TECHNICAL UNIVERSITY OF MUNICH

DEPARTMENT OF INFORMATICS

Bachelor's Thesis in Informatics

Situation-Aware Simplification of Temporal Logic Specifications for Autonomous Vehicles

Paul Manfred Reisenberg

TECHNICAL UNIVERSITY OF MUNICH

DEPARTMENT OF INFORMATICS

Bachelor's Thesis in Informatics

Situation-Aware Simplification of Temporal Logic Specifications for Autonomous Vehicles

Author:	Paul Manfred Reisenberg
Supervisor:	Prof. Dr.-Ing. Matthias Althoff
Advisor:	Florian Lercher, M.Sc.
Submission Date:	TBD

I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

Munich, TBD

Paul Manfred Reisenberg

Abstract

Temporal Logic is a powerful formalism for specifying the behavior of autonomous vehicles over time, ensuring their compliance with traffic rules and verifying their safety. Previous work has explored the use of temporal logic for both specifying traffic regulations and motion planning for autonomous vehicles. However, as the number of traffic participants increases, a significant computational challenge arises when handling Temporal Logic specifications that must be satisfied concerning the interactions with every other traffic participant. In this work, we present a solution to mitigate this problem for Linear Temporal Logic (*LTL*) specifications. We first introduce an algorithm that simplifies *LTL* specifications based on partial knowledge about the state trace, e.g., knowledge about when certain propositions hold and do not hold. We then adapt this algorithm to handle traffic rule specifications from existing literature. Furthermore, we evaluate the algorithm by applying it to real-world traffic scenarios using the CommonRoad framework. Our evaluation demonstrates that the simplified specifications generated by our algorithm provide substantial reductions in computational complexity while maintaining equivalent expressiveness.

Contents

1 Introduction

Traffic regulations are vital for ensuring road safety and efficient transportation systems. Complying with traffic rules is crucial for preventing accidents, maintaining order, and facilitating smooth traffic flow. Although traffic regulations vary by location, the Vienna Convention on Road Traffic (VCoRT) [1] establishes a common framework adopted by numerous countries, including Germany's Road Traffic Regulation (StVO). For autonomous vehicles to operate safely on public roads, they must adhere to all applicable traffic rules at all times. A formalisim to define these traffic rules for autonomous vehicles is temporal logic, which allows the specification of properties over time. Evaluating compliance with these rules, however, can be computationally expensive, and in some cases, unnecessary if a particular rule is trivially satisfied given the available information about the traffic situation. For instance, the right-before-left rule can be ignored if it is known that no vehicle is ever approaching from the right.

Motivated by this idea of ignoring specifications that we must not pay attention to, based on the knowledge we already have about the traffic situation, we first we propose a general simplification algorithm for Linear Temporal Logic (LTL) specifications based on partial about the Boolean values of propositions within the system's state sequence i.e. the traffic situation. If the truth value of a formula can be determined a priori the algorithm returns either $\top$ (true), in which case the entire specification can be ignored or $\bot$ (false). If the truth value of the formula can not be determined a priori, the algorithm returns a semantically equivalent but simplified formula instead. The purpose of this algorithm is to serve as a preprocessing step for algorithms that work with LTL specifications, reducing the complexity of verifying the formulas by simplifying them based on the available information. To demonstrate the effectiveness of our simplification algorithm, we apply it to formalizations of interstate and intersection traffic rules from [2] and [3]. First, we illustrate the theoretical benefit of our approach by simplifying *LTL* specifications for traffic scenarios from the CommonRoad framework [4]. Subsequently, we integrate our algorithm into a specification-compliant reachability analysis from [5], that operates on *LTL* specifications, and show that the simplified specifications generated by our algorithm lead to substantial reductions in computational complexity.

1.1 Related Work

1.1.1 Temporal Logic Simplification

The closest related line of research to our simplification approach of Temporal Logic Specifications is 3-Valued Linear Temporal Logics. This extension introduces a third truth value "?", which means "unknown whether true or false", which allows the reasoning about incomplete state traces. This approach is particularly prominent for runtime verification [6]. The authors in [7] introduce a 3-valued semantic for *LTL* and propose a procedure for generating a deterministic monitor which can identify satisfaction or violation of properties as early as possible. Further automation based approaches for *LTL* can be found in [8], [9]. Another approach is formula rewriting [10]–[13]. Similar to [12] we reduce a *LTL* formula to either $\top$ or $\bot$ if it can be deduced. However, if neither $\top$ nor $\bot$ can be inferred, instead of returning ?, we return a simplified formula that is satisfied if and only if the original formula is satisfied. 3-Valued approaches for Computation Tree Logic (*CTL*) and Signal Temporal Logic (*STL*) can be found in [14] and [15]. Furthermore, extensions such as 4-Valued-*LTL* [16] exist which further splits "?" into (1) "will presumably violate the property" and (2) "presumably conform to the property".

1.1.2 Formalization of Traffic Rules

Research on formalizing traffic rules can broadly be categorized into two areas: interstate rules [2], [17], [18] and intersection rules [3], [19], [20]. Additionally, there are Responsibility-Sensitive Safety (RSS) rules [21]–[24], which define general guidelines for autonomous vehicles to follow, though they lack legal standing. Temporal logic has been the predominant formalization approach for traffic rules. *LTL* and co-safe *LTL* have been employed in [17], [23], [25]–[28], while Metric Temporal Logic (MTL) has been utilized in [2], [3]. *STL* has also been used in [18], [21], [22], [24]. However, some formalizations have been proposed without resorting to temporal logic [19], [20], [29].

1.1.3 Vehicle Occupancy Prediction

A comprehensive analysis of the possible future positions and movements of other vehicles, pedestrians, and obstacles is essential for safe navigation in road traffic. Approaches for occupancy predictions of traffic participants can broadly be divided into three categories: *1) Single future behaviour* [30]–[35] considers a single possibility for the future occupancy of traffic participants. While sufficient for non-safety-related applications, it cannot be used for safety verification as it only considers a single possibility *2) Stochastic approaches* employed in [36]–[40]. Although useful for risk assessment, stochastic approaches cannot be used for formal verification as they offer probabilistic guarantees rather than strict, deterministic proofs of correctness. *3) Set-based predictions* as proposed in [41]–[44] consider all possible future positions that traffic participants can occupy. This approach is suitable for formal verification as it provides guarantees

of correctness by considering all possible future behaviors. We employ the notion of a reachable set from [42] (Definition 4) to determine propositions about future states of the traffic scenario.

2 Preliminaries

This chapter introduces the fundamental concepts that serve as a basis for this thesis. It introduces relevant notation and covers the basics of temporal logic. Additionally, it provides an overview of the road network and vehicle model, as well a concrete example of a traffic rule formalized in *LTL*. Lastly we introduce an algorithm for *LTL* specification compliant reachability analysis [5], used to evaluate the temporal logic simplifications.

2.1 Notation

2.1.1 Minkowski Sum

The Minkowski sum is a fundamental operation in various fields such as convex geometry, computational geometry, and mathematical optimization. Given two sets A and B, $A \oplus B$ is defined as the set obtained by adding each element in A to each element in B.

Formally, the Minkowski sum of sets A and B is defined as follows:

$$A \oplus B := \{a + b \mid a \in A,\ b \in B\}$$

2.1.2 Interval Notation

In this work we often work with intervals defined over $\mathbb{N}_0$. Given $a \in \mathbb{N}_0$ and $b \in \mathbb{N}_0 \cup \{\infty\}$ the resulting set from our interval notation is given by:

$$[a, b] = \{x \in \mathbb{N}_0 \mid a \leq x \leq b\}$$

2.2 Temporal Logic

2.2.1 Linear Temporal Logic

Linear Temporal Logic (*LTL*) is an extension of normal Boolean logic that allows the formalization of propositions and relations over discrete time. An *LTL* formula over a set of atomic propositions $\mathcal{AP}$ is constructed as follows:

$$\varphi ::= a \mid \top \mid \bot \mid \gamma \wedge \psi \mid \neg\gamma \mid \mathbf{X}\gamma \mid \gamma \mathbf{U} \psi \quad ,$$

where $a \in \mathcal{AP}$. The semantic interpretations of the temporal operators are informally given as:

$\mathbf{X}\varphi$: φ holds in the next state (next)

$\varphi_1 \mathbf{U} \varphi_2$: φ_1 holds until φ_2 holds (until)

The formal statisfaction relation is defined over traces represented by infinite words τ over the alphabet $2^{\mathcal{AP}}$. We denote the positions of the trace as $\tau(i)$ and inductively define the satisfaction of an *LTL* formula at the position $i \in \mathbb{N}_0$ as follows:

$$
\begin{array}{lll}
\tau, i \models a \text{ for } a \in \mathcal{AP} & \textit{iff} & a \in \tau(i) \\
\tau, i \models \top & \textit{iff} & \textit{true} \\
\tau, i \models \bot & \textit{iff} & \textit{false} \\
\tau, i \models \gamma \wedge \psi & \textit{iff} & \tau, i \models \gamma \text{ and } \tau, i \models \psi \\
\tau, i \models \neg\varphi & \textit{iff} & \tau, i \not\models \varphi \\
\tau, i \models \mathbf{X}\varphi & \textit{iff} & \tau, i+1 \models \varphi \\
\tau, i \models \gamma \mathbf{U} \psi & \textit{iff} & \exists k \geq 0 : \tau, i+k \models \psi \text{ and } \forall l \in [0, k-1] : \tau, i+l \models \gamma
\end{array}
$$

The derived operators **F** (eventually) and **G** (once) can be defined in terms of the basic syntax as follows:

$$\mathbf{F}\varphi := \top \mathbf{U} \varphi \qquad \mathbf{G}\varphi := \neg\mathbf{F}\neg\varphi$$

2.2.2 *LTL* Interval Notation

We introduce intervals as a syntactic extension for *LTL*, similar to [45]. This interval notation is also supported in Spot [41], a widely used library and model checker for working with temporal logics like LTL. From now on let τ denote a state trace, $i, a \in \mathbb{N}_0$ and $b \in \mathbb{N}_0 \cup \{\infty\}$.

Definition 2.1. Let $c \in \mathbb{N}_0$:

$$\mathbf{X}_{[a]}\varphi := \underbrace{\mathbf{X}\mathbf{X}...\mathbf{X}}_{\text{a times}}\varphi$$

$$\gamma \mathbf{U}_{[a,c]} \psi := \bigvee_{k \in [a,c]} \Big(\big(\bigwedge_{l \in [0,k-1]} \mathbf{X}_{[l]} \gamma \big) \wedge \mathbf{X}_{[k]} \psi \Big) \qquad \gamma \mathbf{U}_{[a,\infty]} \psi := \Big(\bigwedge_{k \in [0,a-1]} \mathbf{X}_{[k]} \gamma \Big) \wedge \mathbf{X}_{[a]} (\gamma \mathbf{U} \psi)$$

$$\mathbf{F}_{[a,b]}\varphi := \top \mathbf{U}_{[a,b]} \varphi \qquad \mathbf{G}_{[a,b]}\varphi ::= \neg\mathbf{F}_{[a,b]}\neg\varphi$$

We now introduce the satisfaction relation for *LTL* operators with intervals in lemmas 2.1- 2.3.

Lemma 2.1. *Satisfaction relation for* $\mathbf{X}_{[a]}\varphi$*:*

$$\tau, i \models \mathbf{X}_{[a]}\varphi \quad \textit{iff} \quad \tau, i+a \models \varphi$$

Proof. We prove the lemma by induction on a

Base Case: $a = 0$

$$\tau, i \models \mathbf{X}_{[0]}\varphi \quad \iff \quad \tau, i+0 \models \varphi$$

Induction Step: $a \in \mathbb{N}_0$
Induction Hypothesis: $\tau, i \models \mathbf{X}_{[a]}\varphi \quad \iff \quad \tau, i+a \models \varphi$

$$\begin{aligned}
&\tau, i \models \mathbf{X}_{[a+1]}\varphi \\
&\iff \tau, i \models \mathbf{X}_{[a]}(\mathbf{X}\varphi) && \mid \text{Def } \mathbf{X}_{[a]} \\
&\iff \tau, i+a \models \mathbf{X}\varphi && \mid \textit{I.H.} \\
&\iff \tau, i+(a+1) \models \varphi && \mid \text{Def. } \mathbf{X} \qquad \square
\end{aligned}$$

Lemma 2.2. *Satisfaction relation for* $\mathbf{U}_{[a,b]}\varphi$*:*

$$\tau, i \models \gamma \mathbf{U}_{[a,b]}\varphi \quad \textit{iff} \quad \exists k \in [a,b] : \tau, i+k \models \varphi \;\; \textit{and} \;\; \forall l \in [0,k-1] : \; \tau, i \models \gamma$$

Proof.
Case 1: $b \in \mathbb{N}_0$

$$\begin{aligned}
&\tau, i \models \gamma \mathbf{U}_{[a,b]}\psi \\
&\iff \tau, i \models \bigvee_{k\in[a,b]} \Big(\big(\bigwedge_{l\in[0,k-1]} \mathbf{X}_{[l]}\gamma \big) \wedge \mathbf{X}_{[k]}\psi \Big) && \mid \text{Def. } \mathbf{U}_I \\
&\iff \exists k \in [a,b] : \tau, i \models \big(\bigwedge_{l\in[0,k-1]} \mathbf{X}_{[l]}\gamma \big) \wedge \mathbf{X}_{[k]}\psi && \mid \text{Def. } \vee \\
&\iff \exists k \in [a,b] : \tau, i+k \models \psi \;\text{ and }\; \tau, i \models \bigwedge_{l\in[0,k-1]} \mathbf{X}_{[l]}\gamma && \mid \text{Def. } \mathbf{X}_{[k]} \\
&\iff \exists k \in [a,b] : \tau, i+k \models \psi \;\text{ and }\; \forall l \in [0,k-1] : \tau, i \models \mathbf{X}_{[l]}\gamma && \mid \text{Def. } \wedge \\
&\iff \exists k \in [a,b] : \tau, i+k \models \psi \;\text{ and }\; \forall l \in [0,k-1] : \tau, i+l \models \gamma && \mid \text{Def. } \mathbf{X}_{[l]}
\end{aligned}$$

Case 2: $b = \infty$

$$\begin{aligned}
&\tau, i \models \gamma \mathbf{U}_{[a,b]}\psi \\
&\iff \tau, i \models \big(\bigwedge_{k\in[0,a-1]} \mathbf{X}_{[k]}\gamma \big) \wedge \tau, i \models \mathbf{X}_{[a]}(\gamma \mathbf{U}\psi) && \mid \text{Def. } \mathbf{U}_I \\
&\iff \forall k \in [0,a-1] : \tau, i \models \mathbf{X}_{[k]}\gamma \;\text{ and }\; \tau, i+a \models \gamma \mathbf{U}\psi && \mid \text{Def. } \wedge \text{ / } \mathbf{X}_{[a]} \\
&\iff \forall k \in [0,a-1] : \tau, i+k \models \gamma \text{ and } \tau, i+a \models \gamma \mathbf{U}\psi && \mid \text{Def. } \mathbf{X}_{[k]} \\
&\iff \forall k \in [0,a-1] : \tau, i+k \models \gamma \text{ and } \exists j \in [a,b] : \tau, i+j \models \psi && \mid \text{Def. } \mathbf{U} \\
&\qquad \text{and } \forall n \in [a,j-1] : \tau, i+n \models \gamma \\
&\iff \exists k \in [a,b] : \tau, i+k \models \psi \;\text{ and }\; \forall l \in [0,k-1] : \tau, i+l \models \gamma && \qquad \square
\end{aligned}$$

Lemma 2.3. *For the derived operators* $\mathbf{F}$ *and* $\mathbf{G}$ *the following satisfaction relations hold:*

$$\tau, i \models \mathbf{F}_{[a,b]}\varphi \quad \textit{iff} \quad \exists k \in [a,b] : \tau, i+k \models \varphi$$

$$\tau, i \models \mathbf{G}_{[a,b]}\varphi \quad \textit{iff} \quad \forall k \in [a,b] : \tau, i+k \models \varphi$$

The proof can easily be constructed by using lemma 2.2 and the respective definitions for **F** and **G**.

