Formalising and Monitoring Traffic Rules for Autonomous Vehicles in Isabelle/HOL

Albert Rizaldi[1](✉), Jonas Keinholz[1], Monika Huber[1], Jochen Feldle[2],
Fabian Immler[1], Matthias Althoff[1], Eric Hilgendorf[2], and Tobias Nipkow[1]

[1] Institut für Informatik, Technische Universität München, Munich, Germany
rizaldi@in.tum.de

[2] Forschungsstelle RobotRecht, Julius-Maximilians-Universität Würzburg,
Würzburg, Germany

Abstract. Recent accidents involving autonomous vehicles prompt us to consider how we can engineer an autonomous vehicle which always obeys traffic rules. This is particularly challenging because traffic rules are rarely specified at the level of detail an engineer would expect. Hence, it is nearly impossible to formally monitor behaviours of autonomous vehicles—which are expressed in terms of position, velocity, and acceleration—with respect to the traffic rules—which are expressed by vague concepts such as "maintaining safe distance". We show how we can use the Isabelle theorem prover to do this by first codifying the traffic rules abstractly and then subsequently concretising each atomic proposition in a verified manner. Thanks to Isabelle's code generation, we can generate code which we can use to monitor the compliance of traffic rules formally.

1 Introduction

Formalising law in a logical language is hard. Since the formalisation of the British Nationality Act in Prolog [19], there has yet to be another major breakthrough in the formalisation of law. Even formalising traffic rules for highway scenarios, which seems straightforward on the surface, possesses many challenges. The challenges are not so much representing natural language specifications as logical entities—which we term "codification"—as concretely interpreting predicates such as *overtaking*, *maintaining safe distance*, or *maintaining enough side clearance*—which we term "concretisation". For example, how large is a distance in order to be categorised as safe?

We are mainly motivated to formalise traffic rules for two purposes: (1) holding autonomous vehicles legally accountable; and (2) clarifying requirements for engineering autonomous vehicles. It is necessary that traffic rules are codified in a logical language so that engineers have a clear and well-defined specification

A. Rizaldi—This work is partially supported by the DFG Graduiertenkolleg 1480 (PUMA), DFG NI 491/16-1, European Commission project UnCoVerCPS 643921.

N. Polikarpova and S. Schneider (Eds.): IFM 2017, LNCS 10510, pp. 50–66, 2017.
DOI: 10.1007/978-3-319-66845-1_4

against which the autonomous vehicles will be verified. However, codifying traffic rules can be done abstractly by leaving predicates such as *overtaking*, *safe-distance*, and *side clearance* undefined which still makes traffic rules unclear. Therefore, these predicates need to be concretised through legal and engineering analyses.

Formalising traffic rules entails choosing the logical language to codify the rules. It must be expressive enough to codify natural language yet simple enough to have automation for checking whether the behaviours of autonomous vehicles satisfy (obey) the formulas (traffic rules). In line with our previous works for formalising traffic rules [17], we advocate the use of higher-order logic (HOL) as follows: we codify the rules in linear temporal logic (LTL)—which can be defined in HOL—by assuming each predicate found in the legal text to be an atomic proposition. We then define these predicates concretely in higher-order logic (HOL). In this setting, HOL provides expressiveness while LTL allows automation.

In this paper, we focus on the German traffic rules *Straßenverkehrsordnung* (StVO) especially on the paragraph about overtaking. We choose this specific paragraph, because we think that it represents the general challenge of codification and concretisation of formalising traffic rules. The formalisation is performed with the help of the Isabelle theorem prover in order to achieve a higher level of trustworthiness. Our contributions are as follows:[1]

- We codify a part of the German overtaking traffic rules in LTL and show that these formalise the traffic rules faithfully (Sect. 3).
- We provide a verified checker for detecting the occurrence of an overtaking from a trace of a vehicle (Sect. 4). This requires a formal model of road network—we use lanelets [3]—and functions for detecting lane occupied by a vehicle.
- We provide a verified checker for determining a safe distance by considering the reaction time of the vehicle (Sect. 5); this is an improvement of our previous work [18].
- We provide a trustworthy Standard ML code for overtaking and safe distance checkers and that for monitoring the satisfaction of a trace against LTL formulas (Sect. 6).

2 Preliminaries

Notations used in this paper closely resemble Isabelle/HOL's syntax. Function application is always written in an uncurried form: instead of writing $f\ x\ y$ as in the λ-calculus, we always write $f(x, y)$. We write $t :: \tau$ to indicate that term t has type τ. Types used in this paper could either be a base type such as $\mathbb{R}$ for real numbers, or constructed via type constructors such as $\alpha\, \mathsf{list}$ and $\mathsf{set}\, \alpha$ for list of type α and set of type α, respectively. For an $xs :: \alpha\, \mathsf{list}$, we can (1) obtain its n-th element by writing $xs\,!\,n$; (2) obtain its length by writing $|xs|$;

[1] Our Isabelle formalisation is in https://github.com/rizaldialbert/overtaking.

(3) drop its first n elements by writing $\mathit{drop}(n, xs)$; (4) obtain the first and the last element by writing $\mathit{hd}(xs)$ and $\mathit{last}(xs)$, respectively. We use $\{t \mid x.\ P\}$ as the set builder notation where t is a term, P is a predicate, and x is a free variable in t, which occurs in P. Another frequently used type in this work is a pair; we can obtain the first element of a pair $p :: \alpha \times \beta$ by the *fst* operator, $\mathit{fst}(p) :: \alpha$, and the second element by the *snd* operator, $\mathit{snd}(p) :: \beta$. For option data type, we use *None* and *Some* instead of Haskell's *Nothing* and *Maybe*; but we use Haskell's **do**-notation for monadic computation. In higher-order logic, a deduction with a single premise is written as $P \implies Q$, and if there are n premises, we write $P_1 \implies P_2 \implies \ldots \implies P_n \implies Q$. In linear temporal logic, we shall use $\mathsf{G}(\phi)$ to denote properties that atomic proposition ϕ should be true at all times.

3 Codification of Traffic Rules

The *Straßenverkehrsordnung* (StVO)—or German traffic rules—is the main traffic code for regulating the behaviours of motorised vehicles in Germany. It covers both the scenarios for urban and highway driving: here we focus on the paragraph about overtaking (§ 5 StVO) on highway scenarios. The English version of § 5(4) StVO is:

> When changing the lane to the left lane during overtaking, no following road user shall be endangered. [...] During overtaking, the driver has to change from the fast lane to the right lane as soon as possible. The road user being overtaken shall not be obstructed.

