TECHNICAL UNIVERSITY OF MUNICH

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

Bachelor's Thesis in Informatics

Situation-Aware Simplification of Temporal Logic Specifications for Autonomous Vehicles

Paul Manfred Reisenberg

TECHNICAL UNIVERSITY OF MUNICH

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

Bachelor's Thesis in Informatics

Situation-Aware Simplification of Temporal Logic Specifications for Autonomous Vehicles

Situationsbezogene Vereinfachung von temporallogischen Spezifikationen für autonome Fahrzeuge

Author:	Paul Manfred Reisenberg
Supervisor:	Prof. Dr.-Ing. Matthias Althoff
Advisor:	Florian Lercher, M.Sc.
Submission Date:	25.06.2024

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

Munich, 25.06.2024 Paul Manfred Reisenberg

Abstract

Temporal logic is a powerful formalism for specifying the behaviour 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 rules 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. 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, i.e., 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 semantic equivalence.

Contents

1 Introduction

Compliance with traffic regulations is crucial to prevent accidents, maintain order, and enable a smooth flow of traffic. For autonomous vehicles to drive safely on public roads, they must comply with all applicable traffic rules at all times. A popular formalism for defining these traffic rules for autonomous vehicles is temporal logic, which allows the specification of properties over time. However, evaluating compliance with these rules 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 example, the right-before-left rule can be ignored if it is known that no vehicles are approaching from the right.

Motivated by this idea of ignoring specifications that we do not need to consider, based on the knowledge we already have about the traffic situation, we first propose a general simplification algorithm for LTL specifications based on partial information about the Boolean values of propositions within the state sequence of the system, i.e., the traffic situation. If the truth value of a formula can be determined given the available information about the system's state sequence, 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 cannot be determined a priori, the algorithm returns a semantically equivalent but simplified formula instead. The purpose of this algorithm is to serve as a prepossessing step for algorithms that work with LTL specifications, reducing the complexity of verifying the formulas by simplifying them based on the available information.

Although traffic regulations vary depending on location, the Vienna Convention on Road Traffic (VCoRT) [1] establishes a common framework adopted by numerous countries for their national traffic regulations, including Germany's Road Traffic Regulation (StVO). This regulation forms the basis for the formalized traffic rules in [2] and [3], which we use to demonstrate the effectiveness of our simplification algorithm. We first illustrate the theoretical benefit of our approach by simplifying the temporal logic specifications for traffic scenarios from the CommonRoad framework [4]. Subsequently, we integrate our simplification algorithm into the LTL-specification-compliant reachability algorithm from [5] and show that the simplified specifications generated by our algorithm lead to substantial reductions in computational complexity.

1.1 Related Work

The closest related line of research to our simplification approach to temporal logic specifications is three-valued LTL. This extension introduces "?", signifying "unknown", alongside the traditional truth values "true" and "false", which allows for reasoning without complete information about the system's state or behaviour over time. This approach is particularly prominent for runtime verification [6]. The authors in [7] introduce a three-valued semantics for LTL and propose a procedure for generating a deterministic monitor that 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 an 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. Three-valued approaches for Computation Tree Logic and Signal Temporal Logic (STL) can be found in [14] and [15]. Furthermore, extensions such as four-valued LTL [16] exist, which further splits "?" into (1) "will presumably violate the property" and (2) "presumably conform to the property".

Related work regarding the formalization of traffic rules can broadly be categorized into two areas: interstate rules [2], [17]–[21] and intersection rules [3], [22], [23]. Additionally, there are Responsibility-Sensitive Safety rules [24]–[28], 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], [19]–[21], [26], [28], while Metric Temporal Logic (MTL) has been utilized in [2], [3]. STL has been used in [18], [24], [25], [27]. However, some formalizations have been proposed without resorting to temporal logic [22], [23], [29].

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* approaches [30]–[35] consider a single possibility for the future occupancy of traffic participants. *2) Stochastic approaches* are employed in [36]–[40]. They are useful for risk assessment, but only 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 behaviours. We employ the notion of a reachable set from [42, Definition 4] to determine propositions in future states of the traffic scenario.

2 Preliminaries

This chapter introduces the relevant notation and the basics of temporal logic that serve as the foundation for this thesis. Additionally, we provide an overview of the road network and vehicle model, as well as a concrete example of a traffic rule formalized in LTL. Lastly, we briefly introduce an algorithm for LTL specification-compliant reachability analysis, used to evaluate our proposed simplification approach.

2.1 Minkowski Sum and Interval Notation

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, their *Minkowski sum* $A \oplus B$ is defined as the set obtained by adding each element in A to each element in B,

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

Unless explicitly stated otherwise, we interpret intervals over the natural numbers $\mathbb{N}_0$ instead of $\mathbb{R}$ as usual. This means that for $a \in \mathbb{N}_0$ and $b \in \mathbb{N}_0 \cup \{\infty\}$, we have

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

2.2 Linear Temporal Logic

Linear Temporal Logic is an extension of 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:

$$\phi ::= p \mid \top \mid \bot \mid \gamma \wedge \psi \mid \neg\gamma \mid \mathbf{X}\gamma \mid \gamma \mathbf{U} \psi \mid \mathbf{F}\gamma \mid \mathbf{G}\gamma \quad ,$$

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

- $\mathbf{X}\gamma$: γ holds in the next state (next)
- $\gamma \mathbf{U} \psi$: γ holds until ψ holds (until)
- $\mathbf{F}\gamma$: γ eventually holds in a future state (eventually)
- $\mathbf{G}\gamma$: γ holds in all future states (globally)

The formal satisfaction relation [45] is defined over state traces represented by infinite

words over the alphabet $2^{\mathcal{AP}}$. We denote the set of propositions position $i \in \mathbb{N}_0$ as $\tau(i)$ and inductively define the satisfaction of an LTL formula at the position i as follows:

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

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

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

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

2.2.1 LTL Interval Notation

We introduce intervals as a syntactic extension for LTL, similar to [46]. This interval notation is also supported by Spot [47], 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 $c \in \mathbb{N}_0$:

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

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

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

We now introduce the satisfaction relation for LTL operators with intervals in Lemmas 2.1 to 2.3.

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

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

Proof. We prove the lemma by induction on a

Base Case: $a = 0$

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

Induction Step: $a \to a+1$
Induction Hypothesis: $\tau, i \models \mathbf{X}_{[a]}\phi \quad \iff \quad \tau, i + a \models \phi$

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

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

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

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

$$\begin{aligned}
&\tau, i \models \gamma \mathbf{U}_{[a,b]} \psi \\
&\iff \quad \tau, i \models \bigvee_{k \in [a,b]} \Big(\big(\bigwedge_{l \in [0,k-1]} \mathbf{X}_{[l]}\gamma \big) \wedge \mathbf{X}_{[k]}\psi \Big) && \mid \text{Def. } \mathbf{U}_{[a,b]} \\
&\iff \quad \exists k \in [a,b] : \tau, i \models \big(\bigwedge_{l \in [0,k-1]} \mathbf{X}_{[l]}\gamma \big) \ \wedge \ \mathbf{X}_{[k]}\psi && \mid \text{Def. } \vee \\
&\iff \quad \exists k \in [a,b] : \tau, i+k \models \psi \ \text{ and } \ \tau, i \models \bigwedge_{l \in [0,k-1]} \mathbf{X}_{[l]}\gamma && \mid \text{Def. } \wedge \ / \text{ Lem. 2.1} \\
&\iff \quad \exists k \in [a,b] : \tau, i+k \models \psi \ \text{ and } \ \forall l \in [0,k-1] : \tau, i \models \mathbf{X}_{[l]}\gamma && \mid \text{Def. } \wedge \\
&\iff \quad \exists k \in [a,b] : \tau, i+k \models \psi \ \text{ and } \ \forall l \in [0,k-1] : \tau, i+l \models \gamma && \mid \text{Lem. 2.1}
\end{aligned}$$

