TECHNICAL UNIVERSITY OF MUNICH

DEPARTMENT OF INFORMATICS

Bachelor's Thesis in Informatics

Traffic Scenario Synthesis from Temporal Logic Constraints

Maximilian Frühauf

TECHNICAL UNIVERSITY OF MUNICH

DEPARTMENT OF INFORMATICS

Bachelor's Thesis in Informatics

Traffic Scenario Synthesis from Temporal Logic Constraints

Synthese von Verkehrsszenarien basierend auf Vorgaben in Temporaler Logik

Author:	Maximilian Frühauf
Supervisor:	Prof. Dr.-Ing. Matthias Althoff
Advisor:	Moritz Klischat, M.Sc.
Submission Date:	15.03.2021

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

Ich versichere, dass ich diese Bachelor's Thesis selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

Munich, 15.03.2021 Maximilian Frühauf

Acknowledgments

This thesis would not have had the results it eventually did without the support of my advisor Moritz Klischat, who was always available for answering questions and providing useful ideas on circumventing road blocks; pun intended ;).

Additionally, I would like to thank my family, girlfriend and friends for helping me to navigate this project without getting stuck at a dead end or running out of gas. As we say in Germany, it was the “yellow of the egg”.

Abstract

Test case generation for autonomous vehicles has gained large scale attention in recent years due to advances in motion planning algorithms, creating a need for verified safety with testing scenarios of these novel approaches. To this end we present in this work a scenario synthesis approach based on a templating system, using which we are able to synthesize many concrete scenarios satisfying any given specification, which are then used for verifying a motion planner.

We represent the problem of generating many concrete realizations from a single specification as a satisfiability problem of a linear Temporal Logic formula, modelling the behavior of every vehicle on given map data. This approach has the advantage of implicitly coupling all vehicle interactions together, allowing for complex behavior to emerge while specifying a minimal number of constraints on the system. Based on a satisfying assignment to such a linear Temporal Logic formula we synthesize a scenario by formulating it as an optimization problem, whose solution yields a concrete scenario with exact behavior for all vehicles.

To show the applicability of our approach, we evaluate it with two templates representing a highway and urban scenario showing the range different types of behavior generated. Therefore, we deem this approach highly relevant to the domain of falsification testing for autonomous vehicles and detail future steps for integrating it into a closed loop testing framework.

Contents

Acknowledgments **iii**

Abstract **v**

1. Introduction **1**
1.1. Motivation 1
1.2. Related Work 1
1.3. Problem Statement 3
1.4. Contributions 3
1.5. Overview 4

2. Terminology **5**
2.1. Lanes & Lanelets 5
2.2. Curvilinear Coordinates 6
2.3. Linear Temporal Logic 6
2.3.1. Büchi Automata 7
2.3.2. General Reactivity 1 7
2.4. Scenario Templates 9

3. Satisfying solutions from Templates **11**
3.1. Reachable Lanes 12
3.2. Predicate Graph 12
3.2.1. Critical Sections 13
3.2.2. Edge Requirements 14
3.2.3. Example 14
3.3. LTL Translation 15
3.3.1. Predicate Graph Encoding 15
3.3.2. Node identifiers 16
3.3.3. Relation Encoding 18
3.4. GR(1) solving and enumeration 24
3.4.1. Exclusion Automaton 24
3.4.2. BDD Caching 26
3.4.3. GR(1) Expression 26
3.5. Accepting Word Parsing 27

4. Optimization based Scenario Synthesis **29**
4.1. Switching Times 29

4.2. Expressing Relations as Mixed-Integer Constraints . . . 30
4.3. Longitudinal Problem . . . 31
4.4. Lateral Problem . . . 32

5. Results **33**
5.1. Highway Scenario . . . 33
5.1.1. Computational Performance . . . 35
5.1.2. Scenario Evaluation . . . 37
5.2. Urban Scenario . . . 38
5.2.1. Computational Performance . . . 39
5.2.2. Scenario Evaluation . . . 40
5.3. Discussion . . . 41

6. Conclusion & Future Work **45**
6.1. Lessons Learned . . . 45
6.2. Future Work . . . 46

A. Appendix **47**

List of Figures **51**

List of Tables **53**

Bibliography **55**

1. Introduction

1.1. Motivation

Studies show that around 90% of vehicle crashes are due to human error [1]. Autonomous vehicles aim to reduce those numbers by cutting the human out of the loop entirely and having a machine make driving decisions. However, such an approach requires the validation of the motion planning algorithm, responsible for vehicle movement. A purely real world evaluation of this algorithm is however infeasible, as the distance which would have to be driven to prove reasonable safety is exceedingly large [2, 3, 4]. Therefore, testing in simulation is performed to offload some required testing time to a synthetic environment which can be executed much faster than real time.

Current approaches are able to collect massive databases of scenarios, usually from human driving data, and validate novel motion planners against those databases. [5] One limitation of this approach is however that only behavior in the database can be tested against and very rare behavior is difficult to capture at scale.

Therefore, in this work we propose a novel approach to generate many scenarios from a single template and accompanying map data, which can be easily defined to express constrained environments for a set of vehicles or specifications with many degrees of freedom within the same system. Using this templating system it is possible to generate large amounts of rare event behavior for a set of vehicles, as well as synthesize concrete scenarios which can in turn be used to test motion planners.

This work is highly relevant for falsification testing of autonomous vehicle motion planning algorithms, as each generated scenario can be regarded as a test case for a motion planner. Specifically, if our approach is able to generate scenarios of sufficient realism, and a particular motion planner fails to act safely within it, we can conclude that the planner must be unsafe.

1.2. Related Work

Generating scenarios is a topic which has gained much interest in the past few years. Many approaches have been proposed to deal with the core difficulty of generating large numbers of such sufficiently different conditions for autonomous driving. Applications of those scenarios range from verification of static scenarios against kinematic feasibility [5], to validation of traffic rule [6, 7] consideration, to synthesis of complete systems from complex specifications [8, 9, 10, 11].

Sampling based techniques As a starting point most systems use a knowledge base of some sort to specify known true facts about the domain of autonomous vehicles, for instance Bagschick et al. take this logical conclusion and generate entire scenarios from such a knowledge base [12]. However, this approach has limitations around synthesizing novel scenarios, which are not directly derivable from a knowledge base. To address this issue sampling techniques can be used in conjunction with a rich scenario specification language [8, 9]. While the results by Fremont et al. are impressive, they do not model a temporal dimension in the specifications they generate, limiting the amount of complex behavior synthesized [8]. Majumdar et al. address this by considering temporal behavior [9], but also rely on sampling techniques, having a big computational impact when searching though large solution spaces with sufficient resolution. However, further cross-entropy based sampling techniques seem promising for identifying dangerous scenarios from a specification [13, 10].

Formal Methods Another approach to the problem are formal methods such as the one presented in this work. Their primary goal is to prove components of an autonomous system correct, or in the case where such a proof becomes intractable due to increasing system complexity, rely on falsification [14, 15, 16]. Since the domain of autonomous vehicles is large and modelling all constraints of the physical world is exceedingly difficult, the latter approach, falsification, has gained popularity. Here, the goal is to generate test scenarios and apply them to a specific algorithm, testing for unsafe behavior. The core difficulty in for all approaches however, is generating only scenarios which provide meaningful test cases, as otherwise the problem becomes computationally intractable. Many concepts have been proposed to approach this problem with formal methods, i.e. by formulating it as an constraint satisfaction problem [17], evolutionary algorithm [11] or optimization problem [18, 19].

Temporal Logic with optimization approaches A promising direction is the synthesis of dynamic behavior using temporal logic, as this class of logic allows specification of constraints over time, it lends itself well to controller synthesis for dynamic systems, such as autonomous vehicles. This approach has existed for some time but was computationally infeasible [20] until a result by Piterman et al. showing that a large subset of temporal logic is solvable in polynomial time [21]. Based on this, general controller synthesis algorithms for dynamic systems have been proposed [22], as well as specific ones applied to human robotics [23]. Additionally, results foregoing an automaton based representation have also been considered [24, 25].

For complex systems however, temporal logic based systems synthesizing a realizable strategy are often coupled with optimization based approaches [26, 27, 25] to yield concrete scenarios. This approach has the advantage that many complex specifications can be encoded easily in various forms temporal logic, which has a large expressive power and concrete synthesis is efficient when given by an optimization problem. This is especially promising as, for instance traffic rules in autonomous vehicles can be directly specified in temporal logic [6, 28]. Additionally, the well-behaved response of these systems to slight disturbances in the inputs is also attainable [29].

1.3. Problem Statement

In this work we cover the generation of concrete Traffic Scenarios from abstract specifications. Based on a given domain, or map, as well as constraints on vehicles acting on that map over time, we synthesize feasible scenarios it. Specifically, our approach provides all such satisfying solutions to an abstract specification.

While doing so the input specifications to our approach have both the ability to define exact behavior of all involved vehicles, or one can refrain from doing so, and our approach will also accept specifications having many degrees of freedom. All of those specifications on such a spectrum of complexity are processed in the same way and yield the complete set of satisfying solution to such specifications.

Additionally, we make this approach practically applicable, by constraining ourselves to techniques for synthesizing those solutions which are computationally feasible and dot not scale exponentially in computation time with the size of the problem. Also, to provide a user with the best feedback possible, we adopt a fail fast approach, meaning that invalid specifications are detected early in the process and give rich feedback on the causing error.

1.4. Contributions

Our main contribution of this work is a novel scenario synthesis approach in the space of Autonomous Vehicles. With its main goal being simple specification of constraints over time by use of a template system on known map data. From such a template we are able to synthesize concrete scenarios over a series of time steps, which are consistent with the original specification.

Firstly, we translate a given template and map data to a representation in linear Temporal Logic (LTL) by considering all behaviors possible within the template's constraints. From those behaviors we build a graph representing every reachable state for a vehicle and the requirements to reach any such state. Based on this graph we additionally derive some universal constraints required to express general truths in the domain of autonomous vehicles, i.e. serial lane movements. This graph and its universal constraints are then translated to LTL and checked for satisfiability. Every such solution to this LTL representation can then be regarded as a run through the previously built predicate graph, while observing general road constraints. From this solution, we interface with the approach introduced by Klischat [18] to synthesize concrete scenarios from it.

Additionally, we use previous solutions of the template's LTL formula to enumerate all satisfying solutions to it and synthesize concrete scenarios for each of them. By repeating this process until no more distinct satisfying solutions are found, we are able to generate the complete set of satisfying solutions for a given template.

Using this enumeration of all satisfying solutions for a template we are able to generate novel driving behavior from a small set of input data, i.e. a lane network and constraints over time in the shape of a template. We then evaluate the quality of generated scenarios using our approach in chapter 5 and give a set of potential use cases in falsification of motion planners in chapter 6.

1.5. Overview

Over the course of this work we introduce important notation and concepts relevant to our underlying vehicle framework, linear Temporal Logic and our template specifications in chapter 2.

Within chapter 3 we present the main body of this work, detailing the process of finding satisfying solutions for a given template. Each of those solutions is then passed to an optimization approach to synthesize concrete scenarios from them. In chapter 5 we evaluate and discuss features of our approach on two common types of scenarios, one in a highway setting and an urban one. Lastly, chapter 6 gives a conclusion to this word and points out future research directions.

2. Terminology

This chapter introduces fundamental knowledge, which we will assume to present in other sections of this work. It covers an introduction to the internal lane representation in Section 2.1, a definition of linear Temporal Logic as well as the subset we use throughout this work, General Reactivity 1 in Section 2.3. In Section 1.3 we cover the used template specification language and give an overview of the contributions made in this work in Section 1.4.

2.1. Lanes & Lanelets

The most basic formulation of the problem covered in this work, is one of synthesizing scenarios for autonomous vehicles from specifications. To create those specifications it becomes clear that a map representation is required, upon which such a specification can be defined. For this work we will use the CommonRoad (CR) [30] framework to define such a map based on lanelets, introduced in [31], as fundamental building block. These lanelets can be connected together by defining adjacent ones, predecessors and successors as shown in Figure 2.1.