3.1 Legal Analysis

Overtaking in right-hand-traffic countries could be divided roughly into three parts: changing to the left lane, passing the vehicle in front, and returning back to the original lane (see Fig. 1). Whenever a vehicle changes to the left lane to overtake another road user, the driver has to ensure that those on the fast lane will not be endangered. If a vehicle that becomes a following vehicle might be endangered in any way, overtaking is prohibited [5, § 5 StVO, recital 33]. However, this does not mean that any interference with the following traffic needs to be avoided. If, by the overtaking manoeuvre, the following road user is led to reduce its speed safely and will not collide with the overtaking vehicle [4, p. 481], [15, p. 248], the vehicle is allowed to change to the left lane. This overtaking decision must consider the speed difference of the overtaking and the following car.

After overtaking a slower vehicle in front, the overtaking vehicle needs to return to the right lane. This is a special manifestation of the "drive on the right"-rule in § 2(2) StVO [7, § 5 StVO Rn. 32]. When returning to the right lane, other road users must not be forced to brake. The overtaking vehicle also needs to keep a safe distance to the following traffic. However, there is no fixed value for this distance and the decisive factor is that, in the case of an unexpected

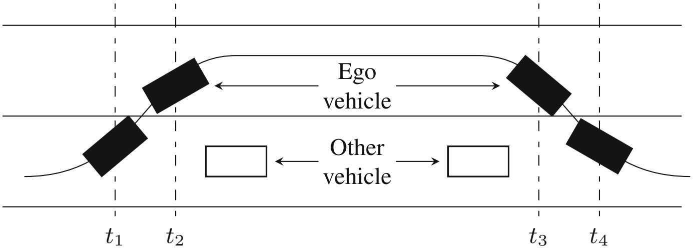

Fig. 1. Illustration of overtaking. The curve represents the overtaking trajectory. We show the positions of the ego vehicle (filled rectangle) at four different time points t_1, t_2, t_3, and t_4. The positions of the other vehicle (empty rectangle) are shown only for t_1 and t_4.

emergency brake, the following vehicle must be able to stop behind the vehicle in front. This depends on the road surface and the speed of both cars [9, § 4 StVO recital 5]. With human drivers, the response time needs to be taken into account.

The last sentence regarding obstruction serves as a protection of the slower vehicle being overtaken. This "no obstruction" rule has the same meaning of keeping a safe distance to the vehicle being overtaken as in the previous paragraphs.

3.2 LTL Formulas of Traffic Rules

In order to codify traffic rules in LTL, we need to identify relevant atomic propositions first. By using the previous legal analysis, we list required atomic propositions with their intended interpretation in Table 1; references to time points t_1, t_2, t_3, and t_4 should be seen in conjunction with illustration in Fig. 1. The LTL formulas of the traffic rules are:

Table 1. Atomic propositions and its intended interpretation

Atomic proposition	Interpretation
`overtaking`	Performing an overtaking manoeuvre—$[t_1; t_4)$
`begin-overtaking`	Overtaking and starting to move to the next lane—$[t_1; t_2)$
`merging`	Starting to merge to the original lane—t_3
`finish-overtaking`	Overtaking and returning back to the original lane—$[t_3; t_4)$
`sd-rear`	Maintaining a safe distance to the rear vehicle on all lanes
`safe-to-return`	Leave large enough distance for merging to the original lane

1. *When changing the lane to the left lane during overtaking, no following road user shall be endangered.*

$$\Phi_1 \;:=\; \boxed{\mathsf{G}\ (\texttt{begin-overtaking} \longrightarrow \texttt{sd-rear})}$$

   As mentioned in the previous legal analysis, the word ‘endangered’ can be concretely interpreted as maintaining a safe distance to the vehicles in the fast lane.
2. *During overtaking, the driver has to change from the fast lane to the right lane as soon as possible.*

$$\Phi_2 \;:=\; \boxed{\mathsf{G}\ (\texttt{merging} \longleftrightarrow \texttt{safe-to-return})}$$

   The phrase ‘as soon as possible’ in this rule is interpreted as the time at which the ego vehicle has left a large enough distance. From this formula, one can infer that atomic proposition `safe-to-return` and `merging` must evaluate to true at the same time; this agrees with the natural language interpretation of the phrase ‘as soon as possible’ too.
3. *The road user being overtaken shall not be obstructed.*

$$\Phi_3 \;:=\; \boxed{\mathsf{G}\ (\texttt{finish-overtaking} \longrightarrow \texttt{sd-rear})}$$

   Here the word ‘obstructed’ is interpreted as maintaining safe distance to the vehicle being overtaken; hence the atomic proposition `sd-rear` in the conclusion of the implication.

3.3 Monitoring Traffic Rules

One intended application of our work is to determine whether the behaviours of an autonomous vehicle recorded in a black box comply with (overtaking) traffic rules or not. This black box is assumed to record not only data from the ego vehicle but also those from other road users observed by the ego vehicle or obtained from vehicle-to-vehicle (V2V) communication. In order to analyse this black box formally, we model the recorded data as discrete time *runs* (or *paths*). Each run is the evolution of a vehicle’s state consisting of continuous data such as position, velocity, and acceleration—all comprise values in x- and y-dimensions. We assume that the black box also contain information about the occupancies of a vehicle; they are represented by rectangles with time-varying width and length.

For formal analysis purposes, we need to convert these runs into *traces*; a corresponding trace of a run is defined here as the evolution of the Boolean values (truth values) over the predefined set of atomic propositions (a word over the set of atomic propositions). This is the next challenge for formalising traffic rules: concretely defining each atomic proposition in Table 1 in terms of the continuous and discrete variables in the runs. Section 4 concretises the first four atomic propositions in Table 1 and Sect. 5 concretises the last two atomic propositions.

4 Concretising the Overtaking Predicate

In this paper, we improve our previous definition of overtaking in [17] by defining four instead of two time points; these points are labelled from t_1 to t_4 in Fig. 1. This is required for concretising `begin-overtaking`, `merging`, and `finish-overtaking`. Overtaking starts at time point t_1, which is the earliest time to touch the lane divider; in $[t_0; t_1)$ the vehicle always stays in the same lane. It then continues until t_2, at which it enters the next lane completely, and stays in this lane until t_3, at which it touches the lane divider again. Overtaking is finished at t_4 when it re-enters the original lane completely. In order to detect and formalise such geometrical interpretations, we need a formal model of lanes and a verified function for lane detection. At t_1 in Fig. 1, for example, the lane detection should tell us that it is in the rightmost lane and starts to touch the lane boundary, and at t_2, it is only in the leftmost lane.