Case 2: $b = \infty$

$$\begin{aligned}
&\tau, i \models \gamma \mathbf{U}_{[a,\infty]} \psi \\
&\iff \quad \tau, i \models \big(\bigwedge_{k \in [0,a-1]} \mathbf{X}_{[k]}\gamma \big) \wedge \mathbf{X}_{[a]}(\gamma \mathbf{U} \psi) && \mid \text{Def. } \mathbf{U}_{[a,\infty]} \\
&\iff \quad \forall k \in [0, a-1] : \tau, i \models \mathbf{X}_{[k]}\gamma \ \text{ and } \ \tau, i+a \models \gamma \mathbf{U} \psi && \mid \text{Def. } \wedge \ / \text{ Lem. 2.1} \\
&\iff \quad \forall k \in [0, a-1] : \tau, i+k \models \gamma \text{ and } \exists j \in [a, \infty] : \tau, i+j \models \psi && \mid \text{Lem. 2.1 / Def. } \mathbf{U} \\
&\qquad \text{and } \ \forall l \in [a, j-1] : \tau, i+l \models \gamma \\
&\iff \quad \exists j \in [a, \infty] : \tau, i+j \models \psi \ \text{ and } \ \forall k \in [0, a-1] : \tau, i+k \models \gamma \\
&\qquad \text{and } \ \forall l \in [a, j-1] : \tau, i+l \models \gamma \\
&\iff \quad \exists j \in [a, \infty] : \tau, i+j \models \psi \ \text{ and } \ \forall l \in [0, j-1] : \tau, i+l \models \gamma && \square
\end{aligned}$$

Lemma 2.3. *For the operators* $\mathbf{F}_{[a,b]}$ *and* $\mathbf{G}_{[a,b]}$ *the following satisfaction relations hold:*

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

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

The proof can be constructed by using Lemma 2.2 and the definitions of $\mathbf{F}_{[a,b]}$ and $\mathbf{G}_{[a,b]}$.

2.2.2 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 [48], only the temporal operators are affected. Given a finite word τ over the alphabet $2^{\mathcal{AP}}$, the satisfaction relation of the temporal operators is defined as follows:

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

2.2.3 Finite Automata

Finite automata are finite-state machines widely used in formal language theory and model checking. They can be used for verifying whether a finite trace τ satisfies an LTL_f formula ϕ. The language of ϕ, denoted as $\mathcal{L}(\phi)$, is the set of finite words τ over the alphabet $2^{\mathcal{AP}}$ that satisfy ϕ, i.e., $\mathcal{L}(\phi) := \{\tau \mid \tau, 0 \models \phi\}$. To verify whether a given finite trace τ satisfies the LTL_f formula ϕ, we can construct an automaton $\mathcal{A}_\phi$ that accepts precisely the language $\mathcal{L}(\phi)$. In other words, the language of the finite automaton $\mathcal{A}_\phi$, i.e., $\mathcal{L}(\mathcal{A}_\phi) := \{\tau \mid \mathcal{A}_\phi \text{ accepts } \tau\}$ should be equal to $\mathcal{L}(\phi)$. The automaton $\mathcal{A}_\phi$ can be represented as a directed graph with a set of states and transitions. Each transition is associated with a condition, and the trace is accepted if and only if it ends in a final state. We use Spot [47] for translating LTL_f formulas to finite automata. The finite automaton generated by Spot will be used for evaluating the simplification algorithm introduced in Section 3.

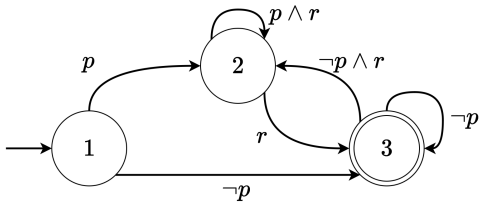


Figure 2.1: Example of a finite automaton for the formula $\phi = \mathbf{G}(p \rightarrow \mathbf{X}r)$ with $p, r \in \mathcal{AP}$, initial state 1 and final state 3.

2.3 Road Network

This section introduces the road network representation. 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, each of which is defined by two polylines that serve as the left and right boundaries. We denote the set of all lanelets as $\mathcal{L} \cup \{\bot\}$, where $\bot$ is a dummy element in case no lanelet exists. 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*$\}$
$attr(l) \in \mathcal{A} \cup \{\bot\}$	Returns the attribute of the lanelet where $\mathcal{A} = \{fork, merge\}$
$\mathrm{l}_{lb}(l)$, $\mathrm{l}_{rb}(l)$	Returns the left and right boundaries of l respectively
$\mathrm{occ}(l)$	Returns the spatial occupancy of the lanelet
$l_{ini}(l), l_{fin}(l)$	Returns 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	$\mathrm{lanes}(l) = \{p \cup s \mid p \in \mathrm{lanes}_{pre}(\emptyset, l) \wedge s \in \mathrm{lanes}_{suc}(\emptyset, l)\}$

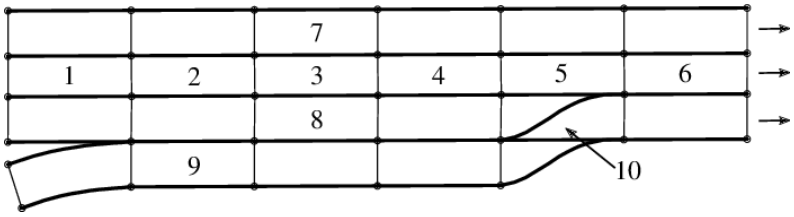


Figure 2.2: Example from [2] of a road network defined by lanelets. Relative to lanelet 3, lanelets 7 and 8 are adjacent left and right and lanelet 2 and 4 are the predecessor and successor lanelets respectively. Lanelets 1 - 6 form a lane. Lanelets 9 and 10 are of type *access_ramp* and lanelet 10 is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

2.4 Vehicle Model

This chapter introduces the vehicle model from [2]. We employ a curvilinear coordinate system [49] 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.

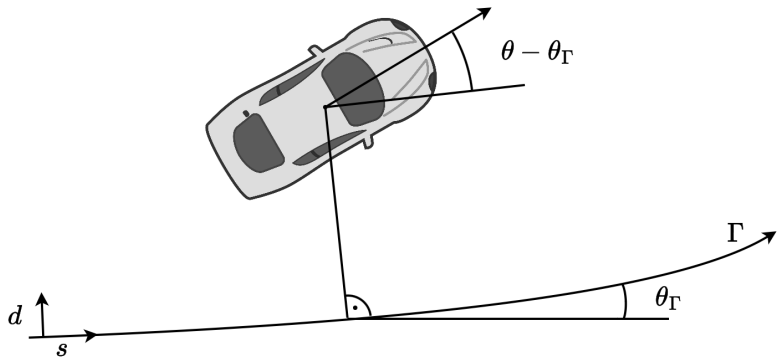


Figure 2.3: Example adapted from [2] of a curve linear coordinate system fitted to the reference path Γ. The longitudinal position is given by s and the lateral distance to the reference path is given by d. The orientation of the reference path and the vehicle are denoted by θ_Γ and θ respectively.