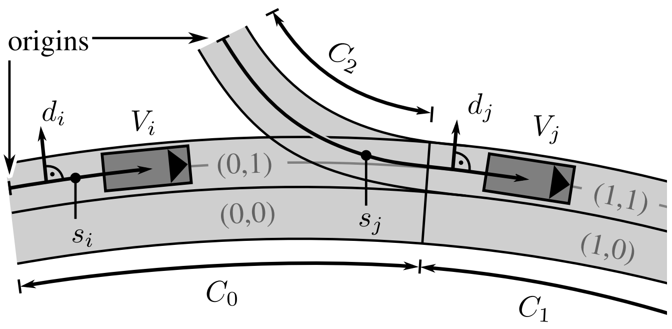


Figure 2.1.: Lane network showing lanes and lanelets. From [18]

To give further structure to the representation in Figure 2.1 we use the concept of lane sections introduced in [18], where a lane section C_{i_c} collects lanes which are coupled laterally by combining adjacent lanes. The *id* of such a section C_{i_c} is given by a tuple $id = (i_c, i_l)$ with $i_c, i_l \in \mathbb{N}$ and i_l enumerates the lanes of a section from right to left. For convenience, we will in the remainder of this work abbreviate the *id* of a lane in a lane section C_{i_c} by l and the set of all lanes in a given lane network by $\mathcal{L}$. The outgoing lanes including succeeding, adjacent left and -right lanes for a lane $l \in \mathcal{L}$ are denoted by $out(l) \subseteq \mathcal{L}$.

2.2. Curvilinear Coordinates

The description of a vehicle's position within this work is in curvilinear coordinates along the current lane $l \in \mathcal{L}$, the vehicle occupies at the moment. In curvilinear coordinates the position of a vehicle V_i is described in terms of (s_i, d_i) with $s_i, d_i \in \mathbb{R}$, where s_i gives the longitudinal offset from the origin of the vehicle's lane l. Specifically, s_i is not the straight line distance from the origin of l, but the length of an arc aligned to the center line of l, from l's origin to the vehicle's position s_i. Along this arc d_i gives the straight line orthogonal deviation from s_i [32]. An example of this is provided in Figure 2.1.

2.3. Linear Temporal Logic

Temporal Logic, in contrast to other logics such as First Order Logic (FOL), allows for specification of properties over discrete time steps. Such specifications are built from basic elements, Atomic Propositions (AP). Each Atomic Proposition is a named Boolean variable $a \in \{\mathbf{true}, \mathbf{false}\}$ and the set of all Atomic Propositions is denoted by AP. To combine Atomic Propositions we define the following syntax of linear Temporal Logic (LTL), which is extended but equivalent to most common definitions found in the literature [33].

Definition 2.3.1 (Linear Temporal Logic, LTL)**.**

$$\begin{aligned} \varphi ::= &\mathbf{true} \mid \mathbf{false} \mid a \mid \neg a \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \\ &\mathbf{X}\varphi \mid \mathbf{F}\varphi \mid \mathbf{G}\varphi \mid \\ &\varphi \mathbf{R} \varphi \mid \varphi \mathbf{W} \varphi \mid \varphi \mathbf{M} \varphi \mid \varphi \mathbf{U} \varphi \text{ with } a \in Ap \end{aligned}$$

Let w be a word over the alphabet 2^{Ap} and φ be a formula then the satisfaction problem $w \vDash \varphi$ is defined as follows:

$w \vDash \mathbf{true}$		$w \vDash \mathbf{X}\varphi$	*iff* $w_1 \vDash \varphi$
$w \nvDash \mathbf{false}$		$w \vDash \mathbf{F}\varphi$	*iff* $\exists k : w_k \vDash \varphi$
$w \vDash a$	*iff* $a \in w_0$	$w \vDash \mathbf{G}\varphi$	*iff* $\forall k : w_k \vDash \varphi$
$w \vDash \neg a$	*iff* $a \notin w_0$	$w \vDash \varphi \mathbf{R} \psi$	*iff* $\forall k : w_k \vDash \psi \vee w \vDash \varphi \mathbf{M} \psi$
$w \vDash \varphi \wedge \psi$	*iff* $w \vDash \varphi \wedge w \vDash \psi$	$w \vDash \varphi \mathbf{W} \psi$	*iff* $\forall k : w_k \vDash \varphi \vee w \vDash \varphi \mathbf{U} \psi$
$w \vDash \varphi \vee \psi$	*iff* $w \vDash \varphi \vee w \vDash \psi$	$w \vDash \varphi \mathbf{M} \psi$	*iff* $\exists k : w_k \vDash \psi \wedge \forall j \leq k : w_j \vDash \varphi$
		$w \vDash \varphi \mathbf{U} \psi$	*iff* $\exists k : w_k \vDash \psi \wedge \forall j < k : w_j \vDash \varphi$

We denote by $\mathcal{M}(\varphi) := \{w \in (2^{AP})^\omega \mid w \vDash \varphi\}$ the model of a formula φ, i.e. the set of all words w that entail φ. Two formulas φ, ψ are then equivalent, denoted by $\varphi \equiv \psi$, if their models are equal:

$$\varphi \equiv \psi := \mathcal{M}(\varphi) = \mathcal{M}(\psi)$$

Additionally, we denote Φ as the set containing all formulas $\varphi \in \Phi$ of finite length.

From the rules defined above we can derive the following shorthand notations, which will be used in the remainder of this work.

$$\varphi \Rightarrow \psi \equiv \neg\varphi \vee \psi \qquad \varphi \Leftrightarrow \psi \equiv \varphi \Rightarrow \psi \wedge \psi \Rightarrow \varphi$$

Satisfiability of a formula φ in LTL is then given if there exists a word w such that $w \vDash \varphi$. If there exists no such w, we call φ unsatisfiable.

2.3.1. Büchi Automata

A Büchi automaton is a finite state machine, which accepts or rejects infinitely long inputs w. Such an automaton consists of a set of states and transitions, defined as follows.

Definition 2.3.2 (Büchi Automaton)**.** The tuple $A := (Q, \Sigma, \delta, q_0, F)$ is called a deterministic Büchi Automaton, with:

Q :	Finite set, called the states of A
Σ :	Finite set, called the alphabet of A
$\delta : Q \times \Sigma \to Q$:	Transition function of A
q_0 :	$q_0 \in Q$, called the initial state of A
$F \subseteq Q$:	Acceptance condition, accepts exactly the state where at least one state $f \in F$ occurs infinitely often in w (Büchi condition).

Where deterministic means, for each state $q \in Q$ and input $\sigma \in \Sigma$ there is exactly one state q' such that $q' = \delta(q, \sigma)$. In other words for each state and input, the next state is exactly determined.

When a word $w \in \Sigma^\omega$, in the language of words of infinite length Σ^ω, leads to the automaton visiting a state in F infinitely often, we call it an accepting word, or A accepts w. To represent such infinitely long words, we split them into two parts: A prefix of w denoted w_p and a cycle w_c in which the first state of w_c is in F and visited infinitely often, $\mathrm{Inf}(w_c^\omega) \cap F \neq \varnothing$. We can then denote the accepting word w as $w = w_p w_c^\omega$, or for practical reasons as $w = (w_p, w_c)$ by omitting the infinite repetition of w_c.

Using the above definition, a formula in LTL φ can be translated to a Büchi Automaton A accepting the same model / language, by various known methods [33, 34, 35]. Even though such a translation to a Büchi automaton is widely used for model checking [36], in the general case, the size of such an automaton A is known to be lower bounded by a doubly exponential function in the number of input states of φ [20].

2.3.2. General Reactivity 1

General checking for satisfiabilty of an LTL formula computationally intractable, but Piterman et. al. have shown that using a subset of LTL, General Reactivity 1 (GR(1)), one can check for satisfiability in polynomial time. More precisely, given a formula φ

in GR(1), they have proposed an algorithm synthesizing a satisfying solution to it in $O(N^3)$ number of steps where N is the number of states of φ [21]. The set of all GR(1) formulas consists of all formulas φ in the following form.

Definition 2.3.3 (General Reactivity 1, GR(1))**.** A formula φ is in GR(1) if it has the following structure:

$$\left(\theta^e \wedge \mathbf{G}\rho^e \wedge \bigwedge_{0<i\leq j} \mathbf{GF}J_i^e\right) \Rightarrow \left(\theta^s \wedge \mathbf{G}\rho^s \wedge \bigwedge_{0<i\leq k} \mathbf{GF}J_i^s\right)$$

With θ^e, θ^s being first order logic propositions, defining the initial state, ρ^e, ρ^s propositions over the current and succeeding states and J_i^e, J_i^s being justice constrains, i.e. constraints on what should hold infinitely often. The premise of the implication is called the environment and the conclusion the system [21]. The goal is then, to for each **true** assignment of the environment, find a satisfying one for the system states.

We use this subset in our work, as it not only significantly faster to compute satisfying solutions for formulas in it, but also Maoz and Ringert have shown that most real-world LTL formulas have a GR(1) representation [37]. Since the computational benefit of the algorithm proposed by Piterman et. al. is derived from its symbolic nature, we use in our work a symbolic implementation of this algorithm implemented by the `omega`[1] tool [22].

Syntactic Sugar

The `omega` tool contains various "syntactic sugar" capabilities, the most important of which to our work being integer valued variables [38]. Each such integer variable v has a fixed integer domain, and allows arithmetic operations to be performed between two integer variables v `op` v' for integer variables v, v' and an operation `op` $\in \{<, \leq, \geq, >, =, \neq, \cdot, \div, \%, +, -\}$. Additionally, operations can also involve integer constants $c \in \mathbb{Z}$, v `op` c and c `op` v.

Each of these expressions above yields a truth value which can be used a larger formula φ. To represent these integer values as binary variables used in GR(1), integer variables and constants are *bitblasted*, conceptually a conversion between a first order logic formula for all values of v to a propositional one. Meaning input variables are converted to their binary representation and each bit is constrained to correspond to the original value v or c.

This allows us to leverage the logarithmically sized encoding of integer variables v when constructing our GR(1) formula in later sections, reducing the number of required variables and therefore system states.

[1]https://github.com/tulip-control/omega

2.4. Scenario Templates

The starting point of the presented approach is a Scenario Template, or simply template. We expect this template as well as a lane network as defined in Section 2.1 to be given and fully specified, meaning that for every vehicle in the template, at least one constraint involving it is given in every scene. Formally such a template is then defined as follows.

Definition 2.4.1 (Scenario Template). A Scenario Template $\mathcal{T}$ is a tuple with $n_{sc} + 1 \in \mathbb{N}$ elements.

$$\mathcal{T} := (\mathcal{A}, \mathcal{S}_1, \mathcal{S}_2, \ldots \mathcal{S}_{n_{sc}}) \quad \mathcal{A} := \{V_1, V_2, \ldots, V_{|\mathcal{A}|}\} \quad \mathcal{S}_i := \left\{r^i_1, r^i_2, \ldots, r^i_{|\mathcal{S}_i|}\right\}$$

$\mathcal{T}$ consists of a set of actors or vehicles $\mathcal{A}$ and $n_{sc} > 0$ scenes. Each of those scenes $\mathcal{S}_i$ is a set of relations which represent constraints having to hold for the entirety of that scene. For two scenes $\mathcal{S}_i, \mathcal{S}_j$, if $i < j$ then $\mathcal{S}_i$ has to hold before $\mathcal{S}_j$, defining a temporal order on scenes in $\mathcal{T}$.

A scene $\mathcal{S}_i$ holds for some LTL formula, denoted by $\mathcal{S}_i \vDash \varphi$, which is a minor abuse of notation meaning the following.

$$\mathcal{S}_i \vDash \varphi \Leftrightarrow \left(r^i_1 \wedge r^i_2 \wedge \cdots \wedge r^i_{|\mathcal{S}_i|}\right) \vDash \varphi$$

A specification of behavior for a set of vehicles $\mathcal{A}$ over time can then be created using multiple scenes as defined above. With such a sequence of scenes $\mathcal{S}_1, \mathcal{S}_\in, \ldots \mathcal{S}_{n_{sc}}$, it is possible to define multiple desired states and their order for all vehicles in $\mathcal{A}$. For practical reasons our approach currently expects at least one constraint for every vehicle $V \in \mathcal{A}$ in every scene. However, this could be removed with some modifications to the process solving a scene sequence given in chapter 3.

The semantics of a single scene $\mathcal{S}_i = \{r^i_1, r^i_2, \ldots r^i_{|\mathcal{S}_i|}\}$ are defined by the conjunction of the relations contained in it as given by definition 2.4.1. These relations can in turn be one of the ones defined in Table 2.1. By using them in an appropriate LTL formula, we afford all relations interactions between each other and are able to model complex behavior.

To give some intuition on the approach presented in this work, we will rely on the following example and show each major step taken on it. For more complicated examples see chapter 5, where strive to provide more realistic use cases as well.