4.1 Lanelets

We use lanelet [3] as a formal model of a lane in this work. A lanelet consists of two nonempty monotone polygonal chains, each for representing the left and right boundary.

Definition 1 (Polygonal chains). *An $xs :: (\mathbb{R}^2 \times \mathbb{R}^2)$ list is a polygonal chain if and only if*

$$\forall i.\ i + 1 < |xs| \longrightarrow \mathsf{snd}\ (xs\,!\,i) = \mathsf{fst}\ (xs\,!\,(i+1)).$$

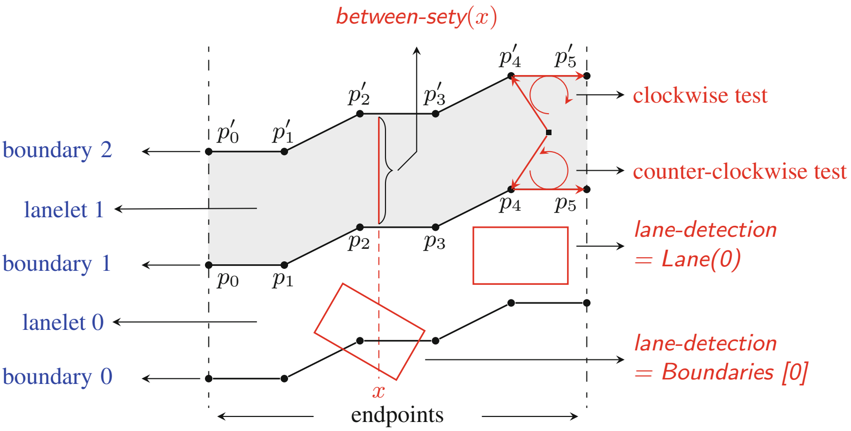

Fig. 2. An example of two lanelets with the direction to the *right*. The upper and lower polygonal chains for lanelet 1 is $\mathsf{points\text{-}le} = [(p'_0, p'_1), (p'_1, p'_2), \ldots, (p'_4, p'_5)]$ and $\mathsf{points\text{-}ri} = [(p_0, p_1), (p_1, p_2), \ldots, (p_4, p_5)]$, respectively. One restriction used in this formalisation is that the endpoints have the same value in x-dimension i. e. $\mathsf{fst}(p_0) = \mathsf{fst}(p'_0)$ and $\mathsf{fst}(p_5) = \mathsf{fst}(p'_5)$. The grey area is the drivable area for lanelet 1. Both the rightmost lanelet and the rightmost boundary are identified with 0, and they increase as we move to the leftmost lanelet and boundary.

Definition 2 (Monotone polygonal chains w.r.t x-dimension). *A monotone polygonal chain w.r.t x-dimension is a polygonal chain whose x-element always increases:*

$$\forall\, i < |xs|. \quad \textsf{fst}\,(\textsf{fst}\,(xs\,!\,i)) \;<\; \textsf{fst}(\textsf{snd}\,(xs\,!\,i)).$$

The property of being monotone for a polygonal chain ensures that for each x, we have a unique y such that (x, y) is in the polygonal chain. Therefore, given a polygonal chain *points*, we can always create a function *f-of-x* from the set of all real numbers in x-dimension to the set of real numbers in y-dimension.

Definition 3 (Lanelets). *A lanelet consists of two nonempty monotone polygonal chains w.r.t. x-dimension, points-le and points-ri, which do not intersect and have the same endpoints in x-dimension.*

As defined in [3], there is no requirement of the relative placement between the two polygonal chains; *points-le* could be positioned above *points-ri* (from a bird's-eye view) or vice-versa. If it is the former then the lanelet has the direction to the right and to the left if it is the latter. Two polygonal chains $\textsf{points}_1$ and $\textsf{points}_2$ are called non-intersecting if there does not exist any two intersecting chains $c_1 \in \textsf{set}\,\textsf{points}_1$, $c_2 \in \textsf{set}\,\textsf{points}_2$.

Note that, with this definition, we could not model a lane which has a 90° turn. This is because our definition of monotone polygonal chain is fixed w.r.t. x-dimension. Lanelets in [3] do not have this restriction, but we can circumvent this problem by using a more general definition of monotone polygonal chains w.r.t to line l and split a polygonal chain into a minimal number of monotone polygonal chains [16]; each with its own coordinate system. We impose this restriction because it eases the following definition of drivable areas and makes the checking of intersecting polygonal chains easier.

Definition 4 (Drivable area). *By using the function representation of the left and right boundary $\textsf{f-of-x}_l$ and $\textsf{f-of-x}_r$, and defining first-point* $:= \textsf{fst}(\textsf{hd}(\textsf{points-le}))$, *last-point* $:= \textsf{fst}(\textsf{last}(\textsf{points-le}))$, *we can define the drivable area as follows (see Fig. 2 for graphical illustration).*

$$\begin{aligned}
\textsf{setx} \;&:=\; \big\{x \;\big|\; x.\ \textsf{first-point} \le x \le \textsf{last-point}\big\},\\
\textsf{between-sety}(x) \;&:=\; \big(\min(\textsf{f-of-x}_l(x), \textsf{f-of-x}_r(x));\ \max(\textsf{f-of-x}_l(x), \textsf{f-of-x}_r(x))\big),\\
\textsf{drivable-area} \;&:=\; \big\{(x, y) \;\big|\; x\,y.\ x \in \textsf{setx} \wedge y \in \textsf{between-sety}(x)\big\}.
\end{aligned}$$

4.2 Lane Detection

In order to detect the lanelet a rectangle is currently occupying, we need first to test whether there is a lanelet in which a rectangle is located completely inside. To achieve this, we need to test whether the four vertices of a rectangle are located in the lanelet, and none of the four edges intersects with any lane boundary of the lanelet. Hence, we need two primitives here: *segment intersection* and *point-in-lanelet test.*

Segments Intersection. First, we differentiate between lines and segments. A line in $\mathbb{R}^2$ is characterised by the line equation $ax + by = c$; a segment is a contiguous subset of a line.