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 and rear bumper can be obtained with $\text{front}(x(t))$ and $\text{rear}(x(t))$ respectively. Similarly, the lateral positions of the leftmost and rightmost points of the car can be obtained with $\text{left}(x(t))$ and $\text{right}(x(t))$. We use $\square_{ego}$ to denote variables associated with the ego vehicle, and we 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 an example later on. The formalization is converted from MTL used in [2] to LTL.

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 carriageway if any of the occupied lanelets has the type *main_carriageway*:

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

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

$$\text{right\_lane}(x_k) \iff \exists l \in \text{lanelets}(x_k) :$$
$$\text{type}(l) = \textit{main\_carriageway} \wedge \text{type}(\text{adj}_r(l)) \neq \textit{main\_carriageway}$$

2.5.2 Entering Vehicles Rule

Given the definitions in the previous section, 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.

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

2.6 Specification-compliant Reachability Analysis

This section briefly introduces the concept of specification-compliant reachability analysis for 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 [46] computes an overapproximation of the reachable states of the ego vehicle that conform to a given set of LTL specifications. These specification-compliant reachable sets 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 Lercher and Althoff [5] builds 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 our simplification algorithm.

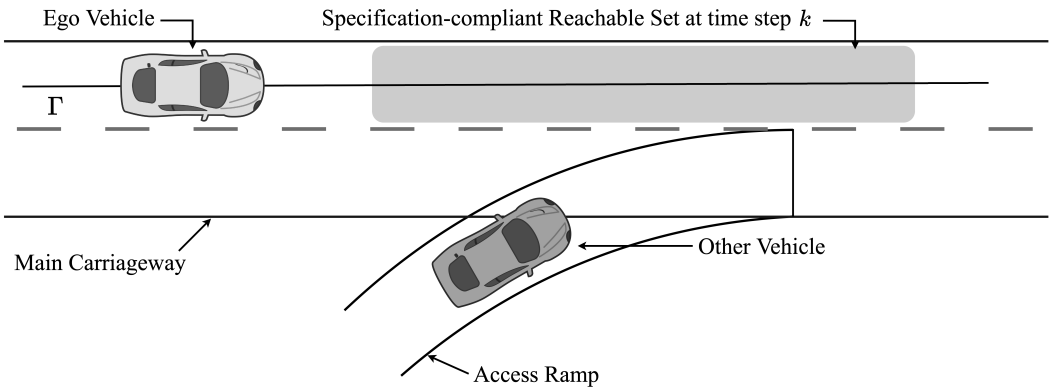


Figure 2.4: Example of a reachability analysis adhering to the Entering Vehicles Rule defined in Section 2.5. As the other vehicle enters the main carriageway via an access ramp, the ego vehicle is not allowed to change to the rightmost lane. Thus, the ego vehicle must be within the grey shaded area at time $t_k = k\Delta t$.

3 Simplification of LTL Formulas

We propose a novel algorithm that simplifies arbitrary LTL formulas by leveraging partial knowledge about the state trace, meaning that we know for certain positions $i \in \mathbb{N}_0$ and certain atomic propositions $p \in \mathcal{AP}$ whether $p \in \tau(i)$ or $p \notin \tau(i)$. The algorithm returns a simplified formula ϕ' that, if evaluated over any trace that conforms to the partial knowledge used for the simplification, 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 knowledge about the state trace to reduce the number of propositions that need to be checked during the verification of the formula.

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 is represented as a mapping

$$\mathcal{P} : \mathcal{AP} \times \mathbb{N}_0 \to \{\top, \bot\} \cup \mathcal{AP} \ .$$

The trace τ *conforms* to the partial knowledge mapping $\mathcal{P}$ iff

$$\forall p \in \mathcal{AP} \ \text{ and } \ \forall i \in \mathbb{N}_0 : \quad \tau, i \models \mathcal{P}(p, i) \iff \tau, i \models p \ .$$

Definition 3.1. The simplification of a formula ϕ is represented as a mapping from trace positions $i \in \mathbb{N}_0$ to simplified LTL expressions:

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

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

$$\phi = \mathbf{G}(p) \quad \text{where } p \in \mathcal{AP}$$

If the proposition a is known to be true at position 4 in the state trace, denoted as $\mathcal{P}(p, 4) = \top$, then the simplification process, given no further information, will produce the following result for $i = 0$:

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

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

Definition 3.2. Let ϕ be a LTL formula. A simplification mapping $\mathcal{S}_\phi$ based on the partial knowledge mapping $\mathcal{P}$ is called *sound* if and only if for any $i \in \mathbb{N}_0$ and any state trace τ conforming to $\mathcal{P}$, the following holds:

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

To better work with simplification mappings, we introduce the following notations:

Definition 3.3. $\mathcal{S}_\phi^{-1}(\phi')$ denotes the set of positions i where the simplified formula $\mathcal{S}_\phi(i)$ is syntactically equivalent to ϕ':

$$\mathcal{S}_\phi^{-1}(\phi') := \{i \mid \phi' \ = \ \mathcal{S}_\phi(i)\}$$

Definition 3.4. $\mathcal{S}_\phi(I)$ denotes the set of simplified formulas for all positions $i \in I$:

$$\mathcal{S}_\phi(I) := \{\mathcal{S}_\phi(i) \mid i \in I\}$$

We define the problem statement as follows: Given an LTL formula ϕ and a partial knowledge mapping $\mathcal{P}$, find the simplification of ϕ at $i = 0$. In addition to defining the simplification, showing that the resulting mapping is sound is the primary objective of this chapter.

3.2 IntervalSimplification Algorithm

The `IntervalSimplification` algorithm works recursively on the structure of the formula ϕ and relies on the following two subfunctions:

1. `PropagateInterval`. Before defining the simplification mapping $\mathcal{S}_\phi$, we need to know for which positions i in the state trace we need to define the mapping. Generating these positions is done by `PropagateInterval` which determines them based on the positions at which the mapping for the input formula has to be defined.

2. `Simplify`. This function computes the simplification mapping of the formula based on the simplification mapping(s) of the subformula(s) at the trace positions generated by `PropagateInterval`.