Example 1 (Lane Merge). Based on the simple lane network of two parallel lanes merging together in Figure 2.2, we define a template with two vehicles $\mathcal{A} = \{V_1, V_2\}$. In the first scene they are on the same lane with V_1 being behind V_2. In the second, we assert that both vehicles should be on the same lane with V_1 having overtaken V_2. This behavior can be expressed with the following template.

Relation	Description
$\texttt{OnLane}(V, l, s)$	Vehicle V is on lane l within the curvilinear interval s
$\texttt{IsBehind}(V_i, V_j)$	Vehicle V_i and V_j are on the same lane with V_i behind V_j
$\texttt{OnSameLane}(V_i, V_j)$	Vehicle V_i is on the same lane as V_j
$\texttt{Overtakes}(V_i, V_j)$	Vehicle V_i overtakes V_j at some step between the previous and current scene
Internal Relations	
$\texttt{OnCritical}(V, l, L, s)$	Subclass of OnLane(V,l,s) with the same semantics but for given critical section with a lane $l \in L$ at most one of them can be active at a given time.

Table 2.1.: Available relations with a definition in natural language

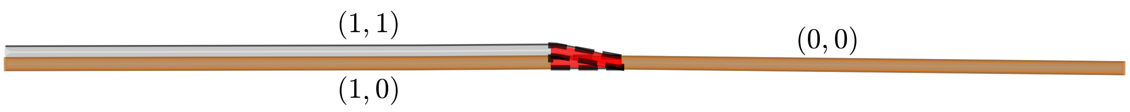


Figure 2.2.: Lane Network for example 1. Brown shows start- and end lanes for both vehicles. The critical section between intersecting lanes is given in red (more in Section 3.2.1).

$$\begin{aligned}
\mathcal{T} &:= (\mathcal{A}, \mathcal{S}_1, \mathcal{S}_2)\\
\mathcal{A} &:= \{V_1, V_2\}\\
\mathcal{S}_1 &:= \{\texttt{OnLane}(V_1, (1, 0)), \texttt{OnLane}(V_2, (1, 0)), \texttt{IsBehind}(V_1, V_2)\}\\
\mathcal{S}_2 &:= \{\texttt{OnLane}(V_1, (0, 0)), \texttt{OnLane}(V_2, (0, 0)), \texttt{Overtakes}(V_1, V_2)\}
\end{aligned}$$

From this template we will create a predicate graph P in Section 3.2, then encode the result in LTL in Section 3.3 and solve that LTL formula, generating many concrete templates $\mathcal{T}^{sol}$ in Section 3.4, which all satisfy the original template $\mathcal{T}$.

Our expectation for this example is, that many scenarios can be generated from this simple template where V_1 overtakes V_2 from on the left and right side respectively, with both vehicles ending on lane $(0, 0)$ with V_2 behind V_1. Even though this scenario is very constrained by design, we will show that even from such a limited template, we are able to synthesize interesting behavior and model the interaction between vehicles V_1 and V_2 correctly. This will demonstrate the rich specification approach we present in this work.

3. Satisfying solutions from Templates

Starting from a given scenario template $\mathcal{T} = (\mathcal{A}, \mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_{n_{sc}})$, we compute all reachable lanes which are on paths satisfying a scene transition $\mathcal{S}_i \rightsquigarrow \mathcal{S}_{i+1} \in \mathcal{T}$. Concatenating them together to get a set of paths from $\mathcal{S}_1 \rightsquigarrow \mathcal{S}_2 \rightsquigarrow \cdots \rightsquigarrow \mathcal{S}_{n_{sc}}$. We use these paths to create a graph representation, called a predicate graph P, based on $\mathcal{T}$, which characterizes all valid states in the lane network satisfying the template $\mathcal{T}$ and their transitions within it in Section 3.2. Then we encode P as a LTL formula and piecewise create satisfying transitions between all scenes $\mathcal{S}_i \rightsquigarrow \mathcal{S}_{i+1}$. Such an independent translation is possible, as each scene $\mathcal{S}_i$ is assumed specify the entire state all vehicles $V \in \mathcal{A}$ needed to satisfy in that scene. For simplicity' sake' will not explicitly state this independent creation of formulas for each scene transition $\mathcal{S}_i \rightsquigarrow \mathcal{S}_{i+1}$ in Sections 3.3 to 3.4, but only assemble them in Section 3.5 to provide the set of all solved templates which satisfy $\mathcal{T}$. Using these templates we individually employ an optimization approach by Klischat [18], formulating the problem of synthesizing a concrete CommonRoad scenario form a solved template as a Mixed-Integer Quadratic Programming (MIQP) problem in chapter 4. This optimization problem is then solved with the solver Gurobi[1], yielding a fully specified CR scenario for every solved template satisfying $\mathcal{T}$.

By employing this process roughly sketched above and depicted in Figure 3.1 we are able to create many realistic scenarios with full control over the actions performed by each vehicle in them.

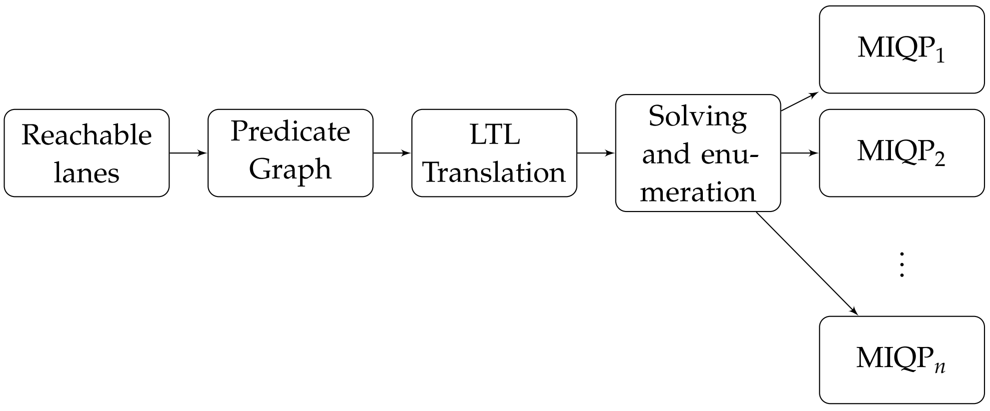


Figure 3.1.: High level overview the pipeline synthesizing all satisfying CommonRoad scenarios from a template

[1]`https://www.gurobi.com/`

3.1. Reachable Lanes

Given a scenario template $\mathcal{T}$ and its corresponding lane network, the problem of generating a concrete scenario from $\mathcal{T}$, can be formulated as an enumeration problem of all states in the given lane network satisfying the constraints in the template $\mathcal{T}$. However, using this approach is computationally very inefficient, as a given lane network can be arbitrarily large, while the template $\mathcal{T}$ might constrain the satisfying states in that lane network to only a small subset of all possible ones, leading to a lot of wasted computation on infeasible states.

We define a state as an $\texttt{OnLane}(V, l, s)$ relation for a vehicle $V \in \mathcal{A}$, a lane $l \in \mathcal{L}$ and longitudinal range s on l. By defining the notion of reachability of a state we are then able to only consider ones which are reachable and thereby largely prune the considered state space. Such an $\texttt{OnLane}(V, l, s)$ is considered reachable, if there exists a scene transition $\mathcal{S}_i \rightsquigarrow \mathcal{S}_{i+1}$ with $\texttt{OnLane}(V, l_i, s_i) \in \mathcal{S}_i$ and $\texttt{OnLane}(V, l_{i+1}, s_{i+1}) \in \mathcal{S}_{i+1}$ such that:

$$\texttt{OnLane}(V, l_, s_i) \rightarrow \cdots \rightarrow \texttt{OnLane}(V, l, s) \rightarrow \cdots \rightarrow \texttt{OnLane}(V, l_{i+1}, s_{i+1}) \tag{3.1}$$

Expressing that, $\texttt{OnLane}(V, l, s)$ is on a path from $\texttt{OnLane}(V, l_i, s_i)$ to $\texttt{OnLane}(V, l_{i+1}, s_{i+1})$ of the same vehicle V.

Using this definition of reachability, we then only need to consider $\texttt{OnLane}(V, l, s)$ relations in all further steps which are reachable, denoted by $R(\mathcal{L})$, as they are the only ones able to satisfy $\mathcal{T}$, giving completeness of $R(\mathcal{L})$. Computability of $R(\mathcal{L})$ is also given, as it can be simply computed by formulating the path constraint in Equation 3.1 as a Bidirectional Search problem from $\texttt{OnLane}(V, l_i, s_i)$ to $\texttt{OnLane}(V, l_{i+1}, s_{i+1})$ and adding all $\texttt{OnLane}(V, l, s)$ relations between them to $R(\mathcal{L})$.

3.2. Predicate Graph

Each vehicle defined in the template $\mathcal{T}$ has to follow a path of lanes to fulfill transitions between its various constraints. To express those transitions between states, and optional requirements (which have to be fulfilled in order to move to that state) we introduce the notion of a predicate Graph.

Definition 3.2.1 (Predicate Graph)**.** A predicate Graph is a directed graph, defined as follows:

$$P := (\mathcal{V}, E, \eta) \quad \mathcal{V} \subseteq \Gamma \quad E \subseteq \mathcal{V} \times \mathcal{V} \quad \eta : E \rightarrow \Phi$$

With $\mathcal{V} \subseteq \Gamma$ being a subset of all relations, $E \subseteq \mathcal{V} \times \mathcal{V}$ the set of directed Edges between relations in $\mathcal{V}$, where and edge represents a transition between two relations for a vehicle. Each edge can also have a label $\eta : E \rightarrow \Phi$ assigning each edge a condition expressed as a boolean function $\varphi \in \Phi$. This condition $\eta(e)$ then has to be satisfied for an transition $e \in E$ to be taken.

This predicate Graph is created from the set of all reachable lanes $R(\mathcal{L})$, with the idea being to for each Lane $l \in R(\mathcal{L})$ create an edge between it and all its outgoing Lanes $out(l) \subseteq R(\mathcal{L})$. Additionally, self loops are inserted for every node, to optionally allow a vehicle to stay in its current state. Following this basic procedure, the resulting predicate graph contains all reachable paths through the given map for each vehicle. Reachability, as in Section 3.1, refers to all paths which start at the $\texttt{OnLane}(V, l_i, s_i) \in \mathcal{S}_i$ constraint for a vehicle in scene $\mathcal{S}_i$ and end in the next scene's constraint for that vehicle $\texttt{OnLane}(V, l_{i+1}, s_{i+1}) \in \mathcal{S}_{i+1}$. Therefore, applying this procedure for all scenes $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_{n_{sc}}$, yields a predicate graph containing all reachable paths for each vehicle.

3.2.1. Critical Sections

So far each Node in the predicate graph as defined in definition 3.2.1 represents a lane $l \in L$ a vehicle V_i can occupy, but no interaction between vehicles is modelled. To address this we introduced $\texttt{OnCritical}(V_i, l, L, s)$ relations in Section 2.4. They are similar to $\texttt{OnLane}(V_i, l, s)$ relations in that they constrain a vehicle V_i to be on lane l within coordinates s but also define this lane usage to be exclusive. So for each lane $l' \in L$ of a given critical section there have to exist $\texttt{OnCritical}(V_j, l', L, s')$ for each vehicle V_j and at most one vehicle can be on any of the lanes $l' \in L$ for a given critical section.