Definition 5 (Closed Segment). *A segment is a pair of points* $(p, q) :: \mathbb{R}^2 \times \mathbb{R}^2$ *and the set of all points on this segment is*

$$\textit{closed-segment}\,(p, q) \;=\; \left\{(1-u)\cdot p + u\cdot q \;\middle|\, u :: \mathbb{R}.\quad 0 \le u \le 1\right\}.$$

With this definition, we can give the correctness and completeness condition for the function *segment-intersect* we wish to define as follows:

$$\textit{segment-intersect}(s_1, s_2) \Longleftrightarrow \exists\, p.\ p \in \textit{closed-segment}(s_1) \,\wedge\, p \in \textit{closed-segment}(s_2)$$

By using the definition of closed segment above, the formula on the right hand side of the bi-implication above can be reformulated as:

$$\exists\, u_1\, u_2.\ 0 \le u_1 \le 1 \wedge ,0 \le u_2 \le 1 \;\wedge\; (1-u_1)\cdot s_1 + u_1 \cdot s_2 = (1-u_2)\cdot s_1 + u_2 \cdot s_2,$$

which is a linear arithmetic formula. Therefore, we can use decision procedures for linear arithmetic problem to define *segment-intersect*. In this work, we implement a specialised instance of Fourier–Motzkin variable elimination algorithm for this problem; readers are encouraged to consult [12] and our implementation in Isabelle for the detailed implementation. We have proved with Isabelle theorem prover that this function indeed satisfies the correctness and completeness condition above.

Point-in-Lanelet Test. Let us consider a lanelet which has the direction to the right and is parameterised by *points-le* and *points-ri* as its left and right boundary, respectively, as defined in Definition 3. To check whether a point is in a lanelet, we need to perform clockwise and counter-clockwise tests. The clockwise test (*cw*) for a triple (p_1, p_2, p_3) checks whether the sequence of points in the triple has a clockwise orientation; the counter-clockwise (*counter-cw*) test does the opposite (see Fig. 2). The point-in-lanelet test is defined by the following function.

$$\begin{aligned}\textit{point-in-lanelet}(p) \;:=\; & \textbf{let } c_1 = \textit{find-segment}(\textit{points-le}, p);\\ & \phantom{\textbf{let }} c_2 = \textit{find-segment}(\textit{points-ri}, p)\\ & \textbf{in }\; \textit{cw}(p, \textit{fst}(c_1), \textit{snd}(c_1)) \;\wedge\; \textit{counter-cw}(p, \textit{fst}(c_2), \textit{snd}(c_2))\end{aligned}$$

The *point-in-lanelet*(p) first finds the two segments c_1 and c_2 in the left and right polygonal chains, respectively, such that *in-x-interval*(c_1, x) and *in-x-interval*(c_2, x) hold (for instance $c_1 = (p'_4, p'_5)$ and $c_2 = (p_4, p_5)$ in Fig. 2). With these two segments, we only need to perform a counter-clockwise test for the triple $(p, \textit{fst}(c_2), \textit{snd}(c_2))$ and a clockwise test for the triple $(p, \textit{fst}(c_1), \textit{snd}(c_1))$ (see Fig. 2). This will guarantee that the point (x, y) is between the segments c_1 and c_2, which in turn ensures that the point is in the drivable area.

Theorem 1. *For a right-direction lanelet defined in Definition 3 with* $\mathsf{points\text{-}le}$ *and* $\mathsf{points\text{-}ri}$ *as its left and right boundary, respectively, we have*

$$\textit{point-in-lanelet}(p) \implies p \in \textit{drivable-area}.$$

Previously, we have explained how to test whether a rectangle is located inside a lanelet completely. However, this is not the only possible result of lane detection; we define a new data type to represent all possible results of our lane detection:

$$\textbf{datatype}\ \textit{detection-opt}\ =\ \textit{Outside} \mid \textit{Lane}\ (n :: \mathbb{N}) \mid \textit{Boundaries}\ (ns :: \mathbb{N}\ \textit{list})$$

Each argument in the constructor *Lane* and *Boundaries* represents the lanelet identifier at which it is currently located or a list of boundaries with which a rectangle is intersecting, respectively. Figure 2 provides two examples of lane detection. The first rectangle intersects with boundary 0 only and hence our lane detection primitive returns *Boundaries*$([0])$. The second rectangle meanwhile is located inside lanelet 0 and hence our lane detection primitive returns *Lane*(0).

The function *lane-detection* takes a rectangle and lane boundaries as arguments and returns an element of type *detection-opt*. It checks first whether there is any lanelet in which a rectangle is completely located and, if there is no such lanelet, it tests for the intersections between the lane boundaries and rectangles. This can be easily done by checking intersections between relevant segments in the lane boundaries and edges in the rectangle. If there is no such lane boundary, we conclude that the rectangle is outside of any lanelet.

4.3 Overtaking Detection

We can use the previously described lane detection function for detecting overtaking as follows. Assuming that the vehicle is located in lane n initially, we use the following function to detect t_1 and t_2:

$$\begin{aligned}\textit{increase-lane}\ \ rects\ :=\ \ \textbf{do}\,\{(t_1, r_1)\ &\leftarrow\ \textit{start-inc-lane}(rects, n, 0);\\ (t_2, r_2)\ &\leftarrow\ \textit{finish-inc-lane}(r_1, (n+1), (t_1+1));\\ \textit{Some}\ ((t_1, t_2), r_2)\}&\end{aligned}$$

The function *start-inc-lane* detects t_1 by continuously checking whether the occupied lane is still n and stops immediately whenever the lane detection returns the boundary $n+1$. Function *finish-inc-lane* detects t_2 by checking whether the lane is still on the boundary $n+1$ and stops immediately on the first occurrence of the lane $n+1$. Notice that $[t_0; t_2)$ and $[t_2; t_4)$ are identical and we can therefore detect t_3 and t_4 similarly as we do for t_1 and t_2, respectively, with function *decrease-lane*. The correctness of the function *increase-lane*[2] is shown next.

[2] We also have similar theorems for decrease-lane but they are omitted for brevity.