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

Algorithm 1 IntervalSimplification

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

match ϕ **with:**
  case $p \in \mathcal{AP}$:
    $\mathcal{S}_\phi \leftarrow (i \mapsto \mathcal{P}(p,i))$

  case $v \in \{\top, \bot\}$:
    $\mathcal{S}_\phi \leftarrow (i \mapsto v)$

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

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

return $\mathcal{S}_\phi$

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 input formula can be evaluated. Let $\mathcal{I}_\phi$ 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 recursively defined as:

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

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

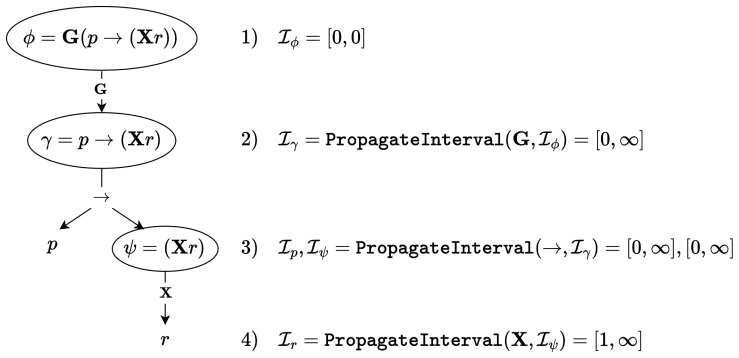


Figure 3.1: Running Example of `PropagateInterval` for $\phi = \mathbf{G}(p \rightarrow (\mathbf{X}r))$

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 input formula at each position in $\mathcal{I}_\phi$. As explained in Section 3.1 our simplification has the goal of eliminating as many proposition evaluations during the validation of the formula as possible with the given information about state trace τ. In the following sections, we define `Simplify` recursively and prove the soundness for each operator.

3.4.1 Simplification of Propositions

The simplification is based on partial knowledge about the state trace, which is encoded in $\mathcal{P}$. As explained in Section 3.1, $\mathcal{P}$ is a 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. The quality of the simplification mainly relies on the quality of the mapping $\mathcal{P}$.

3.4.2 Simplification of Non-Temporal Operators

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

Definition 3.5. Given an LTL formula γ, its simplification mapping $\mathcal{S}_\gamma$ we define the simplification mapping for $\phi = \neg\gamma$ as:

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

Lemma 3.1. *Let γ be an LTL formula, $\mathcal{P}$ a partial knowledge mapping, and $\mathcal{S}_\gamma$ a sound simplification mapping. Then, the simplification mapping $\mathcal{S}_\phi$ with $\phi = \neg\gamma$ is sound.*

Proof. Let τ be a trace that conforms to $\mathcal{P}$. We need to show that $\mathcal{S}_\phi(i) \equiv_{(\tau,i)} \neg\gamma$ 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

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

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 3.6. Given two LTL formulas γ, ψ and their simplification mappings $\mathcal{S}_\gamma$, $\mathcal{S}_\psi$ we define the simplification of $\phi = \gamma \wedge \psi$ 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, $\mathcal{P}$ a partial knowledge mapping, and $\mathcal{S}_\gamma$ and $\mathcal{S}_\psi$ sound simplification mappings. Then, the simplification mapping $\mathcal{S}_{\gamma\wedge\psi}$ 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 given by the simplification of the subformula at the trace position specified by a.

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

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

Lemma 3.3. *Let γ be an LTL formula, $\mathcal{P}$ a partial knowledge mapping, and $\mathcal{S}_\gamma$ a sound simplification mapping. Then, the simplification mapping $\mathcal{S}_\phi$ for $\phi = \mathbf{X}_{[a]}\gamma$ 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 so that ϕ simplifies to the same formula within each interval. Moreover, we require the number of intervals to be minimal, i.e., ϕ must simplify to different formulas in adjacent intervals.

Definition 3.8. Given a set $\Omega \subseteq \mathbb{N}_0$ and a simplification mapping S_ϕ we define `Split` as follows:

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

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

			Ω							
i	0	1	2	3	4	5	6	7	8	9
$\mathcal{S}_\varphi(i)$	ω	ω	ω	γ	γ	γ	ψ	ψ	ψ	ψ

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

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

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

Definition 3.9. Given an LTL formula γ and simplification mapping $\mathcal{S}_\gamma$, we define the simplification mapping for $\phi = \mathbf{G}_{[a,b]}\gamma$ 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 any 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 an LTL formula, $\mathcal{P}$ a partial knowledge mapping, and $\mathcal{S}_\gamma$ a sound simplification mapping. Then, the simplification mapping $\mathcal{S}_\phi$ for $\phi = \mathbf{G}_{[a,b]}\gamma$ is sound.*

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

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

$*_1$ Remark 3.4 and $\mathcal{S}_\gamma(n)$ is the same for all $n \in [x,y]$
$*_2$ Here we can leave out any intervals that simplify to $\top$ because $\mathbf{G}_{[x,y]}(\top)$ will always evaluate to $\top$. Furthermore, this cannot 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 $\top$. □

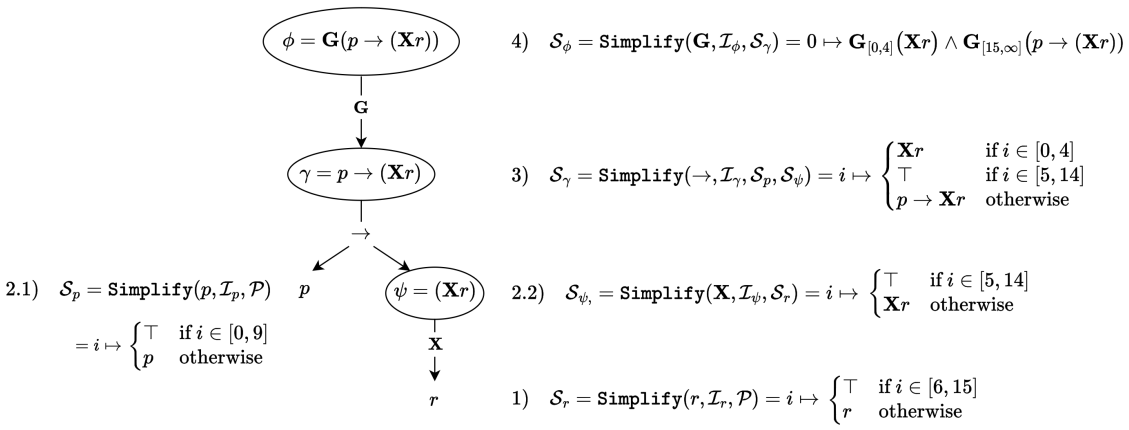


Figure 3.2: Running Example of `Simplify` for $\phi = \mathbf{G}(p \to (\mathbf{X}r))$ given $\forall i \in [0,9] : \mathcal{P}(p,i) = \top$ and $\forall j \in [6,15] : \mathcal{P}(r,j) = \top$

3.4.5 Simplification of the Future Operator

The simplification of the future operator is defined similarly 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 3.10. Given an LTL formula γ and simplification mapping $\mathcal{S}_\gamma$, we define the simplification mapping for $\phi = \mathbf{F}_{[a,b]}\gamma$ 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, $\mathcal{P}$ a partial knowledge mapping, and $\mathcal{S}_\gamma$ a sound simplification mapping. Then, the simplification mapping $\mathcal{S}_\phi$ for $\phi = \mathbf{F}_{[a,b]}\gamma$ is sound.*

Using Lemma 2.3 and substituting $\bot$ for $\top$, the proof is analogous to the proof of Lemma 3.5.

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 simplified subformulas does not change.