The exact insertion of a critical section $\texttt{OnCritical}(V, l, L, s)$ is performed when a relation's lane $r.l$ with $r \in \mathcal{V}$ intersects another relation's lane $r'.l$ with $r' \in \mathcal{V}$. Such an intersection is then iteratively expanded to form a maximal cluster C such that all relations lanes in the cluster have a non-empty intersection.

$$r_1, r_2, \ldots, r_n \in C : \qquad I := r_1.l \cap r_2.l \cap \cdots \cap r_n.l \neq \varnothing$$

The relations $r \in C$ in this maximal cluster are then split into three sections: Before, within and after the cluster's intersection I by projecting I onto the relations lane $\Pi_{r.l} I$, where $\Pi_{r.l}$ is a matrix projecting each point in I onto $r.l$. We can then transform $\Pi_{r.l} I$ from euclidean coordinates to curvilinear ones and define the following new relations:

Definition 3.2.2 (Critical Sections)**.**

$$\begin{aligned}
r^{pre} &:= \texttt{OnLane}\left(r.V, r.l, \left[\min_{p \in r.l} \Pi_{r.l}(p.s), \min_{p \in I} \Pi_{r.l}(p.s)\right)\right) \\
r^{mid} &:= \texttt{OnCritical}\left(r.V, r.l, C, \left[\min_{p \in I} \Pi_{r.l}(p.s), \max_{p \in I} \Pi_{r.l}(p.s)\right)\right) \\
r^{post} &:= \texttt{OnLane}\left(r.V, r.l, \left[\max_{p \in I} \Pi_{r.l}(p.s), \max_{p \in r.l} \Pi_{r.l}(p.s)\right]\right)
\end{aligned}$$

So r^{pre} defines the area before the critical section to be a regular on-lane relation, by including the longitudinal range (s) up to the intersection of lanes I. Within that intersection r^{mid} gives the on-critical relation describing that at most one vehicle can be on this intersection at a time. r^{post} describes the regular on-lane relation after the intersection I similar to r^{pre}. We then update the predicate graph's vertices as well as its edges E accordingly.

$$\mathcal{V} := \mathcal{V} \setminus \{r\} \cup \{r^{pre}, r^{mid}, r^{post}\}$$

Additionally, we need to encode the exclusivity of all on-critical relations r^{mid}, which is not done in the predicate graph but directly in LTL. A detailed description of this encoding is given in paragraph 3.3.3.

3.2.2. Edge Requirements

Interaction between vehicles was encoded into the predicate graph in the previous section. We now concern ourselves with adding requirements or constraints which have to be fulfilled for a vehicle in order to transition from an $\texttt{OnLane}(V, l, s)$ relation $r \in \mathcal{V}$ to an outgoing one $r' \in out(r)$.

Each edge of the predicate graph represents a lane-to-lane connection in the given map. Adding the following assumption, that vehicles have to have a total order in a lane with respect to their longitudinal coordinate from its start point $(r.s)$, we can assert that for a vehicle to move from a lane to any of its successors it has to be the first such vehicle w.r.t. to the ordering. So we obtain the requirement that a vehicle V_i can only move to a succeeding relation if no vehicle is ahead of it. For every edge $(r, r') = e \in E$ for vehicle $V_i = r.v = r'.v$ where $r.l$ succeeds $r'.l$ we update the edge's requirement $\eta(e)$ as follows:

Definition 3.2.3 (Edge Requirements)**.**

$$\eta(e) := \eta(e) \land \neg \left(\bigvee_{\substack{V_j \in \mathcal{A} \\ V_j \neq V_i}} \texttt{IsBehind}(V_i, V_j) \right)$$

So a vehicle V_i can only move from a relation r to r' if there is an edge whose requirement $\eta(e)$ is fulfilled and there exists no vehicle $V_j \in \mathcal{A}$ such that V_i is behind V_j.

3.2.3. Example

By applying the results of the previous sections, specially Definitions 3.2.2 to 3.2.3, to example 1 we create the predicate graph in Figure 3.2. In it one can clearly see the two connected components for vehicles $0, 1$ and their start $\texttt{OnLane}(V_i, l, s)$ relations in $\mathcal{S}_1$ colored in blue with the respective ends in $\mathcal{S}_2$ in orange. Between those, the inserted critical sections are colored in red, corresponding to the red overlapping area in Figure 2.2. Movement between the states is restricted by only allowing to transition $(r, r') \in E$ to a succeeding state r' if the respective condition $\eta((r, r'))$, annotated on

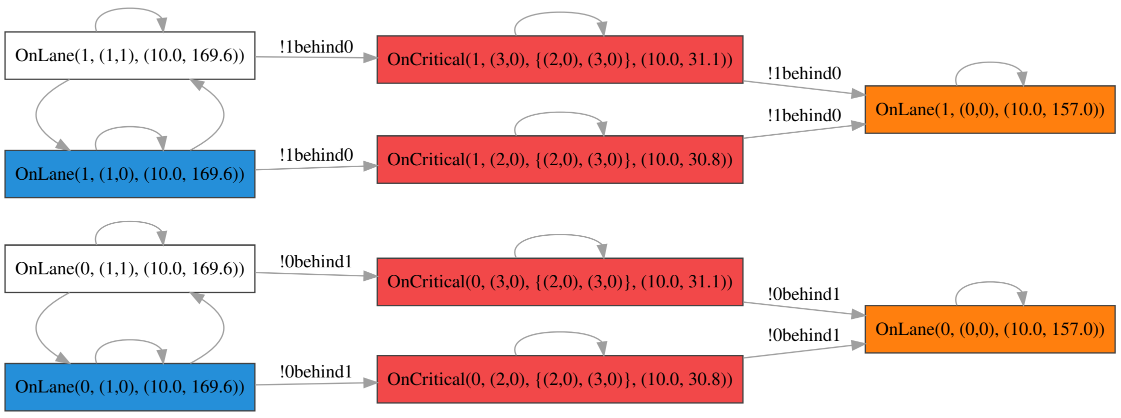

OnLane(1, (1,1), (10.0, 169.6))
!1behind0
OnCritical(1, (3,0), {(2,0), (3,0)}, (10.0, 31.1))
!1behind0
OnLane(1, (0,0), (10.0, 157.0))
!1behind0
OnLane(1, (1,0), (10.0, 169.6))
!1behind0
OnCritical(1, (2,0), {(2,0), (3,0)}, (10.0, 30.8))
OnLane(0, (1,1), (10.0, 169.6))
!0behind1
OnCritical(0, (3,0), {(2,0), (3,0)}, (10.0, 31.1))
!0behind1
OnLane(0, (0,0), (10.0, 157.0))
!0behind1
OnLane(0, (1,0), (10.0, 169.6))
!0behind1
OnCritical(0, (2,0), {(2,0), (3,0)}, (10.0, 30.8))

Figure 3.2.: Predicate graph for example 1.

directly on the edges is fulfilled. For edges $(r, r') \in E$, which do not have an annotation, no condition needs to be met in order to move to their next states $\eta((r, r')) = \textbf{true}$.

3.3. LTL Translation

In this section, we present our approach for converting the Predicate Graph P defined in Section 3.2 to an equivalent representation in LTL. We start by covering P to LTL, by enumerating its nodes and edges and generating an equivalent expression for each (Section 3.3.1). As these expressions are built on the relations defined in Section 2.4 we additionally present an encoding for each of them in Section 3.3.3. This is non-trivial, as our target language is not LTL generally, but GR(1), a very expressive subset of LTL, but a subset nevertheless. Therefore, a general process converting for a LTL formula to one in GR(1) is presented and applied to our specific use cases.

3.3.1. Predicate Graph Encoding

In Section 3.2 we defined the predicate graph for a template. It encodes all feasible states which satisfy the template's constraints and models the requirements to reach those states. In this section we will elaborate on how we encode this predicate graph to LTL expressions. The main idea being, that for each outgoing edge of a node we determine all edges starting from that node whose requirements are entailed by the considered edge. Then we add a formula constraining movement from a state to only the outgoing states which are on any of the before mentioned entailed edges.

Concretely, for nodes $r, r' \in \mathcal{V}$ and an outgoing edge $(r, r') \in E$ let the set $\mathcal{R} := \{(r, r'') \mid (r, r'') \in E \land \eta((r, r'')) \vDash \eta((r, r'))\}$, with r'' being an arbitrary outgoing state, denote the set of all edges whose requirements are entailing $\eta((r, r'))$ Using this we can define the following function to encode a node and its outgoing edges in P:

$$\mathbf{G}\left(r \Rightarrow \bigvee_{(r,r')\in E} \eta((r,r')) \wedge \left(\bigvee_{(r,r'')\in\mathcal{R}} \mathbf{X}r''\right)\right) \tag{3.2}$$

So at all points in time, if an $r = \texttt{OnLane}(V,l,s)$ relation holds, we consider each outgoing edge $(r,r') \in E$ and determine its requirements $\eta((r,r'))$. Then if $\eta((r,r'))$ holds, V can only move to any of the outgoing states r'' if the requirements of that transition $\eta((r,r''))$ are also fulfilled by $\eta((r,r'))$. In particular, since $\eta((r,r'))$ entails itself, a transition from r to r' is possible.

3.3.2. Node identifiers

Instead of encoding each relation $r \in \mathcal{V}$ directly as an LTL Literal defined in definition 2.3.1, we propose an identifier based encoding. The main observation here is that by assigning each lane in the lane network a unique identifier $\alpha(r) \in \mathbb{N}_0$, we can express the position of each vehicle by this identifier and allow comparisons between identifiers. This allows us to for two relations $r, r' \in \mathcal{V}$, express rules such as $\alpha(r) = \alpha(r')$ without specifying the exact relations to compare and additionally provides for a more efficient encoding than regular boolean variables due to a binary encoding for integers in `omega`. However, instead of just assigning each node in the predicate graph an ID based on its lane, we propose a more efficient encoding.

By observing that a predicate graph P as defined in Section 3.2, has exactly one connected component $C \subseteq \mathcal{V}$ per vehicle $V_i \in \mathcal{A}$, and V_i can only move to relations (lanes) within its connected component C, we assign IDs in a consecutive manner $\{0, 1, 2, \dots\}$ for each connected component. However, when doing so we make sure that if two nodes $\texttt{OnLane}(V_i,l,s)$, $\texttt{OnLane}(V_j,l,s)$ reference the same lane, their IDs are equal and vice versa. Precisely we require the following to hold.

$$\forall r, r' \in \mathcal{V} : r.l = r'.l \wedge r.s = r'.s \Leftrightarrow \alpha(\texttt{OnLane}(V_i,l,s)) = \alpha(\texttt{OnLane}(V_j,l,s)) \tag{3.3}$$

This is required, as we check the equality of identifiers of nodes in paragraph 3.3.3 to determine if two vehicles are on the same lane. Using this encoding the relation for each vehicle V_i is uniquely determined by its α value. By additionally to the constraint in Equation 3.3 assigning α values in each connected component C, such that for each vehicle the range of the set $\alpha(C) = \{\alpha(c) \mid c \in C\}$ is minimal, we improve the efficiency of the encoding drastically. This due to each vehicle V_i with its connected component C in paragraph 3.3.3 being assigned a GR(1) variable with the domain $\alpha(C)$. The range of $\alpha(C)$ therefore directly determines the number of auxiliary variables required to represent this variable (in a binary encoding). The exact Python procedure for obtaining such an α assignment is given in Listing 3.1.

For more details on usage of the identifier based encoding see Section 3.3.3.

Listing 3.1. Python code for node ID assignment satisfying Equation 3.3

```python
def alphas(nodes):
    """
    Assigns every node in a connected component a unique id.
    """
    components = defaultdict(set)
    for node in nodes:
        components[node.relation.vehicle_id].add(node)

    same_lane_groups: Dict[OnLane, Set[Node]] = defaultdict(set)
    for node in nodes:
        r = copy(node.relation)
        r.vehicle_id = -1
        same_lane_groups[r].add(node)

    same_lane: Dict[Node, Set[Node]] = dict()
    for group in same_lane_groups.values():
        for node in group:
                same_lane[node] = group - {node}

    shared_nodes = lambda component: len(
        [node for node in component
              if node in same_lane[node] and same_lane[node]]
    )

    alpha = dict()
    idx = -1
    for c in sorted(components.values(), key=shared_nodes):
        for node in c:
                if node not in alpha:
                    idx += 1
                    alpha[node] = idx
                    for other in same_lane[node]:
                        alpha[other] = idx
    return alpha
```

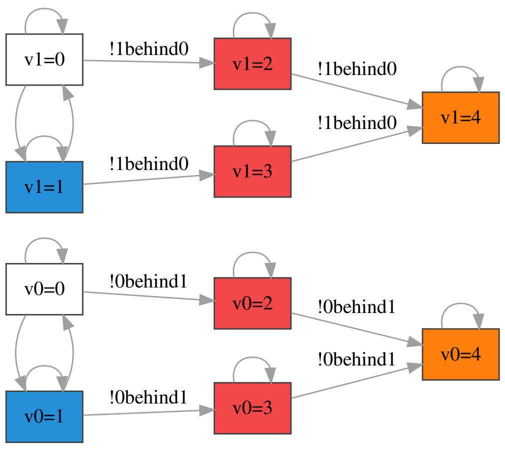


Figure 3.3.: Node ID encoding for all Vehicles V with `v0` and `v1` representing ν in Equation 3.4

3.3.3. Relation Encoding

In Section 2.4 we defined the different types of relations we use as constraints for the concrete templates synthesized in this work. They are: $\texttt{OnLane}(V, l, s)$, $\texttt{OnCritical}(V, l, L, s)$, $\texttt{IsBehind}(V_i, V_j)$, $\texttt{OnSameLane}(V_i, V_j)$ and $\texttt{Overtakes}(V, l, L, s)$. Firstly, we will, in this section, focus on translating basic Relations expressing the various states a vehicle can be in and then progress to high level ones such as $\texttt{Overtakes}(V, l, L, s)$ which are built upon them. In general, we express the minimal amount of information necessary to define a constraint, due to our goal of generating as many feasible scenarios as possible. Especially when using our approach for synthesizing concrete scenarios covered in Chapters 3 to 5, this becomes relevant.