Theorem 2. *Assuming that the initial lane is n, we have the following deduction:*

$$\begin{aligned}
&\textsf{\textit{increase-lane}}\ \mathit{rects} = \textsf{\textit{Some}}\,(t_1, t_2, \mathit{rest}) \implies \mathit{rects}' \ = \ \textsf{\textit{drop}}(t_1 + 1, \mathit{rects}) \implies \\
&\qquad \wedge\ t_1 < |\mathit{rects}| \ \wedge\ t_1 < t_2 \ \wedge\ t_2 < t_1 + |\mathit{rects}'| \\
&\qquad \wedge\ \textsf{\textit{lane-detection}}\ (\mathit{rects}\,!\,t_1) = \textsf{\textit{Boundaries}}\,[n+1] \\
&\qquad \wedge\ \forall\, m.\ m < t_1 \longrightarrow \textsf{\textit{lane-detection}}\ (\mathit{rects}\,!\,m) = \textsf{\textit{Lane}}\,(n) \\
&\qquad \wedge\ \textsf{\textit{lane-detection}}\ (\mathit{rects}\,!\,t_2) = \textsf{\textit{Lane}}\,(n+1) \\
&\qquad \wedge\ \forall\, m > t_1.\ m < t_2 \longrightarrow \textsf{\textit{lane-detection}}\ (\mathit{rects}\,!\,m) = \textsf{\textit{Boundaries}}\,[n+1]
\end{aligned}$$

The overtaking detection can be defined by using *increase-lane* and *decrease-lane* as follows. The primitive looks for t_1 and t_2 with *increase-lane* first and—if we have found this—continues to search for t_3 and t_4 with *decrease-lane*. If it cannot find t_1 and t_2 initially, we can conclude that there is no occurrence of overtaking at all. It could also be that we found t_1 and t_2 without the corresponding pair t_3 and t_4. In this case, we discard t_1 and t_2 and start to look for a new occurrence of overtaking from one lane to the left of the original lane.

5 Concretising the Safe Distance Predicate

The safe distance problem has been previously explored in our previous work [18] and, in this work, we improve it by considering nonzero reaction time. In this problem, we are interested in the scenario where there are two vehicles involved: *ego* and *other* vehicle which are located at $s_{0,\mathrm{e}}$ and $s_{0,\mathrm{o}}$, respectively. These positions are the frontmost part of the ego vehicle and the rearmost part of the other vehicle, respectively. We assume that the other vehicle is located in front of the ego vehicle initially, $s_{0,\mathrm{e}} < s_{0,\mathrm{o}}$, and the other vehicle performs an emergency brake with maximum deceleration $a_\mathrm{o} < 0$. After $0 < \delta$ seconds of reaction time, the ego vehicle also performs an emergency brake with maximum deceleration $a_\mathrm{e} < 0$. We can define the braking movement of the other vehicle mathematically as follows:

$$s_\mathrm{o}(t) := \begin{cases} p_\mathrm{o}(t) & \text{if } \ 0 \leq t \leq t_{\mathrm{stop,o}} \\ p_\mathrm{o}(t_{\mathrm{stop,o}}) & \text{if } \ t_{\mathrm{stop,o}} \leq t \end{cases} \tag{1}$$

where $p_\mathrm{o}(t) := s_{0,\mathrm{o}} + v_\mathrm{o} t + \frac{1}{2} a_\mathrm{o} t^2$ and $t_{\mathrm{stop,o}} := -\frac{v_\mathrm{o}}{a_\mathrm{o}}$. To cater for the reaction time delay, the braking movement of the ego vehicle defined here is slightly different than what we have defined in our previous work [18]:

$$u_\mathrm{e}(t) := \begin{cases} q_\mathrm{e}(t) & \text{if } \ 0 \leq t \leq \delta \\ p_\mathrm{e}^*(t - \delta) & \text{if } \ \delta \leq t \leq t_{\mathrm{stop,e}} + \delta \\ p_\mathrm{e}^*(t_{\mathrm{stop,e}}) & \text{if } \ t_{\mathrm{stop,e}} - \delta \leq t \end{cases} \tag{2}$$

with $q_\mathrm{e}(t) := s_{0,\mathrm{e}} + v_\mathrm{e} t$, $p_\mathrm{e}^*(t) := q_\mathrm{e}(\delta) + v_\mathrm{e} t + \frac{1}{2} a_\mathrm{e} t^2$, and $t_{\mathrm{stop,e}} := -\frac{v_\mathrm{e}}{a_\mathrm{e}}$. As the definition of q_e shows, we can now model that the ego vehicle maintains its current speed for δ seconds before performing an emergency brake. The stopping distances for both vehicles $u_{\text{stop,e}}$ and $s_{\text{stop,o}}$ are the positions where the derivative of $u_{\text{e}}(t)$ and $s_{\text{o}}(t)$, respectively, are equal to zero; we prove that these are

$$u_{\text{stop,e}} := q_{\text{e}}(\delta) - \frac{v_{\text{e}}^2}{2 \cdot a_{\text{e}}} \quad \text{and} \quad s_{\text{stop,o}} := s_{0,\text{o}} - \frac{v_{\text{o}}^2}{2 \cdot a_{\text{o}}}. \tag{3}$$

The problem now is to determine a sufficient distance $s_{0,\text{o}} - s_{0,\text{e}}$ such that for $T = [0; \infty)$ the predicate $\textit{no-collision-react}\,(T) \;:=\; \neg\,(\exists\, t \in T.\; u_{\text{e}}(t) \;=\; s_{\text{o}}(t))$ is true.

Following the methodology in our previous work [18], we analyse all possible cases to obtain the lower bound of the distance that is still safe. There are five possible cases as shown in Fig. 3. These five cases are obtained from case distinction based on stopping distances $u_{\text{stop,e}}$. In case ①, the ego vehicle stops before the initial position of the other vehicle and in case ② it stops after the stopping position of the other vehicle; in case ③, ④, and ⑤, it stops in between. Case ③ is characterised by the condition where the ego position at time δ is already in front of the other vehicle while the fourth and the fifth is not. In case ④, the ego vehicle stops after $s_{\text{o}}(\delta)$ i. e. the position of the other vehicle at time $t = \delta$, while in case ⑤, the ego vehicle stops before.

As Fig. 3 shows, we can deduce that there will be no collision for case ① because the ego vehicle stops before the starting position of the other vehicle.