Definition 3.11. Given a set $\Omega \subseteq \mathbb{N}_0$ and two simplification mappings S_γ and S_ψ:

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

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

			Ω							
i	0	1	2	3	4	5	6	7	8	9
$\mathcal{S}_\gamma(i)$	ω	ω	ω	γ	γ	γ	φ	φ	φ	φ
$\mathcal{S}_\psi(i)$	ψ	ψ	ψ	δ	δ	ρ	ρ	ρ	ρ	ρ

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

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

Definition 3.12. Given two LTL formulas γ, ψ and their simplification mappings $\mathcal{S}_\gamma$ and $\mathcal{S}_\psi$ we define the simplification for $\phi = \gamma \mathbf{U}_{[a,b]} \psi$ as:

$$\mathcal{S}_{\gamma \mathbf{U}_{[a,b]}\psi}(i) = \begin{cases} \top & \text{if } \exists n \in [a,b] : \mathcal{S}_\psi(i{+}n) = \top \wedge \forall n' \in [0,n-1] : S_\gamma(i{+}n') = \top \\ \bot & \text{if } \forall n \in [a,b] : S_\psi(i{+}n) = \bot \vee \exists n' \in [0,a-1] : S_\gamma(i{+}n') = \bot \\ \bigvee\limits_{\substack{[x,y]\in \\ \mathtt{Split}([i+a,i+b],\mathcal{S}_\gamma,\mathcal{S}_\psi)}} \mathcal{S}_{\mathbf{G}_{[0,x-i-1]}\gamma}(i) \wedge \mathbf{X}_{[x-i]}\big(\mathcal{S}_\gamma(x)\mathbf{U}_{[0,y-x]}\mathcal{S}_\psi(x)\big) & \text{otherwise} \end{cases}$$

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 $\mathcal{S}_{\gamma \mathbf{U}_{[a,b]}\psi}$ is sound.*

Proof. The first two cases, $\top$ and $\bot$ follow directly from Lemma 2.2. Let τ be a trace that conforms to $\mathcal{P}$. 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:

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

3.5 Soundness of the IntervalSimplification Algorithm

We now prove that the simplification mapping generated by the `IntervalSimplification` algorithm is sound (Definition 3.2).

Theorem 3.9. *Let ϕ be an LTL formula and $\mathcal{P}$ a partial knowledge mapping. The recursively defined simplification mapping $\mathcal{S}_\phi$ returned by the `IntervalSimplification` algorithm is sound.*

Proof. Let τ be a state trace conforming to $\mathcal{P}$
Base Cases:

$\phi = p \in \mathcal{AP}$: The simplification is given as $\mathcal{S}_\phi(i) = \mathcal{P}(p, i)$. This mapping is sound as τ conforms to $\mathcal{P}$ per assumption.

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

Induction Step:

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

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

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

3.6 Limitations

The `IntervalSimplification` algorithm successfully incorporates all information about the state trace into the simplified formula. Each proposition known to be true or false is incorporated as either $\top$ or $\bot$ in the formula and then subsequently used to simplify the entire expression. Nevertheless, the simplification algorithm has some limitations.

3.6.1 Sliding Window Problem

The problem describes an increase in the formula size for nested temporal operators that are defined over large intervals. This problem is best explained using an example. Given an LTL formula ϕ and a simplification mapping of the subformula γ:

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

We now apply the `IntervalSimplification` algorithm:

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

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 has to incorporate the simplification of γ at a different position, a new formula is created every time. In these cases, we can selectively apply the algorithm to specific subformulas or choose not to apply it to the entire formula. It is generally unlikely to have definitive knowledge about propositions far into the future. Typically, our understanding is more complete for the immediate future, with certainty decreasing as we look further ahead. This pattern of diminishing certainty over time is common in many predictive and planning scenarios. Furthermore, the LTL specifications used for the CommonRoads framework do not suffer from deeply nested temporal operators and demonstrate the practical applicability of the algorithm.

3.6.2 No Semantic Simplifications

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

$$p \wedge \neg p \equiv false$$

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

4 Partial Knowledge Mapping for Traffic Scenario Propositions

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 mapping $\mathcal{P}$ 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 refers to the distance along 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^{min} 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 $\mathcal{S}_k$.

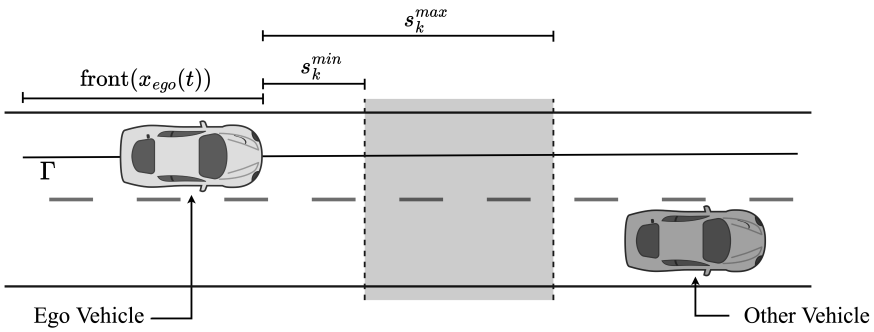


Figure 4.1: Example of a traffic scenario at time $t = 0$ and a longitudinal position overapproximation k time steps in the future. $\mathcal{S}_k$ can be used to bound the longitudinal position of the front bumper using s_k^{min} and s_k^{max} illustrated by the grey shaded area.

We exemplify the usage of longitudinal overapproximation in determining the Boolean values of the proposition in_front_of(x_{ego}, x_o), defined in Section 2.5.2. This proposition 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 $\bot$ 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:

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

The Figure 4.1 shows the states of the two vehicles at $t = 0$. Assuming that the other vehicle does not move backwards, we can set the $\mathcal{P}(\text{in\_front\_of}(\mathrm{x_{ego}}, \mathrm{x_o}), \mathrm{k})$ 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 Set Overapproximation

The longitudinal and lateral overapproximations from Section 4.1 allow us to determine relational propositions like in_front_of and left_of, but they are insufficient for determining positional propositions like on_access_ramp. For this, we need to bound the

potential positions in both the longitudinal and lateral directions by also considering the shape of the ego vehicle. For this, we use the notion of reachable 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 centre point of the vehicle, denoted as $\mathcal{C}$. The initial set can be set to the actual centre 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 *centre point overapproximation* of the ego vehicle at time step k is defined as:

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

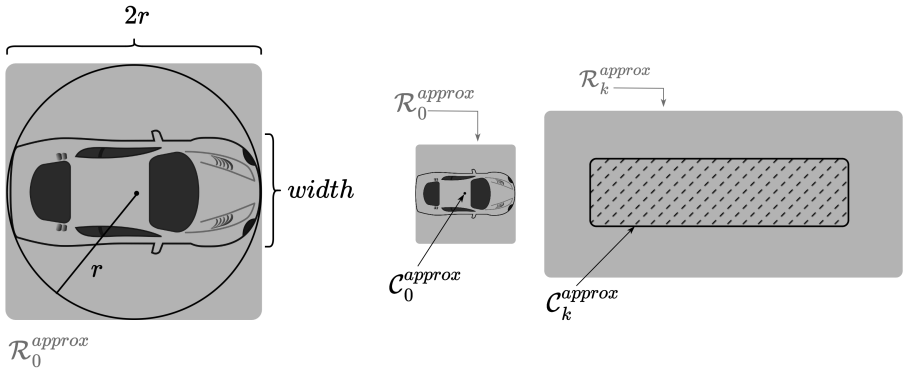
$$\mathcal{C}_k^{approx} = \mathcal{C}_0^{approx} \oplus \left\{ \begin{pmatrix} s_k \\ d_k \end{pmatrix} \in \mathbb{R}^2 \;\middle|\; s_k^{min} \leq s_k \leq s_k^{max} \;\wedge\; d_k^{min} \leq d_k \leq d_k^{max} \right\}$$