2.2.3 LTL over finite traces

To evaluate the simplifications in the context of autonomous driving, we interpret the LTL formula over finite traces (LTL_f), motivated by [5]. The syntax of LTL_f is equivalent to that of LTL. For the satisfaction relation over finite traces [46], only the temporal operators are affected. Given a finite state trace τ over the alphabet $2^{\mathcal{AP}}$, the satisfaction relation of the temporal operators is defined as follows:

$$\begin{aligned}
\tau, i \models \mathbf{X}\varphi \quad & \textit{iff} \quad \tau, i+1 \models \varphi \text{ and } i+1 < \mathit{length}(\tau) \\
\tau, i \models \gamma \mathbf{U} \psi \quad & \textit{iff} \quad \exists k \geq 0 : \tau, i+k \models \gamma \text{ and } i+k < \mathit{length}(\tau) \\
& \quad \text{and } \forall l \in [0, k-1] : \tau, i+l \models \gamma \\
\tau, i \models \mathbf{F}\varphi \quad & \textit{iff} \quad \tau, i \models \top \mathbf{U} \varphi \\
\tau, i \models \mathbf{G}\varphi \quad & \textit{iff} \quad \tau, i \models \neg \mathbf{F} \neg \varphi
\end{aligned}$$

2.2.4 Finite Automata

Finite automata are finite-state machines widely used in formal language theory and model checking. They can be used for verifying whether finite trace τ satisifies a LTL_f formula φ. The language of φ, denoted as $\mathcal{L}(\varphi)$, is the set of finite words τ over the alphabet $2^{\mathcal{AP}}$ that satisfy φ, i.e., $\mathcal{L}(\varphi) := \{\tau \mid \tau, 0 \models \varphi\}$. To verify whether a given finite trace τ satisfies the LTL_f formula φ, we can construct an automaton $\mathcal{A}_\varphi$ that accepts precisely the language $\mathcal{L}(\varphi)$. In other words, the language of the finite automaton $\mathcal{A}_\varphi$ i.e. $\mathcal{L}(\mathcal{A}_\varphi) := \{\tau \mid \mathcal{A}_\varphi \text{ accepts } \tau\}$ should be equal to $\mathcal{L}(\varphi)$.

The automata $\mathcal{A}_\varphi$ can be represented as a directed Graph with a set of states and transitions. Each transition has a condition associated with it and the the trace is accepted if and only if it ends in a final state. We use Spot [47] for the translation of LTL_f formulas to finite automata. The finite automata generated by Spot will be used for evaluating the simplification algorithm introduced in Section 3.

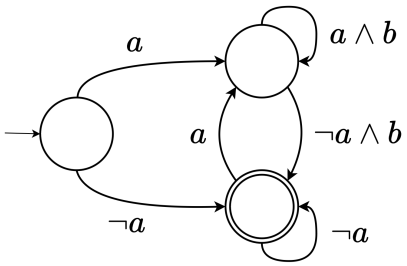


Figure 2.1: Example of a finite automata for the formula $\varphi = \mathbf{G}\big(a \rightarrow \mathbf{X}(b)\big)$ with $a, b \in \mathcal{AP}$

2.3 Road Network

This section introduces the road network representation used in this work. We cover the most important notations and concepts needed to follow the formalization of traffic rules, and we refer the reader to [2] where it is explained in more depth. The drivable area is represented by lanelets of which each is defined by two polylines which serve as the left and right boundary. We denote the set of all lanelets as $\mathcal{L} \cup \{\bot\}$, where $\bot$ represents the reference plane on which all other lanelets lie. We can extract attributes for a single lanelet $l \in \mathcal{L}$ with the functions defined in Table 2.1. Furthermore, the functions defined in Table 2.2 allow us to extract elements from the road network. If $\bot$ is supplied to any of the functions, either $\bot$ or $\emptyset$ is returned.

Table 2.1: Functions for extracting attributes about lanelets [2]

Function	**Description**
$type(l) \in \mathcal{T} \cup \{\bot\}$	Returns the type of the lanlet where $\mathcal{T} = \{$ *access_ramp*, *exit_ramp*, *main_carriageway*, *shoulder*$\}$
$mark_l(l), mark_r(l) \in \mathcal{LM} \cup \{\bot\}$	return the left and right linemarkings respectively with $\mathcal{LM} = \{$*solid*, *dashed*, *broad_solid*, *broad_dashed*$\}$
$attr(l) \in \mathcal{A} \cup \{\bot\}$	return the attribute of the lanelet where $\mathcal{A} = \{$*fork*, *merge*$\}$
$\mathrm{l}_{lb}(l)$, $\mathrm{l}_{rb}(l)$	return the left and right boundaries of l respectively
$\mathrm{occ}(l)$	return the spatial occupancy of the lanelet
$I_{ini}(l), I_{fin}(l)$	return the initial and final points of the lanelet

Table 2.2: Functions for extracting elements form the road network [3]

Name	**Formalization**
Predecessor lanelets	$\mathrm{pre}(l) = \{l' \in \mathcal{L} \mid \mathrm{l}_{ini}(l) = \mathrm{l}_{fin}(l')\}$
Successor lanelets	$\mathrm{suc}(l) = \{l' \in \mathcal{L} \mid \mathrm{l}_{fin}(l) = \mathrm{l}_{ini}(l')\}$
Adjacent left lanelet	$\mathrm{adj}_l(l) = \begin{cases} l' & \text{if } l' \in \mathcal{L} \ \wedge \ \mathrm{occ}(\mathrm{l}_{lb}(l)) \subseteq \mathrm{occ}(l') \\ & \wedge(\mathrm{l}_{ini}(l) \cap (\mathrm{l}_{ini}(l') \neq \emptyset \wedge \mathrm{l}_{fin}(l) \cap \mathrm{l}_{fin}(l') \neq \emptyset \\ & \vee \mathrm{l}_{ini}(l) \cap \mathrm{l}_{fin}(l') \neq \emptyset \ \wedge \ \mathrm{l}_{fin}(l) \cap \mathrm{l}_{fin}(l') \neq \emptyset) \\ \bot & \text{otherwise} \end{cases}$
Adjacent right lanelet	$\mathrm{adj}_r(l)$ - Identical to $\mathrm{adj}_l(l)$, except that $\mathrm{l}_{lb}(l)$ is replaced by $\mathrm{l}_{rb}(l)$
Lane predecessors	$\mathrm{lanes}_{pre}(la, l) = \begin{cases} \left(\bigcup_{pre \in \mathrm{pre}(l)} (\mathrm{lanes}_{pre}(la \cup \{l\}, pre))\right) & \text{if } \mathrm{pre}(l) \neq \emptyset \wedge l \notin la \\ \{la \cup \{l\}\} & \text{otherwise} \end{cases}$
Lane successors	$\mathrm{lanes}_{suc}(la, l)$ - Identical to $\mathrm{lanes}_{pre}(la, l)$, except that $\mathrm{pre}(l)$ is replaced with $\mathrm{suc}(l)$
Lanes	$\mathrm{lanes}(l) = \{p \cup s \mid p \in \mathrm{lanes}_{pre}(\emptyset, l) \wedge s \in \mathrm{lanes}_{suc}(\emptyset, l)\}$

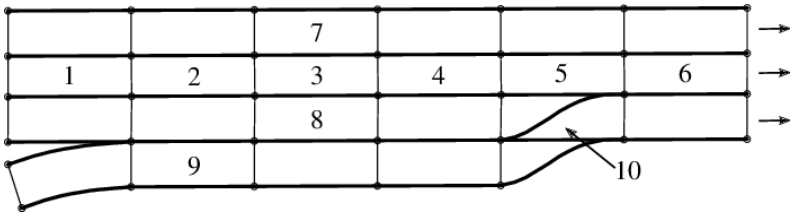


Figure 2.2: Example from [2] of a road network defined by lanelets. Relative to lanelet 3, lanelets 7 and 8 are adjacent left and right and lanelet 2 and 4 are the predecessor and successor lanelets respectively. Lanelets 1-6 form a lane. Lanelets 9 and 10 are of type *access_ramp* and lanelet 10 is additionally of type *merge*. Lanelets 1-8 are of type *main_carriageway*.

2.4 Vehicle Model

This chapter introduces the vehicle model from [2] which is used in this work. We employ a curvilinear coordinate system [48] to describe the position of the ego vehicle, which is defined by the reference path Γ computed by a route planner in a preprocessing step. The state of a vehicle consists of the longitudinal state $x_{lon} = [s \quad v \quad a \quad j]^T$ defined by the position s, velocity v, acceleration a and jerk j and the lateral state $x_{lat} = [d \quad \theta]$ defined by the lateral distance to the reference path d and the orientation θ.

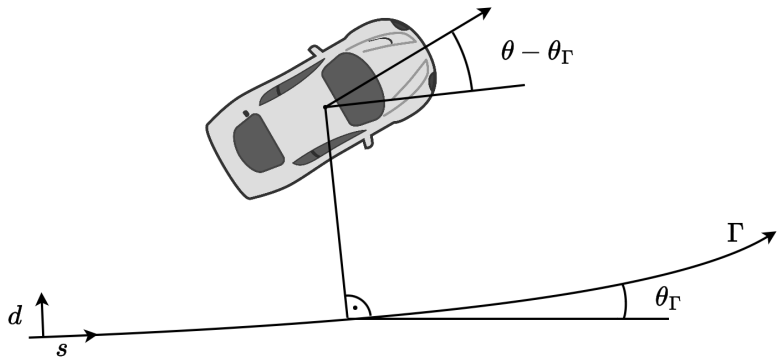


Figure 2.3: Example adapted from [2] of a curve linear coordinate system fitted to the reference path Γ. The longitudinal position is given by s and the lateral distance to the reference path is given by d. The orientation of the reference path and the vehicle are denoted by θ_Γ and θ respectively.

The vehicle input at time t is denoted by $u(t) \in \mathcal{U}$, where $\mathcal{U}$ describes the set of possible inputs. Given the system dynamics described by $\dot{x} = f(x(t), u(t))$ we denote $x(t) = \xi(t; x_0; u(\cdot))$ as the solution of the differential equation at time t, where $u(\cdot)$ denotes the input trajectory and x_0 the initial state. The operator $\text{proj}_\square(x)$ is used to project x to the elements specified by $\square$. The spatial occupancy of a state is given

by $\mathcal{O}(x(t))$. The longitudinal positions of the front bumper and rear bumper can be optained with front$(x(t))$ and rear$(x(t))$ respectively. Similary the lateral positions of the leftmost and rightmost points of the car can be obtained with left$(x(t))$ and right$(x(t))$. We use $\square_{ego}$ to denote variables asscociated with the ego vehicle, and denote all other variables with $\square_o$. We introduce the index $k \in \mathbb{N}_0$ to represent discrete time steps corresponding to continuous time $t_k = k\Delta t$, where $\Delta t \in \mathbb{R}_{>0}$ denotes a fixed time increment.

2.5 Traffic Rule Formalization

This section introduces the formalization of traffic rules using temporal logic. The traffic rules are derived from the German Road Traffic Regulation (StVO) and the Vienna Convention on Road Traffic (VCoRT). We present a traffic rule from [2] and its respective predicates in this section to illustrate how the formalization works and to use it as example later on. The formalization is converted from *MTL* used in [2] to *LTL* for this work.

2.5.1 Functions & Predicates

To be able to define the entering vehicles rule from [2] we first introduce the used functions and predicates building on section 2.3 and 2.4. The set of occupied lanelets of a vehicle k contains any lanelet who's spacial occupancy intersectes with the vehicle shape:

$$\text{lanelets}(x_k) = \{l \in \mathcal{L} \mid \text{occ}(l) \cap \mathcal{O}(x_k) \neq \emptyset\}$$

The lanes occupied by a vehicle are similarly defined as:

$$\text{lanes}(x_k) = \bigcup_{l \in \text{lanelets}(x_k)} \text{lanes}(l)$$

Two vehicles are in the same lane if there exists a lane both of them occupy:

$$\text{in\_same\_lane}(x_k, x_p) \iff \text{lanes}(x_k) \cap \text{lanes}(x_p) \neq \emptyset$$

Vehicle p is in front of vehicle k if the longitudinal position of the rear of vehicle p is greater than the longitudinal position of the front of vehicle k

$$\text{in\_front\_of}(x_k, x_p) \iff \text{front}(x_k) < \text{rear}(x_p)$$

The vehicle k is on the main carriage way if any of the occupied lanelets has the type *main_carriageway*:

$$\text{on\_main\_carriageway}(x_k) \iff \text{main\_carriageway} \in \{\text{type}(l) \mid l \in \text{lanelets}(x_k)\}$$

The vehicle k is on the rightmost lane of the main carriageway if occupies a lanelet of set type which has no right adjacent lanelet of the type *main_carriageway*.

$$\text{right\_lane}(x_k) \iff \exists l \in \text{lanelets}(x_k) :$$

$$\text{type}(l) = \text{main\_carriageway} \wedge \text{type}(\text{adj}_r(l)) \neq \text{main\_carriageway}$$

2.5.2 Entering Vehicles Rule

Given the definitions above we can define the Entering Vehicles Rule (§1(2)StVO;[32] StVO § 18 Rn. 11). This rule asserts that if there is a vehicle on the access ramp which is in front of the ego vehicle, the ego vehicle may not change into the rightmost lane of the main carriageway to not hinder this vehicle from entering the main carriageway.

$$\mathbf{G}\Big(\text{on\_main\_carriageway}(x_{ego}) \wedge \text{in\_front\_of}(x_{ego}, x_0) \wedge \text{on\_access\_ramp}(x_0)$$
$$\wedge\ \mathbf{F}\big(\text{on\_main\_carriageway}(x_0)\big) \implies$$
$$\neg(\neg\text{right\_lane}(x_{ego}) \wedge \mathbf{F}\big(\text{right\_lane}(x_{ego})\big)\Big)\Big)$$

2.6 Reachability Analysis using LTL Specifications

This section briefly introduces the concept of specification-compliant reachability analysis of autonomous vehicles. A reachability analysis is concerned with determining the possible states a vehicle can be in at different points in time. The reachability analysis described in [45] computes an overapproximation of the reachable states of the ego vehicle which conform to a given set of LTL specifications. These specification-compliant driving corridors can then be used to accelerate the generation of feasible vehicle trajectories for motion planners, as non-reachable and non-compliant states can be excluded a priori. The algorithm by Florian Lercher and Matthias Althoff [5] build on this approach by employing *on-the-fly* model checking. The algorithm adopts an automata-based model checking approach, facilitating the early pruning of states that do not comply with the given LTL specifications. It utilizes finite automata (see Section 2.2), generated by the Spot tool. The implementation of this algorithm is used in Section 6 for the evaluation of this work.

