbf{A}\Box\, \forall c\colon (\langle cl(c)\rangle \rightarrow \mathbf{A}\Diamond^{<t}\, \mathit{Good}_i(c)). \tag{6.6}$$

Formula (6.6) states that after a (crossing) claim was set $\mathit{Good}_i(c)$ should always occur in less than t time. With UPPAAL, we can estimate the *probability confidence interval* with which a property holds within some time bound `bound`. For this, the UPPAAL extension SMC offers *evaluation queries* of the type

```
Pr [bound] (<> prop).
```
(6.7)

With such a query, the probability interval $[p-\varepsilon, p+\varepsilon]$ is estimated with which property `prop` holds before `bound` time units (for $p = Pr(<> prop)$). Here, ε is the *uncertainty parameter*. Additionally, with the parameter for *false negatives* α, the *confidence* $1-\alpha$ is estimated with which UPPAAL computes the probability interval. For this, UPPAAL analyses the query within a specific number of system runs that are calculated during runtime. Note that a smaller value for ε results in more slim and more accurate probability intervals, but more system runs have to be examined by UPPAAL, whereas verification time increases. For details on UPPAAL SMC's estimation algorithm, see [DLL+15].

While both the lane change controller $\mathcal{A}_{lc}$ and the crossing controller $\mathcal{A}_{cc}$ are live with a high probability, they are also boundedly live with a high probability.

Fairness

While bounded liveness is an improvement compared to pure liveness, it is still possible that one car has to wait unreasonably longer for a planned manoeuvre than other cars. Thus we introduce *fairness* to our controllers, which in our case means that no car has to wait *unreasonably long* for $Good_i$ to happen.

We omit a CTL interpretation of fairness at this point but explain our notion of this property in the following informally. As we only outline an approach for a fair lane change controller in Sect. 6.2.2, we focus on crossing manoeuvres in the following.

In the previous paragraph about liveness, we indicated that both hitherto presented controllers for highway resp. urban traffic are not completely live. They moreover are not fair. However, we present an implementation of a *fair crossing controller* $\mathcal{A}_{cc}^{f}$ in Sect. 6.4.3. This fair crossing controller negotiates priorities for committing manoeuvres via our guarded broadcast communication. Conveniently, $\mathcal{A}_{cc}^{f}$ also implements total liveness. However, adding communication increases the complexity of the controllers and again leads to questions about security and reliability of communication (cf. Sect. 5.2). Thus, we present an approach on weakening the assumption about having 100% reliable communication in Sect. 6.4.4.

6.2 Properties of the Highway Traffic Lane Change Controller

We examine the system properties of the lane change protocol from [HLOR11] (cf. Sect. 2.4), where this section bases on our contribution [Sch18b]. We use a preferably generic UPPAAL model which we introduce in Sect. 6.2.1. Further on, we explain the needed adaptations of the lane change controller for the implementation in UPPAAL in that section. While safety of the controller was informally proven in [HLOR11], we strengthen their proof result by showing safety and deadlock freedom of the controller in the beginning of Sect. 6.2.2. We then explain and examine with UPPAAL, why both

the lane change controller for highway traffic from [HLOR11] and the overtaking controller from [HLO13] implement unlive behaviour and adapt the original controller from [HLOR11] to a new almost live lane change controller. We show (a high degree of) liveness of the new controller with UPPAAL and also give details on the only special case for which our new controller still shows unlive behaviour. We finally correct this 'only almost' live behaviour by briefly sketching an adaptation to a completely live and fair, now communicating, lane change controller in the end of Sect. 6.2.2.

6.2.1 Implementation of Highway Traffic Manoeuvres in UPPAAL

We introduce the abstract model we examine with UPPAAL and state the considered assumptions and restrictions for it in the following paragraphs. After that, we introduce the UPPAAL implementation of the crossing controller.

An Abstract Model for a Highway Traffic Snapshot in UPPAAL

We now introduce the abstract model that we examine with UPPAAL and state the considered assumptions and restrictions for it. Recall from the previous section that with UPPAAL, our approach is model-checking (for timed automata). Thus, we use the specific model described in the following as our initial traffic snapshot $\mathcal{TS}_0$. The purpose of the used model is to allow for detecting potential incorrectness of the controller, e.g. absence of safety or liveness. Note that we also tried variations of the described model for testing purposes. We give details on the model in the following paragraphs. Note that we also explain the purpose and goals of the chosen model in the last paragraph of this section.

Abstract model. The model we examine with UPPAAL is the traffic situation depicted in Fig. 6.1, where we consider lanes 0 to 3 and the cars A, B and E contained in view $V(E, \mathcal{TS}_0)$ of ego car E. Car D is too far away from car E to be considered in $V(E, \mathcal{TS}_0)$. We set the initial traffic snapshot values for the cars in view $V(E, \mathcal{TS}_0)$ as follows:

- Positions: $pos(A) = 10$, $pos(B) = 12$ and $pos(E) = 30$.
- Sizes of cars as perceived by ego car E: $\Omega_E(A) = 5$, $\Omega_E(B) = 5$ and $\Omega_E(E) = 5$.
- Reservations: $res(A) = \{2\}$, $res(B) = \{0\}$ and $res(E) = \{3\}$.
- Claims: $clm(A) = \{\}$, $clm(B) = \{\}$ and $clm(E) = \{\}$.

Note that we initially require empty sets of claims, as for the model checking procedure with UPPAAL all controllers start in their initial locations without any claim. Thus, we ignore the depicted claims for cars A and B in Fig. 6.1, as these were only included in the figure to illustrate the potential for collisions between cars A and B.

Data structure. For implementing a version of our spatial reasoning about traffic situations in UPPAAL, we first have to encode a representation of our abstract model.

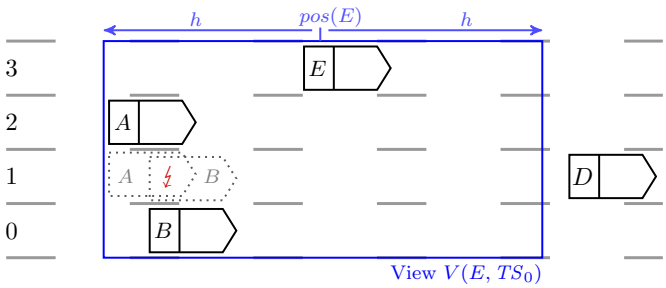

Figure 6.1: Abstract model with adjacent lanes 0 to 3 and cars A, B, E and D, where cars A and B have both a claim on lane 1.

For this, we represent the traffic snapshot $\mathcal{TS}_0$, 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}, 30, 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 is in interval $[30, 35]$ 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 explained before. Note that the 't', e.g. in `pos_t`, is common for standard UPPAAL syntax and does not compulsory denote 'time'.

Speed and distance keeping. The lane change controller is not responsible for distance keeping. However, for cars with different acceleration and speed, a controller for distance keeping would be inevitable to avoid rear-end collisions. We outlined the difficulties for implementing such a distance controller for the multi-lane scenario from [HLOR11] in Sect. 5.1 and thus do not implement one for now. However, if cars would accelerate or brake without any distance control, we would have rear-end collisions. Due to this, 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. That is, for examining the safety and liveness solely

of the lane change manoeuvres with the controller from [HLOR11], we do not need to consider a scenario with cars with different speed and acceleration.

Number of lanes. We decided to limit our abstract model to 4 lanes as this allows for ample lane change manoeuvre possibilities for the cars. We tried up to 20 lanes and the verification run-time seems to increase only linear by about 50 ms each time when we add one lane. Not a surprising discovery, we observed a lower probability for potential collisions with still 3 cars but more lanes. However, we decided to use 4 lanes for all queries for the verification of our system properties in Sect. 6.2.2 as this seems to be a realistic number for nowadays highways.

Number of cars. By restricting our abstract model to the three cars A, B and E, most of the verification queries we present in Sect. 6.2.2 take about 5 to 30 seconds to verify. However, we tried using 4 and 5 cars, where verification run-time appeared to increase exponentially when adding a car, as the most queries took about 15 to 60 minutes for 4 cars and after two days the verification did not finish for 5 cars. This observation is not surprising, as with each new car the UPPAAL system grows by one lane change controller with an additional clock due to which the internal system state space grows significantly during verification. However, we describe in the next paragraph why our model containing 3 cars is still universal enough.

Appropriateness and purpose of the model. Despite the speed limitation and the fact that we restrict the model to 3 cars, we claim to encode an appropriate model to verify both safety and liveness. In particular, we expect the following behaviour for the cars A, B and E and also explain their purposes in the chosen model:

- Cars A and B: These two cars *cannot 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. We thus *expect potential collisions* here, but show that the lane change controller *always* prevents *actual collisions*. Further on, we expect that *liveness* is at least *limited* between these two cars, as possibilities for blockage are foreseeable.
- Car E: We use this car mainly for testing purposes. Of course, 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*, as there is no conflicting car on any neighbouring lane. Thus both, safety and liveness, *should* be guaranteed for this car. However, we later show with car E that liveness is violated as traces exist where even car E never changes lanes due to a livelock.

The Lane Change Controller in UPPAAL

For the UPPAAL implementation, we adapt the lane change controller $\mathcal{A}_{lc}$ from Fig. 2.6, p. 31 to UPPAAL syntax, as neither formulae of Multi-lane Spatial Logic (cf. Sect. 3.5, p. 61) nor controller actions for claiming or reserving lanes (cf. Sect. 4.1, p. 69) are

directly implementable in UPPAAL. The resulting UPPAAL lane change controller `LCP` ('Lane Change Protocol') is depicted in Fig. 6.2. 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. The depicted controller in Fig. 6.2 is a direct implementation of the controller from [HLOR11] and does not yet contain any adaptations, apart from those necessary for the implementation in UPPAAL which we explain in the remainder of this section. Note that internally in our UPPAAL implementation, the cars do not have the names A, B and E, but instead i ranges over $\{1, 2, 3\}$. However, as it is easier to explain together with our model from Fig. 6.1, we stick to `LCP(A)` instead of `LCP(1)` and respectively for the other cars throughout this implementation section.

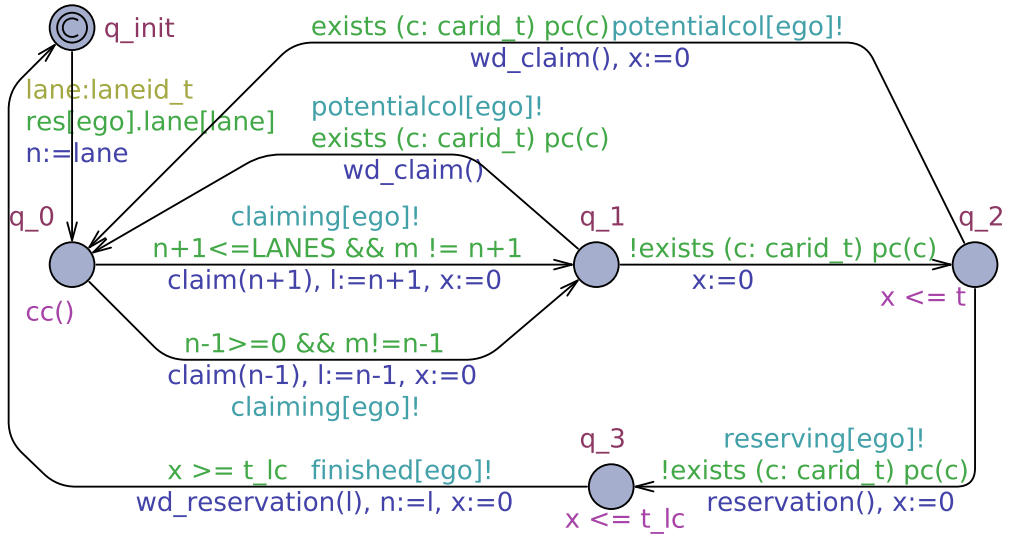

Figure 6.2: UPPAAL implementation `LCP` of the lane change controller $\mathcal{A}_{lc}$ from Fig. 2.6.

Colour coding and fonts. Note that the UPPAAL automata in Figs. 6.2 to 6.12 use the UPPAAL colour coding, where communication via broadcast channels is shown in turquoise, guards are depicted in green, updates in blue, location names and invariants in purple and selection statements are depicted in mustard yellow.

Additionally, we use the following fonts to distinguish ACTA from their respective UPPAAL implementations. While we name ACTA with labels like $\mathcal{A}_{lc}$ as usual (cf. Sect. 4.1), we write UPPAAL automata in typewriter font like `LCP`, to distinguish them. As before, we use the standard math font for ACTA concepts (e.g. location q_2 or a guard $x \leq 1$), while we use a sans serif font for concepts of our UPPAAL automata (e.g. location `q_2` or a guard `x <= 1`).

Timed automata semantics. For implementing $\mathcal{A}_{lc}$ in UPPAAL, we are forced to drop the subformula $\neg\exists c\colon pc(c)$ of the invariant $\mathcal{I}(q_2)$ in location q_2 (cf. Fig. 2.6). This is due to the following feature of the general definition of timed automata semantics, which also applies for the extended UPPAAL timed automata: for an action transition

$\langle l_i, \nu_i \rangle \xrightarrow{\alpha} \langle l_{i+1}, \nu_{i+1} \rangle$ to exist in the semantics of a timed automaton, the invariant $\mathcal{I}(l_i)$ of the location l_i must hold, i.e. $\nu \models \mathcal{I}(l_i)$, on executing the transition (cf. Def. 12, p. 26). However, in the original controller $\mathcal{A}_{lc}$ from [HLOR11], the guard on the transition from q_2 to q_0 contains the formula $\exists c$: $pc(c)$ which contradicts $\mathcal{I}(q_2)$. Thus, it is not possible to ever execute the transition from q_2 to q_0 in $\mathcal{A}_{lc}$, if $\mathcal{I}(q_2)$ remains existent.

Fortunately, the adaptation we propose does not affect crucial construction goals of lane change protocol $\mathcal{A}_{lc}$. The only change is that now the controller LCP from Fig. 6.2 may remain in location q_2 for up to t time units while exists(c : carid_t)pc(c) already holds. However, if this formula still holds after t time, LCP has to change back to location q_0. With that, the original safety result from [HLOR11] is not violated.

Initialisation of the controller. We add an additional initial location q_init before the regular initial location q_0 of the lane change controller from Fig. 2.6, to instantiate internal date variables of the controller. The only data variable considered in the lane change controller is n, encoding the lane on which the car is currently driving. For this, the value lane corresponds to the entry res[ego].lane[lane] in the list of reserved lanes we introduced in the previous section. The entry res[ego].lane[lane] contains the only flag set to 1 in the Boolean list res[ego].lane.

Communication. The original lane change controller $\mathcal{A}_{lc}$ from Fig. 2.6 does not contain any communication. We however add internal sending operations, e.g. claiming[ego]!, for communication of LCP with related *observer automata*, which are used for verification purposes and thus introduced later in Sect. 6.2.2.

Time bounds. For the implementation we have to set specific values for the time bounds in LCP. With t := 1 and t_lc := 2, we choose small time bounds to avoid slowing down the verification unnecessarily. The value 1 for t can be interpreted as the computation time the controller needs to process information in location q_2. Note that 1 is the smallest possible constant > 0 UPPAAL allows for our time bound t. We give detailed information on the influence of these chosen values on our verification results in Sect. 6.2.2 on p. 124.

MLSL formulae. We now explain our UPPAAL representation of MLSL formulae. The only MLSL formulae used by the lane change controller are the collision check cc (cf. formula (2.15), p. 30) in the initial location q_0 and the potential collision check $pc(c)$ (cf. formula (2.16), p. 30) used in several guards and invariants of the controller. Our solution for implementing formulae (2.15) and (2.16) 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]
        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. 6.2.1) 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 clm[A] = $\{\{0,1,0,0\},10,5\}$ and clm[B] = $\{\{0,1,0,0\},12,5\}$, the function call intersect(clm[A], clm[B]) returns *true*.

With the intersect function, we encode the negation of the collision check formula $\neg\, col(\text{ego})$ introduced with MLSL formula (2.15) by the function

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

In the UPPAAL implementation cc() of $\neg\, col(\text{ego})$ it is implied that the check is done for the ego car. We further on encode the potential collision check $pc(c)$ introduced in MLSL formula (2.16) with

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

Note that apart from negating the collision check directly in the formula cc(), we use the functions cc() and pc(c) in the UPPAAL controller LCP in Fig. 6.2 exactly in the same manner as we use the respective MLSL formulae in the original lane change controller $\mathcal{A}_{lc}$ from Fig. 2.6. Also note that this implementation of MLSL formulae is general and is usable for arbitrary traffic snapshot one could implement, not only for our specific traffic snapshot $\mathcal{TS}_0$ from Fig. 6.1.

Controller actions. Besides MLSL formulae, 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, the method

```
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*. Similarly as for the implementation of MLSL formulae, our implementation of controller actions is independently of the specific controller `LCP` from Fig. 6.2 and of the specific model from Fig. 6.1.

6.2.2 Verification of the Properties

We now examine our desired system properties for the UPPAAL implementation `LCP` of the lane change controller from Fig. 6.2 with suitable verification queries. As our model for the verification with UPPAAL, we use the traffic snapshot $\mathcal{TS}_0$ from Fig. 6.1. This means that the UPPAAL system used for verification comprises three lane change controllers `LCP` in parallel, one for each of the cars A, B and E. Note that we thoroughly examine safety and liveness, but only sketch an approach on integrating fairness into the lane change controller.

As a necessary precondition for showing any of our system properties, we examine *deadlock freedom* of our controllers. A *deadlock* would mean that some of our controllers wait for a resource that is permanently blocked by another controller. With only one blocked controller, the system gets stuck and no transition is possible anymore. In our case this resource could e.g. be a space on a lane which is permanently claimed by two different cars. We show deadlock freedom of our controllers, by successfully checking the query

```
A[] not deadlock
```
(6.8)

on a normal work station in averagely 23 seconds with a memory usage peak of roughly 65KB. With this, we globally (‘`[]`’) exclude deadlocks in all (‘`A`’) runs of our system. This result is not surprising, as our controllers can always withdraw an unsuccessful claim, return to the initial location `q_0` and try a new claim later if they are in locations `q_1` or `q_2`. Further on, if a controller already has a reservation and thus is in location `q_3`, the invariant `x <= t_lc` forces the controller to change back to the initial location `q_0` after `t_lc` time. Thus, from each possible location the controllers `LCP(i)` are always able to change back to the initial location `q_0`.

Safety

We confirm the hitherto informally proven *safety* property from [HLOR11] with our UPPAAL implementation from the previous section. Recall from Sect. 6.1 that the queries for the verifier in UPPAAL are formulated in a timed computation tree logic (TCTL) specification language.

To show the validity of the safety formula (6.1), our goal is to show *unreachability* of a *bad system state* with a collision in the overall UPPAAL system. A bad state in this case is a state were the collision check formula cc() from p. 119 holds *for any two arbitrary cars*. Thus, we broaden the method cc() to

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

with which *collisions between any two cars* are considered. Further on, we introduce the observer automaton Observer1, depicted in Fig. 6.3. This automaton is running in parallel with the automata LCP(i) during UPPAAL system runs, observing whether cc_all() holds invariantly. If a collision is detected, Observer1 enters its location unsafe. We use the query

```
A[] not Observer1.unsafe
```
(6.9)

to show in averagely less than 3 seconds with a memory usage peak of 65KB that there exists no system run where the formula cc_all() does not hold. With this query, we thus verify the safety property (6.1) (p. 111) for the lane change controller from [HLOR11].

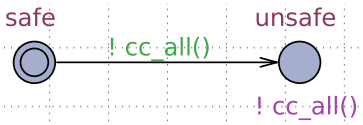

Figure 6.3: Observer1 checking for a collision in the UPPAAL system.

Liveness

With query (6.8), we exclude deadlocks in our system, so the system cannot come into a state where no transition is possible anymore. Nonetheless, the original lane change controller $\mathcal{A}_{lc}$ from Fig. 2.6, and thus also our implementation depicted in Fig. 6.2, is not truly *live*. This is due to the fact that *zeno behaviour* and *livelocks* exist. Zeno behaviour means that in a computation path of the system, time cannot pass beyond some time t_z but infinitely many actions happen. By livelock we mean that while a controller *could* progress it does not but instead remains in one location forever.

To analyse liveness with our UPPAAL implementation, we introduce a second observer automaton Observer(i), as depicted in Fig. 6.4. 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 a message over the channel claiming[A] with which Observer(A) synchronises, such that

both controllers simultaneously change to a new location. Upon reserving a lane, `LCP(A)` sends over reserving[A] and the observer changes to a location success. We approach our liveness property from formula (6.4) by first checking the query

```
A<> (Observer(A).success or Observer(B).success or Observer(E).success),
```
(6.10)

which states that *finally in every trace*, *at least one* of the controllers `LCP(i)` is successful in changing a lane. We later refine this property to a property where we require *each* controller to be finally successful in all system runs. Remember, that we generally expect query (6.10) to be successfully verified, as in our model at least car E should be able to finally change a lane in every possible trace.

However, query (6.10) *does not hold* for the implementation `LCP` from Fig. 6.2, meaning there exist system traces where no car that wishes to do so, finally changes lanes. We discuss and correct the reasons for this faulty behaviour in the following in two steps.

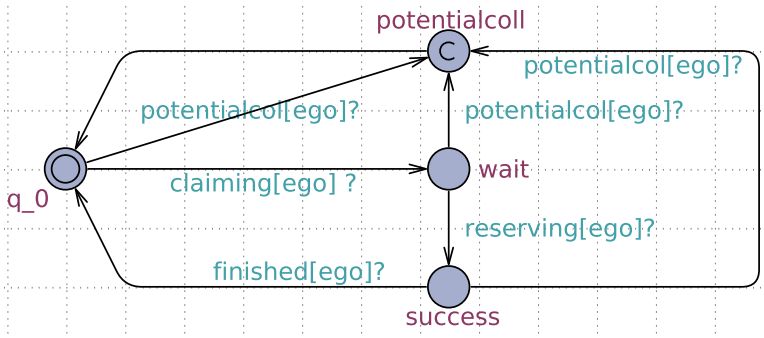

Figure 6.4: `Observer(i)` checking for every instance `LCP(i)`, if whenever car `i` claims a lane, it finally changes lanes, or if a potential collision occurs.

Problem 1 (zeno behaviour and livelock). The first reason why query (6.10) does not hold is that there exists a trace, where only cars A and B both infinitely often set and withdraw their respective claim on lane 1 without any elapse of time and car C does not execute any transition. Thus both controllers `LCP(A)` and `LCP(B)` circle between their respective locations `q_0`, `q_1` and `q_2` infinitely long. This system behaviour is a strong version of the previously sketched zeno behaviour, as time does not pass beyond time 0 in this trace. Additionally, `LCP(E)` is denied a possibility of executing a transition in the sketched trace and thus starves.

The second reason why query (6.10) does not hold is that a trace exists, where each of the automata `LCP(i)` stays in its respective location `q_0` or `q_1` *infinitely long*. Thus, either the cars did not do anything at all or each car made a claim for a lane but never withdraws it or turns it into a reservation. We call this observation livelock, as transitions are available, but never taken.

Solution 1. Both of the sketched undesirable system behaviours are easily solvable by introducing a new time bound t_idle and a respective invariant x <= t_idle to location q_0. Additionally, we introduce the invariant x <= t to location q_1 and place the guard x >= t on the outgoing edges of q_1, as done in the adapted controller LCP′ depicted in Fig. 6.5. For the analysis we set t_idle = 10 and we recall that t = 1 was set in Sect. 6.2.1. The idea is that a controller can wait a while (t_idle) before starting a lane change manoeuvre. However, after a claim was set, t again reflects the computation time LCP′ needs to process data in location q_1.

With these adaptations, we successfully show query (6.10) in less than 0.5 seconds with a memory usage peak of 40KB.

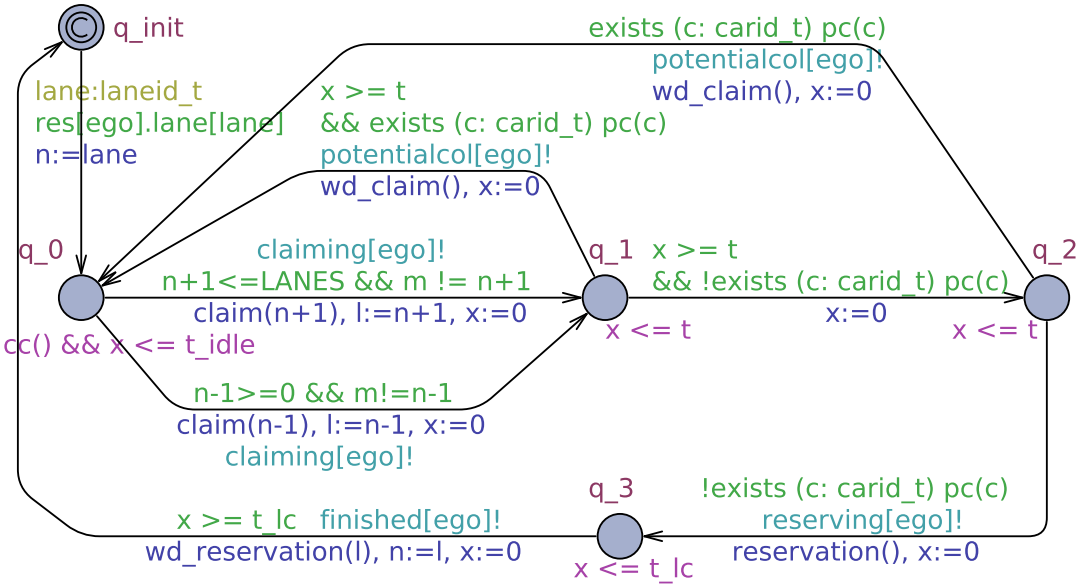

Figure 6.5: Adapted UPPAAL implementation LCP′ without livelocks and zeno behaviour due to new invariant in location q_1 and new guards x >= t on the outgoing edges of q_1.

Problem 2 (unlive behaviour for special case). The verification query (6.10) is already a type of a weak liveness property, as it shows that with the adapted controller LCP′ from Fig. 6.5 in every simulation trace, at least one of the controllers finally changes lanes. However, actually desired is the refined formula

$$\texttt{A<> Observer(i).success}, \tag{6.11}$$

which states for an arbitrary car identifier $i \in \{A, B, E\}$ that the car related to LCP′(i) finally changes lanes and thus implements our liveness property from formula (6.5). With the adapted controller LCP′, property (6.11) only holds for LCP′(E), as anticipated. The reason is, that there still exists the special case, where cars A and B both unsuccessfully

time bound	A	B	E
$k = 5$	$[0.3031, 0.3231]$	$[0.3001, 0.3201]$	$[0.3447, 0.3647]$
$k = 10$	$[0.6691, 0.6891]$	$[0.6568, 0.6880]$	$[0.8336, 0.8536]$
$k = 15$	$[0.8235, 0.8435]$	$[0.7806, 0.8006]$	≥ 0.98
$k = 20$	$[0.8951, 0.9151]$	$[0.8619, 0.8819]$	≥ 0.98
$k = 30$	$[0.9511, 0.9711]$	$[0.9577, 0.9777]$	≥ 0.98
$k = 50$	≥ 0.98	$[0.9789, 0.9989]$	≥ 0.98

Table 6.1: Statistical evaluation of the query `Pr [<= k] (<> Observer(i).success)` for $\texttt{LCP}'$ with different upper time bounds k.

try to change to lane 1 infinitely often and thus create a potential collision infinitely often, preventing both controllers from ever transforming their claim into a reservation.

Note that while the query (6.11) does not hold, moreover the implementation