Given $\mathcal{C}_k^{approx}$ we define the overapproximation of the reachable set $\mathcal{R}_k$ by adding an overapproximation of the shape of the vehicle to $\mathcal{C}_k^{approx}$. We define this vehicle shape overapproximation initially with a circle around the vehicle centre of radius r and then overapproximate this circle with a square, as shown in the example of Definition 4.3. This approach allows us to ignore the orientation of the vehicle, which 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 $\mathcal{C}_k^{approx}$ and $\mathcal{R}_k^{approx}$



Given $\mathcal{R}^{approx}$, we can determine positional propositions such as on_access_ramp(x_{ego}), which is used in Section 2.5 to formalize the entering vehicles rule. 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 $\mathcal{R}_k^{approx}$, and consequently in $\mathcal{R}_k$, lies within an access ramp lanelet, we can set the proposition to $\bot$ at time step k. Similarly, if every point in $\mathcal{R}_k^{approx}$ lies within an access ramp lanelet, the proposition can be set to $\top$. This is formally defined as follows:

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

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, it rarely occurs that all points of the overapproximation $\mathcal{R}_k^{approx}$ lie within a single lanelet or lane, which makes the condition for setting propositions like on_access_ramp(x_{ego}) to $\top$ quite restrictive. 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 attributes it 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:

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

If all points in an intersection set lie within the area for a positional proposition, we can set this proposition to $\top$, as we know that the vehicle at least partially covers that area. Of course, we want the intersection set to be as small as possible, as this will increase the likelihood that all points lie within the area defined by the proposition.

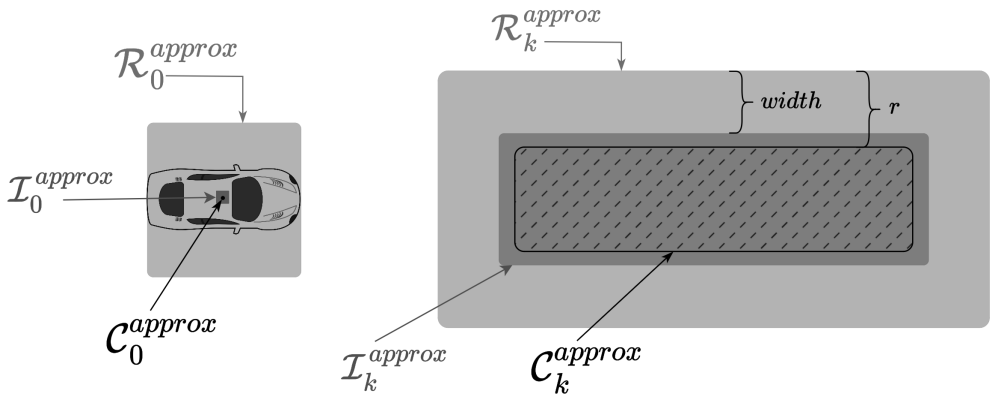
In our pursuit of minimizing the intersection set, it is important to note that there is not a single unique minimal intersection set. For instance, at $k = 0$, any $\{p\}$ with $p \in \mathcal{R}_0$ qualifies as a minimal intersection set. We construct $\mathcal{I}_k^{approx}$ similar to $\mathcal{R}_k^{approx}$, by essentially adding a smaller vehicle approximation, reduced by the width of the car, onto the centre overapproximation $\mathcal{C}_k^{approx}$. 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 $\mathcal{I}_k$ remains covered by the car. We denote the width of the vehicle as *width*.

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

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

Example. Propagation of $\mathcal{C}_k^{approx}$, $\mathcal{R}_k^{approx}$ and $\mathcal{I}_k^{approx}$



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

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

5 Implementation

The python implementation includes the general `IntervalSimplification` algorithm and methods to construct the partial knowledge mapping $\mathcal{P}$ for traffic scenarios within the CommonRoad framework [4]. We have integrated both interstate traffic rules from [2] and intersection traffic rules from [3].

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 5.1: 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 lists the predicates for which we have implemented methods to build the partial knowledge mapping $\mathcal{P}$. The right column indicates which traffic rules utilize these predicates in their temporal logic specifications.

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

Predicate	Rules	Predicate	Rules
on_main_carriageway	R_I2, R_I5	at_stop_sign	R-IN1
on_access_ramp	R_I2, R_I5	passing_stop_line	R-IN1
right_lane	R_I2, R_I5	on_intersection	R-IN3, R-IN4, R-IN5
in_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	turning_(left/right) going_straight	R-IN3, R-IN4, R-IN5

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 the LTL specifications are used to generate a specification-compliant vehicle trajectory for the next 15 time steps. We first generate the partial knowledge map $\mathcal{P}$ and then apply the `IntervalSimplification` algorithm to the LTL specification of the entering vehicles rule.

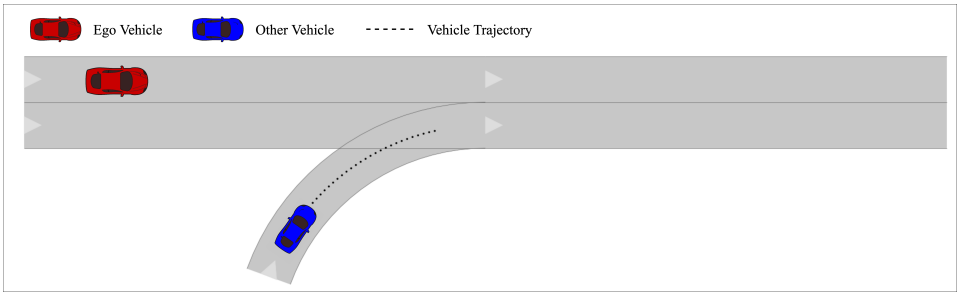


Figure 5.1: Traffic Scenario with one other vehicle entering the main carriageway.

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 rule is instantiated with every other traffic participant, yielding as many formulas as traffic participants. In our case we have only one other traffic participant except the ego vehicle and the simplified formula looks as follows:

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

The 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

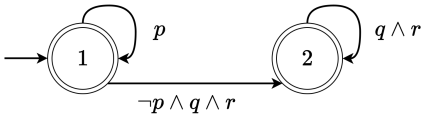
In this section, we present a comprehensive evaluation of our proposed simplification approach. To evaluate its effectiveness, we interpret the simplified formulas over finite traces and use two different metrics: Number of *Unknown Propositions* and number of *Proposition Evaluations*. We compare these metrics for the initial formulas against the simplified formulas. Furthermore, we evaluate the performance of the algorithm as a preprocessing stage to a specification-compliant reachability algorithm [5] by measuring the reduction in computation time by using the simplified LTL formulas.

6.1 Evaluation Metrics

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 assess the quality of our mapping $\mathcal{P}$ by comparing the metric for the initial formula and the simplified formula. Furthermore, it is an upper bound for the number of proposition evaluations needed to verify the simplified formula. For the initial formula, this metric is equal to the product of the number of distinct propositions in the formula and the trace length. The percentage reduction of this metric can be interpreted as the percentage of propositions that were determined a priori. If propositions reoccur in different formulas, they are counted more than once.