Theorem 3 (Case ①). $u_{\text{stop,e}} < s_{0,\text{o}} \implies \textit{no-collision-react}\,[0; \infty)$

From this theorem, we can replace the definition of $u_{\text{stop,e}}$ with the expression in Eq. 3 so that we obtain $\textit{safe-distance}_0 \;:=\; v_{\text{e}} \cdot \delta - v_{\text{e}}^2/(2 \cdot a_{\text{e}})$ as our zeroth safe distance expression. In cases ② and ③, the stopping positions of the ego vehicle are after those of the other vehicles'. Hence, we can deduce collisions by using the Intermediate Value Theorem (IVT). In case ④, we cannot deduce a

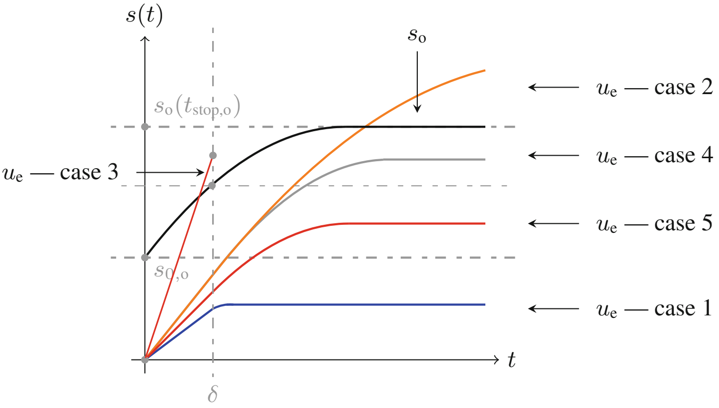

Fig. 3. Cases obtained according to relative positions of ego's stopping distance to that of the other vehicle.

collision or a collision freedom just by looking at the stopping distances of the ego vehicle relative to the other vehicle; the following theorem helps to deduce these.

Theorem 4 (Case ④). *By defining*

$$v_{\mathrm{o}}^* := \begin{cases} v_{\mathrm{o}} + a_{\mathrm{o}}\delta & \textit{if } \delta \le t_{\mathrm{stop,o}} \\ 0 & \textit{otherwise} \end{cases} \tag{4}$$

and $t^*_{\mathrm{stop,o}} := -\frac{v_{\mathrm{o}}^*}{a_{\mathrm{o}}}$, $s^*_{0,\mathrm{o}} := s_{\mathrm{o}}(\delta)$, *and* $s^*_{0,\mathrm{e}} := q_{\mathrm{e}}(\delta)$, *we have*

$$s_{0,\mathrm{o}} \le u_{\mathrm{stop,e}} \implies u_{\mathrm{stop,e}} < s_{\mathrm{stop,o}} \implies \textsf{\textit{no-collision-react}}\,[0;\infty) \iff \neg\left(a_{\mathrm{o}} > a_{\mathrm{e}} \land v_{\mathrm{o}}^* < v_{\mathrm{e}} \land s^*_{0,\mathrm{o}} - s^*_{0,\mathrm{e}} \le \frac{(v_{\mathrm{o}}^* - v_{\mathrm{e}})^2}{2\cdot(a_{\mathrm{o}} - a_{\mathrm{e}})} \land t_{\mathrm{stop,e}} < t^*_{\mathrm{stop,o}}\right).$$

Note that the definition of v_{o}^* depends on the condition whether the reaction time is smaller than the stopping time of the other vehicle $t_{\mathrm{stop,o}}$. When $\delta \le t_{\mathrm{stop,o}}$, we can rearrange the deduction by first weakening the bi-implication into implication and replacing the definition of $u_{\mathrm{stop,e}}$ and $s_{\mathrm{stop,o}}$ with their respective definitions in Eq. 3, and then use de Morgan's rule into:

$$\begin{aligned} & s_{0,\mathrm{o}} \le u_{\mathrm{stop,e}} \implies (a_{\mathrm{o}} > a_{\mathrm{e}} \land v_{\mathrm{o}}^* < v_{\mathrm{e}} \land t_{\mathrm{stop,e}} < t^*_{\mathrm{stop,o}}) \\ & s_{0,\mathrm{o}} - s_{0,\mathrm{e}} > \underbrace{v_{\mathrm{e}}\delta - \frac{v_{\mathrm{e}}^2}{2a_{\mathrm{e}}} + \frac{v_{\mathrm{o}}^2}{2a_{\mathrm{o}}}}_{\textsf{\textit{safe-distance}}_1} \implies s_{0,\mathrm{o}} - s_{0,\mathrm{e}} > \underbrace{\frac{(v_{\mathrm{o}} + a_{\mathrm{o}}\delta - v_{\mathrm{e}})^2}{2\cdot(a_{\mathrm{o}} - a_{\mathrm{e}})} - v_{\mathrm{o}}\delta - \frac{1}{2}a_{\mathrm{o}}\delta^2 + v_{\mathrm{e}}\delta}_{\textsf{\textit{safe-distance}}_2} \\ & \qquad \implies \textsf{\textit{no-collision-react}}\,[0;\infty). \end{aligned}$$

If the first assumption in the deduction above is false, then we are in case ① and Theorem 3 guarantees the situation to be collision free. Hence, to derive a checker, we can safely ignore the first assumption above and put the second assumption as a condition in an **if** statement. Now, we are left with two expressions for safe distance, which can be chosen with the following lemma.

Lemma 1. $a_o > a_e \implies \textsf{\textit{safe-distance}}_1 \le \textsf{\textit{safe-distance}}_2$.

From this lemma, we check whether the distance is larger than $\textsf{\textit{safe-distance}}_2$ when $a_{\mathrm{o}} > a_{\mathrm{e}} \land v_{\mathrm{o}}^* < v_{\mathrm{e}} \land t_{\mathrm{stop,e}} < t^*_{\mathrm{stop,o}}$ is true. Otherwise, we check the distance against $\textsf{\textit{safe-distance}}_1$ because Theorem 4 suggests that this will lead to collision freedom.

In case ⑤, we can deduce a collision freedom. To see this, we can reformulate the problem into a safe distance problem without reaction time delay as we did in our previous work [18] with δ is set to zero. Graphically speaking, we ignore any behaviour that has happened to the left of δ in Fig. 3 and, in this reformulation, case ⑤ becomes case ① in zero reaction time delay setting. As the deduction in [18] suggests, there will be no collision.