```
Observer(i).wait -> Observer(i).success
```
(6.12)

of the more complex liveness formula (6.5) does not hold.

Statistical analysis of Problem 2. While we cannot guarantee 100% liveness of the lane change controller from Fig. 6.5, we can estimate its probability for *time-bounded liveness*. For this, we use the SMC part of our UPPAAL version (cf. Sect. 6.1). In Sect. 6.1, we introduced *evaluation queries*, with which UPPAAL can estimate the *probability confidence interval* with which a property holds.

In our case, we adapt the liveness query (6.11) to the time-bounded evaluation query

```
Pr [<= k] (<> Observer(i).success),
```
(6.13)

which returns the probability with which car i is successful in changing lanes in at most k time ($i \in \{A, B, E\}$). For the probability parameter for false negatives, we use $\alpha = 0.01$ and for the probability uncertainty parameter we use $\varepsilon = 0.01$. This results in slim intervals containing the correct value with a confidence of 99%. We give the respective probabilities for each car for different values for k in Table 6.1. Note that ≥ 0.98 is the highest value for success UPPAAL returned with a confidence of 99%. All queries were evaluated in averagely 5 seconds for $k = 10$ and averagely less than 3 seconds for all other values for k. Note that for $k \geq 60$ all controllers changed lanes with a probability value ≥ 0.98.

Influence of the chosen parameters on the results in Table 6.1. The results in Table 6.1 highly depend on the values for the time constants t, $\mathsf{t\_idle}$ and $\mathsf{t\_lc}$ we set for the controller LCP (cf. Sect. 6.2.1). For this reason, the probabilities for a successful reservation are relatively low for all cars for $k = 5$, as each controller already needs *at least* $\mathsf{t} + \mathsf{t} = 2$ time units to set a reservation and further on may idle up to $\mathsf{t\_idle} = 10$ time units in location $\mathsf{q\_0}$. However, for the time bound $k = 15$, a probability for success greater than

78% can be observed for all cars and after 30 time units each car successfully changes lanes with a probability greater than 95%.

For examining the influence of t_idle on our results more thoroughly, we also tried t_idle = 1 for test purposes. With this, for $k = 5$ each car had already changed lanes with a probability of $> 81\%$, for $k = 10$ with a probability of $> 93\%$ and for $k = 20$ all cars did change lanes with a probability bigger than 98%.

Further on, we varied the value for the time t_lc a car needs to finish a lane change manoeuvre. For t_lc = 3, t_lc = 4 and t_lc = 5, the verification results did not change for cars A and E. However, for B a decrease of averagely 2% for each increase of t_lc was notable for $k \in \{15, 20, 30\}$. We explain this observation by the fact that by taking longer for a lane change manoeuvre, car A blocks car B longer on lane 1.

Additionally, the values in Table 6.1 depend on the arrangement of the cars in the model. Thus, the probabilities for car E are the highest, as E has no other cars as competitors for lanes. However, as E may idle in q_0 for up to t_idle = 10 time units, added with the previously observed 2 time units for claiming a lane, for $k = 5$ and $k = 10$, car E obviously does not have the maximal high probability for success ($\geq 98\%$). However, starting from $k = 15$, car E has the highest probability UPPAAL SMC returns with confidence 99%, as by then it had enough time to change lanes. The values for car A are slightly bigger than those of car B, as initially, car A has two possible lanes for its manoeuvre while car B only has one. Finally, adding further lanes on top of the model from Fig. 6.1 did not change the verification times notably.

An approach on increasing liveness and correcting claims made in [Sch18b]. Although the results given in Table 6.1 are not bad, we tried different approaches to solve Problem 2, in order to fully guarantee liveness or at least reach higher probabilities for success. For this, initially our goal was to stay as close as possible to the construction idea of the original controller $\mathcal{A}_{lc}$ from [HLOR11]. Thus, for now, we do not want to add some kind of (de-) centralised scheduling algorithm, e.g. car-2-car or car-2-X communication. With this we also want to avoid adding too much complexity to the controller.

In [Sch18b], we introduce an additional location q_wait, in which the controller is forced to wait for a bounded non-deterministic time after a potential collision was detected and the controller withdrew its claim. We use a time bound t_w and set t_w = 4. With this, we delimit the waiting time in q_wait by the invariant x <= t_w and the guard x >= 1 on its outgoing edge. Thus, on entering q_wait, a controller has to wait there for some time from the interval $[1, 4]$. After that, the initial location q_0 is entered again and the controller may again claim a lane.

The idea is that with this adaptation, cars A and B should block each other from changing a lane less often. Note that due to a technical error in our implementation in UPPAAL, in our contribution [Sch18b] we incorrectly claimed that with this adaptation the liveness property from query (6.11) holds for each of the cars A, B and E.

However, by only adding q_wait, there still exists the very unlikely case where each time both cars pick the same non-deterministic waiting time in q_wait and continue to block each other infinitely often. Thus, the properties

```
A<> Observer(A).success        and        A<> Observer(B).success        (6.14)
```

still do not generally hold. Also, the probability values in Table 6.1 slightly decrease for this adaptation for lower values for k. The reason for this decrease of liveness is simple: each time a controller leaves the location q_wait, it has to return to the initial location q_0. There it may idle for up to t_idle = 10 time units again. Thus, whenever a controller withdraws a claim and enters q_wait and q_0 subsequently, in the worst case it may wait t_w + t_idle = 14 time units before claiming a lane again. Obviously, with this, the evaluation query (6.13) returns lower probabilities.

Fortunately, we easily solve this problem by introducing the adapted controller LCP″ which is depicted in Fig. 6.6. This controller contains the previously introduced location q_wait, but instead of returning to the initial location q_0 after some waiting time, the controller enters a new committed intermediate location q_im. With this, the controller will try to claim a lane again directly after leaving q_wait without idling for some time in location q_0.

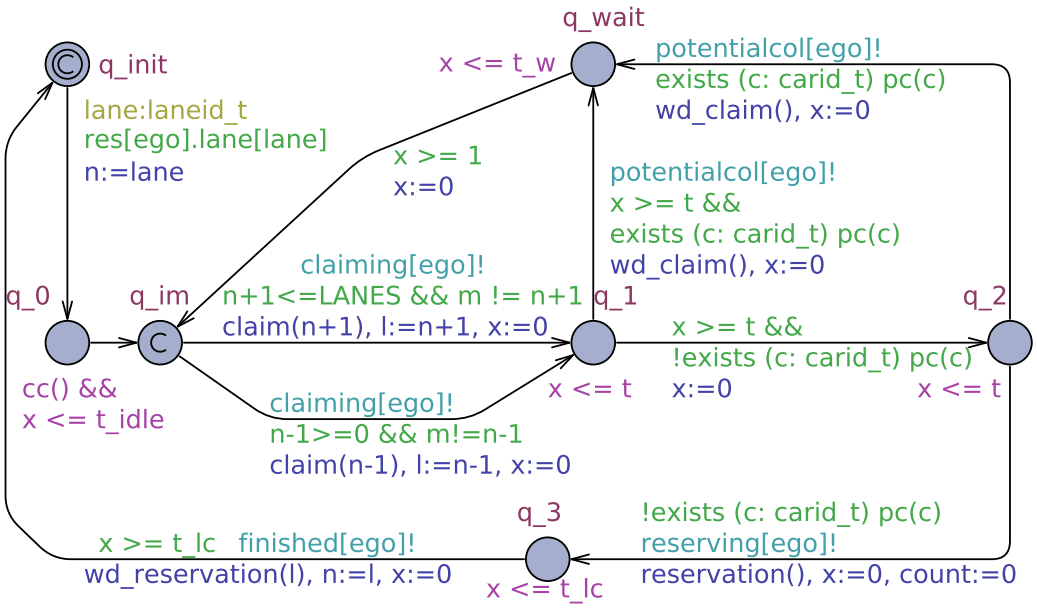

Figure 6.6: Adapted and more alive UPPAAL implementation LCP″, where controllers wait for some time in $[0, \mathsf{t\_w}]$ in location q_wait after withdrawing a claim.

On analysing query (6.13) for the new controller LCP″, we achieve values increased by 2 to 6% for liveness for time bounds $k \geq 15$ as we show in Table 6.2. Note that the values

time bound	A	B	E
$k = 5$	$[0.3043, 0.3243]$	$[0.3102, 0.3302]$	$[0.3423, 0.3623]$
$k = 10$	$[0.6860, 0.7060]$	$[0.6576, 0.6776]$	$[0.8410, 0.8410]$
$k = 15$	$[0.8898, 0.9098]$	$[0.8013, 0.8213]$	≥ 0.98
$k = 20$	$[0.9466, 0.9666]$	$[0.8923, 0.9123]$	≥ 0.98
$k = 30$	≥ 0.98	$[0.9763, 0.9963]$	≥ 0.98
$k = 50$	≥ 0.98	≥ 0.98	≥ 0.98

Table 6.2: Statistical evaluation of the query `Pr [<= k] (<> Observer(i).success)` for LCP″ with different upper time bounds k.

for $k = 5$ and $k = 10$ remain small, as the controllers LCP″(i) may still idle in location q_0 for up to 10 time units once before they start claiming a lane.

On analysing liveness of the crossing controller in Sect. 6.4.2, we also use the waiting location q_wait, which significantly increases liveness there. We explain details for this in the respective section.

Summary and Outlook on Fairness of the Highway Traffic Lane Change Controller.

With the traffic situation from Fig. 6.1, p. 115, and the corresponding implementation, we present one specific scenario for model checking with UPPAAL, designed for the following purposes:

- showing the absence of collisions between any cars in our model (i.e. proving safety (6.1)),
- identifying and analysing the existing zeno behaviour and livelocks (i.e. missing guards and invariants around location q_1),
- eliminating the zeno behaviour and livelocks and showing a high probability for liveness for the adapted lane change controller, and
- introducing location q_wait to further increase liveness.

For this purpose, the restrictions for the scenario, e.g. on 3 cars and 4 lanes were sufficient (cf. explanation on p. 116), although more cars (4) and lanes, as well as different values for the various time constants t, t_i, ... were also examined for test purposes (cf. Sect. 6.2.1). We conclude that our result of achieving only 'almost' live controllers coheres with the fact that distributed scheduling without any means of cooperation and communication between the autonomous cars has its limits.

Thus, for future considerations on achieving real liveness of the lane change controller, it seems a natural choice that two cars which block each other should negotiate who drives first between themselves. Thus, although adding complexity to the controllers, we propose that cars communicate priorities between themselves to solve Problem 2

with a decentralised cooperating scheduling approach. With this, there would be a high chance of finally guaranteeing liveness and additionally achieving fairness of the lane change controller protocol $\mathcal{A}_{lc}$.

For now, we do not implement the sketched fair and cooperative behaviour for the highway traffic case, but instead leave it for future work (cf. Sect. 8.4). However, we refer forward to the UPPAAL implementation for urban traffic in Sect. 6.4, where we implement the here proposed decentralised cooperative scheduling approach to successfully achieve complete liveness and fairness of the crossing controller for urban traffic.

6.3 Safety of Crossing Manoeuvres

Before we analyse the system properties of the crossing controller $\mathcal{A}_{cc}$ from Fig. 5.5 with a dedicated implementation for urban traffic in UPPAAL in Sect. 6.4, we give a mathematical proof of safety for $\mathcal{A}_{cc}$ in this section. However, we also examine safety with our implementation in Sect. 6.4 to further strengthen the following mathematical proof of safety of crossing manoeuvres.

In this section, we first show safety of the crossing controller $\mathcal{A}_{cc}$ with perfect knowledge from Sect. 5.3.1 and after that extend this proof for proving safety of the crossing controller $\mathcal{A}'_{cc}$ with imperfect knowledge from Sect. 5.4. Note that the safety proofs in this section are mathematical proofs by hand over the semantics of the respective crossing controllers (cf. Sect. 4.2).

Safety of Crossing Manoeuvres with Perfect Knowledge

The desired safety property from formula (6.1) states that everywhere the reservations of two arbitrary different cars are not overlapping and thus at any moment the spaces occupied by different cars are disjoint. A traffic snapshot $\mathcal{TS}$ is called *safe* if $\mathcal{TS} \models \mathit{Safe}$ holds. For the following proof, we assume perfect knowledge and thus that every car perceives the complete safety envelope of all other cars in its view. Furthermore, we rely on the following assumptions for the overall safety property to hold.

Assumption A1. The initial traffic snapshot $\mathcal{TS}_0$ is *safe*.

Assumption A2. Every car is equipped with a a crossing controller $\mathcal{A}_{cc}$ (cf. Sect. 5.3.1) and a road controller $\mathcal{A}_{rc}$ (cf. Sect. 5.3.2), together with a distance controller $\mathcal{A}_{dc}$ and a velocity controller $\mathcal{A}_{vc}$ (cf. Sect. 5.1).

Theorem 1 (Urban traffic safety)**.** *If assumptions A1 and A2 hold, every traffic snapshot $\mathcal{TS}$ that is reachable from $\mathcal{TS}_0$ by time transitions and transitions allowed by the road controller$\mathcal{A}_{rc}$ and the crossing controller $\mathcal{A}_{cc}$ is* safe.

Proof. We prove safety from the perspective of an arbitrary car E, because all cars behave similarly. Therefore, we show that all traffic snapshots $\mathcal{TS}$ reachable from $\mathcal{TS}_0$ are safe, for all virtual views $V_i(E, \mathcal{TS}) \in V_M(E, \mathcal{TS})$ of E and all valuations ν with $\nu(\text{ego}) = E$:

$$\mathcal{TS}, V_i(E, \mathcal{TS}), \nu \models \mathit{Safe}_{\text{ego}} \text{ , with } \mathit{Safe}_{\text{ego}} \equiv \neg\exists c \neq \text{ego} \wedge \langle re(\text{ego}) \wedge re(c)\rangle. \tag{6.15}$$

Our approach is a proof by induction over the number of transitions needed to reach a specific traffic snapshot from $\mathcal{TS}_0$. The induction basis holds, as $\mathcal{TS}_0$ is already safe by Assumption A1. Assume that a specific traffic snapshot $\mathcal{TS}_k$ is reachable from $\mathcal{TS}_0$ in $k+1$ steps and that $\mathcal{TS}_k$ is safe. For the induction step we show for $k \mapsto k+1$ that the traffic snapshot $\mathcal{TS}_{k+1}$, reachable from $\mathcal{TS}_k$ by one further transition, is safe as well.

Observe that overlaps of safety envelopes, and thus collisions, can only occur if either the road controller $\mathcal{A}_{rc}$ creates a *reservation on a lane segment*, the crossing controller $\mathcal{A}_{cc}$ creates a *crossing reservation for a crossing segment* or *time passes* and the positions of cars on lanes and crossing segments change. A claim on a lane or crossing segment can at most cause a potential collision, which does not threaten safety property (6.15). Likewise, withdrawing of claims or reservations does not cause (potential) collisions and thus need not be considered in the following proof.

Time passes. For time transitions $\mathcal{TS}_k \xrightarrow{t} \mathcal{TS}_{k+1}$ the distance controller $\mathcal{A}_{dc}$ ensures that the distance to a car or an intersection ahead remains positive. If car E reaches a crossing and $\mathcal{A}_{cc}$ is not permitted to commit a crossing reservation in time and thus leave the locations q_1 to q_3 in $\mathcal{A}_{cc}$, the invariant $ca(\text{ego})$ in these locations will force the distance controller $\mathcal{A}_{dc}$ of car E to decelerate and stop if the required crossing segments are not yet available. In this case there is no automatic crossing reservation for E and the safety property is not violated.

Reservations on lane segments. For lane reservations $\mathcal{TS}_k \xrightarrow{\mathtt{r}(E)} \mathcal{TS}_{k+1}$, the road controller $\mathcal{A}_{rc}$ corresponds to the lane change controller proven safe in [HLO13] except for one additional feature; as explained in Sect. 5.3.2, $\mathcal{A}_{rc}$ will withdraw a committed claim on a lane segment as soon as a crossing is ahead and will not create a new claim or reservation on a lane segment, but E may finish an overtaking manoeuvre already begun. In [HLO13] the authors specify lane change manoeuvres to take at most t_{lc} time to finish, wherefore we simply make sure the distance d_c used in the crossing ahead check is large enough to let E finish the overtaking manoeuvre. For the safety of such overtaking manoeuvres on road segments between intersections, we refer to the safety proof in [HLO13].

Crossing reservations. We now examine a crossing reservation transition from traffic snapshot $\mathcal{TS}_k$, which is safe by assumption, to next traffic snapshot $\mathcal{TS}_{k+1}$:

$$\mathcal{TS}_k \xrightarrow{\mathtt{rc}(E)} \mathcal{TS}_{k+1}. \tag{6.16}$$

For the crossing controller $\mathcal{A}_{cc}$ introduced in Sect. 5.3.1, depicted in Fig. 5.5 on p. 95, the only possibility of a crossing reservation violating safety is the transition from location q_3 to q_4, as we only have a crossing reservation $rc(\text{ego})$ on that edge in $\mathcal{A}_{cc}$:

$$q_3 \xrightarrow{\neg\exists c{:}pc(c)\wedge\neg lc(\text{ego}) \;/\; \texttt{rc}(\text{ego});x:=0} q_4. \tag{6.17}$$

With the transition (6.16) between traffic snapshots $\mathcal{TS}_k$ and $\mathcal{TS}_{k+1}$ and the transition (6.17) from q_3 to q_4 in crossing controller $\mathcal{A}_{cc}$, together with the valuation ν of clock and data variables and a future updated valuation ν', we now examine the following change in the configuration of $\mathcal{A}_{cc}$:

$$\langle \mathcal{TS}_k, \nu, q_3\rangle \xrightarrow{\texttt{rc}(\text{ego})} \langle \mathcal{TS}_{k+1}, \nu', q_4\rangle. \tag{6.18}$$

Of course, we consider a network of ACTA and thus a configuration from this network would actually be considered. This means, previous traffic snapshot changes before $\mathcal{TS}_k$ was reached might have been due to other cars and their controller actions. However, as we examine an internal transition with the considered crossing reservation transition, only the configuration of the specific considered ACTA changes (cf. networks of ACTA, Sect. 4.4). Thus, for reasons of simplicity, we only consider the configuration of the considered car, as this is the only part that changes in the considered network.

By the ACTA semantics defined in Sect. 4.2, the invariant $\mathcal{I}(q_4)$ of future location q_4 has to be satisfied to enable transition (6.18). Formally,

$$\mathcal{TS}_{k+1}, V_i(E, \mathcal{TS}_{k+1}), \nu' \models x \leq t_{cr} \wedge oc(\text{ego})$$

has to be valid. Obviously, $x \leq t_{cr}$ is fulfilled on entering q_4, as the clock x is reset to 0 on the transition from q_3 to q_4. From controller action $rc(\text{ego})$ on the transition to q_4, we can directly deduce the validity of invariant $oc(\text{ego}) \equiv \langle cs \wedge re(\text{ego})\rangle$, because $rc(\text{ego})$ implies that in future traffic snapshot $\mathcal{TS}_{k+1}$ formula $oc(\text{ego})$ holds (c.f. definition of traffic snapshot transitions from Sect. 3.3).

Additionally, the guard on the transition from q_3 to q_4 in $\mathcal{A}_{cc}$ has to be satisfied to allow for transition (6.18). Therefore, we examine

$$\mathcal{TS}_k, V_i(E, \mathcal{TS}_k), \nu \models \neg\exists c : pc(c) \wedge \neg lc(\text{ego}).$$

The part $\neg lc(\text{ego})$ only states that there does not exist a second reservation on another lane segment for car E. This does not endanger safety property (6.15) on crossing segments, because $\mathcal{A}_{rc}$ handles lane change manoeuvres on road segments. For this, we refer to the previous paragraph where we treated reservations on lane segments and again to the safety proof of $\mathcal{A}_{rc}$ in [HLO13]. The validity of the second part $\neg\exists c : pc(c)$ follows directly from the validity of the invariant $\mathcal{I}(q_3)$ which we examine in the following.

Recall the crossing ahead check $ca(\text{ego}) \equiv \langle re(\text{ego}) \frown free^{<d_c} \wedge \neg\langle cs\rangle \frown cs\rangle$ from p. 94. By definition of the semantics of ACTA, traffic snapshot $\mathcal{TS}_k$, view $V_i(E, \mathcal{TS}_k)$ and variable valuation ν have to satisfy the invariant of location q_3 in $\mathcal{A}_{cc}$

$$\mathcal{I}(q_3) \equiv ca(\text{ego}) \wedge \neg\exists c : pc(c) \wedge x \leq t_c.$$

If more than t_c time units pass and $lc(\text{ego})$ still holds or a potential collision is detected, $\mathcal{I}(q_3)$ and the outgoing transitions from location q_3 force $\mathcal{A}_{cc}$ to change back to q_1 and withdraw its crossing claim. Observe that the invariant $ca(\text{ego})$ holds until E is granted a crossing reservation and $\mathcal{A}_{cc}$ therefore changed to location q_4 where $oc(\text{ego})$ holds.

Parts of the following formula transformations for the third subformula $\neg\exists c : pc(c)$ from $\mathcal{I}(q_3)$ are similar to those from the safety proof in [HLOR11] and therefore adapted from there to the urban traffic scenario. In the following, we conclude from the validity of the invariant $\mathcal{I}(q_3)$ and of the second part of the guard $\neg\exists c : pc(c)$ to the validity of the safety formula (6.1) in future traffic snapshot $\mathcal{TS}_{k+1}$.

The subformula $\neg\exists c : pc(c)$ satisfies the following implication:

$$\begin{aligned}\neg\exists c : pc(c) \equiv\ & \neg\exists c :\ c \neq \text{ego} \wedge \langle cl(\text{ego}) \wedge (re(c) \vee cl(c))\rangle \\ \rightarrow\ & \neg\exists c :\ c \neq \text{ego} \wedge \langle cl(\text{ego}) \wedge re(c)\rangle.\end{aligned}$$

Together with the induction hypothesis, we derive

$$\begin{aligned}\mathcal{TS}_k, V_i(E, \mathcal{TS}_k), \nu \models\ & \neg\exists c : c \neq \text{ego} \wedge \langle cl(\text{ego}) \wedge re(c)\rangle \\ & \wedge\ \neg\exists c : c \neq \text{ego} \wedge \langle re(\text{ego}) \wedge re(c)\rangle. \qquad (6.19)\end{aligned}$$

For the further transformation of formula (6.19), amongst other steps, we use the property that our abbreviation somewhere $\langle\ \rangle$ distributes over disjunction (cf. formula (2), page 22):

$$\begin{aligned}& \neg\exists c : c \neq \text{ego} \wedge \langle cl(\text{ego}) \wedge re(c)\rangle \wedge\ \neg\exists c : c \neq \text{ego} \wedge \langle re(\text{ego}) \wedge re(c)\rangle \\ \longleftrightarrow\ & \neg\exists c : c \neq \text{ego} \wedge (\langle cl(\text{ego}) \wedge re(c)\rangle \vee \langle re(\text{ego}) \wedge re(c)\rangle) \\ \overset{(2)}{\longleftrightarrow}\ & \neg\exists c : c \neq \text{ego} \wedge \langle (cl(\text{ego}) \vee re(\text{ego})) \wedge re(c)\rangle. \qquad (6.20)\end{aligned}$$

To show that for crossing reservations we can indeed conclude safety of traffic snapshot $\mathcal{TS}_{k+1}$ from formula (6.20), we exploit the fact that a created crossing reservation occupies the same space as its previous crossing claim. For that, we use Reservation Lemma 1 for crossing reservations, which we present and prove separately after this proof. Applying Lemma 1 to formula (6.20) finally leads to

$$\mathcal{TS}_{k+1}, V_i(E, \mathcal{TS}_{k+1}), \nu \models \neg\exists c : c \neq ego \wedge \langle re(ego) \wedge re(c)\rangle,$$

for all virtual views V_i, which proves the validity of our safety property (6.15). □

In the following, we extend the *reservation lemma* from [HLOR11] for reservations on lanes to a reservation lemma for crossing segments for our urban traffic scenario. Recall that in Sect. 3.5, we did not introduce a new UMLSL atom for crossing reservations, but instead extended the semantics of the existing MLSL atom $re(c)$. Therefore the lemma itself only changes slightly syntactically compared to [HLOR11], while we have to add some semantic arguments to its proof.

Lemma 1 (Reservations on crossing segments)**.** *We consider a crossing reservation transition* $\mathcal{TS} \xrightarrow{rc(C)} \mathcal{TS}'$ *and an UMLSL formula* $\phi' \in \Phi_{\mathbb{U}}$ *which does not contain a (crossing) claim* $cl(c)$ *as a subformula. We further assume another UMLSL formula* $\phi \in \Phi_{\mathbb{U}}$ *results from* ϕ' *by a replacement of all occurrences of* $re(c)$ *in* ϕ' *by* $re(c) \vee cl(c)$*. In that case for all virtual views* $V_i(E, \mathcal{TS}) \in V_M(E, \mathcal{TS})$ *with* $C \in \mathbb{I}$ *and valuations* ν *with* $\nu(c) = C$ *the following statement holds:*

$$\mathcal{TS}, V_i(E, \mathcal{TS}), \nu \models \phi \;\; \textit{iff} \;\; \mathcal{TS}', V_i(E, \mathcal{TS}'), \nu \models \phi'.$$

We now adapt the proof of the reservation lemma for reservations on lanes for standard MLSL from [HLOR11]. The proof is by structural induction on formula ϕ'. Therefore we have to consider only those UMLSL formulae ϕ' closer that are not syntactically contained in standard MLSL or which have changed semantics. For the other cases we refer to the proof from [HLOR11].

Proof. Consider traffic snapshots $\mathcal{TS} = (\mathcal{N}, res, clm, cres, cclm, pth, curr, pos, spd, acc)$ and $\mathcal{TS}' = (\mathcal{N}, res, clm, cres', cclm', pth, curr, pos, spd, acc)$ with updated sets for claimed crossing segments $cclm' = cclm \oplus \{C \mapsto \emptyset\}$ and for reserved crossing segments $cres' = cres \oplus \{C \mapsto cclm(C)\}$ (c.f. definition of transitions between traffic snapshots from Sect. 3.3).

Recall that a virtual view V_i for car E is defined by $V_i(E, \mathcal{TS}) = (L_i, X, E)$, where $L_i \in \mathsf{seq}\,(\overrightarrow{\Pi}_E \cup \overleftarrow{\Pi}_E)$ (cf. Sect. 3.4). A virtual lane $\overrightarrow{\pi_E} \in \overrightarrow{\Pi}_E$ is of the structure $\overrightarrow{\pi_E} = \langle \pi_l \rangle \frown \overrightarrow{\pi_c} \frown \langle \pi_{l'} \rangle$, where $\pi_l, \pi_{l'} \in \mathbb{L}$ and $\overrightarrow{\pi_c} \in \mathsf{seq}_{E_d}\mathbb{CS}$ is a non-empty sequence of crossing segments ($\overleftarrow{\pi_E} \in \overleftarrow{\Pi}_E$ is defined symmetrically).

In Lemma 1, for the considered traffic snapshots $\mathcal{TS}$ and $\mathcal{TS}'$, only crossing reservations or claims are of particular interest, as the sets $clm(C)$ and $res(C)$ remain unchanged in $\mathcal{TS}'$. Therefore, we can reduce the considered view V_i to $V = (L_{cs}, X, E)$, where we retrieve L_{cs} from L_i as follows:

$$L_{cs} = \{\overleftrightarrow{\pi_{cs}} \colon \mathsf{seq}\ \mathbb{CS} \mid \exists \overleftrightarrow{\pi_E} \colon L_i \bullet \#\overleftrightarrow{\pi_{cs}} = \#\overleftrightarrow{\pi_E} - 2 \wedge \forall j \colon 2, .., \#\overleftrightarrow{\pi_E} - 1 \bullet \overleftrightarrow{\pi_{cs}}(j-1) = \overleftrightarrow{\pi_E}(j)\}$$

Induction basis. The cases $\phi' \equiv \mathit{free} \mid u = v \mid \mathit{true} \mid cs$ are trivial. Consider $\phi' \equiv re(c)$, where $\phi \equiv re(c) \vee cl(c)$, by assumption in Lemma 1. Assume V and ν such that

$$\mathcal{TS}, V, \nu \models \phi \text{ resp. } \mathcal{TS}, V, \nu \models re(c) \text{ or } \mathcal{TS}, V, \nu \models cl(c)$$

Case 1: With $\mathcal{TS}, V, \nu \models re(c)$, we have $cres(C) \subseteq cres'(C)$ and thus deduce $\mathcal{TS}', V, \nu \models re(c)$ in both cases.

Case 2: With $\mathcal{TS}, V, \nu \models cl(c)$, we observe $cclm(C) = cres'(C)$ for a crossing claim and deduce again $\mathcal{TS}', V, \nu \models re(c)$.

For the other direction, consider $\mathcal{TS}', V, \nu \models re(c)$. By definition of crossing reservation transitions, we observe $\mathsf{ran}\ \overleftrightarrow{\pi}_{cs} \subseteq cres'(C)$ and furthermore also $\mathsf{ran}\ \overleftrightarrow{\pi}_{cs} \subseteq cres(C)$ or $\mathsf{ran}\ \overleftrightarrow{\pi}_{cs} \subseteq cclm(C)$. Altogether this again leads to $\mathcal{TS}, V, \nu \models re(c) \vee cl(c)$.

Induction step. For $\phi \equiv \neg\phi \mid \phi_1 \wedge \phi_2 \mid \exists c\colon \phi_1 \mid \phi_1 \frown \phi_2 \mid \begin{smallmatrix}\phi_2\\ \phi_1\end{smallmatrix}$ we refer to the proof in [HLOR11], as these UMLSL formulae do not have changed syntax or semantics compared with standard MLSL. □

Safety of Crossing Manoeuvres with Help

Consider the extended crossing controller $\mathcal{A}'_{cc}$ for imperfect knowledge from Sect. 5.4, depicted in Fig. 5.10, for which the previous safety proof only changes slightly. As the controller includes communication, we add one assumption **A3** to the assumptions **A1** and **A2** from the previous section:

Assumption A3. Each car is equipped with reliable communication technology (cf. Sect. 5.2, p. 92).

Theorem 2 (Urban traffic safety with help)**.** *If assumptions A1, A2 and A3 hold, every traffic snapshot $\mathcal{TS}$ that is reachable from $\mathcal{TS}_0$ by time transitions and transitions allowed by the road controller$\mathcal{A}_{rc}$ and the crossing controller $\mathcal{A}'_{cc}$ is* safe*.*

Proof. In $\mathcal{A}'_{cc}$, we observe two transitions with a crossing reservation $\mathtt{rc}(\text{ego})$ that needs to be examined. The first transition is for a crossing manoeuvre without help:

$$q_2 \xrightarrow{\neg\exists c\colon (pc(c)\vee ph(c))\wedge\neg lc(\text{ego})\ /\ \mathtt{rc}(\text{ego});\, x:=0} q_5. \tag{6.21}$$

Compared with the respective reservation transition in $\mathcal{A}_{cc}$ we examined thoroughly in the previous section, the only addition to this guard is the subformula $\neg\exists c\colon ph(c)$. Remember that the long version of the potential helper check from p. 101 is defined by $ph(c) \equiv c \neq \text{ego} \wedge (oc(c) \vee ocac(c)) \wedge \neg lc(c)$. With this, we can easily transform $\neg\exists c\colon ph(c)$ as follows:

$$\neg\exists c\colon ph(c) \equiv \forall c\colon c = \text{ego} \vee (\neg oc(c) \wedge \neg ocac(c)) \vee lc(c) \tag{6.22}$$

From formula (6.22), we conclude that no car different from ego can be on the crossing or approaching the intersection, apart from a car currently changing lanes, which again excludes a crossing reservation for this car. Thus, no car can possibly have an overlap

of reserved crossing segments with the ego car E and E can safely reserve its claimed crossing segments.

The second transition is for a crossing manoeuvre with help:

$$q_3 \xrightarrow{x \geq t_w \wedge \neg\exists c{:}(ph(c) \wedge c \notin \mathbb{H}) \wedge \neg\exists c{:}\ pc(c) \wedge \neg lc(\text{ego})\ /\ \texttt{rc}(\text{ego}); x{:=}0} q_4. \tag{6.23}$$

In the guard of transition (6.23), only the subformulae $x \geq t_w$ and $\neg\exists c : (ph(c) \wedge c \notin \mathbb{H})$ are new compared to the guard we considered in our proof for perfect knowledge in the previous section. Here, $x \geq t_w$ only specifies when the transition is conducted, where t_w is the time we considered in Sect. 5.4 to be the time needed for all potential helpers to answer. With $\neg\exists c : (ph(c) \wedge c \notin \mathbb{H})$, we check whether each car that is a potential helper answered and thus is contained in the set $\mathbb{H}$. If there should be a car that did not answer, we have $\mathcal{TS}, V, \nu \not\models \neg\exists c : (ph(c) \wedge c \notin \mathbb{H})$ and the transition cannot be executed. Thus, even if the communication should fail, safety is still guaranteed. If however each car did answer with *yes* and is thus contained in $\mathbb{H}$, the crossing manoeuvre can be safely started. The case that at least one car answered *no* is covered with the transition from q_3 to q_1. □

6.4 Fairness and Liveness of the Crossing Controller

After showing safety of crossing manoeuvres in the previous section, we now implement our crossing controller $\mathcal{A}_{cc}$ from Fig. 5.5 in UPPAAL Stratego to examine its liveness and fairness properties. The UPPAAL implementation of the abstract model is based on the implementation for highway traffic from Sect. 6.2, but now with the urban traffic model with crossing segments where cars follow paths. Together with examining liveness and fairness of $\mathcal{A}_{cc}$, we also confirm the safety result from the previous section.

We explain the UPPAAL implementation of the urban traffic model in Sect. 6.4.1 and give details on the implemented crossing controller in Sect. 6.4.1. After that, we examine safety and liveness of the controller in Sect. 6.4.2 and present and analyse an adapted fair crossing controller in Sect. 6.4.3. We weaken our assumption on 100% reliable communication in Sect. 6.4.4 by briefly introducing *probabilistic automotive-controlling timed automata* (PACTA) and analysing the effects this extension has regarding our properties with UPPAAL Stratego.

6.4.1 The Urban Traffic Model and Controller in UPPAAL

We now introduce the UPPAAL implementation of our model of urban traffic manoeuvres and of the crossing controller. Note that we re-use and adapt several concepts from the implementation for highway traffic that was introduced in Sect. 6.2.1. For instance, we use an adapted version of the method intersect(c,d) from p. 118 for checking the intersection of the occupied space for two cars c and d.

Abstract Model

For our UPPAAL model, we implement a generic 2-by-2 intersection *cr* consisting of road segments with each two lane segments as depicted in Fig. 6.7. We choose this model, as it is a common type of intersection in urban areas and as our protocol is generalisable to bigger intersections (cf. Sect. 3.6). Also, UPPAAL can only verify a limited number of extended timed automata in parallel. This is due to the fact that model checking for timed automata in general grows exponentially with the number of automata [ACD90, HNSY94, LPY95]. We thus consider only those cars that are of interest for our analysis with UPPAAL, i.e. cars approaching *cr*, where the crossing ahead check *ca*(ego) (cf. formula (5.3), p. 94) holds or cars already on *cr* where the on crossing check *oc*(ego) holds (cf. formula (5.1) p. 89) as only those potentially block some crossing segments for other cars. Cars driving away from *cr* are not interesting, as well as cars driving behind cars for which *ca*(ego) holds, as those are not already allowed to claim or reserve crossing segments.

This dramatically reduces the amount of cars to be considered in our model, where we consider only 4 cars, one car approaching from each of the four roads leading to *cr*, e.g. like in the traffic situation depicted in Fig. 6.7.

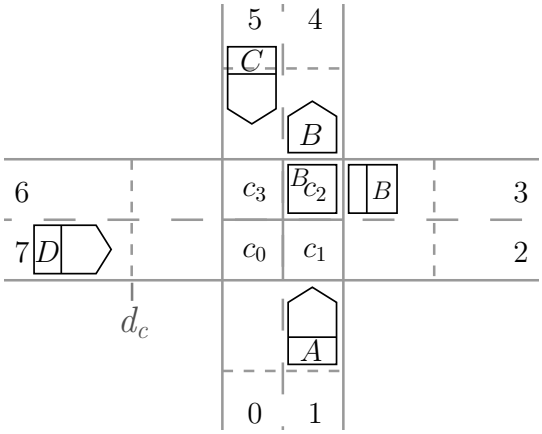

Figure 6.7: Abstract model for UPPAAL implementation.

Number of cars. Generally, we use the described 4 cars in our implementation throughout the remainder of this chapter. However, in the following sections we present a stepwise adaptation of our controller, w.r.t. certain properties. Some of these adaptations increase the systems complexity noticeably, so that the verification of specific queries did not finish within several days. Thus, we use 4 cars in general, but explain whenever we can only use 3 cars. Note that we always use 4 cars for time-bounded queries (cf. Sect. 6.4.2, p. 139), as all of these can be verified comparatively fast due to the time-boundedness of the query.

Path through the intersection. Even in the cases where only 3 cars are used, we cover a large amount of different possible traffic situations at our 2-by-2 intersection. This is because we allow for arbitrary turn manoeuvres at the intersection for each car, by non-deterministically generating a *path through the intersection* for the cars on arriving at the intersection (right-turn, straightforward, left-turn, u-turn). Each time a car successfully finishes a crossing manoeuvre, it is newly spawned in front of the intersection at its last starting point but with a newly generated path through the intersection. With this, we indirectly simulate that more than our actual 4 (resp. 3) cars try to perform a crossing manoeuvre at our intersection. However, we have at most 4 (resp. 3) cars at a time on the intersection. Note that we allow for all possible four crossing manoeuvres at the considered intersection: turning left, driving straight ahead, turning right and doing a u-turn.

Thus, instead of the fixed highway traffic scenario we used in Sect. 6.2 for analysing the system properties of the lane change controller $\mathcal{A}_{lc}$, we now consider a 2-by-2 intersection, where 4 (resp. 3) cars follow arbitrary paths through the intersection. With this, we cover a vast amount of different possible traffic situations at the respective intersection.

Data structure. Similarly to the UPPAAL model for highway traffic from Sect. 6.2.1, we encode information about crossing reservations and claims with a global data structure model_t res resp. model_t clm, where now instead of lane segments crossing segments are represented by a Boolean array with one entry for each of the four considered crossing segments:

```
model_t res[carid_t] = {
  { {0,0,0,0} },
  { {0,0,0,0} },
  { {0,0,0,0} },
  { {0,0,0,0} }     };
```

Note that each line relates to one car and unlike in the implementation for highway traffic, we do not need to represent position and size of each car in model_t res anymore. This is due to the fact that crossing segments are discrete and thus either fully occupied by one car or empty (cf. Chapter 3). As can be seen in the structure above, initially, we assume that none of the cars has a reservation or claim on any crossing segment.

A path through the intersection is represented as follows in our implementation: For each direction from which a car might come, we remember the identifier of the crossing segment the car has to enter *first* in an array int firstsegment[4] = {40, 41, 42, 43}. Here, the numbers from the set {40, 41, 42, 43} are the internal representation of our crossing segments c_0, c_1, c_2 and c_3. We chose this representation, as internally in UPPAAL, it is more convenient to label our crossing segments with integers. We chose integers starting from 40, as lower values are already reserved as identifiers for lane segments. Although we currently do not consider lane segments in our crossing controller, we keep the lane data structure from the highway traffic implementation (cf. Sect. 6.2.1) to allow for a possible future integration of both approaches and implementations (cf. Chapter 8).

For car A from our example we have $\mathsf{firstsegment(A)} = \mathsf{41} \preceq c_1$. On arriving at the intersection, a random crossing segment segment from the set $\{\mathsf{40}, \mathsf{41}, \mathsf{42}, \mathsf{43}\}$ is non-deterministically chosen as the last segment of the path through the intersection. E.g., if $\mathsf{segment} = \mathsf{42} \preceq c_2$ is chosen for car A, the resulting path through the intersection is $\langle \mathsf{41}, \mathsf{42} \rangle \preceq \langle c_1, c_2 \rangle$. Note that if $\mathsf{firstsegment(A)} = \mathsf{segment}$ the path through the intersection contains only that one element segment, meaning the related car turns right at the intersection. Also note that, if in the example $\mathsf{segment} = \mathsf{40} \preceq c_0$ would be chosen for car A, it would do a u-turn using the path through the intersection $\langle \mathsf{41}, \mathsf{42}, \mathsf{43}, \mathsf{40} \rangle \preceq \langle c_1, c_2, c_3, c_0 \rangle$. Whenever a car successfully finished its manoeuvre, it again appears in front of the intersection as a new car with a newly generated path through the intersection.

Representation of the movement of cars. For checking properties for our crossing controller, crossing segments are our critical resource, not lane segments. As crossing segments are discrete, we do not need to consider continuous movement of our cars in our implementation. Instead recall the three phases in which our crossing controller can be (cf. Sect. 5.3, p. 93):

1. away from the intersection (cf. car D in Fig. 6.7),
2. in the crossing ahead phase, where ca(ego) holds and thus the threshold d_c is crossed over (cf. cars A and C in Fig. 6.7) or
3. on the crossing, where oc(ego) holds and possibly some other cars have to wait until a car leaves certain crossing segments (cf. car B in Fig. 6.7).

We abstractly model these three phases a car can be in in our urban traffic scenarios through the implementation of $\mathcal{A}_{cc}$ that we explain in the following section. This includes our notion on how a car 'approaches' an intersection.

Implementation of the Crossing Controller

We now explain the translation of our crossing controller $\mathcal{A}_{cc}$ from Sect. 5.3, Fig. 5.5, into UPPAAL notation. The resulting UPPAAL automaton CRP ('CRossing Protocol') is depicted in Fig. 6.9. Equally as our representation of the abstract model in UPPAAL we introduced in the previous section, CRP uses some of the concepts from the implementation for highway traffic from Sect. 6.2.1. Note that equally as for the highway traffic controller LCP from Sect. 6.2.1, we for now only use internal communication (e.g. claiming[ego]!) to communicate with some observer automata we introduce later.

Initialisation of the controller. The committed initial location q_init is used to let a car (again) appear in front of the intersection, initialised as a new car. For this, UPPAAL offers so-called *select* constructs, whereas in our case with the select construct segment : segmentid_t the last segment for the cars path through the intersection is non-deterministically selected and then stored in an internal variable cs.

In the original initial location q_0, the car is in phase 1, meaning 'away from the intersection' and may stay there for some random time from the interval $[0, \mathsf{t\_away}]$. With this, we simulate that the cars do not spawn directly in front of the intersection, but may take some time until they reach the intersection. Then, $\mathcal{A}_{cc}$ changes to location q_1, which models that the car arrives in the 'crossing ahead' phase 2.

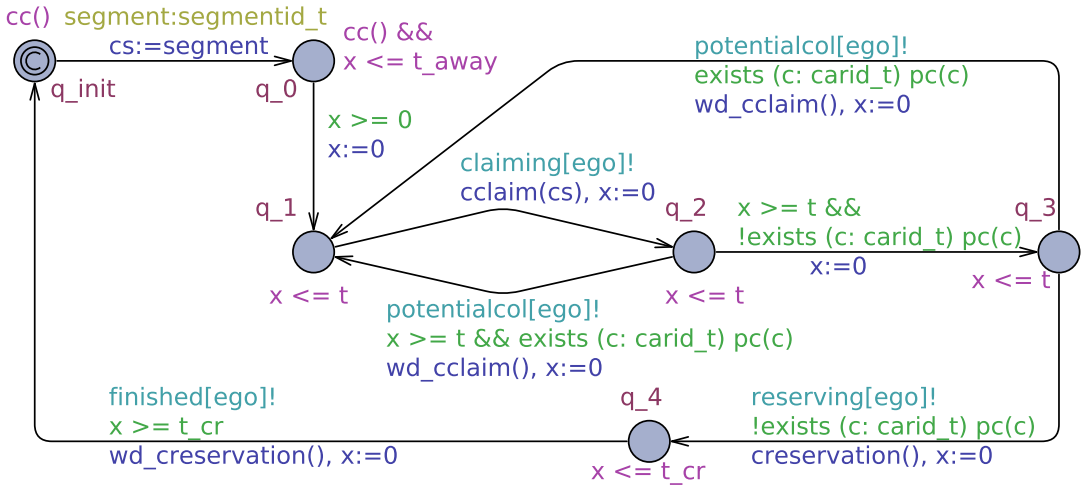

Figure 6.8: Implementation CRP of crossing controller $\mathcal{A}_{cc}$ from Fig. 5.5 in UPPAAL.

Time bounds. Equally as before, we have to assign specific values to our time constants for the UPPAAL implementation. We recall that we use t as a (preferably small) reaction time for the controllers, whereas we set it to the smallest possible integer value > 0 with $\mathsf{t} := 1$. Note that while we use differently labelled constants t and t_c in the controller $\mathcal{A}_{cc}$ from Fig. 5.5, for CRP we use the same constant t as upper bound for the respective invariants in locations q_1 to q_3 for simplification (thus assuming $t = t_c$). With $\mathsf{t\_away} := 10$, we allow a newly spawned car to take up to 10 time units to approach the intersection. Further on we assume a crossing manoeuvre to take 3 time units and set $\mathsf{t\_cr} := 3$. Again, we analyse the influence of these time bounds on our verification results in Sect. 6.4.2.

UMLSL formulae. The implementation of our UMLSL formulae collision check *col*(ego) and potential collision *pc*(ego) use the following simplified version of the method intersect(c,d) from the highway traffic implementation (cf. Sect. 6.2.1):

```
bool intersect(const model_t p1, const model_t p2) {
  return
    exists(i: laneid_t)
        (p1.lane[i] and p2.lane[i])
  or
    exists(i: segmentid_t)
        (p1.segment[i] and p2.segment[i]);          }
```

The simplification is due to the fact that crossing segments are discrete (cf. previous section). Thus, we do not check for an intersection of position intervals but instead check for an intersection of the respective claimed or reserved lane or crossing segments of two cars. With this, we re-use the functions bool cc() and bool pc (carid_t c) from Sect. 6.2.1. Note that we later also check for path intersection.

Our implementation of the formulae crossing ahead *ca*(ego) and on-crossing *oc*(ego) correlates with the cars' phases we introduced in the previous paragraph. As both formulae describe the position of the car with respect to the intersection, these are encoded by the respective location the controller is in. For instance, on entering location q_1 the clock x is reset and we assume CRP enters phase 2, whereas now *ca*(ego) holds.

Controller actions. After some reaction time $\leq t$ in location q_1, CRP claims its required crossing segments with the method call cclaim(cs). This method internally uses the first crossing segment as captured in the array firstsegment and the forwarded parameter cs contains the segment that was randomly chosen in the beginning as last path segment. Then, ego's path through the intersection is generated and simultaneously claimed, meaning the claimed crossing segments are set to 1 in the Boolean array clm[ego]. For instance for car A from Fig. 6.7, again for driving straight ahead, after the method call cclaim(cs) (here: cs = 42), we observe clm[ego] = {0, 1, 1, 0}. The controller actions for withdrawing crossing claims or reservations and setting a crossing reservation are equal as for highway traffic.

The remainder of the controller CRP is a straightforward implementation from the original crossing controller $\mathcal{A}_{cc}$ from Fig. 5.5 and additionally shares some concepts with the lane change controller implementation LCP, described in Sect. 6.2.1.

6.4.2 Verification of Safety and Liveness

Similarly to highway traffic, we start with showing *deadlock freedom* of our crossing controller CRP with the UPPAAL query

```
A[] not deadlock
```
(6.24)

in averagely 27 seconds with a memory usage peak of roughly 126KB. For showing *safety* of CRP, we use the same observer automaton Observer1 as introduced in Sect. 6.2.2 in Fig. 6.3 for the highway traffic controller LCP. Again, with the query

```
A[] not Observer1.unsafe
```
(6.25)

we show in averagely less than 3 minutes with a memory usage peak of 295KB that formula cc_all() holds globally in all possible traces (cf. p. 121). With this, we confirm our safety proof from Sect. 6.3.

For the remainder of this section recall that we consider 4 different cars in our implementation (cf. Sect. 6.4.1), which we name A, B, C and D from now on to distinguish

them. Note that these names do not necessarily relate to the cars depicted in Fig. 6.7, as we consider that the paths for the cars A, B, C and D are chosen non-deterministically on arriving at the intersection. Thus, these cars are arbitrary cars. Also recall that our cars arrive in front of the intersection again as a new car with a new path whenever they successfully finished a crossing manoeuvre. Note that this implies that we expect the same liveness result for each car, as all cars are treated similarly and for no car it is more or less probable to be blocked than for any other car.

Unlive behaviour of `CRP`. Concerning *liveness*, note that our crossing controller `CRP` from Fig. 6.8 does not show the zeno behaviour and livelocks that we sketched for the highway traffic controller `LCP` as 'Problem 1' in Sect. 6.2.2 on p. 122. This is due to the fact that `CRP` already contains the requested invariant `x <= t` in location `q_2` and respective guards including the subformula `x >= t` on the outgoing edges of that location. Thus, we do not have to solve 'Problem 1' as it is already solved. Also note that, compared to `LCP`, after a withdrawn crossing claim `CRP` does not return to the location `q_0`, where it could wait for up to `t_away` time units again. Thus, generally we might except faster verification result for `CRP` than those achieved for the controller `LCP′` without the additional adaptation undertaken in Sect. 6.2.2.

For analysing liveness of `CRP`, we use the same observer automata `Observer(i)` for each controller `CRP(i)` as in Sect. 6.2.2. Again, we consider the query

```
A<> Observer(i).success
```
(6.26)

for verifying liveness of our system. However, without any adaptations, this query does not hold for any of the four cars. This result is not surprising, as similarly to the case for highway traffic from Sect. 6.2.2, we have to cope with 'Problem 2' (p. 123), the special case where two or more cars block each other infinitely often.

Statistical liveness analysis of `CRP`. We give the statistical analysis results for the evaluation query for *time-bounded liveness*