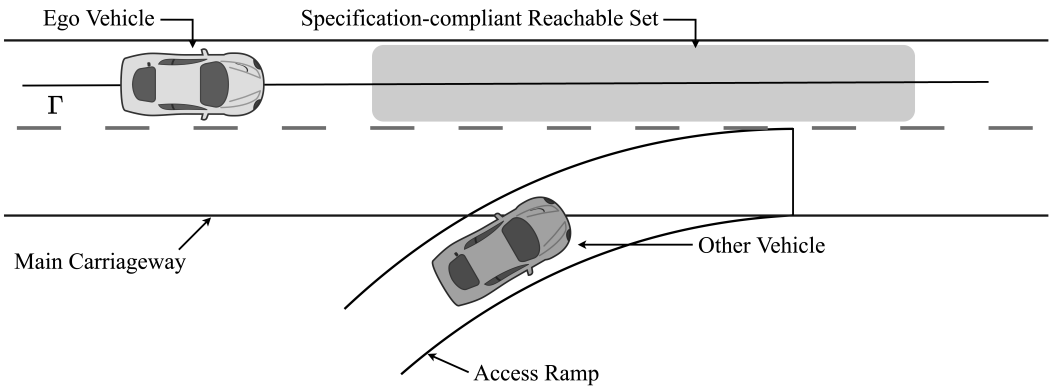


Figure 2.4: Example of a reachability analysis adhering to the Entering Vehicles Rule defined in Section 2.5. As the other vehicle enteres the main carriage way via an access ramp the ego vehicle is not allowed to switch to the rightmost lane.

3 Simplification of LTL Formulas

In this chapter, we introduce a novel algorithm for the simplification of arbitrary LTL formulas using partial knowledge about the state trace. Given a LTL formula φ and information about the Boolean values of propositions at any position in the trace τ, the algorithm returns a simplified formula φ' that is satisfied if and only if φ is satisfied. The goal of the simplification is to reduce the computational effort of checking the validity of the resulting formula. Thus, we use the partial information about the state trace to reduce the number of positions that need to be checked during the verification of the formula as explained in section 3.1.

3.1 Problem Statement

As explained above, the objective of the algorithm, labeled as `IntervalSimplification`, is to produce a simplified formula based on the partial knowledge of the state trace that is satisfied if and only if the original formula is satisfied. This partial knowledge about a state trace τ at $i \in \mathbb{N}_0$ is encoded in $\mathcal{P}$ as follows:

$$\mathcal{P}(a,i) := \begin{cases} \top & \text{we know: } a \in \tau(i) \\ \bot & \text{we know: } a \notin \tau(i) \\ a & \text{otherwise} \end{cases} \quad \text{where } a \in \mathcal{AP}$$

We use $\equiv_{(\tau,i)}$ to denote the semantic equivalence of two formulas at a given position in a trace. Specifically, $\varphi \equiv_{(\tau,i)} \gamma$ iff $\tau, i \models \varphi \Leftrightarrow \tau, i \models \gamma$. For syntactic equivalence, we use $=$.

Definition 3.1. The simplification of a formula φ, evaluated at position $i \in \mathbb{N}_0$ in the trace is represented as a mapping from trace positions to simplified formulas:

$$\mathcal{S}_\varphi(i) := \varphi' \quad , \quad \text{where } \varphi' \text{ is the simplification of } \varphi \text{ at position } i$$

We refer to a formula φ' as a simplification of φ if it reduces the number of positions in the state trace at which propositions need to be evaluated. For instance, consider the formula:

$$\varphi = \mathbf{G}(a) \quad \text{where } a \in \mathcal{AP}$$

If the proposition a is known to be true at position 4 in the state trace, denoted as $\mathcal{P}(a,4) = \top$, then the simplification process, given no further information, will produce the following result:

$$\mathcal{S}_\varphi(0) = \mathbf{G}_{[0,3]}(a) \wedge \mathbf{G}_{[5,\infty]}(a)$$

In this example, φ is simplified by acknowledging that a holds at position 4, thus allowing us to split the globally quantified formula $G(a)$ into two intervals: $[0, 3]$ and $[5, \infty]$.

Definition 3.2. Let φ be a *LTL* formula and τ a state trace. A simplification mapping $\mathcal{S}_\varphi$ is called *sound* if and only if for any position i in the state trace τ, the following holds:

$$\mathcal{S}_\varphi(i) \equiv_{(\tau,i)} \varphi$$

In addition to defining the simplification, showing that the resulting mapping is sound is the primary objective of this chapter. To better work with simplification mappings, we introduce the following notations:

1. $\mathcal{S}_\varphi^{-1}(\varphi')$ denotes the set of positions i where the simplified formula $\mathcal{S}_\varphi(i)$ is syntactically equivalent to φ':
$$\mathcal{S}_\varphi^{-1}(\varphi') := \{i \mid \varphi' = \mathcal{S}_\varphi(i)\}$$
2. $\mathcal{S}_\varphi(I)$ denotes the set of simplified formulas for all positions i in the set I:
$$\mathcal{S}_\varphi(I) := \{\mathcal{S}_\varphi(i) \mid i \in I\}$$

Furthermore, we call a subformula a formula that is part of another formula. For example, for $\varphi = \mathbf{G}\gamma$, γ would be the subformula and φ would be the superformula.

3.2 Interval Simplification Algorithm

We define the problem statement as follows: Given a *LTL* formula φ and partial knowledge $\mathcal{P}$ of the trace τ, find the simplification of φ at $i = 0$. The `IntervalSimplification` algorithm works recursively on the structure of the formula φ and relies on the following two subfuntions:

1. `IntervalPropagation`. Before defining the simplification mapping $\mathcal{S}_\varphi$, we need to know for which positions i in the trace we need to define the mapping. Generating these positions is done by `IntervalPropagation` which determines them based on the positions at which the mapping for the super function has to be defined.
2. `Simplify`. This function computes the simplification mapping of the formula based on the simplification mapping of the subformula(s) at the trace positions generated by `IntervalPropagation`.

In the following definition $\mathscr{B}$ and $\mathscr{U}$ denote binary and unary operators respectively.

Algorithm 1 IntervalSimplification

Input: φ : *LTL* Formula
$\mathcal{I}^{\varphi}$: Evaluation Interval (initally: $[0,0]$)
$\mathcal{P}$: Partial Knowledge Mapping
Output: $\mathcal{S}_{\varphi}$: Mapping of trace positions to *LTL* formulas

match φ **with:**
  case $a \in \mathcal{AP}$:
    $\mathcal{S}_{\varphi} \leftarrow$ Simplify$(a, \mathcal{I}_a, \mathcal{P})$

  case $\mathscr{U}\,\gamma$:
    $\mathcal{I}^{\gamma} \leftarrow$ PropagateInterval$(\mathscr{U}, \mathcal{I}_{\varphi})$
    $\mathcal{S}_{\gamma} \leftarrow$ IntervalSimplification$(\gamma, \mathcal{I}_{\gamma}, \mathcal{P})$
    $\mathcal{S}_{\varphi} \leftarrow$ Simplify$(\mathscr{U}, \mathcal{I}_{\varphi}, S_{\gamma})$

  case $\gamma \mathscr{B} \psi$:
    $\mathcal{I}^{\gamma}, \mathcal{I}^{\psi} \leftarrow$ PropagateInterval$(\mathscr{B}, \mathcal{I}^{\varphi})$
    $S_{\gamma} \leftarrow$ IntervalSimplification$(\gamma, \mathcal{I}^{\phi}, \mathcal{P})$
    $S_{\psi} \leftarrow$ IntervalSimplification$(\psi, \mathcal{I}^{\psi}, \mathcal{P})$
    $S_{\varphi} \leftarrow$ Simplify$(\mathscr{B}, \mathcal{I}_{\varphi}, S_{\gamma}, S_{\psi})$

return $\mathcal{S}_{\varphi}$

3.3 Subfunction: PropagateInterval

The subfunction `PropagateInterval` computes the set of trace positions at which a subformula can be evaluated given the set of positions at which the super formula can be evaluated. Let $\mathcal{I}^{\varphi}$ be the interval for which φ should be simplified, which is initially $[0,0]$, with $a \in \mathbb{N}_0$ and $b \in \mathbb{N}_0 \cup \{\infty\}$. `PropagateInterval` is then recursively defined as follows:

$$
\begin{aligned}
\varphi = \gamma \wedge \psi \quad &: \quad \mathcal{I}^{\gamma} := \mathcal{I}^{\varphi} \quad \text{and} \quad \mathcal{I}^{\psi} := \mathcal{I}^{\varphi} \\
\varphi = \neg\gamma \quad &: \quad \mathcal{I}^{\gamma} := \mathcal{I}^{\varphi} \\
\varphi := \mathbf{X}_{[a]}\gamma \quad &: \quad \mathcal{I}^{\gamma} := \mathcal{I}^{\varphi} \oplus [a,a] \\
\varphi = \gamma \mathbf{U}_{[a,b]} \psi \quad &: \quad \mathcal{I}^{\gamma} := \mathcal{I}^{\varphi} \oplus [0, b-1] \quad \text{and} \quad \mathcal{I}^{\psi} := \mathcal{I}^{\varphi} \oplus [a,b] \\
\varphi := \mathbf{F}_{[a,b]}\gamma \quad &: \quad \mathcal{I}^{\gamma} := \mathcal{I}^{\varphi} \oplus [a,b] \\
\varphi := \mathbf{G}_{[a,b]}\gamma \quad &: \quad \mathcal{I}^{\gamma} := \mathcal{I}^{\varphi} \oplus [a,b]
\end{aligned}
$$

The reader can easily verify the validity of these definitions by referring to the satisfaction relations of the operators, presented in Section 2.2.

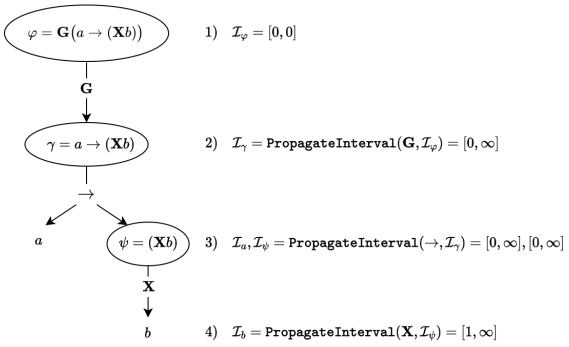


Figure 3.1: Running Example of `PropagateInterval` for $\varphi = \mathbf{G}\big(a \rightarrow (\mathbf{X}b)\big)$

3.4 Subfunction: Simplify

The function `Simplify` is the heart of the algorithm. It takes the respective simplifications of the subformulas and returns the simplifications for the superformula at each position in $\mathcal{I}^\varphi$. As explained in Section 3.1 our simplification has the goal of eliminating all proposition evaluations during the validation of the formula that we can exclude with the given information about state trace τ. In the following sections we will define the recursive definition of this simplification for each operator.

3.4.1 Simplification of Prpopositions

The simplification is based on partial knowledge about the state trace, which is encoded as $\mathcal{P}$. As explained in Section 3.1, $\mathcal{P}$ is a partial mapping of trace positions to boolean values. This mapping is use case specific which makes the algorithm adaptable to any scenario incorporating *LTL* specifications. We demonstrate this in Section 4 with the simplification of traffic rules.

3.4.2 Non-Temporal Operators

The recursive definition of the simplification mapping $\mathcal{S}_\varphi$ for non-temporal operators is trivial, but helpful for understanding the notation. For this reason we only show the proof of the case $\varphi = \neg\gamma$ and the reader can convince himself that the proof of the case $\varphi = \gamma \wedge \psi$ is similar.

Definition 3.3. Given a *LTL* formula γ its simplification mapping $\mathcal{S}_\gamma$ we define the simplification of $\varphi = \neg\gamma$ as:

$$\mathcal{S}_\varphi(i) = \begin{cases} \top & \text{if } S_\gamma(i) = \bot \\ \bot & \text{if } S_\gamma(i) = \top \\ \neg S_\gamma(i) & \text{otherwise} \end{cases}$$

Lemma 3.1. *Let γ be a LTL formula and $\mathcal{S}_\gamma$ a sound simplification mapping. Then, the simplification mapping $\mathcal{S}_{\neg\gamma}$ produced by the* `IntervalSimplification` *algorithm is sound.*

Proof. We need to show that $\mathcal{S}_\varphi(i) \equiv_{(\tau,i)} \neg\gamma$ holds.

1. *Case:* $S_\gamma(i) = \bot$.

   $\tau, i \models \neg\gamma$

   $\iff \tau, i \models \neg S_\gamma(i) \quad | \; S_\gamma$ is sound

   $\iff \tau, i \models \top \quad | \; S_\gamma = \top$

2. *Case:* $S_\gamma(i) = \top$. The proof is analogous to the first case.