Theorem 5 (Case ⑤).

$$s_{\mathrm{o}} \leq u_{\mathrm{stop,e}} \implies u_{\mathrm{stop,e}} < s_{\mathrm{stop,o}} \implies u_{\mathrm{stop,o}} < s_{\mathrm{o}}(\delta) \implies \textit{no-collision-react}\,[0;\infty)$$

By using Eq. 3, the premise $u_{\mathrm{stop,o}} < s_{\mathrm{o}}(\delta)$ in the deduction above can be reformulated so that we can obtained the third safe distance expression for $\delta \leq t_{\mathrm{stop,o}}$ as $\textit{safe-distance}_3 := v_{\mathrm{e}}\delta - v_{\mathrm{e}}^2/2a_{\mathrm{e}} - v_{\mathrm{o}}\delta - a_{\mathrm{o}}\delta^2/2$. To summarise, we can combine all the logical analyses above into the following checker which we have proved in Isabelle theorem prover to be sound and complete.

```
checker :=
        let $\textit{dist} = s_{0,\mathrm{o}} - s_{0,\mathrm{e}}$ in
        if $\underline{\textit{dist} > \textit{safe-distance}_0} \lor (\delta \leq t_{\mathrm{stop,o}} \land \underline{\textit{dist} > \textit{safe-distance}_3})$ then True
        elseif $\delta \leq t_{\mathrm{stop,o}} \land a_0 > a_{\mathrm{e}} \land v_{\mathrm{o}}^* < v_{\mathrm{e}} \land t_{\mathrm{stop,e}} < t_{\mathrm{stop,o}}^*$ then
          $\underline{\textit{dist} > \textit{safe-distance}_2}$ else $\underline{\textit{dist} > \textit{safe-distance}_1}$
```

We can now define the atomic proposition `sd-rear` and `safe-to-return` simply as an application of the safe distance checker. However, the analysis above have the opposite assumption about the relative position of the ego and the other vehicle. Therefore, we have to properly swap the values of position, velocity, and maximum deceleration between the ego and the other vehicle. The key difference between the definition of atomic proposition `sd-rear` and `safe-to-return` is that the former uses the checker w.r.t. the vehicle in the left lanelet, while the latter is w.r.t. the vehicle in the right lanelet.

Since both `sd-rear` and `safe-to-return` are defined by using the notion of safe distance, readers might think that the rule of returning to the right lane as soon as possible (Φ_2) might not be valid because the safe distance condition might hold immediately after the change to the left lane. This would not happen due to the assumption $s_{0,\mathrm{e}} < s_{0,\mathrm{o}}$ in the safe distance problem explained in the beginning of this section; our checker checks this condition implicitly[3]. Hence, during the monitoring of Φ_2, the values between the ego and the other vehicle are swapped and the previous condition becomes $s_{0,\mathrm{o}} < s_{0,\mathrm{e}}$. After the ego vehicle change to the left lane and is still behind the other vehicle, this condition will be false and hence our formalisation of Φ_2 excludes this scenario.

6 Monitoring Overtaking Traffic Rules

With the concrete definition of the atomic propositions in Table 1, we can define a converter from a run to a trace for each atomic proposition. These traces are then combined into a word over the set of all atomic propositions for monitoring the satisfaction of the codified traffic rules in LTL. For this purpose, we need to define the semantics we use in this work. Since runs from an autonomous vehicles' planners are usually finite, we use the semantics in [6][4] which interprets LTL formulas over finite traces.

[3] This can be checked in our Isabelle formalisation.

[4] The semantics for LTL is pretty standard and, hence, we omit them for brevity.

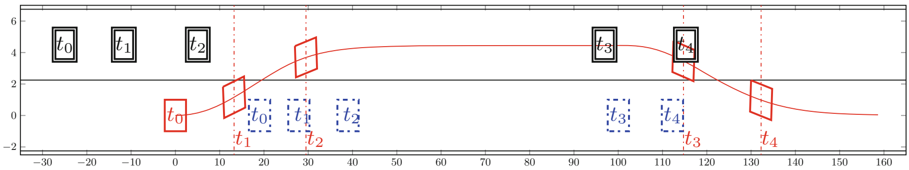

Fig. 4. Example of overtaking scenario. The curve is the trajectory of the ego vehicle we want to monitor. All vehicles drive to the right direction. Ego vehicle (solid rectangle) is positioned at $(0, 0)$ and the first vehicle (double rectangle) at $(-25, 4.5)$ initially. Both vehicles have initial velocity of $16.7\,\mathrm{ms}^{-1}$. The second vehicle (dashed rectangle) is located at $(19, 0)$ with initial velocity of $11.1\,\mathrm{ms}^{-1}$. All vehicles except the ego vehicle drive with constant velocity. For each vehicle we show the position at time $t_1 = 0.8\,\mathrm{s}$, $t_2 = 1.8\,\mathrm{s}$, $t_3 = 7.3\,\mathrm{s}$, and $t_4 = 8.4\,\mathrm{s}$.

As a numerical example, we consider a situation with two lanelets (Fig. 4) in which the first vehicle (double rectangle) is positioned behind the ego vehicle in the leftmost lane. The ego vehicle (solid rectangle) intends to overtake the second vehicle (dashed rectangle); both are at the rightmost lane initially. Each vehicle has the same width (2 m), length (4.8 m), reaction time (1 s), and maximum deceleration ($-8\,\mathrm{ms}^{-2}$). The run for this numerical example is produced by the controller for autonomous vehicles found in [20].

In order to monitor this run on the code level, we need to generate the code for the environment (lanelets). From Definition 3, we need to ensure that the two boundaries of each lanelet are not intersecting. To achieve this, we have implemented a polygonal chain intersection test according to the algorithm in [16] and proved its correctness. The algorithm is based on a sweep-line algorithm which, as the sweeping progresses, it checks whether each pair of chain in each boundary are relevant or not and, if the pair is relevant, it performs an intersection check as explained in Sect. 4.

All code required for monitoring the codified traffic rules, including the semantics of the LTL, are generated automatically by using Isabelle's code generator [8]. Therefore, we only have to trust the Isabelle's code generator for the correctness of our Standard ML's code. However, most of the formalised functions used for monitoring traffic rules require real numbers data type, and code generation with real numbers is usually done with the help of Interval Arithmetic [10] or Affine Arithmetic [11]. Since this is a numerical example only, mapping real numbers to machine's floating-point numbers is sufficient.

The results of the simulation show that all overtaking traffic rules except the merging rule (Φ_2) are satisfied. Particularly, rule Φ_2 is not satisfied due to the 'if' ($\longleftarrow$) fragment. If we weaken rule Φ_2 into $\Phi_2' := \mathsf{G}\,(\texttt{merging} \longrightarrow \texttt{safe-to-return})$, the run from the controller still satisfies rule Φ_2'.