```
Pr [<= k] (<> Observer(i).success)
```
(6.27)

in Table 6.3 using the same statistical parameters as before for highway traffic ($\alpha = \varepsilon = 0.01$, cf. p. 124). Note that in contrast to the observed values for highway traffic in Sect. 6.2.2, the probability intervals do not differ between the cars A, B, C and D for each k. This is because each car spawns in front of the intersection with a random path. Thus, in contrast to the tables we presented in Sect. 6.2.2 for highway traffic, we present only one column with estimation intervals for an arbitrary car $i \in \{A, B, C, D\}$ in this section.

The verification time for each k lay between 30 and 120 seconds and the memory usage had peaks at 300KB. Compared to highway traffic, the observed verification time is notably bigger, but still reasonably low. On the one hand this is due to the select statement we use for the initialisation of `CRP`, with which a path is chosen *non-deterministically* for each car. On the other hand, we ascribe the increased verification time to the fact

time bound	computed probability interval
$k = 5$	$[0.1128, 0.1328]$
$k = 10$	$[0.2003, 0.2203]$
$k = 15$	$[0.2311, 0.2511]$
$k = 20$	$[0.2483, 0.2683]$
$k = 30$	$[0.2706, 0.2906]$
$k = 50$	$[0.3227, 0.3427]$

Table 6.3: Statistical evaluation of the time-bounded liveness query `Pr [<= k] (<> Observer(i).success)` for `CRP` with different upper time bounds k.

that in urban traffic more scenarios with potential collisions exist, as we have 4 cars that can potentially collide instead of 2. This means, more system traces that need to be examined by UPPAAL exist.

The observed values for the probability intervals are distinctly lower than for highway traffic (about 15% to 48%). Further on, after 30 time units cars passed the intersection only with averagely 28% and even after 50 time units the cars finished a crossing manoeuvre only with averagely 33%. In highway traffic, cars with the respective controller `LCP` already changed lanes with around 90% already after 20 time units. The reason for this difference is again that there are a lot possibilities for the 4 cars to block each other on the intersection. This is as with `LCP`, no particular crossing strategy is pursued but the controllers only claim and reserve segments completely uncoordinated. It is also noticeable that the values increase only slowly with each increase of k.

Concerning the influence of the chosen parameters, the influence of the time constant t_cr is relatively low for urban traffic. On decreasing (resp. increasing) t_cr by 1 the probabilities only grow (resp. drop) by averagely 3% each time. Further on, for highway traffic, the influence of t_idle on the verification results was large, as with t_idle $= 1$ the cars were successful in changing lanes with $\geq 93\%$ already after $k = 10$ execution time. In urban traffic, the counterpart of t_idle is t_away. However, on setting t_away $= 1$, instead of increased values, the probabilities for success drop immensely by averagely 30%. The reason for this is that the cars arrive faster at the intersection, whereas more cars try to access crossing segments simultaneously.

An approach on increasing liveness. To increase liveness of `CRP`, we suggest adding a location q_wait, similarly as was done for highway traffic in Sect. 6.2.2 (p. 125). The resulting adapted controller `CRP′` is depicted in Fig. 6.9. As the adaptation is done exactly as was done for highway traffic in Sect. 6.2.2 we directly start with checking our verification queries for `CRP′`.

Naturally, query (6.26) still does not hold for any of the cars. This is because equally as for highway traffic there still exists the case where two cars infinitely often follow the loop of first claiming a crossing segment simultaneously, withdrawing the claim because of the detected potential collison and then waiting for exactly the same time in location

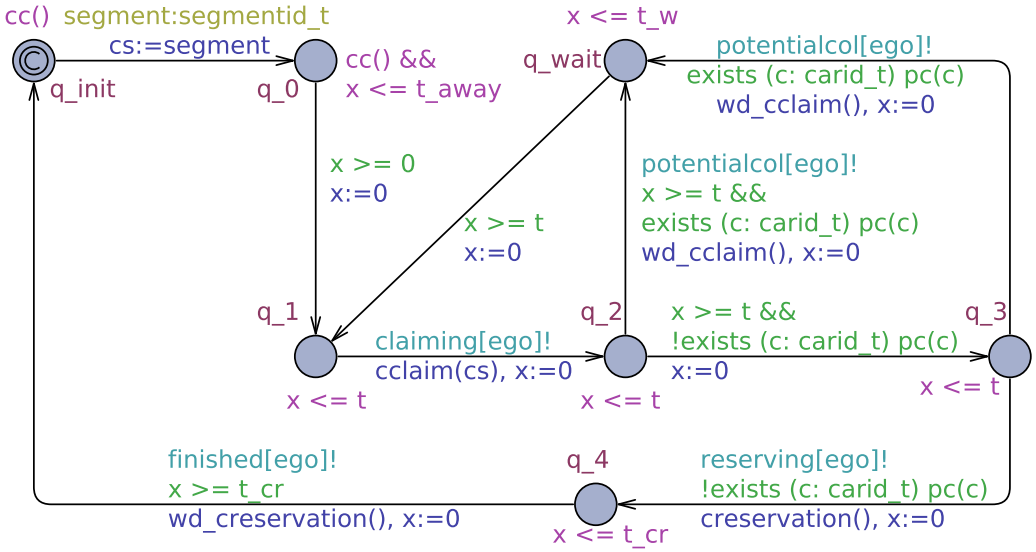

Figure 6.9: Adapted crossing controller implementation CRP′ with additional location q_wait to improve liveness.

time bound	computed probability interval
$k = 5$	$[0.1593, 0.1793]$
$k = 10$	$[0.3561, 0.3761]$
$k = 15$	$[0.5372, 0.5572]$
$k = 20$	$[0.6705, 0.6905]$
$k = 30$	$[0.8324, 0.8524]$
$k = 50$	$[0.9506, 0.9706]$

Table 6.4: Statistical evaluation of the query `Pr [<= k] (<> Observer(i).success)` for the adapted controller CRP′ with different upper time bounds k.

q_wait. However the statistical evaluation `Pr [<= k] (<> Observer(i).success)` of query (6.26) provides quite satisfying results as we provide in Table 6.4. Already for $k = 10$, we observe an increased probability of success from averagely 21% to averagely 36%. Additionally, for $k = 30$ the cars started their crossing manoeuvre with a probability of more than 83%, whereas for CRP the respective value was averagely 28%.

While again the results for statistical liveness of the adapted controller CRP′ are quite satisfying compared to those for CRP, we fail to show actual liveness of our crossing controller due to the sketched special case of cars blocking each other (infinitely) long. Similarly as concluded for highway traffic (cf. p. 127), we argue that our decentralised uncooperative crossing controllers have their limits. Thus, we suggest that cars should communicate with each other instead of blocking each other. With this we achieve both, actual liveness and fairness of our crossing controller as we introduce in the next section.

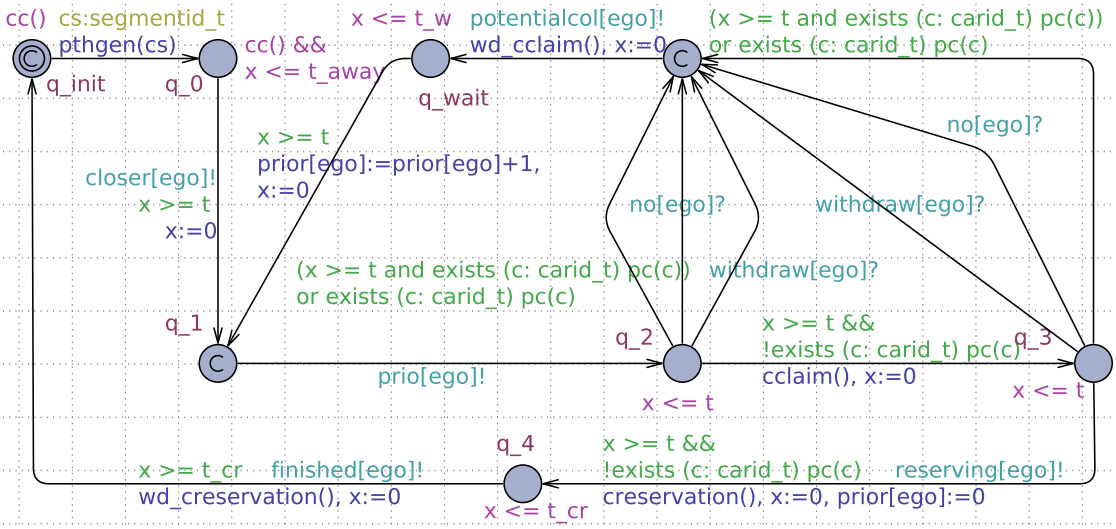

Figure 6.10: Crossing controller $\texttt{CRP}_\texttt{F}$ that sends its priority for reserving some crossing segments via the channel prio[ego].

6.4.3 Ensuring Liveness and Introducing Fairness by Cooperation

To finally show full liveness and introduce *fairness* to our crossing controller $\texttt{CRP}'$, we add communication, so that the controllers may coordinate their crossing manoeuvres. For this purpose, we introduce a concept of *priorities*, where the priority of a car increases the longer it waits in front of an intersection without getting access to it. The results of this section are described in [BS19]. We depict the adapted crossing controller $\texttt{CRP}_\texttt{F}$ in Fig. 6.10. Further on, we introduce a helper controller $\texttt{HP}_\texttt{F}$ which evaluates priorities sent by $\texttt{CRP}_\texttt{F}$. We verify actual liveness and show fairness of $\texttt{CRP}_\texttt{F}$ later in this section, starting on p. 145. In the following, we explain the controller CRP from the perspective of the ego car with its controller instance $\texttt{CRP}_\texttt{F}(\texttt{ego})$.

Crossing controller $\texttt{CRP}_\texttt{F}$ (Fig.6.10). Recall that for $\texttt{CRP}'(\texttt{ego})$, the path through the intersection is built directly on claiming the needed crossing segments with the function cclaim(cs). However for $\texttt{CRP}_\texttt{F}(\texttt{ego})$, ego's path through the intersection is already generated on initialisation with the function call pthgen(cs). Again, cs is the last path segment non-deterministically generated with the select construct cs : segmentid_t. The benefit of generating the path at this point is that cars can already check for path intersections before actually claiming the respective crossing segments. On approaching an intersection and thus entering location q_1, the controller $\texttt{CRP}_\texttt{F}(\texttt{ego})$'s priority prior[ego] is initially set to 0 and immediately sent to all other surrounding cars via a broadcast channel prio[ego].

All other cars i each use a *helper controller* $\texttt{HP}_\texttt{F}(\texttt{i})$ to determine whether the priority of the newly approached car is big enough to enter the crossing, or if their own cars may proceed first. We introduce this helper controller later in this section. If ego's

request to enter an intersection was rejected, it enters location q_wait, similarly as if it detected a potential collision. There, $\texttt{CRP}_\texttt{F}$(ego) waits some time between t and t_w before incrementing its priority with prior[ego] := prior[ego] + 1, changing back to location q_1 and sending it again. Such a reject happens either if $\texttt{CRP}_\texttt{F}$(ego) receives a message withdraw[ego]! by its own helper controller, or a message no[ego]! by some other car's helper controller.

However, if there was no reject detected within t time units while $\texttt{CRP}_\texttt{F}$(ego) is in location q_2, the path through the intersection is claimed with the controller action cclaim() and $\texttt{CRP}_\texttt{F}$(ego) proceeds to location q_3. There again, if a potential collision is detected or a reject is received there, the controller also proceeds to q_wait without entering the intersection. This is needed as there might be another controller $\texttt{CRP}_\texttt{F}$(i) that was forced to wait but now has a significantly bigger priority for entering the intersection than $\texttt{CRP}_\texttt{F}$(ego). However, if no reject or potential collision was detected, the controller reserves its path through the intersection after t time units. On entering the intersection, prior[ego] is set to 0, as after the crossing manoeuvre $\texttt{CRP}_\texttt{F}$(ego) is spawned in front of the intersection as a new car with a new path through the intersection.

Note that with the output actions closer[ego]!, reserving[ego]! and finished[ego]! on some transitions, $\texttt{CRP}_\texttt{F}$(ego)'s own helper controller $\texttt{HP}_\texttt{F}$(ego) is notified in which location the controller is in. This is needed as the helper evaluates other cars' priorities based on this. Additionally, these communications are again used for observer automata to synchronise with $\texttt{CRP}_\texttt{F}$(ego). Note again that we consider these communications as *internal* (i.e. *wired*), as the participating controllers are located in the same car.

Helper controller $\texttt{HP}_\texttt{F}$ (Fig.6.11). If the helper controller $\texttt{HP}_\texttt{F}$(ego) is in its initial location h_0, the related car is not close to an intersection and thus does not need to react to other car's requests. However, on receiving a message closer[ego]? from the related controller $\texttt{CRP}_\texttt{F}$(ego), the controller $\texttt{HP}_\texttt{F}$(ego) changes to location h_1. There, on receiving a message with prio[d]?, the controller checks for intersections between the own (planned or claimed) path through the intersection and the claimed path of the requesting car the variable d relates to. Note that we distinguish between two different types of intersections:

- Transition to the left of h_1: In this case, there exists an intersection between the two paths of the cars ego and d (pthcc(d)) but ego does not yet have a claim and thus there does not exist an intersection between ego's claim and the requested path of car d (!pthclmintersect(d)). Then, the priorities are compared as we describe in the following.
- Transition to the right of h_1: In this case, ego already has a claim which intersects with the requested path of car d (pthclmintersect(d)). For comparing the priorities, ego uses an increased value prior[ego] + s, as we consider ego's claim to be worth more than the request of a potentially newly arrived car. Only if d has a *significantly* (s) higher priority, it has the right to enter the intersection before ego. A significant high priority may occur if a car already waited a long time before the intersection and visited q_wait perhaps several times. Also, we could assign a

significant high priority to an emergency vehicle so that it always has the priority to enter an intersection before all other cars.

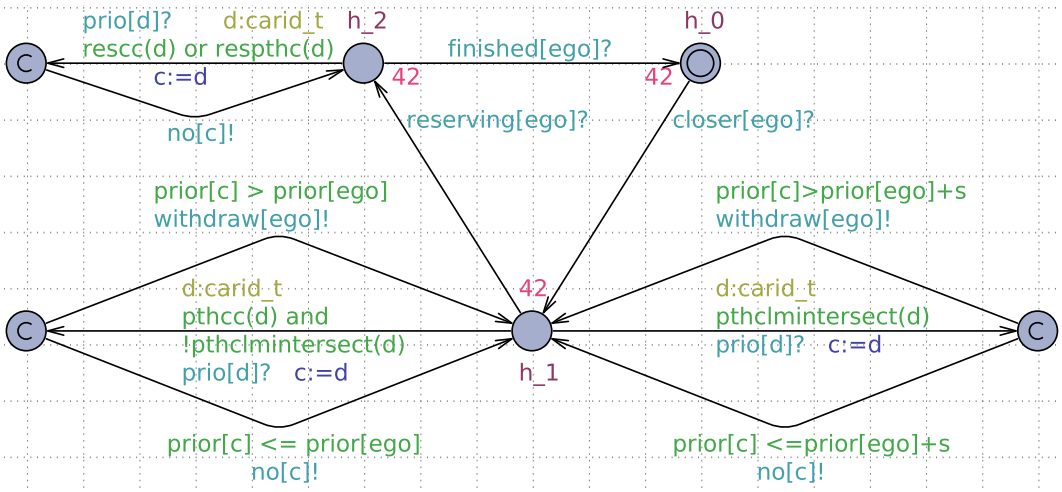

Figure 6.11: Helper controller $\texttt{HP}_\texttt{F}$ comparing priorities of (other) cars.

If an intersection of path segments exists, there are two possible actions for $\texttt{HP}_\texttt{F}[\texttt{ego}]$:

- The priority of ego is (significantly) greater or equal ($\texttt{prior}[\texttt{ego}] \geq \texttt{prior}[\texttt{c}]$) than the priority of the requesting car $c = d$, then $\texttt{no}[\texttt{c}]!$ is sent and the crossing controller $\texttt{CRP}_\texttt{F}(\texttt{c})$ of the car $\nu(c)$ has to withdraw its claim, or
- The priority of ego is (significantly) smaller ($\texttt{prior}[\texttt{ego}] < \texttt{prior}[\texttt{c}]$) than the priority of the requesting car $c = d$, then $\texttt{withdraw}[\texttt{ego}]!$ is sent and $\texttt{CRP}_\texttt{F}(\texttt{ego})$ has to withdraw its own claim.

If the ego car enters an intersection and sends $\texttt{reserving}[\texttt{ego}]!$ to its helper controller, then $\texttt{HP}_\texttt{F}(\texttt{ego})$ further on declines all possible requests concerning crossing segments ego currently occupies. Finally, when ego announces a finished crossing manoeuvre with sending $\texttt{finished}[\texttt{ego}]!$, $\texttt{HP}_\texttt{F}(\texttt{ego})$ changes back to its initial location.

Verification of Actual Liveness and Fairness

We again analyse our queries for an arbitrary car. However, with our described adaptations, verification time increases dramatically for several UPPAAL queries (i.e., queries that are not UPPAAL SMC's evaluation queries). This is due to the added communication and our new helper controllers $\texttt{HP}_\texttt{F}$. However, evaluation queries are not affected by this increase in verification time. Thus, we analyse our evaluation queries for 4 cars as usual, but for the other queries we use a model containing three cars, where $i \in \{A, B, C\}$. Note that we were only able to show safety of CRP using 2 cars, as verification for 3 cars did not finish within a week.

time bound	computed probability interval
$k = 5$	$[0.3186, 0.3386]$
$k = 10$	$[0.6186, 0.6386]$
$k = 15$	$[0.8659, 0.8859]$
$k = 20$	≥ 0.98
$k = 30$	≥ 0.98
$k = 50$	≥ 0.98

Table 6.5: Statistical evaluation of the bounded liveness query $\texttt{Pr}[<= \texttt{k}](<> (\texttt{Observer(i).success}))$ for the controller $\texttt{CRP}_\texttt{F}$ with different upper time bounds k.

Liveness. We finally show full liveness, as with the introduced cooperating controller extension $\texttt{CRP}_\texttt{F}$ the query

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

holds for an arbitrary car $i \in \{A, B, C\}$. The average verification time for this query was already 27 min. and memory usage peaks reached 2, 7GB. For 4 cars, this query did not finish after 2 days. However, as all cars use the same crossing protocol and thus behave similarly, this result is transferable to more cars.

Nevertheless, as before, we also give the respective success intervals for time-bounded liveness. With this, we can show that our property does not only hold *finally* but with a high probability *soon*, within k time. For this, again the query

$$\texttt{Pr}[<= \texttt{k}](<> (\texttt{Observer(i).success})) \tag{6.28}$$

is used for 4 cars. We depict the results in Table 6.5 and use the same statistical parameters as before ($\alpha = \varepsilon = 0.01$, cf. p. 124). We can deduce from Table 6.5 that using $\texttt{CRP}_\texttt{F}$, the cars entered the intersection after $k = 20$ time units with a probability of more than 98%, whereas before using $\texttt{CRP}'$, we only observed a probability of averagely 68% after 20 time units (cf. Table 6.5).

Fairness. For analysing fairness of $\texttt{CRP}_\texttt{F}$, we first introduce a new observer automaton $\texttt{Observer}_\texttt{F}$ (cf. Fig. 6.12). For each instance $\texttt{CRP}_\texttt{F}(\texttt{ego})$ of the fair crossing controller, there exists a related controller $\texttt{Observer}_\texttt{F}(\texttt{ego})$. Whenever the ego car approaches an intersection and $\texttt{CRP}_\texttt{F}(\texttt{ego})$ sends its priority with $\texttt{prio}(\text{ego})!$, $\texttt{Observer}_\texttt{F}(\texttt{ego})$ changes to a location wait. There, if another controller $\texttt{CRP}_\texttt{F}(\texttt{c})$ *reserves* crossing segments that intersect with ego's path through the intersection, although $\texttt{CRP}_\texttt{F}(\texttt{c})$'s priority is significantly lower than ego's, the observer changes to a location bad.

With this, for analysing fairness of $\texttt{CRP}_\texttt{F}$, we introduce the query

$$\texttt{E} <> (\bigvee_{\texttt{i} \in \{\texttt{A},\texttt{B},\texttt{C}\}} \texttt{ObserverF(i).bad}). \tag{6.29}$$

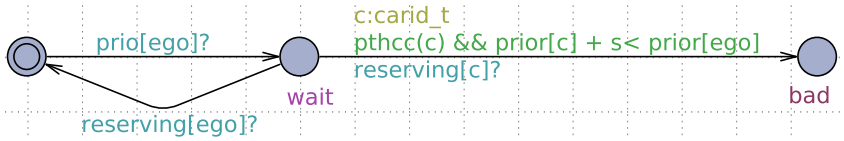

Figure 6.12: $\texttt{Observer}_{\texttt{F}}(\texttt{ego})$ checking for fair treatment of its related crossing controller $\texttt{CRP}_{\texttt{F}}(\texttt{ego})$.

If query 6.29 would hold, this would mean that there exists a system run in which a controller with a (significantly) lower priority and an intersecting path with another controller with a (significantly) higher priority entered the intersection before that controller. This would mean *unfair behaviour* according to our notion. Thus, we expect this property not to be satisfied.

Indeed, we can show in 30 min. average verification time with memory usage peaks of $2,5$GB that there does not exist a computation path, where this property holds. Note that again, only a model with 3 cars was used, as query (6.29) is not a time-bounded evaluation query and for 4 cars the query could not be checked within a reasonable time bound (using approximately 85 GB of RAM, including 54 GB SWAP, within nearly 48 hours).

6.4.4 Introducing Uncertain Communication into the Protocol

In Sect. 5.2, we stated several assumptions we make for our controllers, amongst others that the communication via broadcast channels never fails. This is a very strong assumption that we relax in this section. The results presented in this section are again based on our work published in [BS19].

We only briefly introduce *Probabilistic* ACTA (PACTA), before adding uncertain communication to the crossing controller implementation $\texttt{CRP}_{\texttt{F}}$ from the previous section. After that we analyse this probabilistic extension we name by $\texttt{CRP}_{\texttt{prob}}$ using UPPAAL. For details on PACTA, see [BS19] and moreover [Bis18], where PACTA were first introduced.

Probabilistic ACTA. For the probabilistic extension of our ACTA we use the respective concept that was introduced for *probabilistic timed automata* in [KNPS04]. We directly explain our extension using the PACTA $\mathcal{A}_P$ depicted in Fig. 6.13. This controller waits for 7 time units in its initial location, before proceeding to a committed intermediate location c_1. There, with a probability of $\frac{2}{3}$, the controller proceeds to location q_2, while withdrawing a crossing claim, and with a probability of $\frac{1}{3}$ to q_3, while transferring a crossing claim into a reservation.

Following this example, *probabilistic transitions* are generally partitioned into two parts:

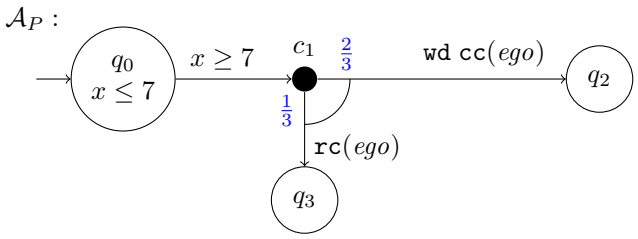

Figure 6.13: A PACTA $\mathcal{A}_P$. When the location c_1 is reached, $\mathcal{A}_P$ proceeds to q_2 with probability $\frac{2}{3}$ and to q_3 with probability $\frac{1}{3}$.

- one part leading to an intermediate committed location, where guards φ and input actions $a?\overrightarrow{\delta} : \varphi$ are allowed on, and
- a second part which contains transitions with probabilities leaving the committed location. The sum of all these probabilities must equal 1. Only output actions $a!\overrightarrow{\delta}$, controller actions c_{act} and data modifications ν_{act} are allowed on this second part of a probabilistic transition.

Crossing controller $\texttt{CRP}_{\texttt{prob}}$ with uncertain communication (cf. Fig. 6.14). For all communication actions in the crossing controller $\texttt{CRP}_{\texttt{F}}$ from the previous section, we use 'normal' broadcast channels. However, some communication channels can be considered to be used for inner-vehicle communication, i.e. all communications with the helper controller $\texttt{HP}_{\texttt{F}}$, as this controller is located in the same car. We assume that these communication channels are therefore wired and have a failure rate near zero. In fact, the only two channels used for real vehicle-to-vehicle (V2V) communication are prio and no. We only use uncertainty on communications via these two V2V channels, as we consider these channels to be unwired and to have a significantly larger rate of failure.

For the extension of $\texttt{CRP}_{\texttt{F}}$ to $\texttt{CRP}_{\texttt{prob}}$ this means that the only change occurs at the transition from location q_1 to q_2, where the controller communicates via the – now uncertain – channel prio. As all other concepts of the controller remain unchanged, we only depict the changes that occur in that part of $\texttt{CRP}_{\texttt{prob}}$ in Fig. 6.14. There, we use a probability p_s for successful sending and a probability p_f for unsuccessful sending. We specify values for these probabilities in the following paragraph on the verification of properties of $\texttt{CRP}_{\texttt{prob}}$.

Likewise, the respective transition where a message is sent via the channel no in the helper controller $\texttt{HP}_{\texttt{F}}$ changes. Note that input actions on uncertain channels do not change, as we assume the uncertainty only occurs on sending a message, not on receiving a message.

Statistical verification of $\texttt{CRP}_{\texttt{prob}}$. Given the probabilistic extension $\texttt{CRP}_{\texttt{prob}}$ of $\texttt{CRP}_{\texttt{F}}$, we now analyse how the properties of the protocol change. For this, recall from the previous section that $\texttt{CRP}_{\texttt{F}}$ was safe, live and fair. As expected, the system is neither live nor fair anymore if communication can fail. However, it remains safe. Due to an exponential

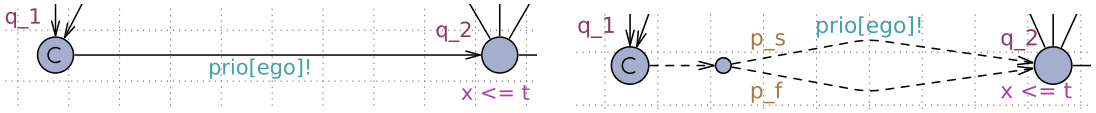

Figure 6.14: Original transition from $\texttt{CRP}_\texttt{F}$ (Fig. 6.10) to the left and transition with uncertain communication from the adapted controller $\texttt{CRP}_\texttt{prob}$ to the right.

increase in complexity, it was only possible to verify the safety property for two cars, as for more than two cars the verification did not finish within several days.

To determine the impact of the lossy channels, we give the results for the query (6.28) on time-bounded liveness `Pr[<= k](<> Observer(i).success)` in Table 6.6. As our fairness query (6.29) does not hold generally with uncertain communication, we adapt query (6.29) to a time-bounded version of fairness

$$\texttt{Pr}[<= \texttt{k}](<> \bigvee_{\texttt{i}\in\{\texttt{A},\texttt{B},\texttt{C}\}} \texttt{ObserverF(i).bad}), \tag{6.30}$$

similarly as done before for examining liveness. The results of this second query (6.30) for showing the absence of fairness are given in Table 6.7. For both queries, we use 4 cars as before for evaluation queries and examine the respective query with different values for k, `p_s` and `p_f`.

k	`p_s` $= 0.99$, `p_f` $= 0.01$	`p_s` $= 0.95$, `p_f` $= 0.05$	`p_s` $= 0.8$, `p_f` $= 0.2$
$k = 5$	$[0.3164, 0.3364]$	$[0.3157, 0.3357]$	$[0.3146, 0.3346]$
$k = 10$	$[0.6154, 0.6354]$	$[0.6139, 0.6339]$	$[0.6112, 0.6312]$
$k = 15$	$[0.8616, 0.8816]$	$[0.8552, 0.8752]$	$[0.8360, 0.8560]$
$k = 20$	≥ 0.98	$[0.9794, 0.9994]$	$[0.9516, 0.9716]$

Table 6.6: Results of the query for time-bounded liveness `Pr[<= k](<> Observer(i).success)` with different parameters for `k` and `p_s`.

The following two observations from Tables 6.6 and 6.7 are especially remarkable: Firstly and unsurprisingly, the system becomes less live and fair the more lossy the channels are. Secondly, the system becomes also less live and fair the longer the time `k` we observe it (compared to the system without uncertainties). The reason for this is that at the very start of the system run, the priorities are all 0 and it is not that likely that one car's priority increases much more than another car's priority, thus communication is not that important. This observation changes over time when cars finish their crossing manoeuvres and start from the beginning again, leading to more significant differences in their respective priorities.

Note that the values in Table 6.6 *decrease* (along the columns), as liveness is less and less present, while in Table 6.7 the values *increase*, because the query asks for *unfairness* instead of fairness. Thus, a probability of 18% for `k` $= 20$ and `p_s` $= 0.8$ means that

after 20 time units fairness is still guaranteed in averagely 82% of the simulated system runs.

Despite loosing liveness and fairness when allowing lossy broadcast channels, the system still remains safe, i.e. there are never two intersecting reservations of two different cars (recall: this query could only be analysed for two cars). Thus, the safety result from [Sch18a] is still preserved in our extended crossing controller protocol. Also, the system remains both live and fair with relatively high probabilities (cf. Tables 6.6 and 6.7). We briefly present ideas where else to apply uncertainties in our approach in future work in Chapter. 8.

k	p_s $= 0.99$, p_f $= 0.01$	p_s $= 0.95$, p_f $= 0.05$	p_s $= 0.8$, p_f $= 0.2$
$k = 5$	$[0, 0.0200]$	$[0, 0.0200]$	$[0, 0.0200]$
$k = 10$	$[0, 0.0200]$	$[0.0022, 0.0222]$	$[0.0109, 0.0309]$
$k = 15$	$[0.0027, 0.0227]$	$[0.0182, 0.0382]$	$[0.0838, 0.1038]$
$k = 20$	$[0.0032, 0.0232]$	$[0.0444, 0.0644]$	$[0.1787, 0.1987]$

Table 6.7: Probability intervals for the time-bounded fairness query $\texttt{Pr}[<= \texttt{k}](<> \bigvee_{\texttt{i} \in \{\texttt{A,B,C}\}} \texttt{ObserverF(i).bad})$ with different parameters for k and p_s.

6.5 Related Work

Safety. MLSL itself is a thoroughly researched and strong formal approach for proving properties of autonomous traffic manoeuvres. Some years ago, the first computer-based assistance for reasoning with a new hybrid extension of MLSL (HMLSL) was introduced by Linker [Lin17b]. *Hybrid* in this case means that concepts from Hybrid Logic [] are introduced to the MLSL approach. The author successfully investigates safety constraints for the motorway traffic scenarios from [HLOR11] with Isabelle/HOL [NPW02]. In contrast to that approach, we investigate safety of the traffic manoeuvre *controller protocols* in this thesis.