3. *Case.* This case is proven in the first two lines of the first case. □

Definition 3.4. Given two *LTL* formulas γ, ψ and their simplification mappings $\mathcal{S}_\gamma$, $\mathcal{S}_\psi$ we define the simplification of $\varphi = \gamma \wedge \psi$ as:

$$\mathcal{S}_{\gamma\wedge\psi}(i) = \begin{cases} S_\gamma(i) & \text{if } S_\psi(i) = \top \\ S_\psi(i) & \text{if } S_\gamma(i) = \top \\ \bot & \text{if } S_\gamma(i) = \bot \text{ or } \; S_\psi(i) = \bot \\ S_\gamma(i) \wedge S_\psi(i) & \text{otherwise} \end{cases}$$

Lemma 3.2. *Let γ and ψ be two LTL formulas and $\mathcal{S}_\gamma$ and $\mathcal{S}_\psi$ sound simplification mappings. Then, the simplification mapping $\mathcal{S}_{\gamma\wedge\psi}$ produced by the* `IntervalSimplification` *algorithm is sound.*

The proof is similar to the proof of lemma 3.1.

3.4.3 Simplification of the Next Operator

If the next operator $\mathbf{X}_{[a]}$ cannot be simplified to either $\top$ or $\bot$, the simplification is simply given by the simplification of the subformula at the trace position specified by a.

Definition 3.5. Given an *LTL* formula γ and simplification mapping $\mathcal{S}_\gamma$, we define the simplification of $\varphi = \mathbf{X}_{[a]}\gamma$ as:

$$\mathcal{S}_\varphi(i) := \begin{cases} \top & \text{if } \mathcal{S}_\gamma(i+a) = \top \\ \bot & \text{if } \mathcal{S}_\gamma(i+a) = \bot \\ \mathbf{X}_{[a]}\mathcal{S}_\gamma(i+a) & \text{otherwise} \end{cases}$$

Lemma 3.3. *Let γ be a LTL formula and $\mathcal{S}_\gamma$ a sound simplification mapping. Then, the simplification mapping $\mathcal{S}_\varphi$ for $\varphi = \mathbf{X}_{[a]}\gamma$ produced by the* `IntervalSimplification` *is sound.*

Using lemma 2.1 the proof can be constructed in a similar way to the proof of lamma 3.1.

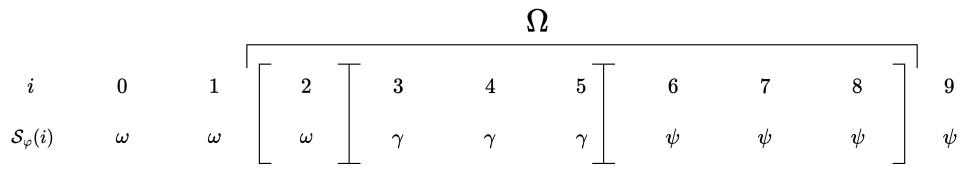
3.4.4 Simplification of the Globally Operator

To define the simplification of the globally operator we define a helper function called `Split`. The function `Split` splits a set of trace positions Ω into intervals that all correspond to the syntactically equivalent simplification of φ:

Definition 3.6. Given a set $\Omega \subseteq \mathbb{N}_0$ and a simplification mapping $S_\varphi(i)$ we define `Split` as follows:

$$\texttt{Split}(\Omega, \mathcal{S}_\varphi) := \Big\{ [x,y] \mid [x,y] \subseteq \Omega \wedge |\mathcal{S}_\varphi([x,y])| = 1 \wedge [x,y] \neq \emptyset \\ \wedge \nexists\, [a,b] \supsetneq [x,y] : |\mathcal{S}_\varphi([a,b])| = 1 \wedge [a,b] \subseteq \Omega \Big\}$$

Example. $\texttt{Split}(\Omega, \mathcal{S}_\varphi) = \{[2,2],[3,5],[6,8]\}$ given $\mathcal{S}_\varphi$ and $\Omega = [2,8]$:



Remark 3.4. *Given a simplification mapping $\mathcal{S}_\varphi$ and $\Omega \subseteq \mathbb{N}_0$, we can easily see that the following holds:*

$$\bigcup_{[x,y] \in \texttt{Split}(\Omega, S_\gamma)} [x,y] \quad = \quad \Omega$$

We can now recursively define the simplification mapping of $\mathbf{G}_{[a,b]}\gamma$.

Definition 3.7. Given an *LTL* formula γ and simplification mapping $\mathcal{S}_\gamma$, we define the simplification mapping of $\varphi = \mathbf{G}_{[a,b]}\gamma$ as:

$$S_{\mathbf{G}_{[a,b]}\gamma}(i) := \begin{cases} \top & \text{if } \forall n \in [a,b] : S_\gamma(i+n) = \top \\ \bot & \text{if } \exists n \in [a,b] : S_\gamma(i+n) = \bot \\ \bigwedge\limits_{\substack{[x,y] \in \texttt{Split}([a+i,b+i], S_\gamma) \\ \wedge\ \mathcal{S}_\gamma(x) \neq \top}} \mathbf{G}_{[x-i,y-i]}\ \mathcal{S}_\gamma(x) & \text{otherwise} \end{cases}$$

In the case that γ holds within the entire interval (case 1) we know that the entire expression must hold and we can thus simplify it to $\top$. Conversely, we know that the entire expression can never be satisfied if γ does not hold at every position in the interval (case 2). The intuitive explanation for the simplification of the third case is that we can split the interval $[a,b]$ in which γ must hold into smaller intervals in which only simpler expressions must hold. This is achieved by the `Split` function.

Lemma 3.5. *Let γ be a LTL formula and $\mathcal{S}_\gamma$ a sound simplification mapping. Then, the simplification mapping $\mathcal{S}_\varphi$ for $\varphi = \mathbf{G}_{[a,b]}\gamma$ produced by the IntervalSimplification is sound.*

Proof. The first two cases $\top$ and $\bot$ follow directly from Lemma 2.3. We have to show that $S_{\mathbf{G}_{[a,b]}\gamma}(i) \equiv_{(\tau,i)} \mathbf{G}_{[a,b]}\gamma$ holds for the third case:

$$
\begin{aligned}
&\tau, i \models \mathbf{G}_{[a,b]}\gamma \\
&\iff \forall n \in [a,b] : \tau, i+n \models \gamma && \mid \text{Lemma 2.3} \\
&\iff \forall n \in [a,b] : \tau, i+n \models S_\gamma(i+n) && \mid \mathcal{S}_\gamma \text{ sound} \\
&\iff \forall n \in [i+a, i+b] : \tau, n \models S_\gamma(n) \\
&\iff \forall [x,y] \in \texttt{Split}([i+a, i+b],\ S_\gamma) : \forall n \in [x,y] : \tau, n \models S_\gamma(x) && \mid *_1 \\
&\iff \forall [x,y] \in \texttt{Split}([i+a, i+b],\ S_\gamma) : \tau, i \models \mathbf{G}_{[x-i,y-i]}\mathcal{S}_\gamma(x) && \mid \text{Lemma 2.3} \\
&\iff \forall [x,y] \in \texttt{Split}([i+a, i+b],\ S_\gamma) \quad \text{and} \quad \mathcal{S}_\gamma(x) \neq \top : && \mid *_2 \\
&\qquad \tau, 0 \models \mathbf{G}_{[x,y]}\mathcal{S}_\gamma(x) \\
&\iff \tau, i \models \bigwedge_{\substack{[x,y] \in \texttt{Split}([a+i,b+i],S_\gamma) \\ \wedge\ \mathcal{S}_\gamma(x) \neq \top}} \mathbf{G}_{[x-i,y-i]}\ \mathcal{S}_\gamma(x)
\end{aligned}
$$

□

$*_1$ Remark 3.4 and $\mathcal{S}_\gamma(n)$ is the same for all $n \in [i+a, i+b]$
$*_2$ Here we can leave out any intervals that simplify to $\top$ because $\mathbf{G}_{[x,y]}(\top)$ will always evaluates to $\top$. This can not result in an empty formula as we assume the condition of the first case is not met and thus there has to be an interval that does not simplify to $\top$.

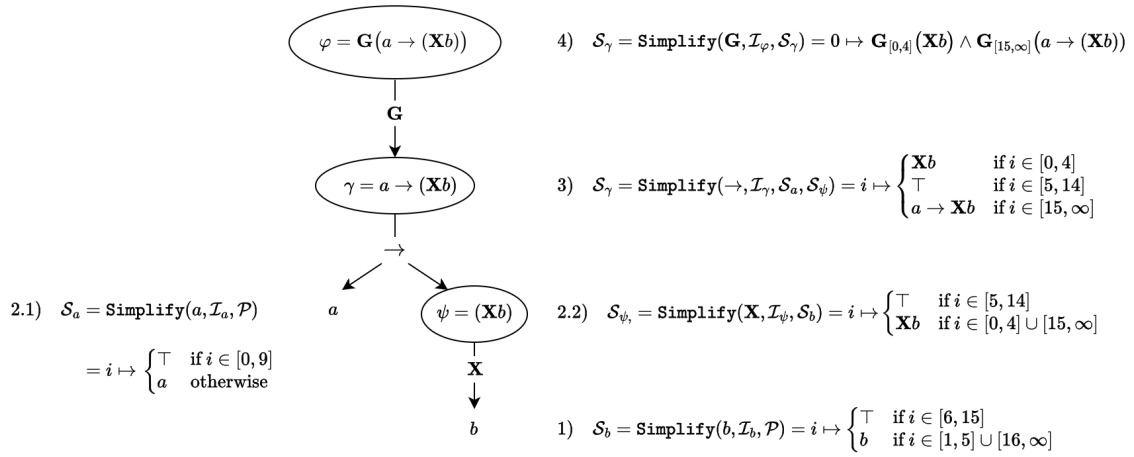


Figure 3.2: Running Example of `Simplify` for $\varphi = \mathbf{G}(a \to (\mathbf{X}b))$ given $\forall i \in [0,9] : a \in \tau(i)$ and $\forall i \in [6,15] : b \in \tau(i)$

3.4.5 Simplification of the Future Operator

The simplification of the future operator is defined similary to the globally operator. We make use of the `Split` function again but now exclude $\bot$s which will not contribute to the satisfaction of the formula.

Definition 3.8. Given an *LTL* formula γ and simplification mapping $\mathcal{S}_\gamma$, we define the simplification mapping of $\varphi = \mathbf{F}_{[a,b]}\gamma$ as:

$$\mathcal{S}_{\mathbf{F}_{[a,b]}\gamma}(i) = \begin{cases} \top & \text{if } \exists n \in [a,b] : S_\gamma(i+n) = \top \\ \bot & \text{if } \forall n \in [a,b] : S_\gamma(i+n) = \bot \\ \bigvee\limits_{\substack{[x,y]\in \texttt{Split}([a+i,b+i],S_\gamma) \\ \wedge\ S_\gamma(x)\neq\bot}} \mathbf{F}_{[x-i,y-i]}\ \mathcal{S}_\gamma(x) & \text{otherwise} \end{cases}$$

Lemma 3.6. *Let γ be a LTL formula and $\mathcal{S}_\gamma$ a sound simplification mapping. Then, the simplification mapping $\mathcal{S}_\varphi$ for $\varphi = \mathbf{F}_{[a,b]}\gamma$ produced by the IntervalSimplification is sound.*

Using Lemma 2.3 and substituting $\bot$ for $\top$, the proof is analogous to **G**.

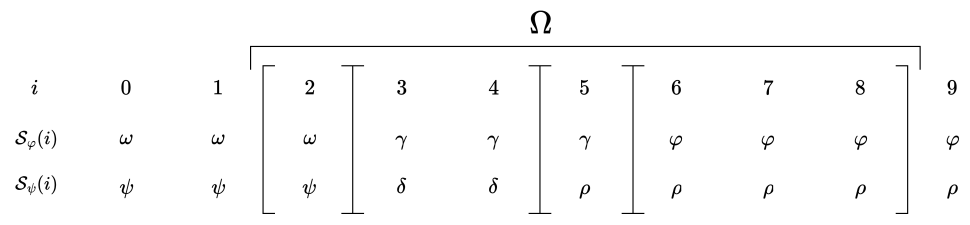
3.4.6 Simplification of the Until Operator

To define the simplification of the until operator we extend the definition of the `Split` function to handle two simplification mappings as input. Now, the returned intervals correspond to sets of trace positions in which the syntactic definition of the two subformulas do not change.

Definition 3.9. Given a set $\Omega \subseteq \mathbb{N}_0$ and two simplification mappings $S_\varphi(i)$ and $S_\psi(i)$:

$$\texttt{Split}(\Omega, S_\gamma, S_\psi) = \Big\{\ [x,y]\ \mid [x,y] = I_\gamma \cap I_\psi\ \wedge\ [x,y] \neq \emptyset \\ \wedge\ I_\gamma \in \texttt{Split}(\Omega, \mathcal{S}_\gamma)\ \wedge\ I_\psi \in \texttt{Split}(\Omega, \mathcal{S}_\psi)\Big\}$$

Example. $\texttt{Split}(\Omega, \mathcal{S}_\varphi, \mathcal{S}_\psi) = \{[2,2],[3,4],[5,5],[6,8]\}$ given $\mathcal{S}_\varphi$, $\mathcal{S}_\psi$ and $\Omega = [2,8]$:



Remark 3.7. *Given two simplification mappings $\mathcal{S}_\varphi$,$\mathcal{S}_\psi$ and $\Omega \subseteq \mathbb{N}_0$, we can again easily convince ourselves that the following holds:*

$$\bigcup_{[x,y]\in \mathtt{Split}(\Omega,\mathcal{S}_\varphi,\mathcal{S}_\psi)} [x,y] \quad = \quad \Omega$$

Definition 3.10. Given two *LTL* formulas γ, ψ and their simplification mappings $\mathcal{S}_\gamma$, $\mathcal{S}_\psi$ we define the simplification of $\varphi = \gamma \mathbf{U}_{[a,b]} \psi$ as:

$$\mathcal{S}_{\gamma \mathbf{U}_{[a,b]}\psi}(i) = \begin{cases} \top & \text{if } \exists n \in [a,b] : \mathcal{S}_\psi(i{+}n) = \top \wedge \forall n' \in [0,n-1] : S_\gamma(t{+}n') = \top \\ \bot & \text{if } \forall n \in [a,b] : S_\psi(i{+}n) = \bot \vee \exists n \in \mathbb{N}_0 : S_\psi(i{+}n) = \bot \\ & \qquad \wedge \forall n' \in [0,n] \cap [a,b] : S_\gamma(i+n') = \bot \\ \bigvee\limits_{\substack{[x,y]\in \\ \mathtt{Split}([a+t,b+t],\mathcal{S}_\gamma,\mathcal{S}_\psi)}} & \mathcal{S}_{\mathbf{G}_{[0,x-i-1]}\gamma}(i) \wedge \mathbf{X}_{[x-i]}\big(\mathcal{S}_\gamma(x)\mathbf{U}_{[0,y-t-x]}\mathcal{S}_\psi(x)\big) \qquad \text{otherwise} \end{cases}$$

To give an intuitive explanation of the simplification, $\gamma \mathbf{U}_{[a,b]} \psi$ must hold in at least one of the intervals returned by `Split` which splits the interval $[a,b]$. This is ensured by the second part after $\wedge$. Additionally γ needs to hold until ψ holds in [a,b] which is ensured by the first part before the $\wedge$.