OnLane The most basic relation we concern ourselves with is the $\texttt{OnLane}(V, l, s)$ one. It requires a vehicle V to be on a lane l and within lane l its longitudinal offset has to be in the given interval, $V.s \in s$. Using the definition of a unique identifier α for every relation in a connected component of the predicate graph P, covered in Section 3.3.2, we can encode this $\texttt{OnLane}(V, l, s)$ relation of a vehicle in LTL as follows.

Since all $\texttt{OnLane}(V, l, s)$ relations V are assigned a unique identifier α, we introduce an integer valued LTL variable ν to track the state of each vehicle V, where:

$$\nu = \alpha(\texttt{OnLane}(V, l, s)) \tag{3.4}$$

So every vehicle V is represented by a variable ν, whose integer value tracks the $\texttt{OnLane}(V, l, s)$ relation V is in over time. The main advantage of such an encoding is that for each vehicle only an integer having approximately the number of states, as V can reach in P needs to be represented, which uses logarithmically many variables in the domain size of v using a binary encoding[2]. Additionally, we do not need to specify an exclusivity behavior for different values of v, which immediately implies that the `OnLane`(V, l, s) relations V can be in are mutually exclusive. For our running example example 1 such an encoding is given in Figure 3.3.

OnCritical Critical sections are areas in the lane network, which at most a single vehicle can occupy at a given time. As defined in Section 3.2.1, all relations forming a critical section `OnCritical`(V, l, L, s) from the intersecting lanes in L can have at most one vehicle at a time on any of them. Encoding this positional constraint on a vehicle V is handled in the same way as described in the previous section. To be consistent with this procedure, we define an `OnCritical`(V, l, L, s) relation to be a subclass of the corresponding `OnLane`(V, l, s) relation. In doing so, both relation types can be regarded the same when convenient, as in the previous section, but an additional exclusivity can be defined across all `OnCritical`(V, l, L, s) relations having the same set of intersecting lanes L. To achieve this explicit exclusivity, we define the following operators.

Definition 3.3.1 (LTL Exclusivity)**.** For a set of relations $\mathcal{R}$ the notion of *exactly one* $r \in \mathcal{R}$ having to be true at any given time step is encoded as follows:

$$Ex(\mathcal{R}) := \bigvee_{r \in \mathcal{R}} r \bigwedge_{o \in R \wedge o \neq r} \neg o$$

When considering the encoding for $Ex(\mathcal{R})$ one can easily see that the size of the resulting formula scales quadratically with the size of $\mathcal{R}$. Even though such an encoding is non-optimal, we use it here for its simplicity and make sure to keep $\mathcal{R}$ as small as possible for practical use.

To allow for *at most one* relation $r \in \mathcal{R}$ to be satisfied we define the $Max_1(\mathcal{R})$ operator:

$$Max_1(\mathcal{R}) := Ex(\mathcal{R}) \vee \bigvee_{r \in \mathcal{R}} \neg r$$

For the special case of $\mathcal{R} = \{r_1, r_2\}$ we can simplify to $Max_1(\mathcal{R}) = \neg r_1 \wedge \neg r_2$.

Then let C be a critical section where for all `OnCritical`$(V, l, L, s) \in C$ the set of intersecting lanes L is the same. For each such critical sections $C_1, C_2, \ldots, C_n$ we encode exclusivity within each section in the following way.

$$\mathbf{G}\left(Max_1(C_1) \wedge Max_1(C_2) \wedge \cdots \wedge Max_1(C_n)\right) \tag{3.5}$$

With this definition, we therefore assert that each critical section C can only have at most one satisfied relation within it at any given time. This guarantees that our encoded predicate graph is, if a solution exists, traversed collision free.

[2]Since v is binary encoded, the number of representable states is always of the form $2^n - 1$ for some minimal $n \in \mathbb{N}$.

OnSameLane For specifying that two vehicles have to be on the same lane in the given lane network, but omitting which lanes exactly, we introduced the $\texttt{OnSameLane}(V_i, V_j)$ relation in Section 2.4. Based on the encoding of an $\texttt{OnLane}(V, l, s)$ relation and its identifier $\alpha(\texttt{OnLane}(V, l, s))$ described in detail in paragraph 3.3.3 it's trivial to encode an $\texttt{OnSameLane}(V_i, V_j)$ relation. As with Equation 3.3 we constrain the identifiers of two relations with the same vehicles to be equal, which is carried forward to the LTL encoding by tracking a vehicle's identifier using the LTL variable ν in Equation 3.4. We encode an $\texttt{OnSameLane}(V_i, V_j)$ relation for two vehicles $V_i, V_j \in \mathcal{A}$ by the following.

$$\texttt{OnSameLane}(V_i, V_j) \Leftrightarrow \nu_i = \nu_j \tag{3.6}$$

In natural language this means that two vehicles V_i, V_j are on the same lane if and only if their corresponding LTL variables ν_i, ν_j are equal.

IsBehind In Section 2.4 we defined an $\texttt{IsBehind}(V_i, V_j)$ relation to represent for the vehicles $V_i, V_j \in \mathcal{A}$, *being on the same lane* with V_i behind V_j. However, the exact position of V_i, V_j on their common lane is not specified. In our LTL representation we do not model continuous values (i.e. s-offset of V_i, V_j) and will only do so after solving for a concrete scenario in chapter 4. Therefore, our approach currently does not constrain the maximum number of vehicles on a lane, and resolves this only during the optimization phase.

To encode the $\texttt{IsBehind}(V_i, V_j)$ relation we firstly, need to encode movement from one relation to all its outgoings in P. This is achieved by comparing the identifiers for vehicles V_i, V_j. If at some time step V_i, V_j are on different lanes $\nu_i \neq \nu_j$, but they move to the same lane on the next step $\mathbf{X}(\nu_i = \nu_j)$, then either V_i has to be behind V_j or the other way around. For the case that V_j does not move to a new lane in the next time step, this clearly implies that V_j stays on its lane and V_i therefore has to be behind it. Since the behavior of V_j in this example is symmetric, we arrive at the following formula.

$$\forall v_i \neq v_j \in \mathcal{A} : \mathbf{G}\left((\nu_i \neq \nu_j \wedge \nu_j = \mathbf{X}(\nu_j) \wedge \mathbf{X}(\nu_i = \nu_j)) \Rightarrow \mathbf{X}\left(\texttt{IsBehind}(V_i, V_j)\right)\right) \tag{3.7}$$

Here, we assert that if V_i and V_j are not on the same lane at some point in time, but move to a common one in the next step, and V_j does not switch lanes, then V_i has to be behind V_j in the next step. However, this constraint omits one type of behavior, namely if V_i and V_j are on adjacent lanes and V_i performs a lane change to V_j's lane then it forces V_i to arrive on its target lane behind V_j, omitting that V_i might also insert before V_j. This behavior was unable to be addressed in the time constraints of this work and is left as a future improvement.

To ensure the consistency of an $\texttt{IsBehind}(V_i, V_j)$ relation over time, we assert that if V_i and V_j are on the same lane at the current time step $\nu_i = \nu_j$ as well as the next and $\texttt{IsBehind}(V_i, V_j)$ is set, then it also has to be set for the next time step.

$$\begin{aligned} v_i \neq v_j \in \mathcal{A} :& \mathbf{G}\left(\nu_i = \nu_j \wedge \texttt{IsBehind}(V_i, V_j) \wedge \mathbf{X}(\nu_i = \nu_j) \Rightarrow \mathbf{X}(\texttt{IsBehind}(V_i, V_j))\right) \\ & \wedge \mathbf{G}\left(\nu_i = \nu_j \wedge \texttt{IsBehind}(V_j, V_i) \wedge \mathbf{X}(\nu_i = \nu_j) \Rightarrow \mathbf{X}(\texttt{IsBehind}(V_j, V_i))\right) \end{aligned}$$

The above constraint then describes this behavior for both cases of $\texttt{IsBehind}(V_i, V_j)$ and $\texttt{IsBehind}(V_j, V_i)$. Also, if V_i, V_j are on the same lane, some behind relation has to be set. This constraint is required as vehicles might start on the same lane but all combinations of behind relations are supposed to be solved for so no order for V_i, V_j is set in the template, which is simply described by:

$$\forall V_i \neq V_j \in \mathcal{A} : \mathbf{G}\left(\nu_i = \nu_j \Rightarrow \texttt{IsBehind}(V_i, V_j) \vee \texttt{IsBehind}(V_j, V_i)\right) \tag{3.8}$$

Lastly, for vehicles V_i, V_j only V_i can be behind V_j or the other way around, but not both at the same time.

$$\forall V_i \neq V_j \in \mathcal{A} : \mathbf{G}\left(Max_1\left(\left\{\texttt{IsBehind}(V_i, V_j), \texttt{IsBehind}(V_j, V_i)\right\}\right)\right) \tag{3.9}$$

Overtaking In this section we will cover the encoding of an overtaking event between two vehicles V_i and V_j. For the purpose of generating as many scenarios from a given template, we intentionally leave this formulation quite simplistic, as this will allow the solver presented in Section 3.4 the most degrees of freedom.

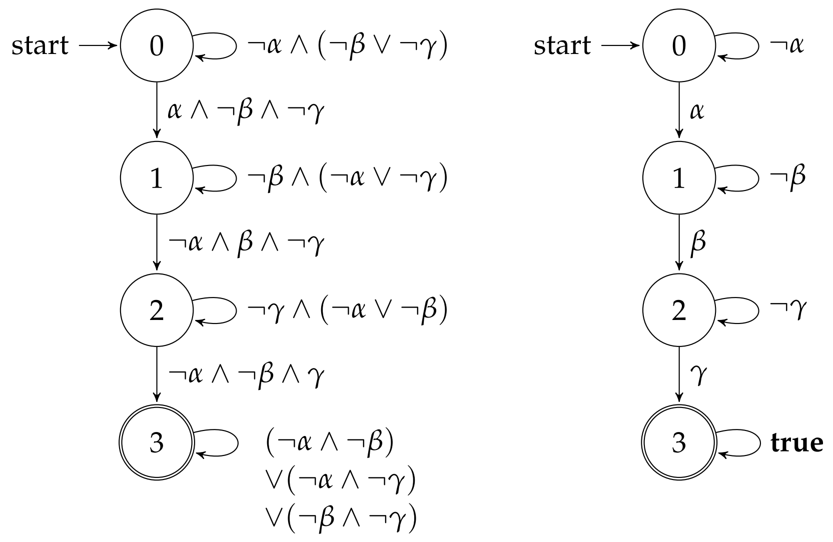
In the following we will consider an overtaking maneuver to be one where two vehicles are at some point in time on the same lane with one driving behind the other. Then the further back vehicle of the two has to switch to a different lane from the leading one, and in the end the two vehicles have to end up on a common lane again, but with the previously following vehicle leading and the leading one now following. This scenario, although it might seem to be overly simplistic, it is an accurate enough expression of the state sequence followed in an overtaking maneuver and leads to interesting behavior as can be seen in chapter 5. If desired, it could however be expanded to include more constraints on the movements of involved vehicles by extending the state description and following the LTL encoding procedure given below. Some of those possible improvements are outlined in [28].