Further on, in [XL17, XLGD19], the authors use a scenario-based approach to show safety of their autonomous driving system (see also Sect. 3.7, p. 67). For this, mixed traffic with manned vehicles is considered where driving decisions of other cars may only estimated and not known. The authors use UPPAAL SMC to estimate driving decisions of other cars and to show *relative safety* of their system. Relative safety here means, that the safety property holds with a certain probability within all reachable states of their abstract model. It is stated that safety can only hold relatively because driving decisions of other cars are uncertain and unobservable.

For more related work on safety aspects of autonomous systems, we refer back to the related work Sect. 1.3, where most of the presented approaches on car manoeuvres investigate safety aspects to some degree.

Liveness. The term of liveness was shaped by [Lam77], where safety and liveness were investigated for a *producer/consumer program*. Further on, the authors use a formal proof method, which is based on temporal logic, to reason about liveness properties in [OL82]. Additionally, earlier in [Dij71], the author proposes a hierarchical ordering of sequential processes with a *director* and a *secretary* granting processor time to the various processes as a scheduling concept to avoid starvation in a system.

Another work to mention here is [Asp18], where a constraint solver is used to prove the correctness of a vehicle coordination protocol for intersections. This work bases on the more detailed described work in [AMB+12]. Similarly to our approach, phases like *farAway*, *close* (to the intersection) are considered for their crossing controller. Both safety and liveness in the sense of progress are considered. In contrast to our work, the work focuses on the effect of parameters for longitudinal movement, whereas we focus more on the effect of time and the choice of probabilities for communication failure. Although uncertain communication is also possible in [Asp18], there are no quantitative statements regarding the choice of such parameters.

Several *centralised scheduling* mechanisms exist for ensuring both safety and liveness at intersections. The main difference to our approach is that we use decentralised scheduling, where the cars plan and negotiate their manoeuvres themselves. For instance, in [AS10], the authors introduce a motion planning algorithm for *autonomous intersection management* (AIM), where with a first come, first served (FCFS) policy, cars approaching at an intersection send reservation requests for the intersection to a centralised scheduler. However, it is possible for cars to get stuck with the FCFS concept. This intersection management concept is thus optimised in [ASS11], where the authors introduce a batch processing method of reservations in AIM. The authors show by simulation that with this adjustment, their intersection management method enforces liveness at intersections. They also propose that a coordination mechanism is called fair, if it e.g. chooses the vehicle waiting the longest to enter a specific position.

Additionally, in [CDV12], a *centralised supervisor* for collision avoidance at intersections is presented. This supervisor is based on a hybrid algorithm and acts as a scheduler for the cars. However, the complexity of this centralised supervisor grows polynomial with the number of controlled agents. To solve this problem, the authors propose a discrete-event system (DES) abstraction in [DCDVL17], where the system is discretised in space and time. Additionally, in [DCDVL17], a limited set of uncontrolled vehicles are taken into account. These uncontrolled vehicles are represented through uncontrollable events of the DES abstraction. With that, now the goal is to synthesise the least restrictive supervisor for the controlled cars, with which the system still remains safe and deadlock-free. The considered state reduction could be a starting point to simplify our UPPAAL implementation.

Fairness. In Sect. 5.5, we referred to *(de-) centralised cooperative intersection management* (CIM) approaches (e.g. [LP11, FFCa+10]) that are widely used for the design and implementation of crossing protocols. Many of these CIM approaches also examine safety aspects of their protocol and several of them are designed and verified w.r.t. liveness and fairness properties comparable to the properties we analysed in this chapter.

Finally worth mentioning is a distributed reservation approach that was proposed earlier based on Petri-Net models [NRTT97]. For implementing fairness, the authors also propose priorities depending on waiting time and velocity and support their claims by simulation results. Only one car at a moment has a send-token and may thus communicate with the other cars. However, time is not explicitly considered with the Petri-Net models.

Chapter summary. We strengthened the respective MLSL approaches from [HLOR11] and from this thesis by implementing the respective controllers for highway and urban traffic in UPPAAL and successfully confirming their safety property. We additionally optimised both controllers by examining and implementing liveness properties for them. Finally, we added a fairness protocol using priorities to the crossing controller and proposed PACTA for introducing uncertainty to MLSL controllers. This is a start for weakening some of the assumptions we have for the controllers.

7 Case study: A Hazard Warning Communication Protocol with MLSL

The previous approaches on MLSL, similarly as the controllers hitherto investigated in this thesis, focus on traffic manoeuvres for different traffic types and desirable properties of controllers conducting these manoeuvres. In this chapter, we examine a different use-case of the MLSL approach: we introduce an MLSL *hazard warning communication protocol*. Note that this chapter reflects the contents of [OS17].

For our case study, we investigate an approach of Müllner, Fränzle and Fröschle [MFF15], where a communication protocol for a timely traffic-hazard warning to other traffic participants was introduced. The authors use simulation techniques to estimate the probability that the hazard warning message is received in time. Their simulation framework works with discrete time steps, where a decentralised environmental notification message is sent at intervals of one time step.

We use the hazard warning scenario sketched in [MFF15] to perform a case study for the MLSL approach. For this, we formalise the timing aspects of their simulation-based analysis of the communication protocol by using automotive-controlling timed automata (ACTA) (cf. Chapter 4). Using this formalisation, we prove the timely warning property with a mathematical proof, assisted by verification with UPPAAL. Note that by using ACTA, we also use a continuous time dimension instead of discrete time steps.

Additionally, we link the timely warning property with spatial reasoning to prove avoidance of hazard collision by introducing the new MLSL extension *Hazard Warning Multilane Spatial Logic* (HMLSL) to cope with hazards. As before, formulae of this logic appear in the guards and invariants of ACTA to establish the desired spatial safety properties. Note that the approach from [MFF15] addresses highway traffic manoeuvres, which is why we also only introduce hazards to the highway traffic MLSL from [HLOR11]. However, we give hints on possible extensions to other traffic scenarios in future work in Sect. 8.4.

This chapter is structured as follows. We start with introducing hazards to the abstract model and logic from [HLOR11] in Sect. 7.1. After that, we introduce our hazard warning communication concept in Sect. 7.2, where we also specify the actual *hazard warning controllers*. Finally, we prove both timely and spatial hazard safety in Sect. 7.3.

7.1 Abstract Model with Hazards and HMLSL

In this section, we adopt the basic scenario for our case study from [MFF15] and we explain the needed modifications for the model from [HLOR11] that we introduced in our preliminary Sect. 2.2. In both approaches, highway traffic scenarios are considered, where all traffic drives in one direction. We assume that all cars are equipped with the hazard warning controllers we introduce later. Furthermore, we demand that all cars are equipped with the lane change, distance and velocity controllers introduced in Sect. 5.1 to guarantee collision freedom while the hazard warning message is propagated.

The approach from [MFF15] focuses on *stationary traffic hazards*, like traffic jams, collisions, slippery street parts (ice, aquaplaning), and limited sight (e.g. due to fog). Such stationary hazards are discussed in the ETSI standard 102638 [ETS09], amongst various other possible hazardous scenarios that we do not consider here. We assume a hazard to stretch over an arbitrary number of lanes and to have a positive extension along the lanes. For now, we restrict the number of occurring hazards to one at a time. We give an example to accompany the following introduction of our model with Fig. 7.1.

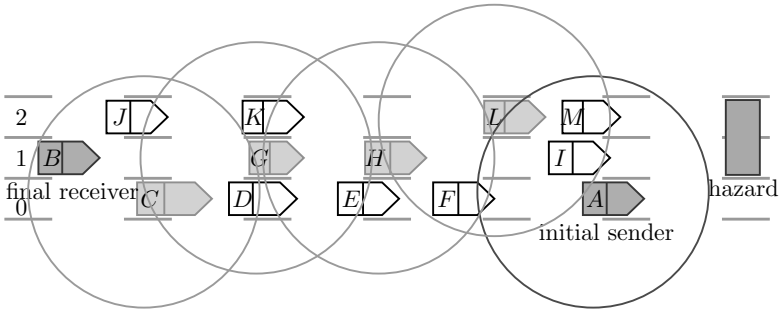

Figure 7.1: Traffic situation with a hazard to the right. The initial sender, car A, detects a hazard on lanes $L_{\lightning} = \{1, 2\}$ and sends a timely warning message via the communication chain $\overrightarrow{c} = \langle A, L, H, G, C, B\rangle$ with intermediate cars L, H, G, C to the final receiver, which is car B.

The first car to approach and perceive a hazardous situation, we call *initial sender* A. In the previous chapters, we considered safety in the sense of collision freedom. The overall safety goal in this chapter is that A transmits a hazard warning as fast as possible to a specific *final receiver* B driving somewhere behind A. We assume that car B is about to reach the hazard in t time units. The safety goal is missed if B reaches the hazard without receiving a hazard warning before t time units after it has occurred. In that case B may not be able to initiate braking or another emergency manoeuvre preventing it from colliding with the hazard (e.g., leaving the highway or changing to a lane without a hazardous situation). Note that we do not consider how such an emergency manoeuvre is conducted, but concentrate on sending timely warnings. Also note that initial sender A sends the warning even if it itself is not affected by the hazard because it is driving on a non-hazardous lane (cf. Fig. 7.1).

The hazard warning is communicated via a broadcast channel *hazard* through a *communication chain* comprising other traffic participants (cf. Fig. 7.1). Following the approach from [MFF15], we assume every car to have the same communication technology and thus the same *communication radius* r. With this, we can determine the minimal amount of cars needed to build a communication chain and send the hazard warning from initial sender A to final receiver B. In the following, we assume that a communication chain between initial sender and final receiver can always be established. Additionally, we assume that the cars in the communication chain $\overrightarrow{c}$ drive spatially behind one another.

Traffic snapshot. We recall the traffic snapshot from [HLOR11] that we presented in Sect. 2.2. In [HLOR11], the focus was on safety of lane change manoeuvres, where the authors showed that reserved spaces of different cars are mutually exclusive. Note that while claimed spaces and acceleration were of some interest in [HLOR11] (cf. Sect. 2.2), they are not in the focus of this case study. This is because we do not consider lane change or overtaking in this chapter.

Instead, we now consider a new feature in the abstract model of road traffic: a *hazard*. Intuitively, we think of a hazard as a space of rectangular shape on a multi-lane road that, from a certain moment on, blocks several adjacent lanes. To this end, we modify the traffic snapshot from Def. 1, Sect. 2.4 (p. 15) by a component *haz* with three attributes:

- a Boolean attribute $haz.on$ ranging over $\{0, 1\}$ and indicating whether the hazard is present ($haz.on = 1$) or not,
- an attribute $haz.lanes$ representing the set of lanes affected by the hazard, which is a contiguous subset of $\mathbb{L}$, and
- an attribute $haz.ext$ representing a fixed horizontal extension of the hazard, which is an interval $[haz.start, haz.end] \subseteq \mathbb{R}$ with $haz.start < haz.end$.

Thus, the *traffic snapshot* now is a structure $\mathcal{TS} = (pos, spd, res, haz)$ which besides *haz* comprises the functions pos, spd, res:

- $pos : \mathbb{I} \to \mathbb{R}$ such that $pos(C)$ is the position of car C along the lanes,
- $spd : \mathbb{I} \to \mathbb{R}$ such that $spd(C)$ is the current speed of the car C,
- $res : \mathbb{I} \to \mathbb{P}(\mathbb{L})$ such that $res(C)$ is the set of lanes C reserves.

As motivated before, we do not consider lane change and acceleration in this chapter. Due to that we omit the functions *clm* and *acc* that are included in the traffic snapshot definition of original MLSL.

Traffic snapshot transitions. We recall the *transitions*, which are used to model changes of the traffic snapshot from Def. 3, p. 16. In such a transition a hazard may occur (by switching $haz.on$ from 0 to 1) and remain present, i.e., $haz.on = 1$ is a stable

predicate. The transition where a hazard occurs is given for a traffic snapshot $\mathcal{TS} = (pos, spd, res, haz)$ with

$$\mathcal{TS} \xrightarrow{haz.on} \mathcal{TS}' \Leftrightarrow \mathcal{TS}' = (pos, spd, res, haz') \text{ and } haz = 0 \wedge haz' = 1.$$

We assume that a hazard occurs passively, e.g. an environment could trigger this transition. Further on, we adapt the time transition, where cars will move, i.e., their position will increase, and they may change their speed. As we consider stationary hazards, a hazard cannot move when time passes. For a time constant $t \in \mathbb{R}_{\geq 0}$, the *time transition* is adapted as follows:

$$\begin{aligned}\mathcal{TS} \xrightarrow{t} \mathcal{TS}' \Leftrightarrow \mathcal{TS}' = (pos', spd', res', haz') \text{ and} \\ \forall C \in \mathbb{I}\colon pos'(C) \geq pos(C) \text{ and } res' = res \text{ and } haz' \geq haz,\end{aligned}$$

where $haz' \geq haz$ abbreviates $haz'.on \geq haz.on$, $haz'.lanes = haz.lanes$ and $haz'.ext = haz.ext$. This means that a hazard which is present in haz remains present in haz', but it does not change its position and size when time is passing.

During a time transition, each car C continues to move, formalised by its increasing position ($pos'(C) \geq pos(C)$), but it does not change its reserved lanes ($res' = res$). Note that the speed may change in an unconstrained manner. However, for collision freedom a distance controller will have to adapt the speed so that a sufficient distance is kept to the cars ahead and thus the reserved spaces remain disjoint.

MLSL with hazards. Recall that the basic MLSL for highway traffic consists of atoms, propositional connectives, quantifiers over car variables and the two spatial chop operators. We now extend the syntax of MLSL from [HLOR11] (cf. Def. 6, p. 21) by a new atom hz representing a hazard at the logical level. The resulting logic we call *Hazard Warning Multi-lane Spatial Logic (HMLSL).*

Definition 53 (HMLSL syntax)**.** *The syntax of the* Hazard Warning Multi-lane Spatial Logic HMLSL *is defined as*

$$\phi ::= true \mid u = v \mid free \mid re(c) \mid hz \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \exists c\colon \phi_1 \mid \phi_1 \frown \phi_2 \mid \begin{smallmatrix}\phi_2\\ \phi_1\end{smallmatrix},$$

where $c, u, v \in \text{Var}$. *We denote the set of all HMLSL formulae by* $\Phi_{\mathbb{H}}$.

The new atom hz expresses that a space is occupied by a hazard. We accordingly extend the semantics of MLSL from Def. 7, p. 21, to define in which circumstances a traffic snapshot, a view and a variable valuation satisfy a given formula with the new atom hz.

Definition 54 (Semantics of hz)**.** *The* satisfaction *of the atom hz with respect to a traffic snapshot* $\mathcal{TS}$*, a view* $V = (L, X, E)$*, and a valuation* ν *of the variables with* $\nu(ego) = E$ *is defined as follows:*

$$\begin{aligned}\mathcal{TS}, V, \nu \models hz \;\Leftrightarrow\; |L| = 1 \text{ and } \|X\| > 0 \text{ and} \\ hz.on = 1 \text{ and } L \subseteq hz.lanes \text{ and } X \subseteq hz.ext.\end{aligned}$$

Note that the atom hz holds only in one lane. To express that a hazard holds in several lanes, the vertical chop operator can be used. Often, we want to express that there is *no* hazard in the considered space. With the extended logic HMLSL, we can specify *hazard safety* with the following formula:

$$\mathit{Safe\text{-}hz} \equiv \neg\exists c : \langle re(c) \wedge hz \rangle \,. \tag{7.1}$$

Formula (7.1) means that no car has some overlap of its reserved space with a hazard. The *hazard detection controller* we introduce in Sect. 7.2 uses the following formula as a transition guard:

$$\langle re(ego) \rangle \frown \langle hz \rangle \,. \tag{7.2}$$

Formula (7.2) specifies that in front of the ego car (more precisely, of the car that the variable ego currently evaluates to in the considered valuation ν) there is a hazard. By the semantics of the two somewhere operators, the hazard needs not be on the same lane that ego is driving on. This is e.g. the case for car A in Fig. 7.1.

7.2 The Hazard Warning Communication Protocol

We first give an intuition for the communication concept for the hazard warning approach and after that present details on the hazard warning controllers.

Communication Concept

In our abstract model, the autonomous cars can be understood as nodes in a *Vehicular ad hoc network* (VANET) [SD14], without a fixed wireless infrastructure and without taking roadside units into account. Following the approach in [MFF15], an instantaneous transitive bridge relay between the initial sender node A and the final receiver node B is required, once a hazard is detected. This bridge relay we call *communication chain* and formalise it as a finite sequence $\overrightarrow{c}$ of cars, the first one being the initial sender and the last one being the final receiver (cf. Fig. 7.1).

The warning message is distributed via a broadcast channel *hazard*, where only cars contained in the communication chain actively forward the message. This approach avoids flooding the message by all traffic participants and thus avoids network overload. Note that due to the use of broadcast channels, all the other cars – even while not involved in the communication chain – receive the hazard warning, but they need not react to it. With this, all cars between initial sender and final receiver will be warned and can react to the hazard by e.g. (emergency) braking or changing to a non-hazardous lane. However, in this chapter we focus on hazard safety of the cars in communication chain $\overrightarrow{c}$, particularly final receiver B.

Communication chain. There are several results on how to calculate an optimal communication chain. We refer to the approach of Claypool and Kannan [CK01], where the authors introduce the concept of *Selective Flooding* for improved *Quality-of-Service Routing*. This approach precomputes routes between all communication nodes, based on static "snapshots" of the topology, which resemble our traffic snapshots $\mathcal{TS}$. These precomputed routes are then stored in a routing table. Whenever one node requests a transitive communication link to another node, the optimal route between those nodes is estimated by flooding control packages through the precomputed routes. The authors furthermore propose a combination of Selective Flooding and *Source Routing* to cope with network topology changes, like moving cars in our case.

In our approach, we assume that whenever one car forwards the warning message, it listens on channel *hazard* if the next car in the communication chain really forwards the message in some time bound t_w. Therefore, the cars have to stay in communication range until the message is successfully forwarded. In [SDD16], several methods are presented for *Cluster-based Routing Protocols* in VANETs. In these hierarchical protocols, a *cluster head* is obligated to communicate with the other nodes in his cluster to maintain the cluster formation. Additionally, the authors describe how routes from a source to a destination node can be established via Cluster-based Routing Protocols, which again is interesting for generating our communication chain $\overrightarrow{c}$.

We use the Z notation for sequences for the communication chain. Recall the functions on Z sequences we introduced in the preliminary Sect. 2.1, where $\mathsf{head}\ s$ returns the first element of a non-empty sequence s, $\mathsf{second}\ s$ returns the second element and $\mathsf{tail}\ s$ returns the part that follows the first element of s.

Hazard warning controllers

For the hazard warning controllers, we again use ACTA with communication as introduced in Chapter 4. On detecting a hazard, a warning message is sent via a broadcast channel *hazard*. The corresponding output action is $\mathit{hazard}!\langle L_{\lightning}, \overrightarrow{c}\rangle$, where $L_{\lightning}$ is the set of lanes affected by the hazard and $\overrightarrow{c}$ is the communication chain, comprising unique car identifiers. With this output action the current values $\nu(L_{\lightning})$ resp. $\nu(\overrightarrow{c})$ of these two data variables are sent over broadcast channel *hazard*.

For synchronisation with this output, consider a related input action $\mathit{hazard}?\langle L, \overrightarrow{d}\rangle$: $\mathsf{head}\ \overrightarrow{d} = \text{ego}$ in another automaton. There, we first store the received values in local variables L and $\overrightarrow{d}$, such that $\nu(L) = \nu(L_{\lightning})$ and $\nu(\overrightarrow{d}) = \nu(\overrightarrow{c})$. This input action synchronises with the given output action only if the HMLSL formula $\mathsf{head}\ \overrightarrow{d} = \text{ego}$ evaluates to true, that is, if the first element of the communicated chain $\nu(\overrightarrow{d}) = \nu(\overrightarrow{c})$ agrees with the value of ego.

For the actual hazard warning controllers, we adapt the formula (7.1) to

$$\mathit{Safe\text{-}hz}(\text{ego}) \equiv \neg \langle \mathit{re}(\text{ego}) \wedge \mathit{hz} \rangle\,. \tag{7.3}$$

Formula (7.3) states *hazard safety* for the ego car, meaning that there does not exist a spatial overlap of ego's reservation with a hazard anywhere in the considered view. The main goal of our hazard warning controllers is to ensure that *Safe-hz*(ego) is never violated. We prove this spatial hazard safety for our controllers later in Sect. 7.3.

In the locations of the controllers, we employ the invariant

$$\mathcal{I}_h \;\equiv\; b \rightarrow \mathit{Safe\text{-}hz}(\text{ego}),$$

where b is a Boolean variable that is set to the value true if the controller detects a hazard or receives a hazard warning message from another car. Informally, this means that whenever a car has knowledge about a hazard, it avoids colliding with it. Thus a kind of distance controller sensitive to hazards is part of the controllers. In the second part of Sect. 7.3, we prove with the assistance of UPPAAL that the hazard warning will arrive in time so that the car can react to it (timely hazard warning).

For the implementation of the communication protocol, we assume every car to be equipped with two controllers. The first one is a *hazard detection controller* that detects the hazard, determines the communication chain, and sends the initial hazard warning message. The second controller is a *forwarding controller* that is used to forward the message to all cars in reach of its communication radius. In order to send a hazard warning message, we use the channel *hazard* to send and receive hazard warnings as described previously. In the following, we will refer to the initial sender as car A and to the final receiver as car B.

In timed automata, transitions may be taken immediately if guards and invariants allow it. For the sake of reality, we assume a communication not to happen immediately, but to take some positive upper time bound $t_c > 0$ to take hardware limitations into account. We assume all cars are equipped with the same communication technology and therefore use the same time bound t_c for all cars.

Hazard detection controller. The constructed hazard detection controller $\mathcal{A}_{det}$ is depicted in Fig. 7.2 and its construction is explained in the following, starting from the initial location.

If the HMLSL formula $\langle re(\text{ego})\rangle \frown \langle hz\rangle$ evaluates to *true*, the hazard detection controller of initial sender A initiates the sending of the hazard warning message by changing from its initial location q_0 to q_1. While doing so, A determines the set of affected lanes using the function *affected_lanes*() and stores them in a set named L_{λ}. By definition of our abstract model, we have *affected_lanes*() = *haz.lanes* (cf. Sect. 7.1). On the same transition, an optimal communication chain is calculated with the function *com_chain*() and stored in $\overrightarrow{c}$. For details about deriving the communication chain we again refer to the beginning of this section. The derived communication chain contains the car identifier A of initial sender as first entry and the car identifier B of final receiver as last entry. If we cannot establish a communication chain, *comm_chain*() returns $\overrightarrow{c} = \langle A\rangle$ and the controller changes back to its initial location, because there is no car to warn in reach ($\#\overrightarrow{c} \leq 1$).

By setting the value of the Boolean control variable b to *true*, we remember whether a warning message was already sent. With this we avoid unlimited resending of the warning message and thus unnecessary flooding of channel *hazard*. As soon as the hazard is detected, until the next car in the communication chain forwards the warning message, initial sender A is obligated to maintain the communication link to the next car in chain $\overrightarrow{c}$.