The *Proposition Evaluations* metric is derived from the finite automaton 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*. The lower bound is calculated by finding the trace that satisfies ϕ and needs the least amount of proposition evaluations in the best case, i.e., we assume that only the necessary propositions are evaluated for determining the next state in the automaton. For the upper bound, we assume the opposite, i.e., that we evaluate the propositions in the worst possible order. For the example in Figure 6.1 for a trace of length 10 the least amount of proposition evaluations is 10 if p is always true and always evaluated first at every position in the trace. The upper bound for the proposition evaluations is 21 if q and r are always true and p is false, at least at the first position in the trace.

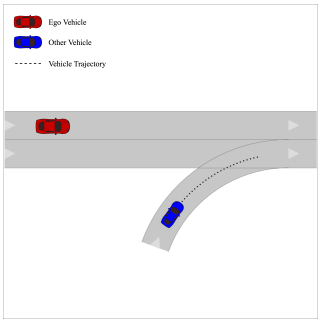
Figure 6.1: Example of the number of *Proposition Evaluations* of a Finite Automata. We count the number of proposition evaluations in best and worst case. a_i denotes the evaluation of proposition a at position i in the trace



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

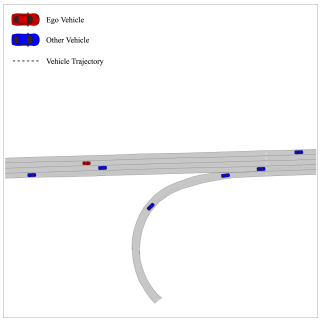
lower bound: 10 $(p_0, ..., p_9)$
upper bound: 21 $(p_0, q_0, r_0, q_1, r_1, ..., q_9, r_9)$

6.2 Evaluation on Interstate Scenarios



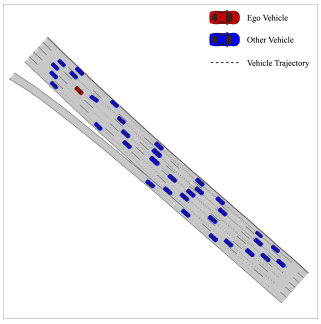
INTERSTATE_1
Trace Length: 16
Traffic Participants: 1

Rule	Unknown Propositions			Proposition Evaluations		
	Initial (#)	Simp. (#)	Red. (%)	Initial (#)	Simp. (#)	Red. (%)
R_G1	64	35	45.3	16 - 62	12 - 30	25.0 - 51.6
R_G4	32	25	21.9	16 - 32	16 - 25	0.0 - 21.9
R_I1	48	16	66.7	16 - 48	16 - 16	0.0 - 66.7
R_I5	80	35	56.3	16 - 80	16 - 35	0.0 - 56.3



INTERSTATE_2
Trace Length: 16
Traffic Participants: 6

Rule	Unknown Propositions			Proposition Evaluations		
	Initial (#)	Simp. (#)	Red. (%)	Initial (#)	Simp. (#)	Red. (%)
R_G1	384	84	78.1	96 - 372	9 - 15	90.6 - 96.0
R_G4	32	32	0.0	16 - 32	16 - 32	0.0 - 0.0
R_I1	48	16	66.7	16 - 48	16 - 16	0.0 - 66.7
R_I5	480	52	89.2	96 - 480	9 - 16	90.6 - 96.7



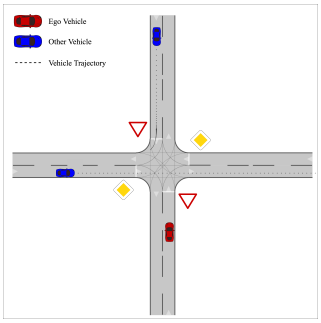
INTERSTATE_3
Trace Length: 16
Traffic Participants: 34

Rule	Unknown Propositions			Proposition Evaluations		
	Initial (#)	Simp. (#)	Red. (%)	Initial (#)	Simp. (#)	Red. (%)
R_G1	2176	632	71.0	544 - 2108	96 - 204	82.4 - 90.3
R_G4	32	32	0.0	16 - 32	16 - 32	0.0 - 0.0
R_I1	48	17	64.6	16 - 48	16 - 17	0.0 - 64.6
R_I5	2720	266	90.2	544 - 2720	16 - 28	97.1 - 99.0

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, which causes the ego vehicle's potential positions to multiply with each time step, making it more difficult to ascertain propositions in 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 determined, which is especially visible for rule R_I5 in the INTERSTATE_2 and INTERSTATE_3 scenarios. 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 over 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 a 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.

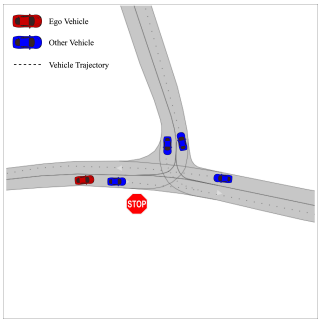
A comparison of the *Proposition Evaluations* metric with the *Unknown Propositions* metric demonstrates the effectiveness of the `IntervalSimplification` algorithm in integrating knowledge about the state trace into the simplified formula. Unsurprisingly, the number of unknown propositions always serves as an upper bound for the number of proposition evaluations required for an accepting trace. However, the algorithm even achieves reductions that go well below this bound, indicating its ability to leverage the available knowledge, to exclude proposition evaluations that do not contribute to the satisfaction of the expression. A notable example of this can be found in the simplification of R_G1 in the INTERSTATE_2 scenario, where a reduction of the *Unknown Propositions* metric of 78% leads to a 96% upper bound reduction of the number of *Proposition Evaluations*. This highlights the algorithm's capability of not only incorporating the given knowledge, but also using this knowledge to further eliminate proposition evaluations.

6.3 Evaluation on Intersection Scenarios



INTERSECTION_1
Trace Length: 16
Traffic Participants: 2

Rule	Unknown Propositions			Proposition Evaluations		
	Initial (#)	Simp. (#)	Red. (%)	Initial (#)	Simp. (#)	Red. (%)
R-IN1	80	42	47.5	16 - 64	0 - 0	100 - 100
R-IN3	608	420	30.9	32 - 608	0 - 0	100 - 100
R-IN4	576	448	22.2	32 - 576	18 - 128	43.8 - 77.8
R-IN5	256	160	37.5	32 - 256	0 - 0	100 - 100



INTERSECTION_2
Trace Length: 16
Traffic Participants: 4

Rule	Unknown Propositions			Proposition Evaluations		
	Initial (#)	Simp. (#)	Red. (%)	Initial (#)	Simp. (#)	Red. (%)
R-IN1	80	51	36.3	16 - 64	9 - 18	43.8 - 71.9
R-IN3	1216	835	31.3	64 - 1216	0 - 0	100 - 100
R-IN4	1152	899	22.0	64 - 1152	57-338	10.9 - 70.7
R-IN5	512	322	37.1	64 - 512	1-37	98.4 - 92.8

As observed in the interstate scenarios, the construction of the knowledge mapping $\mathcal{P}$ works effectively, with 22% being the lowest reduction for the *Unknown Propositions* metric. The reduction of the *Proposition Evaluations* metric compared to the reduction in the *Unknown Propositions* is even greater than for the interstate scenarios, with 70.7% being the lowest upper bound reduction. 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. Using the available knowledge the `IntervalSimplification` algorithm is able to determine the correct case and can thus significantly reduce the size of the expression. In particular, for the INERSECTION_1 scenario, three of the four rules could be simplified to $\top$. We can also see that our original example from the introduction is fulfilled by being able to ignore the 'right before left' rule, as no vehicle is ever approaching from the right.