Lemma 3.8. *Let γ and ψ be two LTL formulas and $\mathcal{S}_\gamma$ and $\mathcal{S}_\psi$ sound simplification mappings. Then, the simplification mapping $\mathcal{S}_{\gamma \mathbf{U}_{[a,b]}\psi}$ produced by the IntervalSimplification algorithm is sound.*

Proof. The first two cases $\top$ and $\bot$ follow directly from Lemma 2.2. We have to show that $S_{\gamma \mathbf{U}_{[a,b]}\psi}(i) \equiv_{(\tau,i)} \gamma \mathbf{U}_{[a,b]} \psi$ holds for the third case:

$$\begin{aligned}
&\tau, i \models \gamma \, \mathbf{U}_{[a,b]} \, \psi \\
&\iff \exists n \in [a,b] : \tau, i{+}n \models \psi \ \wedge \ \forall n' \in [0,n{-}1] : \tau, i{+}n' \models \gamma && \mid \text{Lemma 2.2} \\
&\iff \exists n \in [i{+}a, i{+}b] : \tau, n \models \psi \ \wedge \ \forall n' \in [i,n{-}1] : \tau, n' \models \gamma \\
&\iff \exists [x,y] \in \mathtt{Split}([i+a,i+b],\mathcal{S}_\gamma,\mathcal{S}_\psi) : \exists n \in [x,y] : \tau, n \models \psi && \mid \text{Remark 3.7} \\
&\qquad \wedge \forall n' \in [i,x{-}1] : \tau, n' \models \gamma \ \wedge \forall n' \in [x,n{-}1] : \tau, n' \models \gamma && \mid \text{Lemmas 2.2, 2.3} \\
&\iff \exists [x,y] \in \mathtt{Split}([i+a,i+b],\mathcal{S}_\gamma,\mathcal{S}_\psi) : \\
&\qquad \tau, i \models \mathbf{G}_{[0,x-1-i]}\gamma \ \wedge \ \tau, x \models \gamma \mathbf{U}_{[0,y-x]}\psi && \mid \text{Lemma 3.5} \\
&\iff \exists [x,y] \in \mathtt{Split}([i+a,i+b],\mathcal{S}_\gamma,\mathcal{S}_\psi) : \\
&\qquad \tau, i \models \mathcal{S}_{\mathbf{G}_{[0,x-1-i]}\gamma}(i) \ \wedge \ \tau, x \models \big(\mathcal{S}_\gamma(x)\mathbf{U}_{[0,y-x]}\mathcal{S}_\psi(x)\big) && \mid \mathcal{S}_\gamma \text{ and } \mathcal{S}_\psi \text{ sound} \\
& && \mid \text{Lemma 3.5} \\
&\iff \tau, i \models \bigvee_{\substack{[x,y]\in \\ \mathtt{Split}([a+t,b+t],\mathcal{S}_\gamma,\mathcal{S}_\psi)}} \mathcal{S}_{\mathbf{G}_{[0,x-i-1]}\gamma}(i) \wedge \mathbf{X}_{[x-i]}\big(\mathcal{S}_\gamma(x)\mathbf{U}_{[0,y-x]}\mathcal{S}_\psi(x)\big) && \square
\end{aligned}$$

3.4.7 Correctness of the IntervalSimplification Algorithm

We now proof that the simplification mapping is sound (Definition 3.2).

Theorem 3.9. *Let φ be a LTL formula and τ a state trace. Given the partial knowledge about the state trace $\mathcal{P}$, the recursive simplification mapping $\mathcal{S}_\varphi$ defined in the previous sections is sound.*

Proof.
Base Cases:

$\varphi = a \in \mathcal{AP}$: The simplification is given as $\mathcal{S}_\varphi(i) = \mathcal{P}(i, a)$. This mapping is sound per definition as φ is only simplified if we know if $a \in \tau(i)$ or $a \notin \tau(i)$. Thus $\mathcal{S}_\varphi(i) \equiv_{(\tau,i)} \varphi$ must hold.

$\varphi = \top \mid \bot$: The simplification is given as $\mathcal{S}_\varphi(i) = \varphi$ and is thus trivially sound.

Induction Step:

Induction Hypothesis: $\mathcal{S}_\gamma$ and $\mathcal{S}_\psi$ are sound simplification mappings for γ and ψ.

$\varphi = \neg\gamma \mid \gamma \wedge \psi \mid \mathbf{X}_{[a]}\gamma \mid \mathbf{F}_{[a,b]}\gamma \mid \mathbf{G}_{[a,b]}\gamma \mid \gamma \mathbf{U}_{[a,b]}\psi$

$\mathcal{S}_\varphi(i) \equiv_{(\tau,i)} \varphi$ holds for each case as shown in lemmas 3.1, 3.2, 3.3, 3.5, 3.6, 3.8 □

3.5 Limitations

3.5.1 Sliding Window Problem

The problem describes a blow up of the formula size for nested temporal operators defined over large intervals. This problem is best examplified with an example. Given a *LTL* formula φ and a simplification mapping of the subformula γ:

$$\varphi = \mathbf{G}\big(\mathbf{F}(\gamma)\big) \quad \text{and} \quad \mathcal{S}_\gamma(i) = \begin{cases} \top & \text{if } i = 100 \\ \gamma' & \textit{otherwise} \end{cases},$$

we apply the `IntervalSimplification` algorithm and get:

$$\mathcal{S}_\varphi(i) = \bigwedge_{i \in [0,100]} \mathbf{G}_{[i,i]}\big(\mathbf{F}_{[i,99-i]}(\gamma) \wedge \mathbf{F}_{[100-i,100-i]}(\gamma') \wedge \mathbf{F}_{[101-i,\infty]}(\gamma)\big) \wedge \mathbf{G}_{[101,\infty]}\big(\mathbf{F}(\gamma)\big).$$

As one can see, the formula is much larger than the original formula, although we only defined the simplification mapping for γ at one position in the trace. The name Sliding Window Problem also becomes clear from the resulting formula, as by "moving" the future operator, whose interval always had to incorporate the simplification of γ at a different position, a new formula was created every time. Nevertheless, the *LTL* specifications for the CommonRoads framework do not suffer from this problem and demonstrate that practical applicability of the `IntervalSimplification` algorithm.

3.5.2 No Semantic Simplification

A limitation of the algorithm is that it is not capable of semantic simplifications of temporal and non-temporal formulas. Examples of such a simplification is the following:

$$A \wedge \neg A \equiv \mathit{False}$$

To address this, the responsibility lies with the writer of the specifications to ensure formulas do not contain such redundancies.

4 Simplification of Traffic Rules

This chapter demonstrates the application of the `IntervalSimplification` algorithm to *LTL* specifications within the context of autonomous vehicles. While the simplification algorithm itself remains unchanged, the process of constructing the partial knowledge map $\mathcal{P}$ is specific to the application domain of traffic scenarios. We illustrate this by leveraging various reachability analyses to determine $\mathcal{P}$. We assume a comprehensive understanding of the road network, including all traffic participants and obstacles, along with future states for all vehicles except the ego vehicle. Given that this work serves as a preprocessing step for other algorithms dealing with *LTL* specifications, the techniques employed herein need to strike a balance between effectiveness and computational cost. As explained in section 2.4, $k \in \mathbb{N}_0$ represents the discrete time step corresponding to the continuous time $t_k = k\Delta t$ with $\Delta t \in \mathbb{R}_{>0}$ being a fixed time increment. For the application of *LTL* to these scenarios each time step k corresponds to a position in the state trace τ of the system beginning with $k = 0$.

4.1 Longitudinal and Lateral Overapproximation

The longitudinal and lateral occupancy approximations lay the foundation for subsequent occupancy approximations. As detailed in Section 2.4, the longitudinal component of a position aligns with the reference path Γ, while the lateral component represents the orthogonal distance to the reference path. At time step k, we represent the longitudinal overapproximation as a 4-dimensional vector $\mathcal{S}_k := [s_k^{max}, \dot{s}_k^{max}, s_k^{min}, \dot{s}_k^{min}]^T$. Here, s_k^{max} and s_k^{min} represent the maximum and minimum longitudinal positions, respectively, at time step k. Likewise, $\dot{s}_k^{max}$ and $\dot{s}_k^{min}$ indicate the maximum and minimum longitudinal velocities possible at time step k, given the maximum and minimum acceleration a_{lon}^{max} and a_{lon}^{min} of the vehicle. We initialize s_0^{max} and s_0^{max} to zero, while $\dot{s}_0^{max}$ and $\dot{s}_0^{min}$ are set to the longitudinal velocity of the ego vehicle at time step 0. Subsequent overapproximations are defined using discrete-time system dynamics:

$$\mathcal{S}_{k+1} = \begin{pmatrix} 1 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 1 \end{pmatrix} \mathcal{S}_k + \begin{pmatrix} \frac{1}{2}\Delta t^2 & 0 \\ \Delta t & 0 \\ 0 & \frac{1}{2}\Delta t^2 \\ 0 & \Delta t \end{pmatrix} \begin{pmatrix} a_{lon}^{max} \\ a_{lon}^{min} \end{pmatrix}$$

We set thresholds for the maximum and minimum velocity and impose zero acceleration once these thresholds are reached. Denoting $\mathcal{D}_k$ as the lateral state overapproximation, its definition parallels that of $\mathcal{S}_k$. We exemplify the usage of longitudinal overapproximation

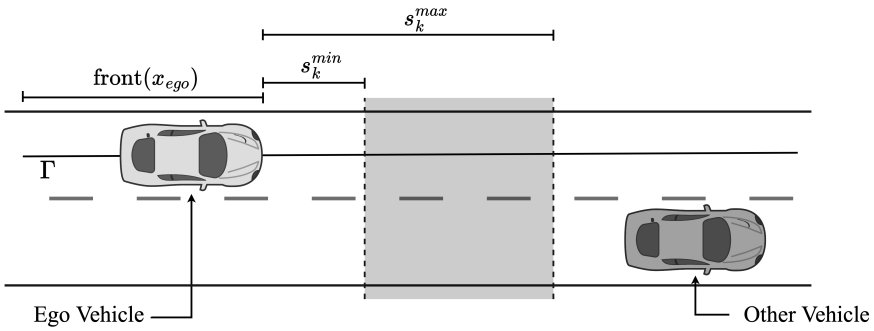


Figure 4.1: Example of the longitudinal position overapproximation at time step k. $\mathcal{S}_k$ can be used to bound the longitudinal position of the front bumper using s_k^{min} and s_k^{max}.

in determining the Boolean values of the predicate in_front_of(x_{ego}, x_o), defined in Section 2.5.2. This predicate evaluates to true if the longitudinal position of the rear bumper of x_o exceeds that of the front bumper of x_{ego}. We ascertain the truth of the proposition when the maximum longitudinal position of the ego vehicle's front remains less than the longitudinal position of the rear of the other vehicle. Conversely, the proposition is set to false if even the minimum longitudinal state of the ego vehicle's front surpasses the longitudinal rear position of the other vehicle. Formally, this is expressed as follows:

$$\mathcal{P}(\text{in\_front\_of}(\text{x}_{\text{ego}}, \text{x}_\text{o}), k) := \begin{cases} \top & \text{if } front(x_{ego}(0)) + s_k^{max} < rear(x_o(k)) \\ \bot & \text{if } front(x_{ego}(0)) + s_k^{min} \geq rear(x_o(k)) \\ ? & \text{otherwise} \end{cases}$$

Assuming that the other vehicle in Figure 4.2 does not move backwards, we can set the $\mathcal{P}(\text{in\_front\_of}(\text{x}_{\text{ego}}, \text{x}_\text{o}), \text{k}$ to $\top$ as the maximum longitudinal position of the front bumper of x_{ego} is still less than the rear position of x_o.

4.2 Reachable Occupancy Sets

Using the longitudinal and lateral overapproximations from 4.1 allows us to determine relational propositions like in_front_of but it is insufficient for determining positional propositions like on_access_ramp. For this we need to bound the potential positions in both longitudinal and lateral direction by also considering the shape of the ego vehicle. For this we use the notion of occupancy sets from [42].

Definition 4.1. The *reachable set* of the ego vehicle at time step k is defined as:

$$\mathcal{R}_k = \left\{ p \in \mathcal{O}(x) \;\middle|\; x = \xi\big(k\Delta t, x_{ego}(0), u(\cdot)\big) \wedge u(\cdot) \in \mathcal{U} \right\}$$

To obtain an overapproximation of this set, we first overapproximate the position of the center point of the vehicle denoted as $\mathcal{C}$. The initial set can be set to the actual center position of the ego vehicle as it is known at the beginning of the scenario. All subesequent approximations are propagated using the longitudinal and lateral overapproximations from section 4.1.

Definition 4.2. The *center point overapproximation* of the ego vehicle at time step k is defined as:

$$\mathcal{C}_0^{approx} = \left\{ \begin{pmatrix} proj_s(x_{ego}(0)) \\ proj_d(x_{ego}(0)) \end{pmatrix} \right\}$$

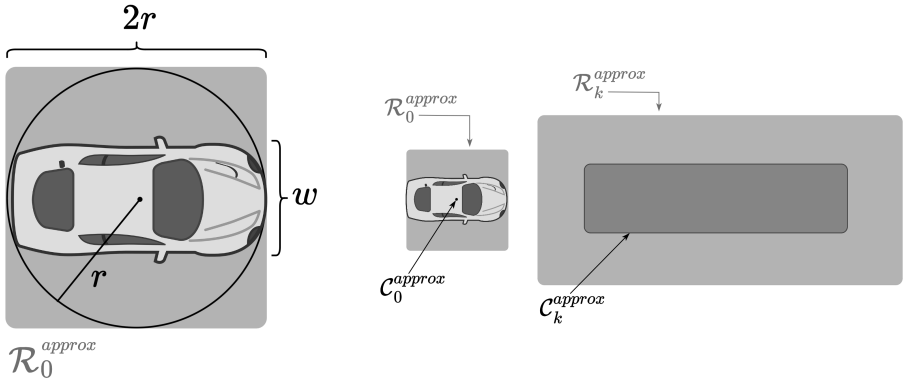
$$\mathcal{C}_k^{approx} = \mathcal{C}_0^{approx} \oplus \left\{ \begin{pmatrix} s_k \\ d_k \end{pmatrix} \in \mathbb{R}^2 \;\middle|\; s_k^{min} \leq s_k \leq s_k^{max} \;\wedge\; d_k^{min} \leq d_k \leq d_k^{max} \right\}$$

Given $\mathcal{C}_k^{approx}$ we define the overapproximation of the reachable set $\mathcal{R}_k$ by adding a overappriximation of the shape of the vehicle to $\mathcal{C}_k^{approx}$. We define this vehicle shape overapproximation initially with a circle of radius r and then overapproximating this circle with a square. This approach reduces the computational complexity of propagating the reachable sets.