Based on the overtaking scenario described in the previous section, we define the following sequence of states α, β, γ, which are expressed using the predicates we defined in Section 2.4. They describe the encoding of an $\texttt{Overtakes}(V_i, V_j)$ relation in the following.

$$\alpha := \texttt{OnSameLane}(V_i, V_j) \wedge \texttt{IsBehind}(V_i, V_j) \tag{3.10}$$

$$\beta := \neg\texttt{OnSameLane}(V_i, V_j) \tag{3.11}$$

$$\gamma := \texttt{OnSameLane}(V_i, V_j) \wedge \texttt{IsBehind}(V_j, V_i) \tag{3.12}$$



(b) Simplified edge based Büchi automaton

(a) Transition based Büchi automaton

Figure 3.4.: Deterministic Büchi automata for Equation 3.13

Noting that by using the predicates `OnSameLane`(V_i, V_j) and `IsBehind`(V_i, V_j) directly in an LTL formula, we implicitly reference the encoding for these predicates given in paragraph 3.3.3. Finally, we then need to convert the sequence α, β, γ to LTL, where the states have to follow one another. However, in the natural language description of an overtaking scenario we implicitly assumed that an arbitrary amount of time can pass between each state α, β, γ. Such an encoding in LTL is expressed as follows, with the states α, β, γ being mutually exclusive.

$$\mathbf{F}(\alpha \wedge \mathbf{F}(\beta \wedge \mathbf{F}(\gamma))) \quad \wedge \quad \mathbf{G}\left(Ex(\{\alpha, \beta, \gamma\})\right) \tag{3.13}$$

However, as we express all our formulas in a subset of LTL, namely GR(1) to check them for satisfiability in polynomial time (more on this process in Section 3.4), we need to convert this expression to GR(1), which does not include nested **F** operators [21].

Overtaking Automaton Fortunately, Maoz, Ringert and Jacobs have presented a general approach to the issue of converting a LTL formula to GR(1) if such a conversion is possible [37, 39]. Maoz and Ringert propose to convert the desired LTL formula to a deterministic Büchi automaton and give an automatic conversion to GR(1), while Jacobs performs the same deterministic Büchi automaton translation and converts the result manually. In the following we will apply this conversion technique manually to Equation 3.13 resulting in an equivalent GR(1) specification.

The first step in such a conversion is the translation of the formula in LTL to a deterministic Büchi automaton, where deterministic means that the automaton can only be in one state at a time and that state is always uniquely determined by its inputs. Such a conversion can be automatically performed by tools such as `spot` [34], which build a more general form of Büchi automaton (Transition Generalized Büchi Automaton) from a LTL formula like the one in Equation 3.13 and convert it to a deterministic Büchi automaton. The result of this translation is given in Figure 3.4a (minimized using a SAT approach). For representing these automata we give two different perspectives, one transition based and an edge based one. In the transition based automaton from Figure 3.4a transitions are annotated with the exact boolean formulas that are required for the automaton to take that transition. So in our case the projection of each boolean literal α, β, γ all formulas at each transition. In the edge based one, given in Figure 3.4b, we omit this exact specification of each literal and only give the dominant literals describing each edge. Such a simplification is equivalent, as the states making up α, β, γ are already mutually exclusive by their encoding in Section 3.3.3.

Based on the edge based deterministic Büchi automaton in Figure 3.4b we can now encode the chain of states α, β, γ with their definition in Equations (3.10) to (3.12) in GR(1) by introducing a new variable $o_i \in \{0, 1, \dots, 3\}$ to describe the current state the automaton is in. With this o_i each edge in the automaton can then be expressed in GR(1) using the following formula.

$$\mathbf{G}(o_i = q \wedge \sigma \Rightarrow \mathbf{X}(o_i = \delta(q, \sigma))) \tag{3.14}$$

Where $q \in Q$ is a state and $\sigma \in \Sigma$ is a word in the alphabet of the automaton. $\delta(q, \sigma)$ is the transition function giving a succeeding state for a state q and input word σ (details in Section 2.3.1). Equation 3.14 intuitively then describes that if the automaton is in a given state q and an input word σ is given then it has to be in state $\delta(q, \sigma)$ in the next state, which exactly encodes the transition function δ. Therefore, the semantics of the resulting formula are the same as the ones expressed in the original deterministic Büchi automaton.

For our case of converting Equation 3.13 to GR(1), we get the following result when applying Equation 3.14 to the automaton in Figure 3.4b.

Definition 3.3.2 (Overtaking Automaton encoding)**.**

$$o_i = 0 \tag{3.15}$$

$$\begin{aligned}
&\mathbf{G}((o_i = 0 \land \texttt{OnSameLane}(V_i, V_j) \land \texttt{IsBehind}(V_i, V_j)) && \Rightarrow \mathbf{X}(o_i = 1)) \\
\land&\mathbf{G}((o_i = 0 \land \neg\texttt{OnSameLane}(V_i, V_j) \lor \texttt{IsBehind}(V_i, V_j)) && \Rightarrow \mathbf{X}(o_i = 1)) \\
\land&\mathbf{G}((o_i = 1 \land \neg\texttt{OnSameLane}(V_i, V_j)) && \Rightarrow \mathbf{X}(o_i = 2)) \\
\land&\mathbf{G}((o_i = 1 \land \texttt{OnSameLane}(V_i, V_j)) && \Rightarrow \mathbf{X}(o_i = 2)) \\
\land&\mathbf{G}((o_i = 2 \land \texttt{OnSameLane}(V_i, V_j) \land \texttt{IsBehind}(V_j, V_i)) && \Rightarrow \mathbf{X}(o_i = 3)) \\
\land&\mathbf{G}((o_i = 2 \land \neg\texttt{OnSameLane}(V_i, V_j) \lor \neg\texttt{IsBehind}(V_j, V_i))) && \Rightarrow \mathbf{X}(o_i = 3)) \\
\land&\mathbf{G}((o_i = 3) && \Rightarrow \mathbf{X}(o_i = 3))
\end{aligned} \tag{3.16}$$

$$\mathbf{GF}(o_i = 3) \tag{3.17}$$

The above formulas fully encode the automaton in GR(1) by Equation 3.15 adding to the initial state θ^s. Transitions are expressed as described in the previous section and explicitly given in Equation 3.16 and added to the formula's system transitions ρ^s. Lastly, Equation 3.17 specifies the justice constraint which should hold infinitely often in the future and gets added to J_1^s.

3.4. GR(1) solving and enumeration

On of the main contributions of this work is, to generate multiple scenarios from a given template. Since any such template is expressed as an LTL formula φ using the methods described in the previous section, we will in this section cover, how to generate multiple distinct satisfying solutions for any such given formula φ.

Piterman et al. have shown that a reduced set of LTL, GR(1), can be solved in $\mathcal{O}(N^3)$ time, where N is the number of states of the input formula [21]. This drastically improves on the previously known doubly exponential lower time bound for general LTL formulas [20]. We utilize the result by Piterman et al. by way of the `omega` solver presented in Section 2.3.2. However, since the approach presented in [21] can only be used to check for realizability of a specification and then constructs an automaton realizing this solution if one exists, enumerating all solutions of such a system is not directly possible.

3.4.1. Exclusion Automaton

Therefore, we present an approach of enumerating satisfying solutions to a GR(1) specification φ. The main idea here is that by excluding the solution $w^{(i)}$ for step i we can generate a new specification $\varphi^{(i+1)}$, which is satisfiable if and only if a new solution $w^{(i+1)}$ exists. However, since any solution to a GR(1) specification is a word w of infinite length, we can not exclude every step of w directly, but split it into a prefix of finite length w_p and a finite cyclic component w_c such that $w = w_p w_c w_c \ldots$ and only exclude the finite prefix w_p. We do so, as the last step of w_c is the first step in w satisfying the specification φ.

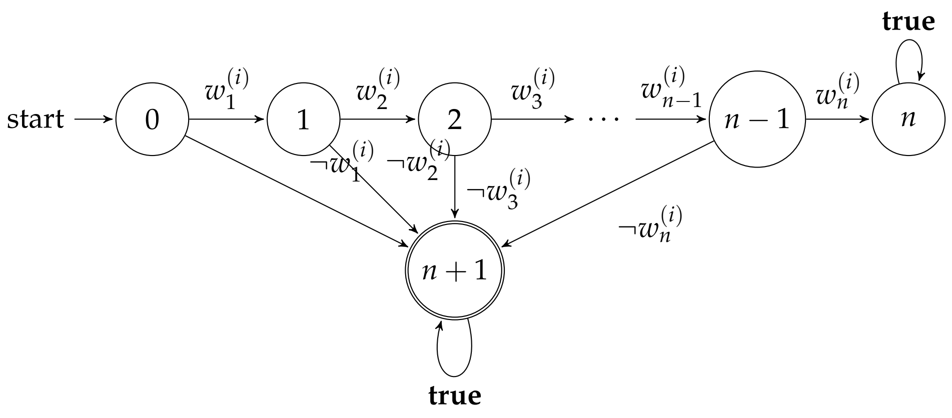


Figure 3.5.: Edge based deterministic Büchi automaton for excluding of a previously seen word $w^{(i)}$

In detail, we perform this exclusion using the automaton in Figure 3.5. It graphically represents the process of excluding future words $w_p^{(j)}$ for $j > i$ by matching each step $w_1^{(i)}, w_2^{(i)}, \dots, w_n^{(i)}$ of $w_p^{(i)}$ and staying in state n if a complete match to $w_p^{(i)}$ was found, moving to state $n+1$ and staying in it if $w_p^{(j)}$ differs at any point from $w_p^{(i)}$. Semantically we can now define state n to be equivalent to $false$ and state $n+1$ to $true$. To express this graph in GR(1), we use the following conversion, introducing a new variable $s_i \in [\![0..n+1]\!]$ representing the state of the automaton.

Definition 3.4.1 (Exclusion Automaton encoding)**.**

$$s_i = 0 \tag{3.18}$$

$$\begin{aligned}
&\mathbf{G}((s_i = 0 \land w_1^{(i)}) \Rightarrow \mathbf{X}(s_i = 1)) \\
&\land\mathbf{G}((s_i = 0 \land \neg w_1^{(i)}) \Rightarrow \mathbf{X}(s_i = n+1)) \\
&\land\mathbf{G}((s_i = 1 \land w_2^{(i)}) \Rightarrow \mathbf{X}(s_i = 2)) \\
&\land\mathbf{G}((s_i = 1 \land \neg w_2^{(i)}) \Rightarrow \mathbf{X}(s_i = n+1)) \\
&\vdots \\
&\land\mathbf{G}((s_i = n-1 \land w_n^{(i)}) \Rightarrow \mathbf{X}(s_i = n)) \\
&\land\mathbf{G}((s_i = n-2 \land \neg w_n^{(i)}) \Rightarrow \mathbf{X}(s_i = n+1)) \\
&\land\mathbf{G}((s_i = n) \Rightarrow \mathbf{X}(s_i = n)) \\
&\land\mathbf{G}((s_i = n+1) \Rightarrow \mathbf{X}(s_i = n+1)))
\end{aligned} \tag{3.19}$$

$$\mathbf{GF}(s_i = n+1) \tag{3.20}$$

Since we use the same process as in paragraph 3.3.3 where we've presented an argument for the validity of this conversion, it is by the same logic sound and complete here as well. Identically, Equation 3.18 is added to the system initial state θ^s in the final GR(1) formula. Equation 3.19 exactly encodes the transition relation for the exclusion automaton in Figure 3.5 which is added to the transition constraints ρ^s and Equation 3.20 encodes the automaton's goal state, which gets added to J_1^s as a justice constraint. Using these GR(1) formulas we can therefore exclude $w_p^{(i)}$ from all future solutions while adding only a number of states linear in the length of $w_p^{(i)}$. By repeating this approach until the formula is no longer satisfiable, we enumerate all its solutions.

3.4.2. BDD Caching

The `omega` solver we utilize for generating a satisfying solution $w_p^{(i)}$ for a GR(1) formula φ, first converts it to a graph-based representation of that formula, namely a Binary Decision Diagram (BDD) introduced in [40], using the Python library `dd`[3]. Even though, the approach presented in the previous section on excluding each word $w_p^{(i)}$ to enumerate all satisfying words of a formula φ in GR(1) only increases the resulting formula length linearly, it still requires a complete conversion of φ to BDDs. This process however, is computationally expensive and through the simple observation that a formula φ' generated by the exclusion process from φ still contains φ entirely, we cache the BDD for φ and convert the added formulae by the exclusion process separately to BDDs. Then both are cached for future use and combined to form φ'. By applying this process successively, only the formulae added by the exclusion process are converted to BDDs when enumerating the solutions for a template, while combination of BDDs ($\varphi \wedge \varphi'$) can be performed efficiently using the procedure defined in [40].