After t_c time units, the hazard detection controller sends the initial warning message to the next car in $\overrightarrow{c}$ over broadcast channel *hazard*, along with the affected lanes $L_{\lightning}$ and the communication chain $\overrightarrow{c}$ and changes to location q_2. The identifier of the next car in $\overrightarrow{c}$ is stored in a variable *next*. The controller then listens on channel *hazard* to ensure that the forwarding controller of the next car really resends the message. If the forwarding controller does not forward the message within a (reasonable short) time bound t_w, the hazard detection controller changes back to location q_1 and repeats the warning message. Note that one could implement a continuous warning message sending – as proposed for common *decentralised environmental notification messages* – by assuming a time bound $t_w = 0$ (cf. ETSI standard 102 637-3 [ETS10]).

If the controller receives the forwarded message from the next car in $\overrightarrow{c}$, the warning was successful and it changes back to its initial location q_0.

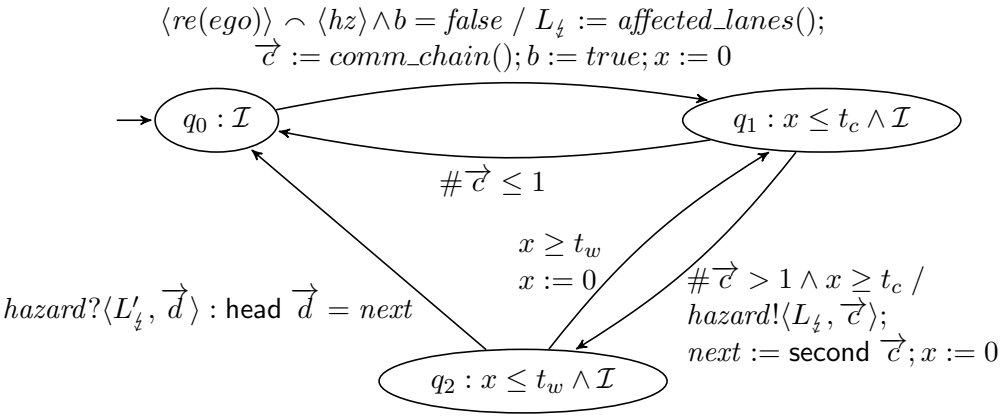

Figure 7.2: Hazard detection controller $\mathcal{A}_{det}$ for detecting a hazard and starting the hazard warning communication chain.

Forwarding controller. The constructed forwarding controller $\mathcal{A}_{for}$ is depicted in Fig. 7.3 and we explain its construction in the following, as usual starting from the initial location.

The forwarding controller copies part of the behaviour of the hazard detection controller. The main difference is the transition from r_0 to r_1, where the forwarding controller does not detect a hazard, but listens on channel *hazard* whether a hazard warning message is forwarded. If a warning message is received, the forwarding controller of

a car synchronises with the sender only if its car identifier is the second entry in the communication chain. With this behaviour we prevent that every car that listens on channel *hazard* resends the warning message.

If a car is second in the communication chain $\overrightarrow{c}$, the forwarding controller synchronises with the sender. Furthermore, we remove the first element of the communication chain $\overrightarrow{c}$, so that the car identifier of the active forwarding controller now is the first element of the resulting shortened communication chain $\overrightarrow{c}' = \mathsf{tail}\ \overrightarrow{c}$. If $\overrightarrow{c}'$ contains more than one element, the forwarding controller sends the new communication chain via channel *hazard*. The following behaviour is the same as that from the hazard detection controller; the forwarding controller listens whether the next car in chain again forwards the warning.

Provided the newly derived communication chain only contains the identifier of the current car ($\#\overrightarrow{c}' \leq 1$), the controller confirms that the message was delivered and changes back to the initial location. Note that this is only the case if the current car is the final receiver.

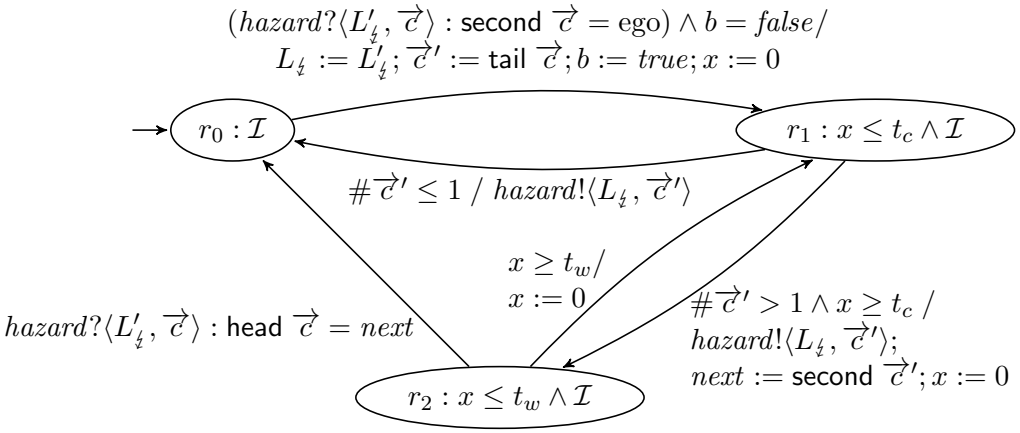

Figure 7.3: Forwarding controller $\mathcal{A}_{for}$ for forwarding hazard messages.

7.3 Analysis of the Protocol and Proof of Hazard Safety

In this section, we start with briefly explaining the implementation of the hazard warning controllers in UPPAAL in Sect. 7.3.1. After that we conduct two proofs in Sect. 7.3.2: The first proof shows that the warning messages are always sent timely and the second proof shows spatial hazard safety for all cars.

7.3.1 Implementation in UPPAAL

For our implementation in UPPAAL we conduct the following two adaptations of our hazard detection controller $\mathcal{A}_{det}$ and forwarding controller $\mathcal{A}_{for}$ to the type of extended timed automata UPPAAL uses. We distinguish these adapted automata from the controllers introduced in Sect. 7.2 by naming the detection controller by `DET` and the forwarding controller by `FOR`. As before, we name the first car to perceive the hazard by the car identifier A and the car which is supposed to receive the timely warning message by B (cf. Fig. 7.1).

With UPPAAL, we verify the timely behaviour of our automata and therefore abstract from the spatial aspects. We do not consider the affected lanes L_{ℓ} in our UPPAAL implementation because it is not relevant for the timely forwarding process and only interesting for manoeuvres that cars could conduct to avoid the hazard. Also, we do not need the spatial invariant $\mathcal{I}$ described in Sect. 7.2 and hence neither the Boolean variable b.

Setting and results. In our implementation, we successfully performed several system executions with N cars for different values for N with $2 \leq N \leq 100$. Following [MFF15] our goal was that a warning message from A is delivered to B in at most t time units, where $t = 100$, as proposed there. In the simulation with UPPAAL each car i owns a detection controller `DET(i)` and a forwarding controller `FOR(i)`. Additionally, we introduce an environment and two observer controllers needed for the verification in the following paragraphs. The overall number of UPPAAL timed automata for every execution is thus $2 \cdot N + 3$. Each one of the verification properties introduced later was checked in less than 0.1 seconds with a memory usage peak each time less than 85 KB on a normal work station. Note that only those automata are actively interacting with each other through the broadcast channel *hazard* that are neighbouring in the communication chain $\overrightarrow{c}$ (e.g., the fourteenth car in $\overrightarrow{c}$ is only answering a forwarding request from the thirteenth car). In the following paragraphs, we explain the adaptations of our controllers to UPPAAL and the additional controllers needed for the verification as well as the verification properties.

HMLSL formulae. The hazard detection controller $\mathcal{A}_{det}$ (cf. Fig. 7.2) is using the formula $\langle re(\text{ego})\rangle \frown \langle hz\rangle$ as a guard at the transition from q_0 to q_1. Instead of using an HMLSL formula – which is not available in UPPAAL – for hazard detection, we introduce an additional automaton `Environment` that places and removes hazards. On placing a hazard, the Environment informs controllers via a broadcast channel `att`(ention) about the existence of a hazard and additionally sets $\mathsf{init\_id} = A$. Initially, all detection controllers are in location q_0 and listen on channel `att`. As every of those controllers additionally checks the guard $\mathsf{id} == \mathsf{init\_id}$, only the detection controller `DET` of initial sender A synchronises with this output. The resulting detection controller `DET` for the implementation in UPPAAL is depicted in Fig. 7.4.

Sending data over channels. Sending data over channels is not provided by UPPAAL,

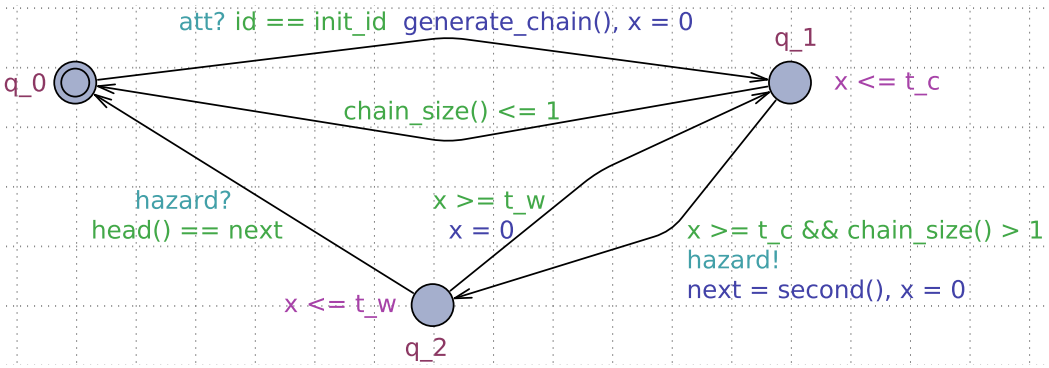

Figure 7.4: Adapted UPPAAL detection controller DET.

therefore we cannot pass our communication chain from car to car. But a distinction between local variables, only accessible and changeable by one specific automaton, and global variables, accessible and changeable by all automata in the system, is available. However, by using a global variable for the communication chain $\overrightarrow{c}$, we have to restrict the access to it.

Consider again the forwarding controller $\mathcal{A}_{for}$ from Fig. 7.3. On sending a hazard warning message (transition from location r_1 to r_2), the controller remembers the second element of $\overrightarrow{c}$ in a local variable *next*. The controller in the next car in $\overrightarrow{c}$ synchronises with this output (transition from its location r_0 to r_1), and removes the head of $\overrightarrow{c}$ on the same transition with the function *tail* $\overrightarrow{c}$. This is consistent with our definition of sending data, because only a local version $\overrightarrow{c}'$ of $\overrightarrow{c}$ is shortened and sent later. However, with a global communication chain in our UPPAAL implementation, next is likely to be valuated with the wrong element, because write and read access for $\overrightarrow{c}$ is uncontrolled.

We overcome this problem in our forwarding controller FOR by simply separating the function tail $\overrightarrow{c}$ from the transition from r_0 to r_1. For this, we introduce a new intermediate location r_im between r_0 and r_1. The location r_im is committed, so no interleaving transitions from any other automata is allowed until r_im was left. On the transition from r_0 to r_m, we keep the following constructs: the input message on the channel *hazard*, all guards and also the reset operation for clock the x. Then, on the transition from r_im to r_1, the global communication chain is shortened with tail $\overrightarrow{c}$. Since no synchronisation happens on this transition, there is no read access on the global communication chain at the same time.

Note that besides the initial generation of $\overrightarrow{c}$ in the first detection controller through function generate_chain(), only the function tail() manipulates the communication chain, by removing its first element. The functions head(), second() and chain_size() only return the respective elements or the current size of $\overrightarrow{c}$. Thus, there is no need to separate these functions from their respective transitions.

Because of the committed location r_im, the special case where the current forwarding controller is located in final receiver B, and thus $\#\overrightarrow{c} \leq 1$, is handled slightly differently in our UPPAAL implementation. Particularly, no transition from r_1 to r_0 exists, but the behaviour of that transition is implemented within the function second(): if the communication chain contains only one element, second() returns 0 and the additional guard id! = 0 ensures that the transition is not taken. In detail, this means that the last controller in $\overrightarrow{c}$ takes that transition once, namely when its identifier is the second in $\overrightarrow{c}$. This ensures that its predecessor receives the confirmation that the message was sent successfully. With an additional transition vom r_2 to r_0, it is ensured that the last controller does not wait in location r_2 infinitely long for a confirmation, as no next sender exists. The adapted forwarding controller FOR for the implementation in UPPAAL is given in Fig. 7.5.

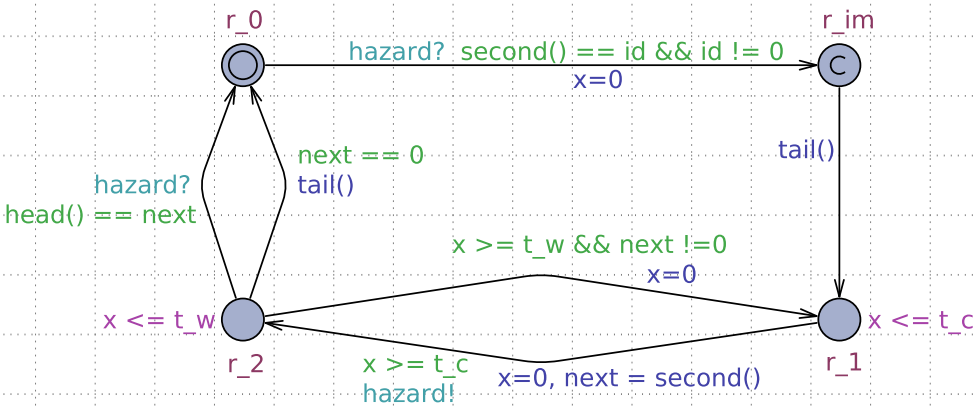

Figure 7.5: Adapted forwarding controller FOR for the UPPAAL implementation with an additional committed location r_im.

Observer Automata and Verification.

We pursue two verification goals with UPPAAL. To this end, we introduce automata Observer1 resp. Observer2. Remember that the overall goal of our approach is a hazard warning message delivery before the final receiver B reaches the hazard after t time units and that we assume one single communication to take t_c time units.

Following [MFF15], we set $t := 100$ and assume one communication to take $t\_c := 1$ time units due to hardware restrictions (cf. Sect. 7.2). With these assumptions, the hazard warning is supposed to be timely delivered if at most $N = 100$ cars are considered for the communication chain and the warning delivery is finished before t time units. This fits the maximal amount of cars that were used in the simulation in [MFF15].

Observer1 (end-to-end message delivery). This observer checks the end-to-end latency of the warning message delivery between the initial sender and the final receiver. The time bound t introduced in Sect. 7.1 we use as a failure time bound in `Observer1`. If t is exceeded, `Observer1` changes to a distinct bad location `Observer1.fail` and our timely message delivery verification goal is missed. `Observer1` monitors the following three events in the given order:

1. The environment places a hazard (change to location `Observer1.hz_on`).
2. The detection controller of the initial sender (first element in $\overrightarrow{c}$) sends the initial warning message (change to location `Observer1.warning_sent`).
3. The forwarding controller of the final receiver (last element in $\overrightarrow{c}$) confirms that it received the warning message
(change to location `Observer1.warning_received`).

The automaton `Observer1` is depicted in Fig. 7.6.

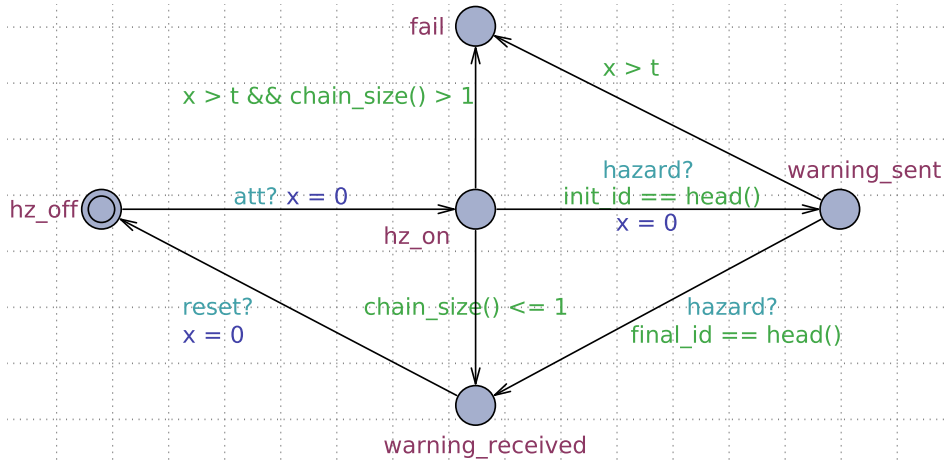

Figure 7.6: `Observer1` monitors that when initial sender A forwards the hazard warning, car B finally receives the message in less than t time units, where t is the time in which car B would arrive at the hazard and our timely warning goal would be missed.

`Observer1` enters Observer1.fail if either 2. does not occur in less than t time units, or if 2. occurred timely, but 3. was not reached in less than t time units. This would be the case if the final receiver reaches the hazard without receiving a warning. Note that the transition from warning_received to hz_off is used for resetting the controller to its initial location.

For a system consisting of N controllers `DET`, N controllers `FOR` and one entity of `Observer1` we examined the following requirements with UPPAAL for $N \leq 100$. Here, the symbol `->` is the leads-to operator in the logic of UPPAAL (cf. Sect. 6.1, p. 112).

Unreachability of fail I: `A[] not Observer1.fail`

Liveness I: `Observer1.hz_on ->`
`(Observer1.warning_sent or chain_size <= 1))`

Liveness II: `Observer1.warning_sent ->`
`(Observer1.warning_received and Observer1.x <= t)`

All three queries hold and were verified within seconds.

Observer2 (observing message delivery between two arbitrary cars). This observer checks the timely forwarding of the hazard warning message between two consecutive cars in the communication chain. In `Observer2` we use the time bound `t_c` for one single communication as failure time bound. If a communication is not resent in less than `t_c` time units, the bad location `Observer2.fail` is entered. The automaton monitors the following three events in the given order:

1. Wait for a hazard warning (by idling in location `Observer2.wait`).
2. A hazard warning of an arbitrary forwarding controller n is received (change to location `Observer2.listening`).
3. Forwarding controller $n + 1$ forwarded the warning (and thus changes to location `Observer2.success`).

We depict `Observer2` in Fig. 7.7.

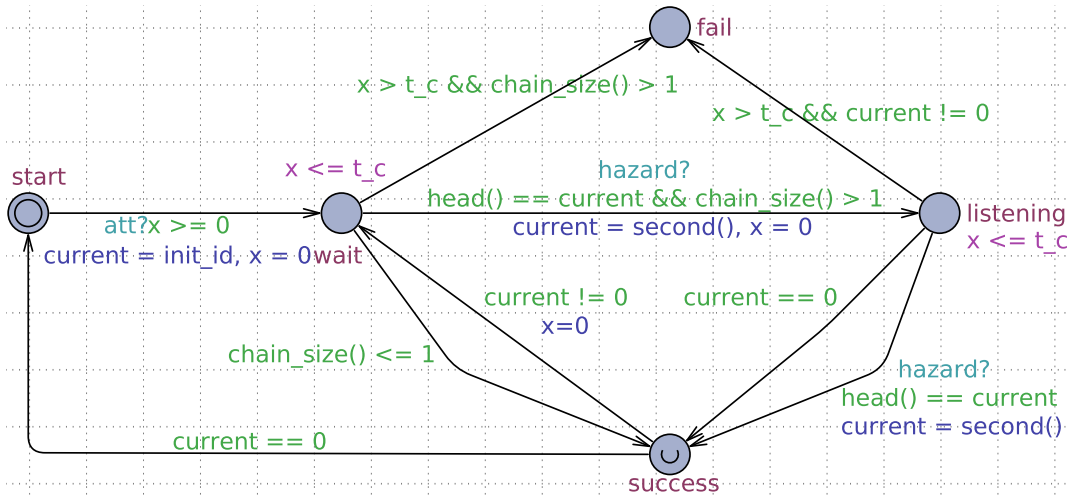

Figure 7.7: `Observer2` monitors that whenever a car in the communication chain $\overrightarrow{c}$ forwards a warning message, the next car in $\overrightarrow{c}$ really reforwards this message in less than `t_c` time units. This verification result is used in our proof by induction for the timely hazard warning safety property.

Note that `Observer2` monitors not only one hazard warning procedure, but listens if every single warning message itself is resent timely by the next automaton in communication chain $\overrightarrow{c}$. We use this timed liveness property later in our inductive proof in Sect. 7.3.2. The fail location `Observer2.fail` is entered, if a single communication is not forwarded in less than `t_c` time units.

We consider the last forwarding as a special case: If forwarding controller $n+1$ belongs to the last car B in $\overrightarrow{c}$, `Observer2` does not change back to `Observer2.wait`, because not further forwarding messages will occur. For a system consisting of N controllers `DET`, N controllers `FOR`, N controllers `Observer2` and one entity of `Observer1`, we examine the following requirements with UPPAAL for $N \leq 100$:

Unreachability of fail II: `A[] not Observer2.fail`

Liveness III: `Observer2.wait -> Observer2.success`

Both queries were successfully verified within seconds. We use the verification results of our implementation in UPPAAL from this section to prove the timely message propagation in the following section.

7.3.2 Proof of Timely Warning and Hazard Safety

In this section we stipulate that every car is equipped with a hazard detection controller $\mathcal{A}_{det}$ and a forwarding controller $\mathcal{A}_{for}$ as introduced in Sect. 7.2. We now prove that warned cars do not collide with the hazard. The safety proof is divided into two steps.

Firstly, we show a timing property: whenever a hazard is detected by a car A, the final receiver B in a communication chain from A to B is warned within some time bound depending on the length of the chain. This proof is supported by the model checker UPPAAL. Secondly, we link the timing property to a spatial property: when the established time bound is below the time it takes car B to reach the hazard then we can guarantee hazard safety of B in the sense that it satisfies the spatial property

$$\mathit{Safe\text{-}hz}(\mathrm{ego}) \;\equiv\; \neg \langle \mathit{re}(\mathrm{ego}) \wedge \mathit{hz} \rangle \,,$$

i.e., the reserved space of B does not overlap with the hazard.

Timely Hazard Warning Message Propagation

We begin with the timing property.

Theorem 3 (Timely warning)**.** *Suppose a communication chain $\overrightarrow{c} = \langle A, ..., B \rangle$ of length $N \geq 2$ is built up after a hazard detection by the initial car $\overrightarrow{c}(1) = A$ in the chain. Then a hazard warning message is received by the final car $\overrightarrow{c}(N) = B$ in the chain within $(N-1) \cdot t_c$ time after it has occurred.*

Proof. We show by induction over i that

($*$) for every $i \in \{1, ..., N\}$ a hazard warning message is received by car $\overrightarrow{c}(i)$ within $(i-1) \cdot t_c$ time after it has occurred, and if $i < N$ holds, the hazard warning message is forwarded to car $\overrightarrow{c}(i+1)$ at most t_c time later.

Induction basis: $i = 1$. By construction of $\mathcal{A}_{det}$, the initial car $\overrightarrow{c}(1) = A$ senses the hazard immediately, i.e., within 0 time after it has occurred, and forwards it to $\overrightarrow{c}(2)$ at most t_c time later.

Induction step: $i \to i+1$, where $i+1 \leq N$. By induction hypothesis, the hazard warning message is received by car $\overrightarrow{c}(i)$ within $(i-1) \cdot t_c$ time after it has occurred, and car $\overrightarrow{c}(i)$ forwards it at most t_c time later. This communication is instantaneously received by the next car $\overrightarrow{c}(i+1)$ in the chain. Thus altogether the hazard warning is received by car $\overrightarrow{c}(i+1)$ within $i \cdot t_c$ time after it has occurred. By construction of $\mathcal{A}_{for}$, if $i+1 < N$ holds, car $\overrightarrow{c}(i+1)$ forwards the hazard warning at most t_c time later.

We checked the induction step for fixed values in our implementation of the controllers with UPPAAL. We refer to `Observer2` which monitors the properties **Unreachability of fail II** and **Liveness III** (cf. Sect. 7.3.1). There we showed for a communication chain $\overrightarrow{c}$ with $N = 100$ and a fixed constant t_c that if an arbitrary car $\overrightarrow{c}(i)$ with $i < N$ receives a hazard warning message, this message is really resent to the next car $\overrightarrow{c}(i+1)$ and that this communication takes less than t_c time units: The location `Observer2.fail` is entered, iff one single communication exceeds t_c time units. Unreachability of `Observer2.fail` proves that indeed no single communication exceeds t_c time units. The property **Liveness III** verifies that whenever an arbitrary element $\overrightarrow{c}(i)$ receives a warning, it is successfully forwarded to $\overrightarrow{c}(i+1)$.

As we can only verify our properties in UPPAAL for a fixed and finite amount of cars N, a forwarding exception is the special case $\overrightarrow{c}(N) = B$, where B is the final receiver and therefore last element in $\overrightarrow{c}$. In this case, the message is not forwarded, because the communication goal is reached. From ($*$) the statement of the theorem follows. □

As described, for the induction step we used the properties monitored by `Observer2`, which observes if one single timely message forwarding process from an arbitrary car $\overrightarrow{c}(i)$ to the next car $\overrightarrow{c}(i+1)$ is successful. Additionally to that, we derived interesting verification results from `Observer1`, which monitors the overall timely message sending from initial sender A to final receiver B. In several iterations, we showed for various values of N with $2 \leq N \leq 100$ that a warning message from A indeed is delivered to B in at most t time units, where again $t = 100$, as proposed in [MFF15]. With `Observer1`, the property **Liveness I** verifies that if A has knowledge of the hazard, it actually sends the initial warning and **Liveness II** verifies that that message is finally received by B. **Unreachability of fail I** shows that the overall message sending happens in at most t time units.

Avoidance of Hazard Collisions

We now turn to the spatial property. For the following safety theorem, we state the following assumptions:

A1. $\mathcal{TS}_0$ is the traffic snapshot where the hazard first occurred. In $\mathcal{TS}_0$ all cars satisfy the property *Safe-hz*.

A2. Car A is closest to the hazard and detects it in $\mathcal{TS}_0$, thereby building up a communication chain $\overrightarrow{c} = \langle A, ..., B\rangle$ of length $N \geq 2$ to car B.

A3. Car B needs t time to reach the hazard and during this time it satisfies the property *Safe-hz*.

A4. For the time bound t_c used in the controllers $\mathcal{A}_{det}$ and $\mathcal{A}_{for}$ the inequality $(N-1)\cdot t_c \leq t$ holds.

Theorem 4 (Hazard safety)**.** *Suppose the assumptions* **A1**–**A4** *hold. Then in every traffic snapshot $\mathcal{TS}^*$ that is reachable from $\mathcal{TS}_0$ via time transitions car B satisfies the property Safe-hz(ego) (under the valuation $\nu(\text{ego}) = B$).*

Proof. Note that by **A2**, Theorem 3 is applicable. Let $\mathcal{TS}^*$ be reachable from $\mathcal{TS}_0$ via time transitions. Then $\mathcal{TS}_0 \xrightarrow{t^*} \mathcal{TS}^*$ for some time $t^* \in \mathbb{R}_{\geq 0}$. If in $\mathcal{TS}^*$ car B has not yet received the hazard warning message sent by car A via the communication chain $\overrightarrow{c}$, we know by Theorem 3 and **A4** that $t^* < (N-1)\cdot t_c \leq t$ holds. Thus by **A3**, car B satisfies *Safe-hz*(ego) in $\mathcal{TS}^*$.

If in $\mathcal{TS}^*$ car B has received the hazard warning message via its controller $\mathcal{A}_{for}$, this controller guarantees *Safe-hz*(ego) from the moment on that the hazard warning message has first been received by B, say in the traffic snapshot $\mathcal{TS}_1$. Thus we can split the time t^* into $t^* = t_1 + t_2$ such that

$$\mathcal{TS}_0 \xrightarrow{t_1} \mathcal{TS}_1 \xrightarrow{t_2} \mathcal{TS}^*,$$

where $t_1 \leq (N-1)\cdot t_c \leq t$ due to Theorem 3 and **A4**. Then car B satisfies *Safe-hz*(ego) in $\mathcal{TS}^*$ by the invariant of its controller $\mathcal{A}_{for}$. □

By a similar argument, we can extend the above theorem and show that every car in the communication chain $\overrightarrow{c}$ satisfies the property *Safe-hz*(ego).

Outlook (cars outside of $\overrightarrow{c}$)

So far, we only prove hazard safety for the cars in communication chain $\overrightarrow{c}$, because only those synchronise with a hazard warning (cf. transition in $\mathcal{A}_{for}$ from r_0 to r_1). However, with our broadcast communication we can also reach all other cars not involved in the communication chain. We therefore assume the communication radius r to stretch over all lanes $\mathbb{L}$ and to have a positive extension along the lanes. For formalisation, we refer to the definition of a view from Section 3.4, as the communication radius of a car can be considered to be a *communication view* V_C. Only cars inside the communication view $V_C(E)$ of a warning car E can synchronise with E.

For cars outside $\overrightarrow{c}$ to synchronise with a warning in their communication view, we add a transition in the forwarding controller $\mathcal{A}_{for}$. The new transition leads from initial location r_0 to a new location r_4, where the invariant $\mathcal{I}$ is required to hold (cf. Section 7.2). The new transition gets the guard

$$(\mathit{hazard}?\langle L_{\lambda}, \overrightarrow{c}\rangle : \mathit{not\text{-}element}(\overrightarrow{c}, \text{ego}) \wedge b = \mathit{false}$$

and the variable update $b := \mathit{true}$. The function $\mathit{not\text{-}element}(\overrightarrow{c}, \text{ego})$ evaluates to true if the car identifier that ego evaluates to is not included in $\overrightarrow{c}$.

Chapter summary. In this chapter, we formalised the timing aspects of the simulation-based analysis of a communication protocol for timely traffic hazard warning to other traffic participants from [MFF15] by using our automotive-controlling timed automata (ACTA) (cf. Chapter 4). With this formalisation we prove the timely warning property, partly supported by the model checker UPPAAL [BDL04].

Also, we linked the timely warning property with spatial reasoning to prove avoidance of hazard collision using a new extension of the Multi-lane Spatial Logic MLSL [HLOR11] dealing with hazards, called HMLSL. Formulae of this logic appear in the guards and invariants of ACTA to establish the desired spatial safety properties.

8 Conclusion

We split our conclusion into four parts. First, we summarise the results of this thesis in Sect. 8.1. After that, we informally evaluate our results w.r.t real-world demands in Sect. 8.2. The goal of this evaluation is to analyse the degree of abstraction of our approach and to illustrate possibilities for approaching a more realistic approach in the future. For this, we briefly summarise some of the assumptions that we already weakened within this thesis and we point out possible directions for weakening further assumptions. After that, we give a very brief overview over a topic we started working on recently in Sect. 8.3, and finally conclude with some directions for future work in Sect. 8.4.

8.1 Summary