Definition 4.3. The overapproximation $\mathcal{R}_k^{approx}$ of the reachable set $\mathcal{R}_k$ is defined as:

$$\mathcal{R}_k^{approx} = \mathcal{C}_k^{approx} \oplus \left\{ \begin{pmatrix} s_0 \\ d_0 \end{pmatrix} \in \mathbb{R}^2 \;\middle|\; |s_0|, |d_0| \leq r \right\}$$

Example. Propagation of reachable sets $\mathcal{R}_t^{approx}$ and $\mathcal{C}_t^{approx}$



Given $\mathcal{R}^{approx}$, we can now determine positional propositions such as on_access_ramp(x_{ego}), which is utilized in section 2.5. This proposition is true if the shape of the vehicle intersects with a lanelet of type *access_ramp*. Therefore, if at time $t = k\delta t$ no point in $\mathcal{R}_k^{approax}$, and consequently in $\mathcal{R}_t$, lies within an access ramp lanelet, we can set the proposition

to $false$ at time step k. Similarly, if every point in $\mathcal{R}_t^{approax}$ lies within an access ramp lanelet, the proposition can be set to $true$. This is formally defined as follows:

$$\mathcal{P}(\text{on\_access\_ramp}(\text{x}_{\text{ego}}), t) := \begin{cases} \top & \text{if } \mathcal{R}_t^{approax} \subseteq \{occ(l) | l \in \mathcal{L} \wedge \text{type}(l) = access\_ramp\} \\ \bot & \text{if } \mathcal{R}_t^{approax} \cap \{occ(l) | l \in \mathcal{L} \wedge \text{type}(l) = access\_ramp\} = \emptyset \\ ? & \text{otherwise} \end{cases}$$

Similarly to the proposition of on_access_ramp other positional propositions can be determined using $\mathcal{R}^{approx}$.

4.3 Occupancy Intersection Sets

As one might have noticed, the condition for setting propositions like on_access_ramp(x_{ego}) to $\top$ is quite restrictive, as it rarely occurs that all points of the overapproximation $\mathcal{R}_t^{approax}$ lie within a single lanelet or lane. Already the initial overapproximation $\mathcal{R}_0^{approx}$ almost always covers the adjacent left and right lanelets as well. However, looking at the definition of these positional attibutes if suffices that the ego vehicle only partially covers the area i.e. the shape of the ego vehicle intersects with the area of for example the access ramp. This leads us to introduce an additional positional overapproximation called *Intersection Sets*.

Definition 4.4. A set $\mathcal{I}_k$ is called *intersection set* for time step k if for every possible future state of the ego vehicle at least one point in $\mathcal{I}_k$ is covered by the vehicle, formally defined as:

$$\mathcal{I}_k \text{ is a intersection set for timestep } k$$
$$\text{iff. } \forall u(\cdot) \in \mathcal{U} : \mathcal{O}\big(\xi(k\Delta t, x_{ego}(0), u(\cdot))\big) \cap \mathcal{I}_k \neq \emptyset$$

If all points in a intersection set lie within a the area for a positional proposition we can set this position to true, as we know that the vehicle at least prartially covers that area. Of course we want the intersection set to be as small as possible as this will increase the likelyhood that all points lie within the area defined by the proposition.

In our pursuit of minimizing the intersection set, it's important to note that there isn't a single unique minimal intersection set. For instance, at $k = 0$, any $\{p\}$ with $p \in \mathcal{R}_0$ qualifies as a minimal occupancy set. We construct $\mathcal{I}_t^{approx}$ similar to $\mathcal{R}_t^{approx}$, by essentially shrinking the shape covered by $\mathcal{R}_t^{approx}$ by the width of the car from all sides. This adjustment guarantees that even in the worst-case scenario, where the vehicle is at the edge of the reachable set, at least one point in $\mathcal{O}_t$ remains covered by the car. We denote the width of the vehicle as $width$. The initial occupancy set is defined as follows:

Definition 4.5. The intersection set $\mathcal{I}_k^{approx}$ is defined as:

$$\mathcal{I}_k^{approx} = \mathcal{C}_k^{approx} \oplus \left\{ \begin{pmatrix} s_0 \\ d_0 \end{pmatrix} \in \mathbb{R}^2 \;\middle|\; |s_0|, |d_0| \leq r - width \right\}$$

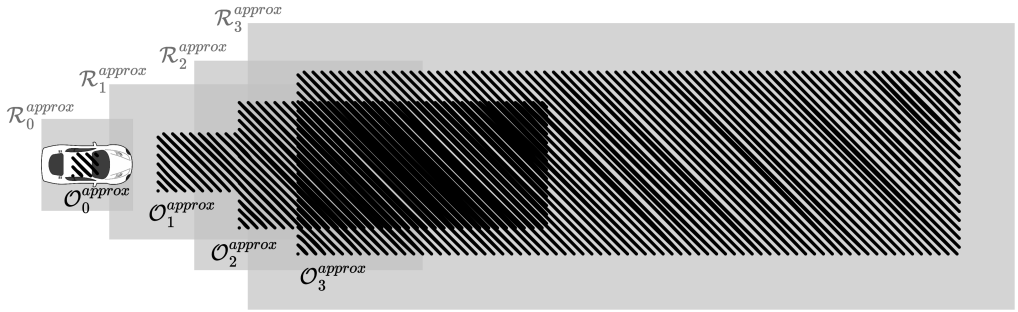


Figure 4.2: Example of propagation of reachable sets $\mathcal{R}_k^{approx}$ and $\mathcal{I}_k^{approx}$

Now, with $\mathcal{I}_k$ available, we can refine the definition of $\mathcal{P}$ for the predicate on_access_ramp by utilizing $\mathcal{I}_k$ for verification and $\mathcal{R}_k$ for falsification of the resulting proposition:

$$\mathcal{P}(\text{on\_access\_ramp}(\text{x}_{\text{ego}}), t) := \begin{cases} \top & \text{if } \mathcal{I}_t^{approax} \subseteq \{occ(l) | l \in \mathcal{L} \wedge \text{type}(l) = \textit{access\_ramp}\} \\ \bot & \text{if } \mathcal{R}_t^{approax} \cap \{occ(l) | l \in \mathcal{L} \wedge \text{type}(l) = \textit{access\_ramp}\} = \emptyset \\ ? & \text{otherwise} \end{cases}$$

5 Implementation

The implementation includes the general `IntervalSimplification` algorithm and methods to construct the partial knowledge mapping $\mathcal{P}$ for traffic scenarios within the CommonRoad framework. We have integrated both interstate traffic rules from [2] and intersection traffic rules from [3]. This implementation is utilized in the subsequent chapter for evaluation purposes.

5.1 Implemented Traffic Rules & Predicates

Table 5.1 provides an overview of the traffic rules that have been implemented, including four interstate traffic rules and four intersection traffic rules. For a formal definition of these traffic rules, please refer to the respective papers. Table 5.2 lists the predicates for which we have implemented methods to build the partial knowledge map $\mathcal{P}$. The right column indicates which traffic rules utilize these predicates in their temporal logic specifications.

Table 5.1: Implemented Traffic Rules from [2] and [3]

Rule	Informal Definition
R_G1	*Safe Distance Rule*: The ego vehicle must maintain a safe distance to preceding vehicles.
R_G4	*Traffic Flow*: No vehicle is allowed to impend the traffic flow
R_I1	*Stopping Rule*: It is forbidden to stop in the main carriageway if not required by the situation
R_I5	*Entering Vehicles Rule*: Defined in Section 2.5.2
R-IN1	*Stop Sign Rule*: The ego vehicle must stop at a stop sign.
R-IN3	*Right Before Left Rule*: The ego vehicle must yield to vehicles approaching from the right.
R-IN4	*Priority Rule*: The ego vehicle must yield to vehicles with the right of way.
R-IN5	*Turning Left Rule*: Asserts that the left-turning ego vehicle, which lacks priority, can only enter the oncoming lane if it does not endanger any other vehicle.

Table 5.2: Implemented Predicates in Partial Knowledge Mapping $\mathcal{P}$

Predicate	Rules	Predicate	Rules
at_stop_sign	R-IN1	on_main_carriageway	R_I2, R_I5
stop_line_in_front	R-IN1	on_access_ramp	R_I2, R_I5
turning_(left/right)	R-IN3, R-IN4, R-IN5	right_lane	R_I2, R_I5
on_intersection	R-IN3, R-IN4, R-IN5	lane_same_lane	R_G1, R_G4, R_I1
relevant_traffic_light	R-IN3	cut_in	R_G1
on_incoming_left_of	R-IN3	keeps_safe_distance	R_G1
at_(red/yellow)_light	R_IN1, R_IN3	in_front_of	R_G1, R_G4, R_I1, R_I5

5.2 Example Use Case

We show an exemplary use case of the simplification for the entering vehicles rule (section 2.5.2) for the traffic scenario below. We assume that we want to generate a specification compliant vehicle trajectory for the next next 15 time steps. We first generate the parial knowlede map $\mathcal{P}$ and then apply the `IntervalSimplification` algorithm to the LTL_f specification of the entering vehicles rule.

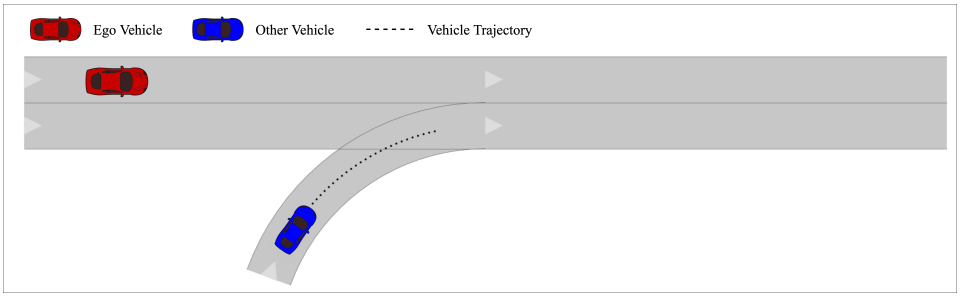


Figure 5.1: Traffic Scenario with one other vehicle entering the main carriage way on an access ramp.

5.2.1 Generation of the Partial Knowledge Mapping

Using the techniques described in section 4 we can generate the partial knowledge mapping $\mathcal{P}$ for the propositions used in the entering vehicles rule:

Proposition	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
on_main_carriageway(x_{ego})	⊤	⊤	⊤	⊤	⊤	⊤	⊤	⊤	?	?	?	?	?	?	?	?
on_main_carriageway(x_o)	⊥	⊥	⊥	⊥	⊥	⊥	⊤	⊤	⊤	⊤	⊤	⊤	⊤	⊤	⊤	⊤
in_front_of(x_{ego}, x_o)	⊤	⊤	⊤	⊤	⊤	?	?	?	?	?	?	?	?	?	?	?
on_access_ramp(x_o)	⊤	⊤	⊤	⊤	⊤	⊤	⊤	⊤	⊤	⊤	⊤	⊤	⊤	⊤	⊤	⊤
right_lane(x_{ego})	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?

5.2.2 IntervalSimplification

Given the knowledge mapping $\mathcal{P}$, we can now apply the `IntervalSimplification` algorithm to the entering vehicles rule. The resulting formula looks as follows:

$$
\begin{aligned}
\mathcal{S}_\varphi(0) \quad = \quad & \mathbf{G}_{[0,4]} \Big(\mathbf{F}\big(\text{right\_lane}(x_{ego})\big) \implies \text{right\_lane}(x_{ego}) \Big) \\
\wedge \ & \mathbf{G}_{[5,7]} \Big(\text{in\_front\_of}(x_{ego}, x_0) \implies \neg \Big(\neg \text{right\_lane}(x_{ego}) \wedge \mathbf{F}\big(\text{right\_lane}(x_{ego})\big) \Big) \Big) \\
\wedge \ & \mathbf{G}_{[8,15]} \Big(\text{on\_main\_carriageway}(x_{ego}) \wedge \text{in\_front\_of}(x_{ego}, x_0) \\
& \qquad \implies \neg \Big(\neg \text{right\_lane}(x_{ego}) \wedge \mathbf{F}\big(\text{right\_lane}(x_{ego})\big) \Big) \Big)
\end{aligned}
$$

The simplified formula clearly shows how the algorithm incorporates the knowledge about the future states of the traffic participants into the formula by splitting the globally operator into intervals for which only structurally smaller expressions need to hold compared to the original formula (section 2.5.2).

6 Evaluation

In this chapter, we present a comprehensive evaluation of our proposed simplification approach. To assess its effectiveness, we innterpret the simplified formulas over finite traces and employ two distinct metrics: *Unknown Propositions* and *No. of Proposition Evaluations*. We compare these metrics for original formula against the simplified formulas. Furthermore, we evaluate the algorithm's performance as a prepossessing step for a specification-compliant reachability algorithm [5], measuring the computation time reduction.

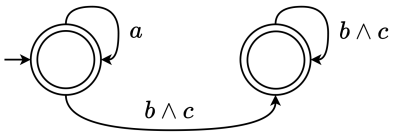
6.1 Evaluation Metrics

6.1.1 Unknown Propositions

The *Unknown Propositions* metric quantifies the total number of proposition values that are unknown in the state trace. It is therefore a metric that allows us to asses the quality of our mapping $\mathcal{P}$ by comparing the metirc for the original formula (Original) and after simplification (Simplified). Furthermore it is an upper bound for the number of proposition evaluations needed for the verification of the formula. Recurrent propositions in different formulas are counted more than once.

6.1.2 Number of Proposition Evaluations

The *No. of Proposition Evaluations* metric is derived from the finite automata representation of the LTL_f formula, generated by the Spot tool [47]. It measures the number of proposition evaluations required for accepting runs through the finite automaton. This metric demonstrates how effectively the information from $\mathcal{P}$, quantified by the *Unknown Propositions* metric, translates into an actual reduction of proposition evaluations during the verification process. The metric is given as *lower bound* $-$ *upper bound*.