3.4.3. GR(1) Expression

Encoding of a scene transition in a template $\mathcal{T}$ from $\mathcal{S}_i \rightsquigarrow \mathcal{S}_{i+1}$ for $\mathcal{S}_i, \mathcal{S}_{i+1} \in \mathcal{T}$ is done by assigning the conjunction of all relations in $\mathcal{S}_i = \left\{r_1^i, r_2^i, \ldots, r_{|\mathcal{S}_i|}^i\right\}$ to the initial system state of our GR(1) formula θ^s.

$$\theta^s = r_1^i \wedge r_2^i \wedge \cdots \wedge r_{|\mathcal{S}_i|}^i \tag{3.21}$$

The succeeding scene's relations $\mathcal{S}_{i+1} = \left\{r_1^{i+1}, r_2^{i+1}, \ldots, r_{|\mathcal{S}_{i+1}|}^{i+1}\right\}$ are similarly conjoined together and added to the first justice constraint J_1^s of the formula.

$$J_1^s = r_1^{i+1} \wedge r_2^{i+1} \wedge \cdots \wedge r_{|\mathcal{S}_{i+1}|}^{i+1} \tag{3.22}$$

If not explicitly stated otherwise, we then add the remaining general constraints presented in the previous section to the system transition ρ^s, with initial and final states

[3]`https://github.com/tulip-control/dd`

for the overtaking (definition 3.3.2) and exclusion automata (definition 3.4.1) added to θ^s and J_1^s respectively. Based on these we give the following GR(1) formula.

Definition 3.4.2 (GR(1) Expression)**.**

$$(\mathbf{true} \wedge \mathbf{G}(\mathbf{true}) \wedge \mathbf{GF}(\mathbf{true})) \Rightarrow (\theta^s \wedge \mathbf{G}\rho^s \wedge \mathbf{GF}J_1^s)$$

As given by the above definition, we have found this GR(1) formula to be sufficient in our work. By defining all environment variables to true we cause our solver `omega` to find solutions irrespective of the environment. Each solution to the formula in definition 3.4.2 then, by construction, satisfies all constraints in the original template $\mathcal{T}$.

3.5. Accepting Word Parsing

Based on the automaton satisfying the GR(1) formula we gave in definition 3.4.2, we in this section cover the parsing of it to a new scenario template $\mathcal{T}^{sol}$, which concretizes the original template $\mathcal{T}$. This concretization, will provide all intermediate scenes necessary, for transitioning between the ones specified in $\mathcal{T}$, therefore giving a complete specification of all vehicles in $\mathcal{T}$ over all time steps.

In definition 3.4.2 we gave the definition of our GR(1) formula, which can be interpreted as a Streett Game [41]. The solution to which is expressed as an automaton, where each state is an input to the system with transitions between them. Since this automaton satisfies a GR(1) formula whose justice constraints have to occur infinitely often, each trace w of this automaton is infinitely long. Therefore, we split this word w in to a prefix w_p and cyclic component w_c of finite length such that $w = w_p w_c w_c \dots$ similar to Section 3.4. The specific point at which to start for this cyclic component w_c is selected such that the first step of w_c is the first in w to satisfy the justice constraint J_1^s. Since we only define a single justice constraint in Section 3.4.3 and all justice constraints in a GR(1) formula have to be fulfilled infinitely often, such separation w_p, w_c has to exist with the steps in w_c repeating infinitely often in w. An example of this split for a given automaton solving our running example problem, is given in Figure 3.6.

Each state in the accepting word sections w_p, w_c defined above is made up of a Boolean function (specifically a conjunction of literals) $b \in \Phi$, defining the values of each literal in the original GR(1) formula to constitute a satisfying run. By parsing every such formula b using the parser `lark`[4], we reconstruct the corresponding relations and their states from b. Since every formula b corresponds to one discrete state the system needs to reach in order to satisfy the original GR(1) formula, we collect this set of parsed relations in a new scene $\mathcal{S}$. When applying this process for every state in w_p and the first one in w_c we construct a new sequence of scenes $(\mathcal{S}_{i,1}, \mathcal{S}_{i,2}, \dots, \mathcal{S}_{i+1,n_i})$. This new sequence then gives a valid transition between two scenes in the original template from scene $\mathcal{S}_i = \mathcal{S}_{i,1}$ to $\mathcal{S}_{i+1} = \mathcal{S}_{i+1,n_i}$. By solving each GR(1) formula for every scene transition $\mathcal{S}_i \rightsquigarrow \mathcal{S}_{i+1}$ in the original template $\mathcal{T}$, we obtain one such sequence

[4]`https://github.com/lark-parser/lark`

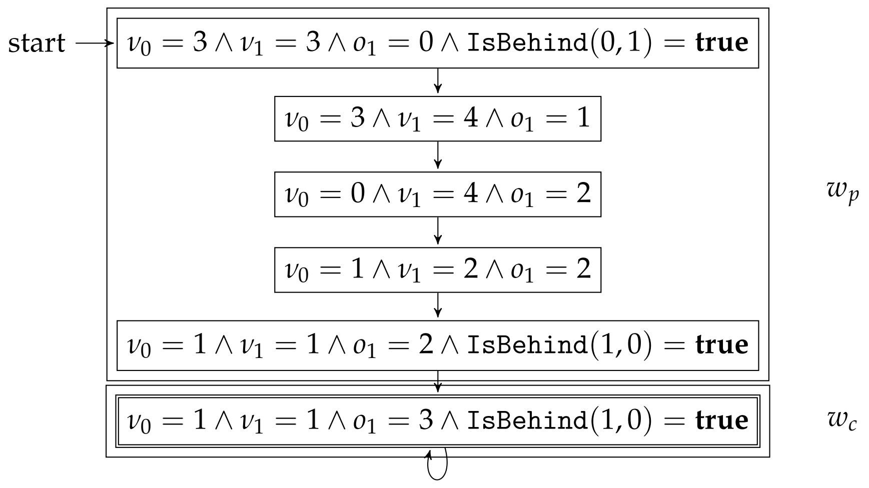

start
$\nu_0 = 3 \wedge \nu_1 = 3 \wedge o_1 = 0 \wedge \texttt{IsBehind}(0,1) = \textbf{true}$
$\nu_0 = 3 \wedge \nu_1 = 4 \wedge o_1 = 1$
$\nu_0 = 0 \wedge \nu_1 = 4 \wedge o_1 = 2$
$\nu_0 = 1 \wedge \nu_1 = 2 \wedge o_1 = 2$
$\nu_0 = 1 \wedge \nu_1 = 1 \wedge o_1 = 2 \wedge \texttt{IsBehind}(1,0) = \textbf{true}$
$\nu_0 = 1 \wedge \nu_1 = 1 \wedge o_1 = 3 \wedge \texttt{IsBehind}(1,0) = \textbf{true}$
w_p
w_c

Figure 3.6.: Accepting Büchi automaton for example 1. The split of $w = w_p w_c w_c \dots$ is given by the respective boxes.

of concrete scenes for every transition. Concatenating them together while leaving out duplicates, we can create the following solved template $\mathcal{T}^{sol}$ for a given template $\mathcal{T}$.

Definition 3.5.1 (Solved Template $\mathcal{T}^{sol}$).

$$\begin{aligned} \mathcal{T}^{sol} := & \left(\mathcal{A}, \mathcal{S}_{1,1}, \mathcal{S}_{1,2}, \dots, \mathcal{S}_{i-1,n_{i-1}}, \mathcal{S}_{i,1}, \mathcal{S}_{i,2}, \dots, \mathcal{S}_{n_{sc}}\right) \\ = & \left(\mathcal{A}, \mathcal{S}_1, \dots, \mathcal{S}_{n_{sol}}\right) \end{aligned} \tag{3.23}$$

The last simplification in the above equation is a re-indexing of scenes in the same order for simplicity' sake', with $n_{sol} \geq n_{sc}$.

4. Optimization based Scenario Synthesis

Based on a concrete scene $\mathcal{T}^{sol}$ for one of our solved GR(1) formulas, we will in the following concern ourselves with synthesizing a concrete scenario from it. Based on work by Klischat [18] we formulate this as a convex optimization problem with the constraints adapted from the previously generated GR(1) solution. So every relation specified in that solution will be expressed as a set of linear constraints on the state of its associated vehicle. This section is intentionally kept brief as most of its content is work by Klischat [18] and the primary purpose of generating concrete scenarios using this approach in this work is to visually verify the specified behavior defined in the previous sections.

Concrete trajectories are synthesized over a time horizon $h \in \mathbb{N}$, in multiples of a time step, by formulating the problem as a Mixed-Integer Quadratic Programming (MIQP) whose solution gives a trajectory for all vehicles. For computational efficiency reasons this problem is split up into two independently solvable MIQPs, one for the longitudinal position of the vehicle and one for the lateral one. In the following we will first show a formulation for the longitudinal problem and using its solution will solve for the lateral positions of each vehicle [18, 19].

4.1. Switching Times

Solving for a concrete solution in each of the optimization problems outlined above, requires the progression of scenes in the solved template $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_{n_{sol}}$ to be expressed as well [18]. To this end for each scene $\mathcal{S}_i$ with $i \in [\![1..n_{sol}]\!]$, Klischat defines a switching time $\underline{k}_i \in [\![0..h]\!]$ when $\mathcal{S}_i$ becomes active. The predicates defined in $\mathcal{S}_i$ then have to hold until the switching time of the next scene $\mathcal{S}_{i+1}$. Since specifying exact switching times before knowing the solution of the optimization problem is difficult, Klischat proposes to set upper and lower bounds for the duration of each scene δ_i as follows.

$$\forall i \in [\![1..n_{sol}]\!] : h_{\min} \leq \delta_i \leq h_{\max} \quad \text{with } h_{\min}, h_{\max} \in \mathbb{N}$$

From these bounds on the duration of a scene δ_i switching times $\underline{k}_i$ are encoded as a binary vector $\beta_i \in \{0,1\}^h$ with $i \in [\![1..n_{sol}]\!]$ whose values switch at $\underline{k}_i$.

$$\beta_i(k) = \begin{cases} 0 & \text{if } k < \underline{k}_i \\ 1 & \text{else} \end{cases}$$

The required scene progression is expressed via linear constraints on β_i.

$$\beta_1(0) = 1 \qquad (4.1)$$
$$\forall k \in [\![1..h]\!], \forall i \in [\![1..n_{sol} - 1]\!] : \beta_i(k) \leq \beta_{i+1}(k)$$

Here at the initial time step Klischat requires $\beta_1(0) = 1$ as an initial condition, expressing that $\mathcal{S}_1$ has to hold at the first time step. For all time steps $k : \beta_i(k) \leq \beta_{i+1}(k)$ gives the desired constraint on scene $\mathcal{S}_i$ coming before $\mathcal{S}_j$ with $1 \leq i < j \leq n_{sol}$. If any given scene $\mathcal{S}_i$ is active or not is then easily given by $\alpha_i \in \{0, 1\}^h$.

$$\forall k \in [\![0..h]\!] : \alpha_i(k) = \begin{cases} \beta_i(k) - \beta_{i+1}(k) & \text{if } 0 \leq i < n_{sol} \\ \beta_i(k) & \text{else} \end{cases}$$

The constraints on duration of scene $\mathcal{S}_i$ are then given by:

$$\forall k \in [\![0..h]\!] : h_{\min} \leq \sum_{k=0}^{h} \alpha_i(k) \leq h_{\max} \qquad (4.2)$$

4.2. Expressing Relations as Mixed-Integer Constraints

After solving for a template $\mathcal{T}^{sol}$ the position of each vehicle $V_i \in \mathcal{A}$ is specified by a corresponding `OnLane`(V_i, l, s) relation. In contrast to the original template $\mathcal{T}$, $\mathcal{T}^{sol}$ specifies all intermediate lanes as well, such that all lanes visited by a vehicle V_i are fully specified by its `OnLane`(V_i, l, s) relations in $\mathcal{T}^{sol}$. Within a given lane, two vehicles V_i, V_j might still occupy it at the same time, in which case a `IsBehind`(V_i, V_j) relation is also given. Therefore, it is sufficient to only express `OnLane`(V_i, l, s) and `IsBehind`(V_i, V_j) relations for all vehicles $V_i, V_j \in \mathcal{A}$ as constraints in the optimization problem.