The innovation of our approach is the level of abstraction from car dynamics to merely spatial reasoning for safety properties in urban traffic scenarios. In [HLOR11] and [HLO13] this was shown for motorways and country roads. We were able to build on these settings, by introducing methodologies for an extension to urban traffic for both the abstract model and the controllers. Additionally, we introduced a formal semantics for the traffic controllers with ACTA and examined different system properties for the controllers. We introduced a case study on a hazard-warning communication protocol to further strengthen our work and show its flexibility.

For the extension of the abstract model, we started with adding new topological features to the abstract model; an urban road network allowing intersections where arbitrary numbers of lanes meet and in which cars follow infinite paths. We included this new urban road network into the traffic snapshot definition from [HLOR11] and also introduced sanity conditions for both the topology and the new urban traffic snapshot. We then introduced virtual multi-views, to cope with the bended views for the different turn manoeuvres at intersections. We also added a representation of crossing segments to Multi-lane Spatial Logic and defined the evaluation of UMLSL formulae over the extended urban traffic snapshot and our virtual multi-views.

Furthermore, we constructed a crossing controller with new controller actions for crossing manoeuvres (crossing claim and crossing reservation). With the syntax and semantics of automotive-controlling timed automata (ACTA), we presented a generic type of automaton to formally implement our crossing controller as well as the lane change and overtaking controllers from [HLOR11] and [HLO13] and other traffic controllers. We also

gave an overview over the different parts of (dynamic and discrete) control within one autonomous system and delimited where our crossing controller can be placed in into that bigger picture. To weaken some of our assumptions, we also introduced a version of our crossing controller which uses communication to cope for an imperfect knowledge.

Next, we turned towards desirable system properties for both our crossing controller as well as for the highway traffic lane change controller from [HLOR11]. The properties we considered where safety, (bounded) liveness and fairness. For this, we proved safety of our crossing controller with a mathematical proof, using an induction over all possible reachable traffic snapshots starting from an arbitrary initial safe traffic snapshot. For showing the other system properties, we used an implementation of our abstract model and controllers in UPPAAL SMC. With this, we were able to detect and resolve unlive behaviour of both the lane change and the crossing controller. We improved the controllers and showed (bounded) liveness for both controllers. We also introduced a communication concept with priorities to the crossing controller, with which we were able to show its fairness. Again to weaken some of the strong assumptions we make, we introduced uncertain communication to the crossing controller.

We complete our work with the hazard warning case study for highway traffic, for which we include hazards into the model and logic for highway traffic and introduce hazard warning controllers. We show the timely hazard warning with a mathematical proof by induction assisted with verification results obtained for an implementation of our controllers in UPPAAL SMC.

8.2 Evaluation – How Realistic is our Approach?

As indicated before, some of the assumptions for our model that we give in Sect. 3.1, as well as the assumptions for our controller that we give in Sect. 5.2, are quite strong and lead to a high level of abstraction of our approach. However, abstraction from some details is necessary if one strives to use formal methods for purely logical reasoning about a system [CGL94]. Further on, if we can prove a result on a high level of abstraction, a next step can be to weaken some of the strong assumptions that were made, to achieve a more realistic approach.

We summarise which assumptions we already weakened within this thesis and give ideas for further possibilities of approaching scenarios closer to reality with our approach.

We weakened the assumption about having perfect knowledge about perceiving the braking distances of other cars (cf. sensor function Ω_E from Sect. 3.4.3, p. 59). This was done by introducing a crossing controller with imperfect knowledge in Sect. 5.4, which only perceives the physical size of cars within their safety envelopes with Ω_E. This controller copes for the imperfect knowledge by using communication. Of course, that notion of 'imperfect' knowledge results in a model, where still a large amount of knowledge is assumed and a high level of abstraction exists. However, using communication, we could

also compensate other types of lack of knowledge. For instance, we could assume a more realistic sensor function Ω_E, where also sensor failure is considered. With this, a car that is not able to perceive sufficient information for its reasoning process could commit traffic manoeuvres with the help of other cars or road-side units.

The second assumption that we weakened is the assumption about having 100% reliable communication. For this, we introduced uncertain communication to the fair crossing controller $\texttt{CRP}_\texttt{F}$ in Sect. 6.4.4 by introducing PACTA. Unsurprisingly, with our notion of uncertain communication, the considered liveness and fairness properties now only hold with a certain probability. However, safety remains invariantly valid. One goal for future work could be to investigate more thoroughly what realistic values are for how lossy and unreliably channels actually are. For this, results from the Vehicle-2-Vehicle or Vehicle-2-X community are of importance [SD14].

Further on, we could apply uncertainties to more of the MLSL traffic scenarios. For instance, in the country roads approach [HLO13] (cf. Sect. 2.4, p. 29), communication with a helper car is used to safely overtake other cars. On using uncertain communication channels in that approach, one could analyse how the system properties change.

Also, we previously sketched that sensor uncertainty is a topic worthwhile investigating. We could use PACTA not only for introducing uncertainty to communication channels to the MLSL controllers, but also to formalise uncertainty of the sensor function Ω_E from Sect. 3.4.3. Whenever an autonomous car detects uncertain sensor information, it could e.g. communicate with an appropriate *sensor helper controller* for additional information to complete a driving manoeuvre.

Communication could also be used if sensors are blocked by objects. For instance, consider the traffic situation depicted in Fig. 8.1. There, the view of car E is blocked by an obstacle. However, communication should be unaffected by the obstacle. Nonetheless, if we assume communication failure, cars E and F might not perceive each other in time. For this, car D can operate as a helper car and share its sensors' information with cars E and F.

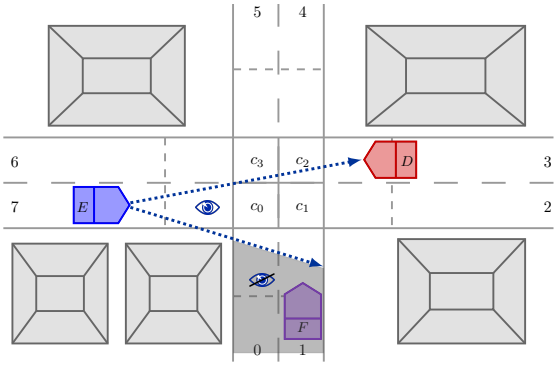

Figure 8.1: Intersection where obstacles restrict the view of car E.

One of the strongest assumptions that we have is to consider that all cars drive autonomously and use the controllers that we introduce in Sect. 5.1. This assumption is provided in Sect. 5.2 and assumed to hold for showing our system properties in Chapter 6. Weakening this assumption would mean to consider unknown vehicles, which are either human-controlled vehicles or vehicles using an (autonomous) driving technology different from our approach. On facing such an unknown vehicle, we cannot know which driving rules and driving plans the vehicle follows. However, several approaches exist for estimating the possible driving trajectories of unknown vehicles. Taking the estimated trajectories into account, our controllers could adapt their traffic manoeuvres accordingly. However, with only estimated trajectories of other cars, our system properties – including safety – may only hold with a certain probability. An approach on including human-controlled vehicles into the highway traffic approach from [HLOR11] was sketched in [Bis18].

8.3 Recent Work: Explainability

A more recent topic that we started working on is the *explainability of cyber-physical systems (CPS)*. We did not include it in this thesis, as it is a topic beyond the scope of this thesis and hitherto rather a vision than a thoroughly elaborated theory. The goal of the work is that increasingly complex cyber-physical systems should gain the ability of *self-explaining* their past, current and future behaviour. We claim that explainability of CPS is essential to increase confidence and trust between the user and the system, as well as to enhance possibilities of collaboration between different CPS. With that, in our case, a system's explanation is not only user-centred but also includes internal explanations that can be provided to other systems. We introduce details on our *Monitor, Analyse, Build, Explain* (MAB-EX) framework in [BGG+19]. MAB-EX is based on the *Monitor, Analyse, Plan, Execute* (MAPE) loop for self-adaptive systems from IBM [Sin06] and includes the following four phases:

- *Monitor* system data, the environment and the recipient of explanations (user, other system,...),
- *Analyse* the monitored data to detect the need for an explanation,
- *Build* an explanation by evaluating an internal model of the system (explanation model), and
- *Explain* the behaviour in question to the recipient.

In case of the autonomous urban traffic controllers from this thesis, there are several observations that could be explained. For instance, if another car passes the intersection before the own car, it could be explained that the other car had a larger priority for passing the intersection. Equally, a car could estimate the time needed to pass an intersection and pass this information to its driver or another car. Finally, in case of a traffic accident involving an autonomous car, it would be useful if the car can provide information about the cause of the accident, e.g. for legal and insurance questions.

8.4 Directions for Future Work

Besides the approaches already sketched in the evaluation in Sect. 8.2, we see several more possibilities to extend the work presented in this thesis.

Virtual view construction. In Sect. 3.4 on p. 57, we observed that with our view construction, we loose vertical spatial information in the resulting virtual views, caused by the different numbers n, m of lanes meeting at an intersection. Additionally, we informally introduced placeholder lanes for one-way lanes in Remark 1, p. 56. We could overcome the loss of vertical spatial information by filling up n-by-m intersections with fake lanes upon creating a virtual view. This would result in an x-by-x intersection with only one distinct virtual view again. However, we would have to introduce fake crossing segments, too, to determine the connections of real lane segments with fake lanes. It is non-trivial to determine where the fake lanes and fake crossing segments should be added to the topology.

Decidability. In Sect. 1.1, p. 2, we describe related work [Lin15, Ody15] concerning the undecidability of (robust) satisfiability of MLSL. This undecidability result also applies for our UMLSL from Sect. 3.5. However, we also describe the approach from [FHO15] in Sect. 1.1, where the authors prove that MLSL is decidable if only cars within some specific scope are considered. It surely would be interesting to examine the applicability of the results from [FHO15] to our approach with urban traffic.

Proof system. A proof system for an extension of MLSL including length measurement and modal operators was proposed in [Lin15]. This proof system could be extended for UMLSL to prove (safety) properties of our controllers by deduction. However, our urban traffic approach is a lot more complex than the highway traffic approach, due to the more complex topology and view construction. Thus, an extension of the proof system would be complex and success is hardly foreseeable.

Controllers. An interesting point for future work concerning the controllers is to connect and combine the existing approaches (highway traffic, urban traffic, ...) and controllers (e.g. lane change controller, crossing controller, ...). As a vision, one could think of one controller on top, coordinating the more specialised controllers. Further on, controllers for currently not covered scenarios could be constructed using ACTA (e.g. parking controller, highway-exit controller, ...).

Implementation. It is of interest to compare our UPPAAL implementation for highway traffic from Sect. 6.2 and our implementation for urban traffic from 6.4 with other approaches regarding its efficiency. For this, in the highway traffic case, a distance controller should be integrated into the approach. However, we refer back to Sect. 5.1, where we gave details why this is a complex task. Nonetheless, one optimisation for the highway traffic case could be to include that different models are automatically checked as initial traffic snapshots $\mathcal{TS}_0$.

For the urban traffic case, an implementation allowing more complex intersections than the 2-by-2 crossing used for our UPPAAL model in Sect. 6.4 might be helpful. A more complex priority system where cars can increase their priority even if they are not directly in front of the crossing might help to increase the efficiency of the fairness protocol further. Using such a system, it might be reasonable to examine the efficiency of urban road networks instead of single intersections. Another point for future work is an optimisation of our controllers, to allow for more than 2 resp. 3 cars for the time-unbounded verification (cf. Sects. 6.4.2 and 6.4.3).

Hazard warning. In Chapter 7, we consider only highway traffic scenarios. A linkage of the hazard warning approach with the country roads approach from [HLO13] and the urban traffic approach from this thesis is highly interesting for future work. In case of country roads, the hazard warning message must be sent in the two driving directions, using two communications chains (cf. Fig. 8.2). In urban traffic, the message possibly needs to be sent in even more directions, depending on the location of the hazard.

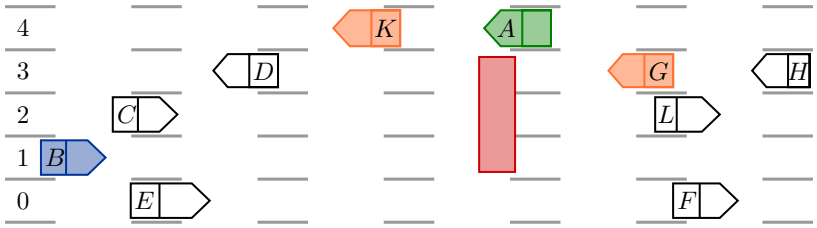

Figure 8.2: Extension of the hazard warning approach to country roads, where a hazard may stretch over lanes with different driving directions.

Optimisation of traffic flow. Although this has not been in the scope of this thesis, there are several starting points for an optimisation of traffic flow in our proposed urban traffic scenario. For instance, one could consider including a concept of platooning [LGS98, LUS12] (cf. Sect. 1.3, p. 7). If for instance two cars, one driving behind the other, are approaching an intersection and both plan to conduct the same crossing manoeuvre, they could conduct the traffic manoeuvre together as a temporary platoon. For the availability of such a platoon, it is possible to take heavily trafficked streets into account, as building a platoon is more likely on those kinds of roads. The sketched procedure is e.g. described in [NB04, LW06] (cf. Sect. 1.3, p. 7). In both approaches, cars use communication to build virtual platoons on approaching an intersection. In future work, we could examine how to include approaches like these into our urban traffic approach.

More dynamical reservation of crossing segments. Further on, the crossing controller $\mathcal{A}_{cc}$ that we introduced in Sect. 5.3 reserves all needed crossing segments for its manoeuvre at once, instead of one segment after the other. With this, it is for instance not possible that two cars simultaneously turn left in front of each other. Surely, traffic flow may be optimised if we reserve the needed segments more dynamically one after the other. However, this is deadlock-prone, so this adaptation requires an additional mechanism to prevent deadlocks.

Bibliography

[ACD90] Rajeev Alur, Costas Courcoubetis, and David L. Dill. Model-checking for real-time systems. In *5th IEEE Symposium on Logic in Computer Science, Proc.*, pages 414–425, 1990.

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[AD14] Matthias Althoff and John M. Dolan. Online verification of automated road vehicles using reachability analysis. *IEEE Transactions on Robotics*, 30(4):903–918, 2014.

[ADNL+15] Yehia Abd Alrahman, Rocco De Nicola, Michele Loreti, Francesco Tiezzi, and Roberto Vigo. A calculus for attribute-based communication. In *30th ACM Symp. on Applied Computing, Proc.*, pages 1840–1845. ACM, 2015.

[AM16] Matthias Althoff and Silvia Magdici. Set-based prediction of traffic participants on arbitrary road networks. *IEEE Trans. Intelligent Vehicles*, 1(2):187–202, 2016.

[AMB+12] Mikael Asplund, Atif Manzoor, Mélanie Bouroche, Siobhàn Clarke, and Vinny Cahill. A formal approach to autonomous vehicle coordination. In Dimitra Giannakopoulou and Dominique Méry, editors, *International Symposium on Formal Methods, Proc.*, pages 52–67. Springer, 2012.

[ANT12] Mikael Asplund and Simin Nadjm-Tehrani. Worst-case latency of broadcast in intermittently connected networks. *International Journal on Ad Hoc Ubiquitous Computing*, 11(2/3):125–138, 2012.

[AS10] Tsz-Chiu Au and Peter Stone. Motion planning algorithms for autonomous intersection management. In *AAAI Workshop on Bridging The Gap Between Task And Motion Planning BTAMP, Proc.*, pages 2–9. AAAI Press, 2010.

[Asp18] Mikael Asplund. Automatically proving the correctness of vehicle coordination. *ICT Express*, 4:51–54, 2018.

[ASS11] Tsz-Chiu Au, Neda Shahidi, and Peter Stone. Enforcing liveness in autonomous traffic management. In *25th Conference on Artificial Intelligence, Proc.*, pages 1317–1322. AAAI Press, 2011.

[BDH+19] Yougang Bian, Jieyun Ding, Manjiang Hu, Qing Xu, Jianqiang Wang, and Keqiang Li. An advanced lane-keeping assistance system with switchable assistance modes. *IEEE Transactions on ITS*, pages 1–12, 2019.

[BDL04] Gerd Behrmann, Alexandre David, and Kim G. Larsen. A tutorial on uppaal. In Marco Bernardo and Flavio Corradini, editors, *Formal Methods for the Design of Real-Time Systems*, pages 200–236. Springer, 2004.

[BGG+19] Mathias Blumreiter, Joel Greenyer, Francisco Javier Chiyah Garcia, Verena Klös, Maike Schwammberger, Christoph Sommer, Andreas Vogelsang, and Andreas Wortmann. Towards self-explainable cyber-physical systems. In *22nd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion*, pages 543–548, 2019.

[BHLO17] Gregor v. Bochmann, Martin Hilscher, Sven Linker, and Ernst-Rüdiger Olderog. Synthesizing and verifying controllers for multi-lane traffic maneuvers. *Formal Aspects of Computing*, 29(4):583–600, 2017.

[Bis18] Cristopher Bischopink. Moving hazards - reasoning about humand drivers in autonomous traffic. Master's thesis, University of Oldenburg, 2018.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

[Bra] Technische Universität Braunschweig. Project stadtpilot. https://www.tu-braunschweig.de/stadtpilot. Project webpage, accessed: 11/26/2019.

[Brä11] Torben Bräuner. *Hybrid Logic and its Proof-Theory*. Springer, 2011.

[BS19] Christopher Bischopink and Maike Schwammberger. Verification of fair controllers for urban traffic manoeuvres at intersections. In Emil Sekerinski, Nelma Moreira, José N. Oliveira, Daniel Ratiu, Riccardo Guidotti, Marie Farrell, Matt Luckcuck, Diego Marmsoler, José Campos, Troy Astarte, Laure Gonnord, Antonio Cerone, Luis Couto, Brijesh Dongol, Martin Kutrib, Pedro Monteiro, and David Delmas, editors, *Formal Methods. FM 2019 International Workshops - Porto, Portugal, October 7-11, 2019, Revised Selected Papers, Part I*, volume 12232 of *Lecture Notes in Computer Science*, pages 249–264. Springer, 2019.

[BTK07] Kostas E. Bekris, Konstantinos I. Tsianos, and Lydia E. Kavraki. A decentralized planner that guarantees the safety of communicating vehicles with complex dynamics that replan online. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, Proc.*, pages 3784–3790. IEEE, 2007.

[CDV12] Alessandro Colombo and Domitilla Del Vecchio. Efficient algorithms for collision avoidance at intersections. In *15th ACM Int. Conf. on Hybrid Systems: Comp. and Control, HSCC, Proc.*, pages 145–154. ACM, 2012.

[CE82] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs, Workshop*, pages 52–71. Springer, 1982.

[CE16] Lei Chen and Cristofer Englund. Cooperative intersection management: A survey. *IEEE Trans. on Int. Transportation Systems*, 17(2):570–586, 2016.

[CGL94] Edmund M. Clarke, Orna Grumberg, and David E. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems*, 16(5):1512–1542, 1994.

[CGP99] Edmund M. Clarke, Jr., Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, 1999.

[CHR91] Zhou Chaochen, C. A. R. Hoare, and Anders P. Ravn. A calculus of durations. *Information Processing Letters*, 40(5):269–276, 1991.

[CK01] Mark Claypool and Gangadharan Kannan. Selective flooding for improved quality-of-service routing. In *Int. Symp. on Convergence of IT and Comm., ITCom, Proc.*, pages 33–44. Int. Society for Optics and Photonics, 2001.

[Com11] European Commission. White paper: Roadmap to a single European transport area – towards a competitive and resource efficient transport system. Technical Report COM(2011) 144 final, 2011.

[Cou18] European Transport Safety Council. Ranking eu progress on road safety – 12th road safety performance index report. Technical report, 2018.

[Cou19] European Transport Safety Council. Road safety priorities for the EU 2020-2030 – briefing for the Europ. parliamentary elections. Technical report, 2019.

[CRT09] Alessandro Cimatti, Marco Roveri, and Stefano Tonetta. Requirements validation for hybrid systems. In Ahmed Bouajjani and Oded Maler, editors, *21st International Conference on Computer Aided Verification, CAV, Proc.*, pages 188–203. Springer, 2009.

[CSB+17] Rajashekar Chandra, Yuvaraj Selvaraj, Mattias Brännström, Roozbeh Kianfar, and Nikolce Murgovski. Safe autonomous lane changes in dense traffic. In *IEEE 20th Int. Conf. on ITS, Proc.*, pages 1–6. IEEE, 2017.

[DB16] Murat Dikmen and Catherine M. Burns. Autonomous driving in the real world: Experiences with tesla autopilot and summon. In *8th International Conference on Automotive User Interfaces and Interactive Vehicular Applications (AutomotiveUI), Proc.*, pages 225–228. ACM, 2016.

[DCDVL17] Eric Dallal, Alessandro Colombo, Domitilla Del Vecchio, and Stéphane Lafortune. Supervisory control for collision avoidance in vehicular networks using discrete event abstractions. *Discrete Event Dynamic Systems*, 27(1):1–44, 2017.

[DFWB12] Louise A. Dennis, Michael Fisher, Matthew P. Webster, and Rafael H. Bordini. Model checking agent programming languages. *Automated Software Engineering*, 19(1):5–63, 2012.

[DH01] Werner Damm and David Harel. LSCs: Breathing life into message sequence charts. *Formal Methods in System Design*, 19(1):45–80, 2001.

[DHO06] Werner Damm, Hardi Hungar, and Ernst-Rüdiger Olderog. Verification of cooperating traffic agents. *Int. Journal of Control*, 79(5):395–421, 2006.

[Dij71] Edsger W. Dijkstra. Hierarchical ordering of sequential processes. *Acta Informatica*, 1(2):115–138, 1971.

[DJL+15] Alexandre David, Peter Gjøl Jensen, Kim G. Larsen, Marius Mikučionis, and Jakob H. Taankvist. Uppaal stratego. In Christel Baier and Cesare Tinelli, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 9035 of *LNCS*, pages 206–211. Springer, 2015.

[DLL+15] Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikučionis, and Danny Bøgsted Poulsen. Uppaal SMC tutorial. *STTT*, 17(4):397–415, 2015.

[DMPR18] Werner Damm, Eike Möhlmann, Thomas Peikenkamp, and Astrid Rakow. A formal semantics for traffic sequence charts. In Marten Lohstroh, Patricia Derler, and Marjan Sirjani, editors, *Principles of Modeling - Essays Dedicated to Edward A. Lee on the Occasion of His 60th Birthday*, pages 182–205. Springer, 2018.

[DMR14] Werner Damm, Eike Möhlmann, and Astrid Rakow. Component based design of hybrid systems: A case study on concurrency and coupling. In *17th International Conference on Hybrid Systems: Computation and Control, HSCC, Proc.*, pages 145–150. ACM, 2014.

[DPRW13] Werner Damm, Hans-Jörg Peter, Jan Rakow, and Bernd Westphal. Can we build it: formal synthesis of control strategies for cooperative driver assistance systems. *Math. Structures in Comp. Science*, 23(4):676–725, 2013.

[DSB16] Shweta N. Dethe, Varsha S. Shevatkar, and R. P. Bijwe. Google driverless car. *Int. Journal of Scientific Research in Science, Engineering and Technology IJSR SET*, 2:133 –137, 2016.

[EHK+17] Andreas B. Eriksen, Chao Huang, Jan Kildebogaard, Harry Lahrmann, Kim G. Larsen, Marco Muniz, and Jakob H. Taankvist. Uppaal Stratego for intelligent traffic lights. In *12th ITS European Congress, Proc.*, 2017.

[Elm01] Wilfried Elmenreich. An introduction to sensor fusion. Research Report 47/2001, TU Wien, Institut für Technische Informatik, 2001.

[ERSO12] European Road Safety Observatory. *Basic Fact Sheet "Junctions"*, 2012.

[ETS09] ETSI. *Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Definitions (ETSI TR 102 638 V1.1.1)*, 2009.

[ETS10] ETSI. *Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 3: Specifications of Decentralized Environmental Notification Basic Service (ETSI TS 102 637-3 V1.1.1)*, 2010.

[FFCa+10] Michel Ferreira, Ricardo Fernandes, Hugo Conceição, Wantanee Viriyasitavat, and Ozan K. Tonguz. Self-organized traffic control. In *7th ACM Int. Workshop on VehiculAr InterNETworking, VANET, Proc.*, pages 85–90. ACM, 2010.

[FHO15] Martin Fränzle, Michael R. Hansen, and Heinrich Ody. No need knowing numerous neighbours. In Roland Meyer, André Platzer, and Heike Wehrheim, editors, *Correct System Design: Symposium in Honor of Ernst-Rüdiger Olderog on the Occasion of His 60th Birthday, Proc.*, volume 9360 of *LNCS*, pages 152–171. Springer, 2015.

[FJM+01] Lino Figueiredo, Isabel S. Jesus, José A. Machado, José R. Ferreira, and J.L. Martins de Carvalho. Towards the development of intelligent transportation systems. *Intelligent Transportation Systems*, 88:1206–1211, 2001.

[FVP+13] Negin Fathollahnejad, Emília Villani, Risat Pathan, Raul Barbosa, and Johan Karlsson. On reliability analysis of leader election protocols for virtual traffic lights. In *43rd IEEE/IFIP Conference on Dependable Systems and Networks Workshop, DSN-W*, pages 1–12, 2013.