6.4 Specification-Compliant Reachability Analysis

To evaluate the benefits of our simplification approach for algorithms that work with LTL specifications, we integrated it into the specification-compliant reachablity analysis by Lercher and 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 specification-compliant motion planning (as discussed in Section 2.6). We benchmarked the algorithm on the three interstate traffic scenarios defined in Section 6.2, applying 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 incorporating our simplification step.

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

Scenario	**Initial**	**With Simplification**			**Reduction**
	Total	Simplification	Reachability	Total	
INTERSTATE_1	0.057 *s*	0.009 *s*	0.042 *s*	0.051 *s*	10.53%
INTERSTATE_2	1.099 *s*	0.067 *s*	0.068 *s*	0.135 *s*	87.71%
INTERSTATE_2	/	0.371 *s*	0.081 *s*	0.452 *s*	/

The "Initial" column shows the total computation time without simplification, while the "With Simplification" columns break down the time into the simplification step itself ("Simplification") and the subsequent reachable set computation ("Reachability"). The "Total" column under "With Simplification" represents the combined time for both steps, and the "Reduction" column quantifies the computation time reduction 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 10.53% was observed. However, for the more intricate INTERSTATE_2 scenario, the simplification step led to a remarkable reduction of 87.71%. For the INTERSTATE_3 scenario, the reachability analysis using unsimplified formulas could not be computed, as the Spot tool failed to generate the finite automaton. Consequently, we conducted the reachability analysis only for rules related to traffic participants on the access ramp, resulting in a computation time of 0.119 *s*. This represents a 46.9% increase compared to the time required for the reachability analysis with simplified formulas. These results demonstrate that the performance gain stems not only from our simplification algorithm's ability to identify ignorable rules but also from its capacity to simplify necessary rules. Furthermore, 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 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 LTL specifications for autonomous vehicles by leveraging partial knowledge about the state trace. We developed an algorithm that can reduce LTL formulas to either $\top$, $\bot$, or a semantically equivalent but simplified form, utilizing information about when propositions hold or do not hold during the system's state trace.

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 semantic equivalence. 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 information about the state trace. This opens up opportunities for broader applications in areas such as robotics, power 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 STL or MTL. Moreover, investigating techniques for more efficiently obtaining the partial knowledge about the state trace could further enhance the effectiveness of the simplification algorithm.

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.

[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," in *Proceedings of the IEEE Intelligent Vehicles Symposium (IV)*, 2024.

[6] M. Leucker and C. Schallhart, "A brief account of runtime verification," *The Journal of Logic and Algebraic Programming*, vol. 78, no. 5, pp. 293–303, 2009.

[7] A. Bauer, M. Leucker, and C. Schallhart, "Runtime verification for LTL and TLTL," *ACM Trans. Softw. Eng. Methodol.*, vol. 20, no. 4, 2011.

[8] S. Vijzelaar and W. Fokkink, "Creating büchi automata for multi-valued model checking," in *Proc. of the International Conference on Formal Techniques for Distributed Objects, Components, and Systems*, 2017, pp. 210–224.

[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, 2023.

[10] K. Havelund and G. Rosu, "Monitoring programs using rewriting," in *Proceedings 16th Annual International Conference on Automated Software Engineering (ASE 2001)*, pp. 135–143.

[11] G. Fraser and F. Wotawa, "Using LTL rewriting to improve the performance of model-checker based test-case generation," in *Proceedings of the 3rd International Workshop Advances in Model Based Testing, AMOST 2007*, pp. 64–74.

[12] L. Zhao, T. Tang, J. Wu, and T. Xu, "Runtime verification with multi-valued formula rewriting," in *2010 4th IEEE International Symposium on Theoretical Aspects of Software Engineering*, pp. 77–86.

[13] G. Roşu and K. Havelund, "Rewriting-Based Techniques for Runtime Verification," *Automated Software Engineering*, vol. 12, pp. 151–197, 2005.

[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, "Property-directed verified monitoring of signal temporal logic," in *Runtime Verification*, J. Deshmukh and D. Ničković, Eds., Cham: Springer International Publishing, 2020, pp. 339–358.

[16] A. Bauer, M. Leucker, and C. Schallhart, "Comparing LTL semantics for runtime verification," *J. Log. Comput.*, vol. 20, pp. 651–674, 2010.

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

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

[19] A. Rizaldi, F. Immler, and M. Althoff, "A formally verified checker of the safe distance traffic rules for autonomous vehicles," in *Proceedings of the 8th International Symposium on NASA Formal Methods*, 2016, pp. 175–190.

[20] A. Rizaldi, J. Keinholz, M. Huber, J. Feldle, F. Immler, M. Althoff, E. Hilgendorf, and T. Nipkow, "Formalising and Monitoring Traffic Rules for Autonomous Vehicles in Isabelle/HOL," in *13th International Conference on integrated Formal Methods*, 2017.

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

[22] 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, 2021.

[23] A. Karimi and P. S. Duggirala, "Formalizing traffic rules for uncontrolled intersections," in *2020 ACM/IEEE 11th International Conference on Cyber-Physical Systems (ICCPS)*, pp. 41–50.

[24] M. Hekmatnejad, S. Yaghoubi, A. Dokhanchi, H. B. Amor, A. Shrivastava, L. Karam, and G. Fainekos, "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*, 2019.

[25] 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)*, pp. 1050–1056.

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

[27] N. Aréchiga, "Specifying Safety of Autonomous Vehicles in Signal Temporal Logic," in *2019 IEEE Intelligent Vehicles Symposium (IV)*, pp. 58–63.

[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)*, pp. 1477–1483.

[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)*, pp. 1471–1476.

[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*, pp. 458–464.

[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, 2010.

[32] A. Eidehall, "Multi-target threat assessment for automotive applications," in *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pp. 433–438.

[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, 2009.

[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)*, pp. 1220–1225.

[35] A. Barth and U. Franke, "Where will the oncoming vehicle be the next second?" In *2008 IEEE Intelligent Vehicles Symposium*, pp. 1068–1073.

[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, 2008.

[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, 2009.

[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.

[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, 2011.

[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*, pp. 1043–1048.

[41] M. Koschi and M. Althoff, "SPOT: A tool for set-based prediction of traffic participants," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1686–1693.

[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, 2016.

[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, 2014.

[44] M. Koschi and M. Althoff, "Interaction-aware occupancy prediction of road vehicles," in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1–8.

[45] C. Baier and J.-P. Katoen, *Principles of Model Checking*. MIT Press, 2008, pp. 235–239, Section 5.1.2.

[46] 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.

[47] Alexandre Duret-Lutz, E. Renault, M. Colange, F. Renkin, A. Gbaguidi Aisse, P. Schlehuber-Caissier, T. Medioni, A. Martin, J. Dubois, C. Gillard, *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)*, vol. 13372, 2022, pp. 174–187.

[48] 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*, Beijing, China: AAAI Press, 2013, pp. 854–860.

[49] 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.