7 Related Work and Conclusions

The monitoring part of our work belongs to the research area called *runtime verification*; Küster [13] provides a complete overview of this research area. Specifically, our work can be categorised as *runtime monitoring.* Our work does not construct a monitor automaton [2] as in most monitoring techniques but simply executes the semantics of LTL over finite-length traces. This is sufficient because we wish to verify traces produced by autonomous vehicles' planners whose duration are usually not very long. The other intended application of our work is to perform automated offline checking of a recorded trace for compliance with traffic rules.

In terms of the logic used for specifying properties, there is signal temporal logic (STL) [14] which is expressive enough to specify real-time properties. This is particularly useful for relaxing the requirement to satisfy a rule within a certain duration of time such as in the 'if' part of Φ_2. Another expressive logic for runtime monitoring is metric first-order temporal logic (MFOTL) [1] which is capable of handling relations that change over time such as safe distance. However, we stick to logic without first-order fragment because we do not need to reason about first-order structure for formalising the presented subset of traffic rules.

In comparison to [18], we have improved the analysis by taking the reaction time into consideration. This is required because autonomous vehicles will interact with human drivers and they clearly do not have zero reaction time. To achieve this, we have to perform five instead of three case analyses in terms of the relative stopping distance between the ego and the other vehicles. Additionally, during our formalisation effort, we found out that we need two additional case analyses regarding the relative duration of the stopping time and reaction time. As a result, we obtain a general checker (safe distance expression) which has been proved to be sound and complete in Isabelle/HOL.

In comparison to [17], we now have identified four instead of two time points when detecting overtaking occurrences. To achieve this, we need to implement two additional checkers for detecting the additional time points and formally verify them in Isabelle/HOL. As a result, we are now able to specify traffic rules which requires these two time points: safe distance during the beginning of the overtaking manoeuvre and merging.

To conclude, we have formalised a subset of overtaking traffic rules in Isabelle/HOL. By formalising traffic rules, we do not mean only the codification of traffic rules in a logical language where abstract concepts such as overtaking and safe distance are left unspecified: we went deeper by concretising these abstract concepts through legal and mathematical analysis. Through these two analyses, we obtain unambiguous, precisely defined specifications from overtaking traffic rules for autonomous vehicles. Furthermore, from these formalised traffic rules, we show how to monitor the satisfaction of a run obtained from a planner for autonomous vehicles.

References

1. Basin, D.A., Klaedtke, F., Müller, S., Zalinescu, E.: Monitoring metric first-order temporal properties. J. ACM **62**(2), 15:1–15:45 (2015)
2. Bauer, A., Leucker, M., Schallhart, C.: Runtime verification for LTL and TLTL. ACM Trans. Softw. Eng. Methodol. **20**(4), 14:1–14:64 (2011)
3. Bender, P., Ziegler, J., Stiller, C.: Lanelets: efficient map representation for autonomous driving. In: Proceedings of the IEEE Intelligent Vehicles Symposium, Michigan, MI, USA, pp. 420–425 (2014)
4. Bundesgerichtshof: Sorgfaltspflichten beim überholen auf der autobahn, pp. 481–482. Neue Juristische Wochenschrift: NJW (1954)
5. Burmann, M., Heß, R., Hühnermann, K., Jahnke, J., Janker, H.: Straßenverkehrsrecht: Kommentar, 24th edn. C.H. Beck (2016)
6. Giannakopoulo, D., Havelund, K.: Automata-based verification of temporal properties on running programs. In: Proceedings of the 16th Annual International Conference on Automated Software Engineering (ASE), San Diego, CA, USA, pp. 412–416 (2001)
7. Gutt, S.: Gesamtes Verkehrsrecht, 1st edn. Nomos (2014)
8. Haftmann, F., Nipkow, T.: Code generation via higher-order rewrite systems. In: Blume, M., Kobayashi, N., Vidal, G. (eds.) FLOPS 2010. LNCS, vol. 6009, pp. 103–117. Springer, Heidelberg (2010). doi:10.1007/978-3-642-12251-4_9
9. Hentschel, P., König, P., Dauer, P.: Straßenverkehrsrecht: Kommentar, 43rd edn. C.H. Beck (2015)
10. Hölzl, J.: Proving inequalities over reals with computation in Isabelle/HOL. In: Proceedings of the ACM SIGSAM International Workshop on Programming Languages for Mechanized Mathematics Systems (PLMMS 2009), Munich, pp. 38–45 (2009)
11. Immler, F.: A verified algorithm for geometric zonotope/hyperplane intersection. In: Proceedings of the International Conference on Certified Programs and Proofs (CPP), Mumbai, India, pp. 129–136 (2015)
12. Kroening, D., Strichman, O.: Decision Procedures - An Algorithmic Point of View. Texts in Theoretical Computer Science. EATCS Series, 2nd edn. Springer, Heidelberg (2016)
13. Küster, J.C.: Runtime verification on data-carrying traces. Ph.D. thesis, The Australian National University, October 2016
14. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Lakhnech, Y., Yovine, S. (eds.) FORMATS/FTRTFT -2004. LNCS, vol. 3253, pp. 152–166. Springer, Heidelberg (2004). doi:10.1007/978-3-540-30206-3_12
15. Oberlandesgericht Karlsruhe: Gefährdung des Nachfolgendes beim Überholen. Neue Zeitschrift für Verkehrsrecht (NZV), pp. 248–249 (1992)
16. Park, S.C., Shin, H.: Polygonal chain intersection. Comput. Graph. **26**(2), 341–350 (2002)
17. Rizaldi, A., Althoff, M.: Formalising traffic rules for accountability of autonomous vehicles. In: Proceedings of the 18th IEEE International Conference on Intelligent Transportation Systems, Las Palmas de Gran Canaria Canary Islands, Spain, pp. 1658–1665 (2015)
18. Rizaldi, A., Immler, F., Althoff, M.: A formally verified checker of the safe distance traffic rules for autonomous vehicles. In: Rayadurgam, S., Tkachuk, O. (eds.) NFM 2016. LNCS, vol. 9690, pp. 175–190. Springer, Cham (2016). doi:10.1007/978-3-319-40648-0_14

19. Sergot, M.J., Sadri, F., Kowalski, R.A., Kriwaczek, F., Hammond, P., Cory, H.T.: The British Nationality Act as a logic program. Commun. ACM **29**(5), 370–386 (1986)
20. Werling, M., Ziegler, J., Kammel, S., Thrun, S.: Optimal trajectory generation for dynamic street scenarios in a frenet frame. In: Proceedings of the IEEE International Conference on Robotics and Automation, Anchorage, AK, USA, pp. 987–993 (2010)