[Gro02] ERTMS User Group. *ERTMS/ETCS System requirements specification*, 2002.

[Har] Mark Harris. The 2,578 problems with self-driving cars. `https://spectrum.ieee.org/cars-that-think/transportation/self-driving/the-2578-problems-with-self-driving-cars`. IEEE Spectrum, last checked-out on 20.11.2019.

[HCvS06] Luc C. G. J. M. Habets, Pieter J. Collins, and Jan H. van Schuppen. Reachability and control synthesis for piecewise-affine hybrid systems on simplices. *IEEE Transactions on Automatic Control*, 51(6):938–948, 2006.

[Hei16] Dirk Heinrichs. Autonomous driving and urban land use. In Markus Maurer, J. Christian Gerdes, Barbara Lenz, and Hermann Winner, editors, *Autonomous Driving: Technical, Legal and Social Aspects*, pages 213–231. Springer, 2016.

[HLO13] Martin Hilscher, Sven Linker, and Ernst-Rüdiger Olderog. Proving safety of traffic manoeuvres on country roads. In Zhiming Liu, Jim Woodcock, and Huibiao Zhu, editors, *Theories of Programming and Formal Methods – Essays Dedicated to Jifeng He on the Occasion of His 70th Birthday*, volume 8051 of *LNCS*. Springer, 2013.

[HLOR11] Martin Hilscher, Sven Linker, Ernst-Rüdiger Olderog, and Anders P. Ravn. An abstract model for proving safety of multi-lane traffic manoeuvres. In Shengchao Qin and Zongyan Qiu, editors, *13th Int. Conference on Formal Engineering Methods, ICFEM, Proc.*, pages 404–419. Springer, 2011.

[HNSY94] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193 – 244, 1994.

[Hoa78] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21:666–677, 1978.

[HS16] Martin Hilscher and Maike Schwammberger. An abstract model for proving safety of autonomous urban traffic. In Augusto Sampaio and Farn Wang, editors, *13th Int. Colloquium on Theor. Aspects of Computing ICTAC, Proc.*, volume 9965 of *LNCS*, pages 274–292. Springer, 2016.

[Int18] SAE International. J 3016: Surface vehicle recommended practice – (r) taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles, 2018.

[IRW+18] Ramón Iglesias, Federico Rossi, Kevin Wang, David Hallac, Jure Leskovec, and Marco Pavone. Data-driven model predictive control of autonomous mobility-on-demand systems. In *IEEE International Conference on Robotics and Automation ICRA, Proc.*, pages 1–7. IEEE, 2018.

[Kal60] Rudolph E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[KDM+17] Maryam Kamali, Louise A. Dennis, Owen McAree, Michael Fisher, and Sandor M. Veres. Formal verification of autonomous vehicle platooning. *Science of Computer Programming*, 148:88–106, 2017.

[KFS13] Savas Konur, Michael Fisher, and Sven Schewe. Combined model checking for temporal, probabilistic, and real-time logics. *Theoretical Computer Science*, 503:61 – 88, 2013.

[Kle99] Lawrence A. Klein. *Sensor and Data Fusion Concepts and Applications*. Society of Photo-Optical Instrumentation Eng. (SPIE), 2nd edition, 1999.

[KLF19] Maryam Kamali, Sven Linker, and Michael Fisher. Modular verification of vehicle platooning with respect to decisions, space and time. In Cyrille Artho and Peter Csaba Ölveczky, editors, *Formal Techniques for Safety-Critical Systems*, pages 18–36. Springer, 2019.

[KNPS04] Marta Kwiatkowska, Gethin Norman, David Parker, and Jeremy Sproston. Performance analysis of probabilistic timed automata using digital clocks. In Kim G. Larsen and Peter Niebert, editors, *Formal Modeling and Analysis of Timed Systems*, pages 105–120. Springer, 2004.

[KRS+12] Tim Köhler, Christian Rauch, Martin Schröer, Elmar Berghöfer, and Frank Kirchner. Concept of a biologically inspired robust behaviour control system. In Chun-Yi Su, Subhash Rakheja, and Honghai Liu, editors, *Intelligent Robotics and Applications*, pages 486–495. Springer, 2012.

[LAB+11] Jesse Levinson, Jake Askeland, Jan Becker, Jennifer Dolson, David Held, Sören Kammel, J. Zico Kolter, Dirk Langer, Oliver Pink, Vaughan R. Pratt, Michael Sokolsky, Ganymed Stanek, David Michael Stavens, Alex Teichman, Moritz Werling, and Sebastian Thrun. Towards fully autonomous driving: Systems and algorithms. In *IEEE Intelligent Vehicles Symposium (IV), Proc.*, pages 163–168. IEEE, 2011.

[Lam77] Leslie Lamport. Proving the correctness of multiprocess programs. *IEEE Transactions on Software Engineering*, 3(2):125–143, 1977.

[LGS98] John Lygeros, Datta N. Godbole, and Shankar Sastry. Verified hybrid controllers for automated vehicles. *IEEE Transactions on Automatic Control*, 43(4):522–539, 1998.

[LH15] Sven Linker and Martin Hilscher. Proof theory of a multi-lane spatial logic. *Logical Methods in Computer Science*, 11(3):1–27, 2015.

[Lin15] Sven Linker. *Proofs for Traffic Safety – Combining Diagrams and Logic.* PhD thesis, University of Oldenburg, 2015.

[Lin17a] Sven Linker. Hybrid multi-lane spatial logic. *Archive of Formal Proofs*, 2017.

[Lin17b] Sven Linker. Spatial reasoning about motorway traffic safety with isabelle/hol. In Nadia Polikarpova and Steve Schneider, editors, *Integrated Formal Methods iFM, Proc.*, pages 34–49. Springer, 2017.

[LLL00] Carolos Livadas, John Lygeros, and Nancy A. Lynch. High-level modeling and analysis of the traffic alert and collision avoidance system (TCAS). *Proceedings of the IEEE*, 88(7):926–948, 2000.

[LMT15] Kim G. Larsen, Marius Mikučionis, and Jakob H. Taankvist. Safe and optimal adaptive cruise control. In Roland Meyer, André Platzer, and Heike Wehrheim, editors, *Correct System Design: Symposium in Honor of Ernst-Rüdiger Olderog on the Occasion of His 60th Birthday*, LNCS, pages 260–277. Springer, 2015.

[LP11] Sarah M. Loos and André Platzer. Safe intersections: At the crossing of hybrid systems and verification. In Kyongsu Yi, editor, *14th Int. IEEE Conf. on Intelligent Transportation Systems (ITSC)*, pages 1181–1186, 2011.

[LPY95] Kim G. Larsen, Paul Pettersson, and Wang Yi. Model-checking for real-time systems. In Horst Reichel, editor, *Fundamentals of Computation Theory*, pages 62–88. Springer, 1995.

[LUS12] Maider Larburu, Arkaitz Urquiza, and Javier Sánchez. Safe road trains for the environment (SARTRE): Validation of SARTRE platoon service and the SARTRE HMI. Technical report, 2012.

[LW06] Li Li and Fei-Yue Wang. Cooperative driving at blind crossings using intervehicle communication. *IEEE Transactions on Vehicular Technology*, 55(6):1712–1724, 2006.

[MC81] Jayadev Misra and Kanianthra M. Chandy. Proofs of networks of processes. *IEEE Transactions on Software Engineering*, SE-7(4):417–426, 1981.

[Mea08] Michael Montemerlo et al. Junior: The stanford entry in the urban challenge. *Journal of Field Robotics*, 25(9):569–597, 2008.

[MFF15] Nils Müllner, Martin Fränzle, and Sibylle Fröschle. Estimating the probability of a timely traffic-hazard warning via simulation. In *48th Annual Simulation Symp. (ANSS), Proc.*, pages 130–137. Society for Computer Simulation Intern., 2015.

[MFHR08] Roland Meyer, Johannes Faber, Jochen Hoenicke, and Andrey Rybalchenko. Model checking duration calculus: A practical approach. *Formal Aspects of Computing*, 20(4–5):481–505, 2008.

[Mil82] Robin Milner. *A Calculus of Communicating Systems*. Springer, 1982.

[MM05] Sven Maerivoet and Bart De Moor. Cellular automata models of road traffic. *Physics Reports*, 419(1):1–64, 2005.

[MMP15] Andreas Müller, Stefan Mitsch, and André Platzer. Verified traffic networks: Component-based verification of cyber-physical flow systems. In *IEEE 18th Int. Conf. on ITS*, pages 757–764, 2015.

[Mos85] Ben Moszkowski. A temporal logic for multilevel reasoning about hardware. *Computer*, 18(2):10–19, 1985.

[MPW92] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, I. *Inf. Comput.*, 100(1):1–40, 1992.

[MRO02] Thomas Moor, Jörg Raisch, and Siu O'Young. Discrete supervisory control of hybrid systems based on l-complete approximations. *Discrete Event Dynamic Systems*, 12(1):83–107, 2002.

[NB04] Norbert Neuendorf and Torsten Bruns. The vehicle platoon controller in the decentralised, autonomous intersection management of vehicles. In *IEEE Int. Conf. on Mechatronics ICM, Proc.*, pages 375–380, 2004.

[NBCF17] Julia Nilsson, Mattias Brännström, Erik Coelingh, and Jonas Fredriksson. Lane change maneuvers for automated vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 18(5):1087–1096, 2017.

[NHO+11] Tobias Nothdurft, Peter Hecker, Sebastian Ohl, Falko Saust, Markus Maurer, Andreas Reschka, and Jürgen R. Böhmer. Stadtpilot: First fully autonomous test drives in urban traffic. In *14th Int. IEEE Conf. on Intelligent Transportation Systems (ITSC)*, pages 919–924, 2011.

[NPW02] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[NRTT97] Rolf Naumann, Rainer Rasche, Jürgen Tacken, and Christoph Tahedl. Validation and simulation of a decentralized intersection collision avoidance algorithm. In *IEEE ITS Conf., Proc.*, pages 818–823, 1997.

[OD08] Ernst-Rüdiger Olderog and Henning Dierks. *Real-time systems - formal specification and automatic verification*. Cambridge University Press, 2008.

[Ody15] Heinrich Ody. Undecidability results for multi-lane spatial logic. In Martin Leucker, Camilo Rueda, and Frank D. Valencia, editors, *ICTAC*, volume 9399 of *LNCS*, pages 404–421. Springer, 2015.

[Ody17] Heinrich Ody. Monitoring of traffic manoeuvres with imprecise information. *Electronic Proceedings in Theoretical Computer Science*, 257:43–58, 2017.

[Ody19] Heinrich Ody. *Monitoring of Traffic Manoeuvres with Imprecise Information*. PhD thesis, University of Oldenburg, 2019. (submitted Nov. 2019).

[OL82] Susan Owicki and Leslie Lamport. Proving liveness properties of concurrent programs. *ACM Trans. on Prog. Lang. and Syst.*, 4(3):455–495, 1982.

[Old18] Ernst-Rüdiger Olderog. Space for traffic manoeuvres: An overview. In Cliff Jones, Ji Wang, and Naijun Zhan, editors, *Symposium on Real-Time and Hybrid Systems: Essays Dedicated to Professor Chaochen Zhou on the Occasion of His 80th Birthday*, pages 211–230. Springer, 2018.

[ORW17] Ernst-Rüdiger Olderog, Anders P. Ravn, and Rafael Wisniewski. Linking spatial and dynamic models, applied to traffic maneuvers. In Mike Hinchey, Jonathan P. Bowen, and Ernst-Rüdiger Olderog, editors, *Provably Correct Systems*, NASA Monographs in System and Software Engineering, pages 95–120. Springer, 2017.

[OS17] Ernst-Rüdiger Olderog and Maike Schwammberger. Formalising a hazard warning communication protocol with timed automata. In Luca Aceto, Giorgio Bacci, Giovanni Bacci, Anna Ingólfsdóttir, Axel Legay, and Radu Mardare, editors, *Models, Algorithms, Logics and Tools – Essays Dedicated to Kim G. Larsen on the Occasion of His 60th Birthday*, volume 10460 of *LNCS*, pages 640–660. Springer, 2017.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *18th Symposium on Foundations of Computer Science, Proc.*, SFCS, pages 46–57. IEEE, 1977.

[PT02] Adrian Perrig and J. D. Tygar. *Secure Broadcast Communication in Wired and Wireless Networks*. Kluwer Academic Publishers, 2002.

[QS82] Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in CESAR. In Mariangiola Dezani-Ciancaglini and Ugo Montanari, editors, *5th Int. Symposium on Programming, Proc.*, pages 337–351. Springer, 1982.

[RIA16] Albert Rizaldi, Fabian Immler, and Matthias Althoff. A formally verified checker of the safe distance traffic rules for autonomous vehicles. In *NASA 8th Int. Symp. on Formal Methods, Proc.*, pages 175–190, 2016.

[Sch05] Andreas Schäfer. A calculus for shapes in time and space. In Zhiming Liu and Keijiro Araki, editors, *Int. Conf. on Theoretical Aspects of Computing (ICTAC), Proc.*, volume 3407 of *LNCS*, pages 463–478. Springer, 2005.

[Sch14] Maike Schwammberger. Semantik von Controllern für sicheren Fahrspurwechsel. Master's thesis, University of Oldenburg, 2014.

[Sch15] Maike Schwammberger. Properties of communicating controllers for safe traffic manoeuvres. In Bernhard K. Aichernig and Alessandro Rossini, editors, *Doctoral Symp. of Formal Methods, Proc.*, pages 3–7, 2015.

[Sch17] Maike Schwammberger. Imperfect knowledge in autonomous urban traffic manoeuvres. *Electr. Proc. in Theor. Comp. Sci.*, 257:59–74, 2017.

[Sch18a] Maike Schwammberger. An abstract model for proving safety of autonomous urban traffic. *Theoretical Computer Science*, 744:143–169, 2018.

[Sch18b] Maike Schwammberger. Introducing liveness into multi-lane spatial logic lane change controllers using UPPAAL. *Electronic Proceedings in Theoretical Computer Science*, 269:17–31, 2018.

[SD14] Christoph Sommer and Falko Dressler. *Vehicular Networking*. Cambridge University Press, 2014.

[SDD16] Dhawale Satyajeet, Atul R. Deshmukh, and Sanjay S. Dorle. Heterogeneous approaches for cluster based routing protocol in vehicular ad hoc network VANET. *Int. Journal of Computer Applications*, 134(12):1–8, 2016.

[Sin06] David Sinreich. An architectural blueprint for autonomic computing. *IBM Autonomic Computing – White Paper*, 2006.

[Spi89] J. Michael Spivey. *The Z Notation: A Reference Manual*. Prentice-Hall, Inc., 1989.

[SS16] Miranda A. Schreurs and Sibyl D. Steuwer. Autonomous driving – political, legal, social, and sustainability dimensions. In Markus Maurer, J. Christian Gerdes, Barbara Lenz, and Hermann Winner, editors, *Autonomous Driving: Technical, Legal and Social Aspects*, pages 149–171. Springer, 2016.

[Tim14] Nils Timm. Spotlight abstraction with shade clustering – automatic verification of parameterised systems. In *Theoretical Aspects of Software Engineering Conference TASE, Proc.*, pages 18–25. IEEE, 2014.

[TWR10] Tobe Toben, Bernd Westphal, and Jan-Hendrik Rakow. Spotlight abstraction of agents and areas. In *Quantitative and Qualitative Analysis of Network Protocols, Dagstuhl Seminar Proc.*, 2010.

[VMDS08] Mark Van Middlesworth, Kurt Dresner, and Peter Stone. Replacing the stop sign: Unmanaged intersection control for autonomous vehicles. In *7th Int. Conf. on Auton. Agents and Multiagent Syst. AAMAS, Proc.*, pages 1413–1416. Int. Found. for Auton. Agents and Multiagent Syst., 2008.

[WB18] David Welch and Elisabeth Behrmann. Who's winning the self-driving car race? `https://www.bloomberg.com/news/features/2018-05-07/who-s-winning-the-self-driving-car-race`, 2018. last checked-out on 25.11.2019.

[WD96] Jim Woodcock and Jim Davies. *Using Z – Specification, Refinement, and Proof*. Prentice Hall, 1996.

[WF08] Jorn M. Wille and Thomas Form. Realizing complex autonomous driving maneuvers the approach taken by team CarOLO at the DARPA urban challenge. In *IEEE Int. Conf. on Vehicular Electronics and Safety, Proc.*, pages 232–236, 2008.

[WHLS15] Hermann Winner, Stephan Hakuli, Felix Lotz, and Christina Singer, editors. *Handbuch Fahrerassistenzsysteme, Grundlagen, Komponenten und Systeme für aktive Sicherheit und Komfort*. Springer, 2015.

[WW07] Björn Wachter and Bernd Westphal. The spotlight principle. In *8th Int. Conf. on Verification, Model Checking and Abstract Interpretation VMCAI, Proc.*, pages 182–198, 2007.

[XL16] Bingqing Xu and Qin Li. A spatial logic for modeling and verification of collision-free control of vehicles. In *21st Int. Conf. on Engineering of Complex Computer Systems (ICECCS)*, pages 33–42, 2016.

[XL17] Bingqing Xu and Qin Li. A bounded multi-dimensional modal logic for autonomous cars based on local traffic and estimation. In *Int. Symposium on Theoretical Aspects of Software Engineering (TASE)*, pages 1–8, 2017.

[XLGD19] Bingqing Xu, Qin Li, Tong Guo, and Dehui Du. A scenario-based approach for formal modelling and verification of safety properties in automated driving. *IEEE Access*, 7:140566–140587, 2019.

Index

List of Symbols

Abstract Model

; Chop operator in ITL logic. 20

$\langle\varphi\rangle$ Somewhere abbr. in MLSL. 22

$\oslash$ Horizontal chop operator for views. 58

$\ominus$ Vertical chop operator for views. 58

$\sqsubset$ Operator for expressing that a finite sequence is contained in an infinite path, e.g. $\overrightarrow{\pi} \sqsubset pth$. 40

$\frown$ Chop operator in MLSL. 20

acc Set of real acceleration values, e.g. $acc(C) = 0$. 15, 45

c_i A crossing segment $c_i \in \mathbb{CS}$ ($i \in \mathbb{N}$). 36

$cclm$ Set of crossing claims, e.g. $cclm(C) = \{c_0, c_1\}$. 45

$cl(c)$ Claim atom (MLSL). 20

clm Set of claims, e.g. $clm(C) = \{0\}$. 15, 44

clm_V Visible claims in a view V. 19

cr_i An intersection $cr \in \mathbb{CR}$. 36

$\mathbb{CR}$ Set of intersections, e.g. cr, cr_1. 36, 42

$cres$ Set of crossing reservations, e.g. $cres(C) = \{c_0, c_1\}$. 45

cs Crossing segment atom (UMLSL). 62

$\mathbb{CS}$ Set of crossing segm., e.g. c_0, c_1. 36

$curr(C)$ Current path index of $pth(C)$. 44

CVar Set of car variables, e.g. c, d. 20

E Set of edges in $\mathcal{N}$, $E = E_u \cup E_d$. 39

$E^{\sim}$ Relational inverse of E. 40

$E_{\mathcal{C}}$ Set of directed edges in $\mathcal{N}_{\mathcal{C}}$. 42

E_d Set of directed edges in $\mathcal{N}$. 39

E_u Set of undirected edges in $\mathcal{N}$. 39

ego Variable for car under consideration. 14, 93

$f_{\mathbb{CS}} \colon \mathbb{CS} \to \mathbb{P}(\mathbb{CS})$ Equivalence class of crossing segments. 42

$f_{\mathbb{L}} \colon \mathbb{L} \to \mathbb{P}(\mathbb{L})$ Equivalence class of lanes. 42

free Free space (MLSL atom). 20

haz Hazard component in abstract model with hazards. 155

haz.ext Horizontal extension of a hazard. 155

haz.lanes Set of lanes a hazard occupies. 155

haz.on Boolean attribute for a hazard. 155

hz HMLSL atom for a hazard. 156

$\mathbb{I}$ Set of car identifiers, e.g. A, B. 14, 37

ℓ Length measurement in EMLSL. 32, 62

L Interval of lanes $L = [l, n]$ in view. 19

$\mathbb{L}$ Set of lane segments, $\mathbb{L} \subset \mathbb{N}$. 14, 36

len_V Visible parts of cars in view V. 19

LVar Set of lane variables, e.g. l, n. 20

$\mathcal{N}$ Urban road network. 39

$\mathcal{N}_{\mathcal{C}}$ Coarse road network. 42

ν Variable valuation. 20

$\mathcal{V}$ Set of nodes in $\mathcal{N}$, $\mathcal{V} = \mathbb{L} \cup \mathbb{CS}$. 39

$\mathcal{V}_{\mathcal{C}}$ Set of nodes in $\mathcal{N}_{\mathcal{C}}$, $\mathcal{V}_{\mathcal{C}} = \mathbb{CR} \cup \mathbb{RS}$. 42

$next(C)$ Next path index of $pth(C)$. 44

Ω_E Sensor function for car E. 19, 59

$\overrightarrow{\pi}$ Sequence from the set $\mathsf{seq}_E\,\mathcal{V}$. 40

$\overrightarrow{\pi_{\mathcal{C}}}$ Sequence from the set $\mathsf{seq}_{E_{\mathcal{C}}}\,\mathcal{V}_{\mathcal{C}}$. 43

φ_M MLSL formula. 21

$\Phi_{\mathbb{M}}$ Set of all MLSL formulae. 21

$\varphi_{\mathbb{U}}$ UMLSL formula. 72

$\Phi_{\mathbb{U}}$ Set of all UMLSL formulae. 72

pos Set of position values for cars, e.g. $pos(C) = 40$. 15, 45

pth Infinite path in $\mathcal{N}$. 40, 45

$pth(C)$ Path of car C. 44

$pth_{\mathcal{C}}$ Infinite path in $\mathcal{N}_{\mathcal{C}}$. 43

$pth_{\mathcal{N}}$ Set of all infinite paths in $\mathcal{N}$. 40, 45

r_i A road segment $r_i \in \mathbb{RS}$ ($i \in \mathbb{N}$). 36

$re(c)$ Reservation of car (MLSL atom). 20

res Set of reservations of cars, e.g. $res(C) = \{1\}$. 15, 44

res_V Visible reservations in view V. 19

$\mathbb{RS}$ Set of road segments, e.g. r_0, r_1. 36, 42

RVar Set of all real variables. 31, 62

$seg_V(C)$ Set of visible segments in view. 59

$\mathsf{seq}\,\mathcal{V}$ Set of sequences over $\mathcal{V}$. 40

$\mathsf{seq}_E\,\mathcal{V}$ Set of sequences respecting edges E over $\mathcal{V}$. 40

$\mathsf{seq}_{E_{\mathcal{C}}}\,\mathcal{V}_{\mathcal{C}}$ Set of sequences respecting edges $E_{\mathcal{C}}$ over $\mathcal{V}_{\mathcal{C}}$. 43

$\mathsf{seq}_{E^{\sim}}\,\mathcal{V}$ Set of inverted sequences respecting edges $E^{\sim}$ over $\mathcal{V}$. 40

$\mathsf{seq}_{E_d}\,\mathcal{V}$ Set of finite sequences respecting only directed edges E_d over $\mathcal{V}$. 40

spd Set of speed values, e.g. $spd(C) = 90$. 15, 45

$\mathcal{TS}$ Traffic snapshot. 15, 44

$\mathbb{TS}$ Set of traffic snapshots. 16, 45

V View $V = (L, X, E)$ (highway traffic). 19

Var Set of all variables. 20, 31

V_C Communication view, comprising several multi-views V_M. 100

V_i Virtual view. 58

V_M Multi-view, comprising several virtual views. 58

ω Weight of nodes in $\mathcal{N}$, e.g. $\omega(v) = 20$. 39

X Extension $X = [r, t]$ of a view. 19

(Autom. Contr.) Timed Automata

$\|$ Parallel comp. operator for (AC)TA. 27, 84

$\mathcal{A}$ (automotive-controlling/ extended) timed automaton. 24, 74, 80

$a?$ Input action for TA. 24

$a!$ Output action for TA. 24

Act Set of all actions for TA. 24

B Set of channels $B \subseteq Chan$. 24, 80

C Set of commited states. 74

$\mathcal{C}$ Configuration of an (AC)TA. 26, 75

c_{act} Controller action. 73

$c_{act,\mathsf{syn}}$ Controller action after synchronisation. 82

$\mathcal{C}_{ini}$ Initial configuration of an (AC)TA. 26, 76

$\mathsf{c}(c, \psi_{\mathbb{D}_{\mathbb{L}}})$ lane claim from set $Ctrl_{Act}$. 73

$\mathsf{cc}(c)$ crossing claim from set $Ctrl_{Act}$. 73

$Chan$ Set of comm. channels. 24

$Conf(\mathcal{A})$ Set of all configurations of an (AC)TA $\mathcal{A}$. 26, 75

$Ctrl_{Act}$ Set of all controller actions. 73

$\mathbb{D}$ Set of data variables. 70

d Data variable from $\mathbb{D}$. 70

$\overrightarrow{d}$ Data sequence of type $\mathsf{seq}\,\mathbb{D}$. 79

$\mathbb{D}_{\mathbb{CS}}$ Set of crossing segment data variables. 70

$\mathbb{D}_{\mathbb{I}}$ Set of car data variables. 70

$\mathbb{D}_{\mathbb{L}}$ Set of lane data variables. 70

E Transition relation of a TA 24

I Identifying tuple $I \in \mathbb{D}_{\mathbb{I}} \cup \{\text{ego}\} \times \mathbb{I}$. 73

$\mathcal{I}$ Set of invariants in an (AC)TA. 24, 74

in Input action for ACTA. 79

IN Set of input actions for ACTA. 79

l A location $l \in L$ of an TA. 24

L Set of locations for TA. 24

l_{ini} Initial location for TA. 24

ν Data/ clock variable valuation. 25, 70

ν_{act} Variable modification. 72

$\nu_{act,\mathsf{syn}}$ Variable modification after synchronisation. 82

ν_{ini} Initial variable valuation. 26, 76

$\mathcal{V}_{Act}$ Set of all variable modifications. 72

out Output action for ACTA. 79

OUT Set of output actions for ACTA. 79

φ (Communication) guard or invariant. 24, 72, 79

Φ Set of guards and invariants. 24, 72

$\varphi_{\mathbb{D}}$ Data constraint from set $\Phi_{\mathbb{D}}$. 71

$\Phi_{\mathbb{D}}$ Set of data constraints $\varphi_{\mathbb{D}}$. 71

$\varphi_{\mathbb{X}}$ Clock constraint from set $\Phi_{\mathbb{X}}$. 23

$\Phi_{\mathbb{X}}$ Set of clock constraints $\varphi_{\mathbb{X}}$. 23

$\psi_{\mathbb{D}}$ Data term, 71

$\Psi_{\mathbb{D}}$ Set of data terms. 71

q Location of an ACTA. 74

Q Set of locations of an ACTA. 74

q_{ini} Initial location of an ACTA. 74

$\mathtt{r}(c)$ lane reservation from set $Ctrl_{Act}$. 73

$\mathtt{rc}(c)$ crossing reservation from set $Ctrl_{Act}$. 73

syn Synchronisation expression. 81

τ Internal, null or empty action. 24, 73, 79

T Transition relation of an ACTA. 74

$\mathcal{T}(\mathcal{A})$ Transition system of $\mathcal{A}$. 26, 76

$Time$ Time dimension for clocks. 23

$\mathtt{wd\,c}(c)$ withdraw lane claim from set $Ctrl_{Act}$. 73

$\mathtt{wd\,cc}(c)$ withdraw crossing claim from set $Ctrl_{Act}$. 73

$\mathtt{wd\,r}(c, \psi_{\mathbb{D}_{\mathbb{L}}})$ withdraw lane reservation from set $Ctrl_{Act}$. 73

$\mathtt{wd\,rc}(c)$ withdraw crossing reservation from set $Ctrl_{Act}$. 73

$\mathbb{X}$ Set of clock variables, e.g. x, y. 23

Controller

$\Box$ Globally operator (temp. logic). 110

$\Diamond$ Finally operator (temp. logic). 110

A Quantifier over paths (CTL). 110

$\mathcal{A}_{cc}$ Crossing controller. 93

$\mathcal{A}'_{cc}$ Crossing controller with communication. 101

$\mathcal{A}^f_{cc}$ Fair crossing controller. 113

$\mathcal{A}_{rc}$ Helper controller. 104

$\mathcal{A}_{lc}$ Lane change controller. 29

$\mathcal{A}_{rc}$ Road controller. 93

Bd_Life_i Bounded liveness property. 111

$\overrightarrow{c}$ Communication chain. 157, 158

$ca(c)$ Crossing ahead check. 94

$\mathtt{CRP}$ UPPAAL implementation of $\mathcal{A}_{cc}$. 137

$\mathtt{CRP'}$ Liveness extension of $\mathtt{CRP}$. 141

$\mathtt{CRP_F}$ Fair extension of $\mathtt{CRP'}$. 143

$\mathtt{CRP_{prob}}$ Version of $\mathtt{CRP}$ with uncertainties. 147

$\mathtt{CRP_F(ego)}$ Controller $\mathcal{A}_{cc}$ for ego car. 143

$col(\text{ego})$ Collision check formula. 30, 94

d_c Constant for distance to crossing. 93

$\mathtt{DET}$ Detection controller (UPPAAL). 162

$dir(c)$ Function returning driving direction of car $\nu(c)$. 101

E Quantifier over paths (CTL). 110

ε Parameter for uncertainty (UPPAAL SMC). 113, 124, 140

$\mathbb{H}$ Set of helper cars' identifiers. 103
FOR Forwarding controller (UPPAAL). 164
hazard Channel for hazard warnings. 158
$\mathtt{HP_F}$ Helper controller for $\mathtt{CRP_F}$. 143
$\mathcal{I}_h$ Invariant in hazard controllers. 159
$lc(c)$ Lane change check. 95
$L_{\lightning}$ Set of lanes affected by a hazard. 158
LCP UPPAAL implementation of $\mathcal{A}_{lc}$. 117
LCP′ Adapted implementation of LCP. 123
LCP″ Liveness extension of LCP′. 126
LCP(i) Controller LCP for car $i \in \{A, B, E\}$. 117
$Life_i$ Liveness property. 111
$ocac(c)$ Opposing car approaching the intersection check. 101
$oc(c)$ On crossing check. 89, 96
$pc(c)$ Potential collision check. 30, 95
$ph(c)$ Potential helper check. 101
$ph^{-1}(c, cs)$ Inverse potential helper check. 104
prio[ego] Broadcast channel for ego for priorities. 143, 146
prior[ego] Priority of controller $\mathtt{CRP_F}$[ego], prior[ego] $\in \mathbb{Z}$. 143
α Parameter for false negatives (UPPAAL SMC). 113, 124, 140
Safe Safety property. 87
r Communication radius. 155
t, t_i Frequently used time constant. 93, 154

Z Notation

$\#$ Length/ size (of a sequence/ set). 40
$\frown$ Sequence concatenation. 13
$\langle\rangle$ Sequence delimiters. 13, 41
$\oplus$ Overriding operator. 12, 16, 48
$\lhd$ Domain restriction. 12, 48
$\nrightarrow$ Partial finite function. 12, 41
$\rightarrowtail$ Total bijective function. 12, 41
dom Domain of a function. 12, 41
head s Function on a sequence s. 13
$\mathbb{P} X$ Power set of a set X. 11, 36
ran Range of a function. 12, 48
second s Function on a sequence s. 13
seq X Finite sequence over a set X. 12, 40
squash Squash operator, defined over $\mathbb{N}$. 13
$\mathsf{squash}_{\mathbb{Z}}$ Squash operator defined over $\mathbb{Z}$. 41
tail s Function on a sequence s. 13
Z Z specification language. 11