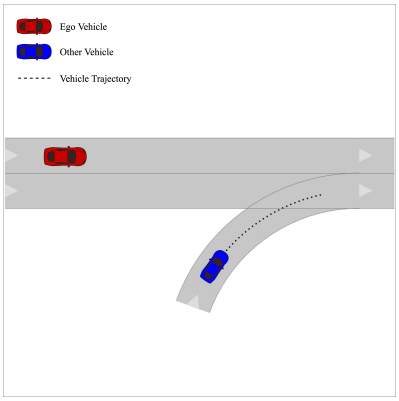
Given a trace τ represented as a finite word over $2^{\{a,b,c\}}$ of length 10 we get the following bounds for the cost for the automata on the left:

lower bound: 10 $(a_0, ..., a_9)$
upper bound: 21 $(a_0, b_0, c_0, b_1, c_1, ..., b_9, c_9)$

Figure 6.1: Example of the Number of Proposition Evaluations of a Finite Automata. We count the number of proposition evaluations in best and worst case. a_i denotes the evaluation of proposition a at position i in the trace

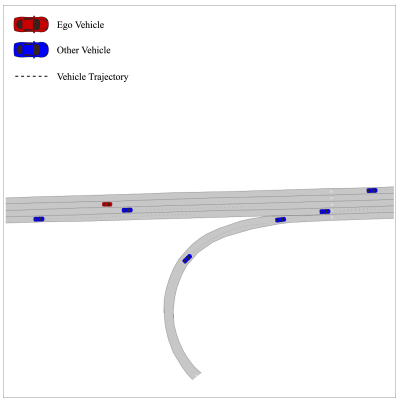
6.2 Evaluation on Interstate Scenarios

6.2.1 Experiment Results



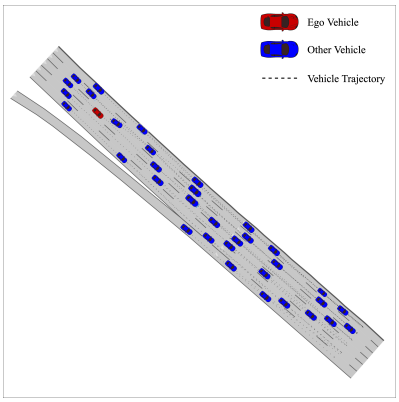
INTERSTATE_1
Planning Horizon: 15
Traffic Participants: 1

Rule	Unknown Propositions			No. of Proposition Evaluations		
	Original	Simp.	Red.	Original	Simp.	Red.
R_G1	64	35	45%	[16,62]	[12,30]	[25%,52%]
R_G4	32	25	22%	[16,32]	[16,25]	[0%,22%]
R_I1	48	16	67%	[16,48]	[16,16]	[0%,67%]
R_I5	80	35	56%	[16,80]	[16,35]	[0%,56%]



INTERSTATE_2
Planning Horizon: 15
Traffic Participants: 6

Rule	Unknown Propositions			No. of Proposition Evaluations		
	Original	Simp.	Red.	Original	Simp.	Red.
R_G1	384	84	78%	[96,372]	[9,15]	[90%, 96%]
R_G4	32	32	0%	[16,32]	[16,32]	[0%,0%]
R_I1	48	16	67%	[16,48]	[16,16]	[0%,67%]
R_I5	480	52	89%	[96,480]	[9,16]	[91%,97%]



INTERSTATE_3
Planning Horizon: 15
Traffic Participants: 34

Rule	Unknown Propositions			No. of Proposition Evaluations		
	Original	Simp.	Red.	Original	Simp.	Red.
R_G1	2176	829	62%	[544,2108]	[160,416]	[70%, 80%]
R_G4	32	32	0%	[16,32]	[16,32,]	[0%,0%]
R_I1	48	31	35%	[16,48]	[16,31]	[0%,35%]
R_I5	2720	92	97%	[544,2720]	[0,0]	[100%,100%]

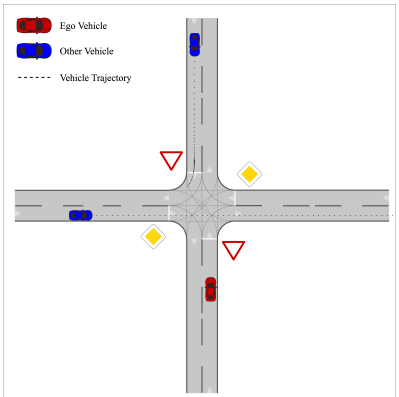
6.2.2 Interpretation of Results

Examining the *Unknown Propositions* metric, it is evident that our knowledge mapping functions effectively. On average, around 51% of the propositions could be determined a priori. However, propositions that depend on the state of the ego vehicle become increasingly challenging to determine as the time horizon extends. This is due to the expansion of the reachable sets (ref section 4), which causes the ego vehicle's potential positions to multiply with each time step, making it more difficult to ascertain future states such as whether it will remain or switch lanes. This effect is particularly evident in rule R_G4, where only a small percentage of propositions can be determined since this rule solely contains propositions dependent on the ego vehicle's state. Furthermore, it is notable that with an increasing number of traffic participants, a larger proportion of propositions can be reduced, which is especially visible for rule R_I5 involving entering vehicles. This can be attributed to the fact that when more vehicles are present, most of them tend to be far away from the ego vehicle, as the number of vehicles in the surrounding area of the ego vehicle is limited due to spatial constraints. This makes it easier to determine propositions related to these distant vehicles for longer time horizons. For example, if a vehicle is far behind the ego vehicle, we can be certain that the ego vehicle will remain in front of that vehicle for longer period of time. This observation suggests that the simplification approach will be particularly useful for complex scenarios involving numerous traffic participants, as a higher number of vehicles increases the likelihood of having more distant vehicles whose states can be reliably determined over longer time frames.

A comparison of the *No. of Proposition Evaluations* metric with the *Unknown Propositions* metric demonstrates the effectiveness of the `IntervalSimplification` algorithm in integrating knowledge about the state trace into the simplified formula. Not surprisingly, the number of unknown propositions always serves as an upper bound for the number of proposition evaluations required for an accepting accepting trace. However, the algorithm even achieves reductions that go well below this bound, indicating its ability to leverage the available knowledge. A notable example is the INTERSTATE_2 scenario, where despite having 52 unknown propositions, the upper bound on the number of proposition evaluations is reduced to a mere 16. This highlights the algorithms capability of not only incorporating the given knowledge but also using this knowledge to further eliminate proposition evaluations.

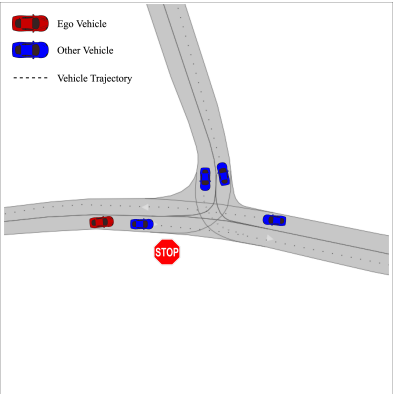
6.3 Evaluation on Intersection Scenarios

6.3.1 Experiments Results



INTERSECTION_1
Planning Horizon: 15
Traffic Participants: 2

Rule	Unknown Propositions			No. of Proposition Evaluations		
	Original	Simp.	Red.	Original	Simp.	Red.
R-IN1	80	42	47.5%	16 - 64	0 - 0	100% - 100%
R-IN3	528	372	29.5%	32 - 608	0 - 0	100% - 100%
R-IN4	512	400	21.8%	32 - 576	18 - 128	43.7% - 77.8%
R-IN5	256	176	31.3%	32 - 256	0 - 0	100% - 100%



INTERSECTION_2
Planning Horizon: 20
Traffic Participants: 4

Rule	Unknown Propositions			No. of Proposition Evaluations		
	Original	Simp.	Red.	Original	Simp.	Red.
R-IN1	80	51	36.3%	16-64	9-18	43.8% - 71.9%
R-IN3	976	691	29.2%	64-1216	0-0	100% - 100%
R-IN4	960	755	21.4%	64-1152	57-338	10.9% - 70.7%
R-IN5	512	370	27.7%	64-512	1-37	98.4% - 92.8%

6.3.2 Interpretation of Results

As observed in the interstate scenarios, the the construction of the knowledge mapping $\mathcal{P}$ works effectively, as reflected in the reduction of unknown propositions. Furthermore, the proportional reduction in the *No. of Proposition Evaluations* metric for the intersection rules is even larger compared to the interstate rules. This can be attributed to the fact that the intersection rules contain numerous case distinctions based on the maneuvers of the traffic participants, such as turning left, going straight, or turning right. By determining the specific maneuver of a traffic participant, the algorithm can immediately discard all other cases, leading to a significant reduction in number of potential proposition evaluations compared to the number of unknown propositions remaining after simplification.

For instance, in the INTERSECTION_1 scenario, while the reduction in unknown propositions for rule R-IN4 is only 21.8%, the upper bound of the number of proposition

evaluations has been reduced by a much larger percentage of 77.8% due to the algorithm's ability to resolve the case distinctions based on the known information. Similarly, in the more complex INTERSECTION_2 scenario, involving four traffic participants, the algorithm achieves an impressive 70.7% reduction in no. of proposition evaluations for rule R-IN4. In both cases rule R_I3 have been simplified to *true* with only knowing around 30% of the propositions a priori.

These results demonstrate the effectiveness of the simplification approach in leveraging the available knowledge to not only reduce the number of unknown propositions but also to make intelligent inferences about the specific cases that apply, thereby significantly minimizing the computational complexity of verifying the *LTL* specifications, especially for rules with intricate case distinctions.

6.4 Specification-Compliant Reachability Analysis

To evaluate the benefits of our specification simplification approach for algorithms operating on LTL specifications, we integrated it into the on-the-fly model checking algorithm proposed by Florian Lercher and Matthias Althoff [5]. This algorithm computes an overapproximation of the reachable states of the ego vehicle that conform to a given set of LTL specifications, enabling the generation of specification-compliant driving corridors for motion planning (as discussed in Section 2.6). We benchmarked the algorithm on the three interstate traffic scenarios defined in Section 6.2, focusing on the "Entering Vehicles Rule" (R_I5) specified in Section 2.5.2. Table 6.1 presents a comparison of the reachable set computation times with and without the incorporation of our specification simplification step.

Table 6.1: Comparison of Reachable Set Computation using [5] with and without specification simplification.

Scenario	**Original**	**With Simplification**			**Reduction**
	Total	Simplification	Reachability	**Total**	
INTERSTATE_1	**0.057 s**	0.012 *s*	0.042 *s*	**0.054 s**	**5.26**%
INTERSTATE_2	**2.495 s**	0.079 *s*	0.068 *s*	**0.147 s**	**94.11**%
INTERSTATE_3	**912.4 s**	0.13 *s*	35.57 *s*	**35.7 s**	**96.09**%

The "Original" column shows the total computation time without simplification, while the "Simplified" columns break down the time into the simplification step itself ("Simplification") and the subsequent reachable set computation ("Set Computation"). The "Total" column under "Simplified" represents the combined time for both steps, and the "Reduction" column quantifies the performance improvement achieved by incorporating the simplification algorithm. The results demonstrate substantial computational gains across all scenarios, with the benefits becoming more pronounced as the scenario complexity increases. For the INTERSTATE_1 scenario, a modest reduction of 5.26% was observed. However, for the more intricate INTERSTATE_2 and INTERSTATE_3

scenarios, the simplification step led to remarkable reductions of 94.11% and 96.09%, respectively. These findings align with our expectations that the simplification algorithm would be increasingly beneficial as the number of traffic participants and the complexity of the scenario increase. The significant performance boosts in the INTERSTATE_2 and INTERSTATE_3 scenarios can be attributed to the algorithm's ability to effectively leverage partial knowledge about the state trace, resulting in simpler LTL specifications that require fewer proposition evaluations during the reachable set computation. This highlights the practical utility of our algorithm in enabling feasible computations for complex scenarios that may otherwise be intractable within reasonable time constraints.

Overall, the results from this benchmark demonstrate the substantial computational benefits of our simplification algorithm when integrated into reachability analysis and formal verification tasks involving LTL specifications in the context of autonomous driving scenarios.

7 Conclusion

This thesis investigated an approach to simplify Linear Temporal Logic (LTL) specifications for autonomous vehicles by leveraging partial knowledge about the state trace. We developed an algorithm that can reduce LTL formulas to either true, false, or a semantically equivalent but simplified form, utilizing information about when propositions hold or do not hold during the system's state sequence.

The proposed algorithm was applied to formalized traffic rules from literature, demonstrating its effectiveness in simplifying specifications based on the available knowledge about the traffic situation. Our evaluation on real-world traffic scenarios using the CommonRoad framework showed that the simplified specifications generated by our algorithm provide substantial reductions in computational complexity while maintaining equivalent expressiveness.The simplification approach was particularly beneficial for complex traffic scenarios involving numerous participants, where the computational cost of verifying LTL specifications grows significantly. By integrating our algorithm as a preprocessing step, we could reduce the number of required proposition evaluations and achieve considerable performance improvements for a specification-compliant reachability analysis operating on LTL specifications.

Furthermore, the proposed algorithm is not limited to the domain of autonomous driving but can be applied to any system that relies on LTL specifications and has access to partial knowledge about the state trace. This opens up opportunities for broader applications in areas such as robotics, cyber-physical systems, and formal verification of software and hardware systems. While this work focused on LTL specifications, future research could explore extending the simplification approach to other temporal logics, such as Computation Tree Logic (CTL) or Metric Temporal Logic (MTL). Additionally, investigating techniques for more efficiently obtaining the partial knowledge about the state trace could further enhance the effectiveness of the simplification algorithm. Overall, this thesis contributed a novel approach to simplifying temporal logic specifications, demonstrated its applicability in the context of autonomous driving, and highlighted its potential for improving the computational efficiency and scalability of formal verification tasks across various domains.

List of Figures

List of Tables

Bibliography

[1] United Nations Economic Commission for Europe, *Convention on road traffic*, United Nations Conference on Road Traffic, (consolidated version of 2006), 1968. [Online]. Available: https://www.unece.org/fileadmin/DAM/trans/conventn/Conv_road_traffic_EN.pdf.

[2] S. Maierhofer, A.-K. Rettinger, E. C. Mayer, and M. Althoff, "Formalization of interstate traffic rules in temporal logic," in *Proc. of the IEEE Intelligent Vehicles Symp. (IV)*, 2020, pp. 752–759.

[3] S. Maierhofer, P. Moosbrugger, and M. Althoff, "Formalization of intersection traffic rules in temporal logic," in *Proc. of the IEEE Intelligent Vehicles Symp. (IV)*, 2022, pp. 1135–1144.

[4] M. Althoff, M. Koschi, and S. Manzinger, "CommonRoad: Composable benchmarks for motion planning on roads," in *Proc. of the IEEE Intelligent Vehicles Symposium*, 2017, pp. 719–726.

[5] F. Lercher and M. Althoff, "Specification-compliant reachability analysis for autonomous vehicles using on-the-fly model checking."

[6] M. Leucker and C. Schallhart, "A brief account of runtime verification," *The Journal of Logic and Algebraic Programming*, The 1st Workshop on Formal Languages and Analysis of Contract-Oriented Software (FLACOS'07), vol. 78, no. 5, pp. 293–303, May 2009, ISSN: 1567-8326. DOI: 10.1016/j.jlap.2008.08.004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1567832608000775 (visited on 05/26/2024).

[7] A. Bauer, M. Leucker, and C. Schallhart, "Runtime verification for ltl and tltl," *ACM Trans. Softw. Eng. Methodol.*, vol. 20, no. 4, Sep. 2011, ISSN: 1049-331X. DOI: 10.1145/2000799.2000800. [Online]. Available: https://doi.org/10.1145/2000799.2000800.

[8] S. Vijzelaar and W. Fokkink, *Creating Büchi Automata for Multi-valued Model Checking*. May 2017, Pages: 224, ISBN: 978-3-319-60224-0. DOI: 10.1007/978-3-319-60225-7_15.

[9] F. Belardinelli, A. Ferrando, and V. Malvone, "3vLTL: A Tool to Generate Automata for Three-valued LTL," *Electronic Proceedings in Theoretical Computer Science*, vol. 395, pp. 180–187, Nov. 2023, arXiv:2311.09787 [cs], ISSN: 2075-2180. DOI: 10.4204/EPTCS.395.13. [Online]. Available: http://arxiv.org/abs/2311.09787 (visited on 05/26/2024).

[10] K. Havelund and G. Rosu, "Monitoring programs using rewriting," in *Proceedings 16th Annual International Conference on Automated Software Engineering (ASE 2001)*, ISSN: 1938-4300, Nov. 2001, pp. 135–143. DOI: 10.1109/ASE.2001.989799. [Online]. Available: https://ieeexplore.ieee.org/document/989799 (visited on 05/30/2024).

[11] G. Fraser and F. Wotawa, *Using LTL rewriting to improve the performance of model-checker based test-case generation*. Jul. 2007, Journal Abbreviation: Proceedings of the 3rd International Workshop Advances in Model Based Testing, AMOST 2007 Pages: 74 Publication Title: Proceedings of the 3rd International Workshop Advances in Model Based Testing, AMOST 2007. DOI: 10.1145/1291535.1291542.

[12] *Runtime Verification with Multi-valued Formula Rewriting | IEEE Conference Publication | IEEE Xplore*. [Online]. Available: https://ieeexplore.ieee.org/document/5587725 (visited on 05/26/2024).

[13] G. Roşu and K. Havelund, "Rewriting-Based Techniques for Runtime Verification," *Automated Software Engineering*, vol. 12, pp. 151–197, Jan. 2005. DOI: 10.1007/s10515-005-6205-y.

[14] G. Bruns and P. Godefroid, "Model checking partial state spaces with 3-valued temporal logics," in *International Conference on Computer Aided Verification*, 1999, pp. 274–287.

[15] T. Wright and I. Stark, *Technical Report: Property-Directed Verified Monitoring of Signal Temporal Logic*, arXiv:2008.06589 [cs], Aug. 2020. DOI: 10.48550/arXiv.2008.06589. [Online]. Available: http://arxiv.org/abs/2008.06589 (visited on 05/26/2024).

[16] A. Bauer, M. Leucker, and C. Schallhart, "Comparing ltl semantics for runtime verification," *J. Log. Comput.*, vol. 20, pp. 651–674, Jun. 2010. DOI: 10.1093/logcom/exn075.

[17] A. Rizaldi and M. Althoff, "Formalising Traffic Rules for Accountability of Autonomous Vehicles," in *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, 2015, pp. 1658–1665. DOI: 10.1109/ITSC.2015.269.

[18] L. Gressenbuch and M. Althoff, "Predictive Monitoring of Traffic Rules," in *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, 2021, pp. 915–922. DOI: 10.1109/ITSC48978.2021.9564432.

[19] Q. Zhang, D. K. Hong, Z. Zhang, Q. A. Chen, S. Mahlke, and Z. M. Mao, "A Systematic Framework to Identify Violations of Scenario-dependent Driving Rules in Autonomous Vehicle Software," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 5, no. 2, Jun. 2021, Place: New York, NY, USA Publisher: Association for Computing Machinery. DOI: 10.1145/3460082. [Online]. Available: https://doi.org/10.1145/3460082.

[20] A. Karimi and P. Duggirala, "Formalizing traffic rules for uncontrolled intersections," Apr. 2020, pp. 41–50. DOI: 10.1109/ICCPS48487.2020.00012.

[21] M. Hekmatnejad, S. Yaghoubi, A. Dokhanchi, *et al.*, "Encoding and monitoring responsibility sensitive safety rules for automated vehicles in signal temporal logic," in *Proceedings of the 17th ACM-IEEE International Conference on Formal Methods and Models for System Design*, ser. MEMOCODE '19, event-place: La Jolla, California, New York, NY, USA: Association for Computing Machinery, 2019, ISBN: 978-1-4503-6997-8. DOI: 10.1145/3359986.3361203. [Online]. Available: https://doi.org/10.1145/3359986.3361203.

[22] J. Karlsson, S. van Waveren, C. Pek, I. Torre, I. Leite, and J. Tumova, "Encoding Human Driving Styles in Motion Planning for Autonomous Vehicles," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 1050–1056. DOI: 10.1109/ICRA48506.2021.9561777.

[23] J. Karlsson and J. Tumova, "Intention-aware motion planning with road rules," in *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*, 2020, pp. 526–532. DOI: 10.1109/CASE48305.2020.9217037.

[24] N. Aréchiga, "Specifying Safety of Autonomous Vehicles in Signal Temporal Logic," in *2019 IEEE Intelligent Vehicles Symposium (IV)*, 2019, pp. 58–63. DOI: 10.1109/IVS.2019.8813875.

[25] *A Formally Verified Checker of the Safe Distance Traffic Rules for Autonomous Vehicles*. Jun. 2016, Pages: 190, ISBN: 978-3-319-40647-3. DOI: 10.1007/978-3-319-40648-0_14.

[26] A. Rizaldi, J. Keinholz, M. Huber, *et al.*, "Formalising and Monitoring Traffic Rules for Autonomous Vehicles in Isabelle/HOL," Sep. 2017, ISBN: 978-3-319-66844-4. DOI: 10.1007/978-3-319-66845-1_4.

[27] K. Esterle, L. Gressenbuch, and A. Knoll, "Formalizing Traffic Rules for Machine Interpretability," in *2020 IEEE 3rd Connected and Automated Vehicles Symposium (CAVS)*, 2020, pp. 1–7. DOI: 10.1109/CAVS51000.2020.9334599.

[28] C. Pek, P. Zahn, and M. Althoff, "Verifying the safety of lane change maneuvers of self-driving vehicles based on formalized traffic rules," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, Jun. 2017, pp. 1477–1483. DOI: 10.1109/IVS.2017.7995918. [Online]. Available: https://ieeexplore.ieee.org/document/7995918 (visited on 05/30/2024).

[29] M. Buechel, G. Hinz, F. Ruehl, H. Schroth, C. Gyoeri, and A. Knoll, "Ontology-based traffic scene modeling, traffic regulations dependent situational awareness and decision-making for automated vehicles," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, 2017, pp. 1471–1476. DOI: 10.1109/IVS.2017.7995917.

[30] J. Wei, J. M. Snider, T. Gu, J. M. Dolan, and B. Litkouhi, "A behavioral planning framework for autonomous driving," in *2014 IEEE Intelligent Vehicles Symposium Proceedings*, ISSN: 1931-0587, Jun. 2014, pp. 458–464. DOI: 10.1109/IVS.2014.6856582. [Online]. Available: https://ieeexplore.ieee.org/document/6856582 (visited on 05/30/2024).

[31] M. Brännström, E. Coelingh, and J. Sjöberg, "Model-Based Threat Assessment for Avoiding Arbitrary Vehicle Collisions," *IEEE Transactions on Intelligent Transportation Systems*, vol. 11, no. 3, pp. 658–669, Sep. 2010, Conference Name: IEEE Transactions on Intelligent Transportation Systems, ISSN: 1558-0016. DOI: 10.1109/TITS.2010.2048314. [Online]. Available: https://ieeexplore.ieee.org/document/5466072 (visited on 05/30/2024).

[32] *Multi-target threat assessment for automotive applications | IEEE Conference Publication | IEEE Xplore*. [Online]. Available: https://ieeexplore.ieee.org/document/6082801 (visited on 05/30/2024).

[33] N. Kaempchen, B. Schiele, and K. Dietmayer, "Situation Assessment of an Autonomous Emergency Brake for Arbitrary Vehicle-to-Vehicle Collision Scenarios," *IEEE Transactions on Intelligent Transportation Systems*, vol. 10, no. 4, pp. 678–687, Dec. 2009, Conference Name: IEEE Transactions on Intelligent Transportation Systems, ISSN: 1558-0016. DOI: 10.1109/TITS.2009.2026452. [Online]. Available: https://ieeexplore.ieee.org/document/5161310 (visited on 05/30/2024).

[34] J.-H. Kim and D.-S. Kum, "Threat prediction algorithm based on local path candidates and surrounding vehicle trajectory predictions for automated driving vehicles," in *2015 IEEE Intelligent Vehicles Symposium (IV)*, ISSN: 1931-0587, Jun. 2015, pp. 1220–1225. DOI: 10.1109/IVS.2015.7225849. [Online]. Available: https://ieeexplore.ieee.org/document/7225849 (visited on 05/30/2024).

[35] A. Barth and U. Franke, "Where will the oncoming vehicle be the next second?" In *2008 IEEE Intelligent Vehicles Symposium*, ISSN: 1931-0587, Jun. 2008, pp. 1068–1073. DOI: 10.1109/IVS.2008.4621210. [Online]. Available: https://ieeexplore.ieee.org/document/4621210 (visited on 05/30/2024).

[36] A. Eidehall and L. Petersson, "Statistical Threat Assessment for General Road Scenes Using Monte Carlo Sampling," *IEEE Transactions on Intelligent Transportation Systems*, vol. 9, no. 1, pp. 137–147, Mar. 2008, Conference Name: IEEE Transactions on Intelligent Transportation Systems, ISSN: 1558-0016. DOI: 10.1109/TITS.2007.909241. [Online]. Available: https://ieeexplore.ieee.org/document/4414362 (visited on 05/30/2024).

[37] M. Althoff, O. Stursberg, and M. Buss, "Model-Based Probabilistic Collision Detection in Autonomous Driving," *IEEE Transactions on Intelligent Transportation Systems*, vol. 10, no. 2, pp. 299–310, Jun. 2009, Conference Name: IEEE Transactions on Intelligent Transportation Systems, ISSN: 1558-0016. DOI: 10.1109/TITS.2009.2018966. [Online]. Available: https://ieeexplore.ieee.org/document/4895669 (visited on 05/30/2024).

[38] T. Gindele, S. Brechtel, and R. Dillmann, "Learning Driver Behavior Models from Traffic Observations for Decision Making and Planning," *IEEE Intelligent Transportation Systems Magazine*, vol. 7, no. 1, pp. 69–79, 2015, Conference Name: IEEE Intelligent Transportation Systems Magazine, ISSN: 1941-1197. DOI: 10.1109/MITS.2014.

2357038. [Online]. Available: https://ieeexplore.ieee.org/document/7014400 (visited on 05/30/2024).

[39] M. Althoff and A. Mergel, "Comparison of Markov Chain Abstraction and Monte Carlo Simulation for the Safety Assessment of Autonomous Cars," *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, no. 4, pp. 1237–1247, Dec. 2011, Conference Name: IEEE Transactions on Intelligent Transportation Systems, ISSN: 1558-0016. DOI: 10.1109/TITS.2011.2157342. [Online]. Available: https://ieeexplore.ieee.org/document/5875884 (visited on 05/30/2024).

[40] A. Lambert, D. Gruyer, G. S. Pierre, and A. N. Ndjeng, "Collision Probability Assessment for Speed Control," in *2008 11th International IEEE Conference on Intelligent Transportation Systems*, ISSN: 2153-0017, Oct. 2008, pp. 1043–1048. DOI: 10.1109/ITSC.2008.4732692. [Online]. Available: https://ieeexplore.ieee.org/document/4732692 (visited on 05/30/2024).

[41] M. Koschi and M. Althoff, "SPOT: A tool for set-based prediction of traffic participants," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, Jun. 2017, pp. 1686–1693. DOI: 10.1109/IVS.2017.7995951. [Online]. Available: https://ieeexplore.ieee.org/document/7995951 (visited on 05/30/2024).

[42] M. Althoff and S. Magdici, "Set-Based Prediction of Traffic Participants on Arbitrary Road Networks," *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 2, pp. 187–202, Jun. 2016, Conference Name: IEEE Transactions on Intelligent Vehicles, ISSN: 2379-8904. DOI: 10.1109/TIV.2016.2622920. [Online]. Available: https://ieeexplore.ieee.org/document/7725548 (visited on 05/30/2024).

[43] M. Althoff and J. M. Dolan, "Online Verification of Automated Road Vehicles Using Reachability Analysis," *IEEE Transactions on Robotics*, vol. 30, no. 4, pp. 903–918, Aug. 2014, Conference Name: IEEE Transactions on Robotics, ISSN: 1941-0468. DOI: 10.1109/TRO.2014.2312453. [Online]. Available: https://ieeexplore.ieee.org/document/6784493 (visited on 05/30/2024).

[44] M. Koschi and M. Althoff, "Interaction-aware occupancy prediction of road vehicles," in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, ISSN: 2153-0017, Oct. 2017, pp. 1–8. DOI: 10.1109/ITSC.2017.8317852. [Online]. Available: https://ieeexplore.ieee.org/document/8317852 (visited on 05/30/2024).

[45] E. I. Liu and M. Althoff, "Specification-compliant motion planning using reachability analysis and model checking," *IEEE Transactions on Intelligent Vehicles*, pp. 1–17, 2023, Early access.

[46] G. De Giacomo and M. Y. Vardi, "Linear temporal logic and linear dynamic logic on finite traces," in *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, ser. IJCAI '13, Beijing, China: AAAI Press, Aug. 2013, pp. 854–860, ISBN: 978-1-57735-633-2. (visited on 05/31/2024).

[47] Alexandre Duret-Lutz, E. Renault, M. Colange, *et al.*, "From Spot 2.0 to Spot 2.10: What's new?" In *Proceedings of the 34th International Conference on Computer Aided Verification (CAV'22)*, ser. Lecture Notes in Computer Science, vol. 13372, Springer, Aug. 2022, pp. 174–187. DOI: 10.1007/978-3-031-13188-2_9.

[48] E. Héry, S. Masi, P. Xu, and P. Bonnifait, "Map-based curvilinear coordinates for autonomous vehicles," in *Proc. of the IEEE International Conference on Intelligent Transportation Systems*, 2017, pp. 1–7.