In the optimization approach by Klischat interaction between different vehicles is achieved by specifying a reference path R_{V_i} for every vehicle V_i [18]. Such a path is essentially an in-order concatenation of all lanes, vehicle V_i visits in $\mathcal{T}^{sol}$. Using this reference path, the vehicle's position is specified in curvilinear coordinates on R_{V_i}, i.e. its longitudinal state x_{s,V_i} and the lateral state x_{d,V_j} are given by the following equations.

$$x_{s,V_i} = (s_{V_j}, \dot{s}_{V_i}, \ddot{s}_{V_i}) \qquad x_{d,V_i} = (d_{V_i}, \dot{d}_{V_i}, \ddot{d}_{V_i}) \qquad (4.3)$$

With s_{V_i} being the longitudinal- and d_{V_i} the lateral offset of V_i with respect to R_{V_i} (more on curvilinear coordinates in Section 2.2).

Satisfying the relations of a template $\mathcal{T}^{sol}$ for each vehicle V_i with respect to its reference lane R_{V_i} is then easily expressed with the projection operator $\pi_{R_{V_i}}(s)$, which projects curvilinear coordinates with respect to a lane l to the vehicles reference lane R_{V_i}.

1. `OnLane`(V_i, l, s)
   A vehicle V_i is on lane l and within that lane's s-range $s = (\underline{s}, \overline{s})$, if the longitudinal position of V_i, s_{V_i} is within the required range $(\underline{s}, \overline{s})$ when projected onto its

reference lane R_{V_i}. Additionally, its lateral offset d_{V_i} has to be within the lane's minimal and maximal lateral offset $\underline{d}(s_{V_i}), \overline{d}(s_{V_i})$ at point s_{V_i}.

$$\texttt{OnLane}(V_i, l, s) \Leftrightarrow \pi_{R_{V_i}}(\underline{s}) \leq s_{V_i} \leq \pi_{R_{V_i}}(\overline{s}) \land \underline{d}(s_{V_i}) \leq d_{V_i} \leq \overline{d}(s_{V_i}) \tag{4.4}$$

2. $\texttt{IsBehind}(V_i, V_j)$
   Since a $\texttt{IsBehind}(V_i, V_j)$ relation is only given in $\mathcal{T}^{sol}$ if V_i and V_j are on the same lane, we can simply express this as a constraint on the longitudinal offset of V_i, having to be less or equal than the one of V_j. However, since V_i, V_j might be on different reference lanes R_{V_i}, R_{V_j}, Klischat introduces a common reference point x_{ref} somewhere on the common lane of V_i, V_j. By then constraining the vehicle's offsets with respect to x_{ref}, we can correct for this difference in reference lane coordinate systems. Additionally, a minimal safety distance $r \in \mathbb{R}$ is given, which the vehicle's distance to each other cannot not fall below.

$$\begin{aligned}\texttt{IsBehind}(V_i, V_j) &\Leftrightarrow s_{V_i} - \pi_{R_{V_i}}(x_{ref}) + r \leq s_{V_j} - \pi_{R_{V_j}}(x_{ref}) \\ &\Leftrightarrow s_{V_j} - s_{V_i} \geq \pi_{R_{V_j}}(x_{ref}) - \pi_{R_{V_i}}(x_{ref}) + r\end{aligned} \tag{4.5}$$

The constraints in Equations (4.4) to (4.5) can then be rewritten as general linear functions and directly used in the optimization problem. A large constant M is used to only activate them in their corresponding scene $\mathcal{S}_i$ [18].

$$\underline{g}_i(x_s, k) > \underline{c}_i(k) - M(1 - \alpha_i(k)) \tag{4.6}$$

$$\overline{g}_i(x_s, k) > \overline{c}_i(k) + M(1 - \alpha_i(k)) \tag{4.7}$$

4.3. Longitudinal Problem

Solutions for the longitudinal problem are always with respect to some user-defined cost function $J_s(x_s, u_s, w)$, where the index s refers to constraints pertaining to the longitudinal problem. $x_s = (x_{s,1}^T, \ldots, x_{s,|\mathcal{A}|})$, $u_s = (\dddot{s}_1, \ldots, \dddot{s}_{|\mathcal{A}|})$ refer to the states and jerk for all vehicles weighted by $w \in \mathbb{R}^\rho$.

Definition 4.3.1 (Longitudinal optimization problem).

$$\underset{x_s(0),u_s}{\operatorname{argmin}} \sum_{k=0}^{b} J_s(x_s(k), u_s(k), w) \tag{4.8}$$

$$\forall k \in [\![0..h]\!] : \quad x_s(k+1) = A_s x_s(k) + B_s u_s(k) \tag{4.9}$$

$$u_{s,\min} \leq u_s(k) \leq u_{s,\max}$$

$$x_{s,\min} \leq x_s(k) \leq x_{s,\max}$$

$$\forall i \in [\![0..n_{sol}]\!] : \quad \underline{g}_i(x_s, k) < \underline{c}(k) + M(1 - \alpha_l(k)) \tag{4.10}$$

$$\overline{g}_i(x_s, k) > \overline{c}(k) + M(1 - \alpha_l(k))$$

$$\beta_0(0) = 1 \tag{4.11}$$

$$\forall i \in [\![0..n_{sol} - 1]\!] : \quad \beta_i(k) \leq \beta_{i+1}(k)$$

$$h_{\min} \leq \sum_{k=0}^{h} \alpha_(k) \leq h_{\max} \tag{4.12}$$

The above MIQP system then minimizes Equation 4.8 with respect to the dynamic constraints in Equation 4.9 describing the physical properties of each vehicle. Equation 4.10 gives the relational constraints and Equations (4.11) to (4.12) encode the switching times.

4.4. Lateral Problem

After solving the longitudinal problem Klischat uses its solution to solve the lateral problem and synthesize the trajectory for each vehicle. This problem is simpler, as a solution to the longitudinal problem already includes solutions for the switching times $\beta_i(k)$, reducing the lateral problem to a quadratic problem. Additionally, using the solution to the longitudinal problem, the exact lanes a vehicle is on for every time step k are known, which is used to compute the lateral bounds $[\underline{d}(s(k)), \overline{d}(s(k))]$ for every k.

Definition 4.4.1 (Lateral optimization problem).

$$\underset{u_d(\cdot)}{\operatorname{argmin}} \sum_{k=0}^{h} J_d(x_d(k), u_d(k), d_{ref}(k), w) \tag{4.13}$$

$$\forall k \in [\![0..h]\!] : \quad x_d(k+1) = A_d x_d(k) + B_d u_d(k) \tag{4.14}$$

$$u_{d,\min}(k) \leq u_d(k) \leq u_{d,\max}$$

$$x_{d,\min}(k) \leq x_d(k) \leq x_{d,\max}$$

$$\underline{d}_L(s(k)) \leq x_d(k) \leq \overline{d}_L(s(k)) \tag{4.15}$$

By combining the solutions to the longitudinal- and lateral problems, the exact state for every vehicle at every time step $k \in [\![0..h]\!]$ is specified and a CommonRoad scenario is created. Examples for such scenarios are given in chapter 5.

5. Results

We demonstrate the effectiveness of our approach on two scenarios, which are chosen to represent two distinct settings in autonomous driving. The first is a multi vehicle highway scenario in which we demonstrate overtaking of different vehicles and unconstrained interactions between them. The second is an urban scenario in which we show how our generated scenarios behave when tightly constrained by their environment. We also empirically evaluate the applicability of the results for both settings.

Since our contribution in this work is primarily on formula generation and enumeration, we will not focus on the optimization aspect of scenario synthesis, but instead on satisfiability enumeration. All parameters required for the optimization approach by Klischat [18], are given in Tables A.1 to A.3, with the specific ones for each template ($\Delta t, t_h$, max iteration) given separately in Table 5.1 and Table 5.2. All our evaluation is performed on an Intel i7-8565U mobile processor with 16 GB RAM.

5.1. Highway Scenario

One of the use cases for our approach is to create highly dynamic scenarios on highways. We consider the event of one vehicle overtaking another on a main carriage highway. Additionally, we add another unconstrained vehicle parallel to the merging ones, representing another traffic participant. The lane network for this scenario is given in Figure 5.1 with the respective template in the following.

Parameter	**Value**	
Δt	0.2	(s)
t_h	12	(s)
max iteration	100	

Table 5.1.: Highway Scenario parameters

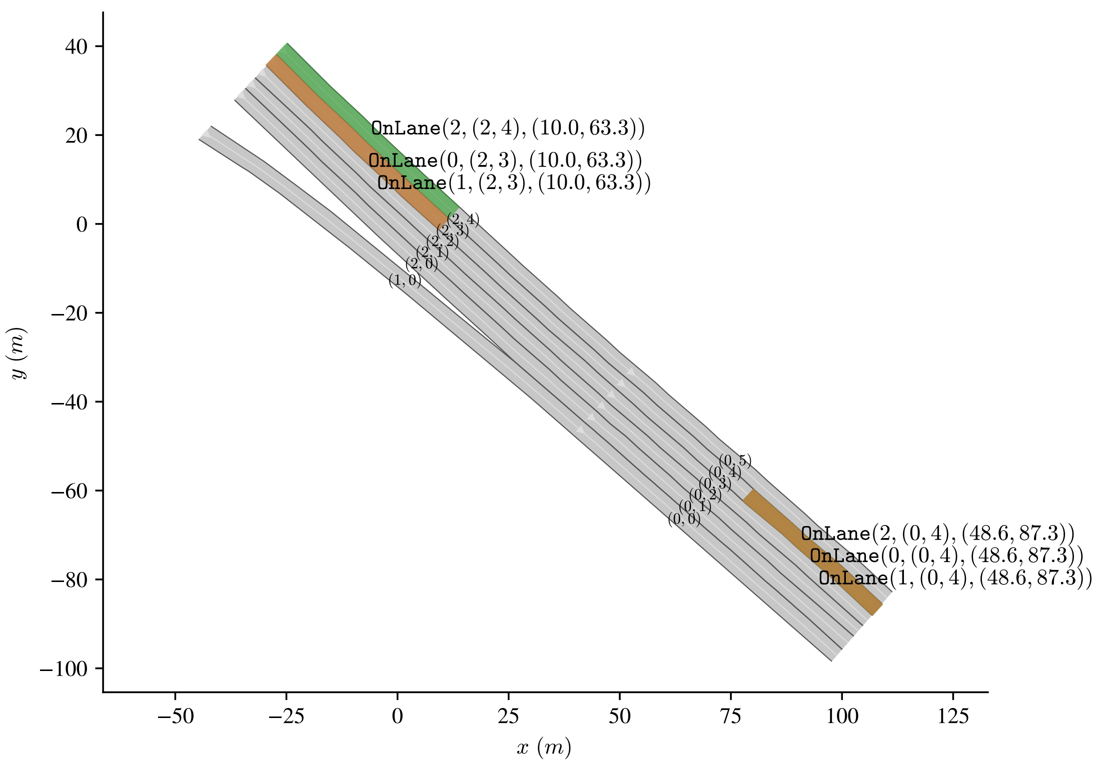


Figure 5.1.: Lane network with color coded $\texttt{OnLane}(V_i, l, s)$ specifications for each vehicle from the respective template

Example 2 (Highway Scenario Template).

$$\begin{aligned}
\mathcal{T}_{hw} =& (\mathcal{A}, \mathcal{S}_1, \mathcal{S}_2) \\
\mathcal{A} =& (0, 1, 2) \\
\mathcal{S}_1 =& \{\texttt{OnLane}(0, (2, 3), [10.0, 63.3]),\ \texttt{OnLane}(1, (2, 3), [10.0, 63.3]), \\
& \texttt{OnLane}(2, (2, 4), [10.0, 63.3]),\ \texttt{IsBehind}(0, 1)\} \\
\mathcal{S}_2 =& \{\texttt{OnLane}(0, (0, 4), [48.6, 87.3]),\ \texttt{OnLane}(1, (0, 4), [48.6, 87.3]), \\
& \texttt{OnLane}(2, (0, 4), [48.6, 87.3]),\ \texttt{Overtakes}(0, 1), \\
& \texttt{IsBehind}(1, 0),\ \texttt{IsBehind}(1, 2)\}
\end{aligned}$$

A visual representation of this template $\mathcal{T}_{hw}$ is given in Figure 5.1. In natural language, this template defines vehicles $0, 1$ to be on the same lane $(2, 3)$ with 0 behind 1 in the first scene $\mathcal{S}_1$. Simultaneously, vehicle 2 is on the adjacent lane $(2, 4)$ to vehicles $0, 1$. In the next scene we require 0 to have overtaken 1 with both ending on lane $(0, 4)$, while vehicle 2 also moves from its starting lane to $(0, 4)$. On that lane, as 1 was overtaken, we assert it to behind vehicles $1, 2$.

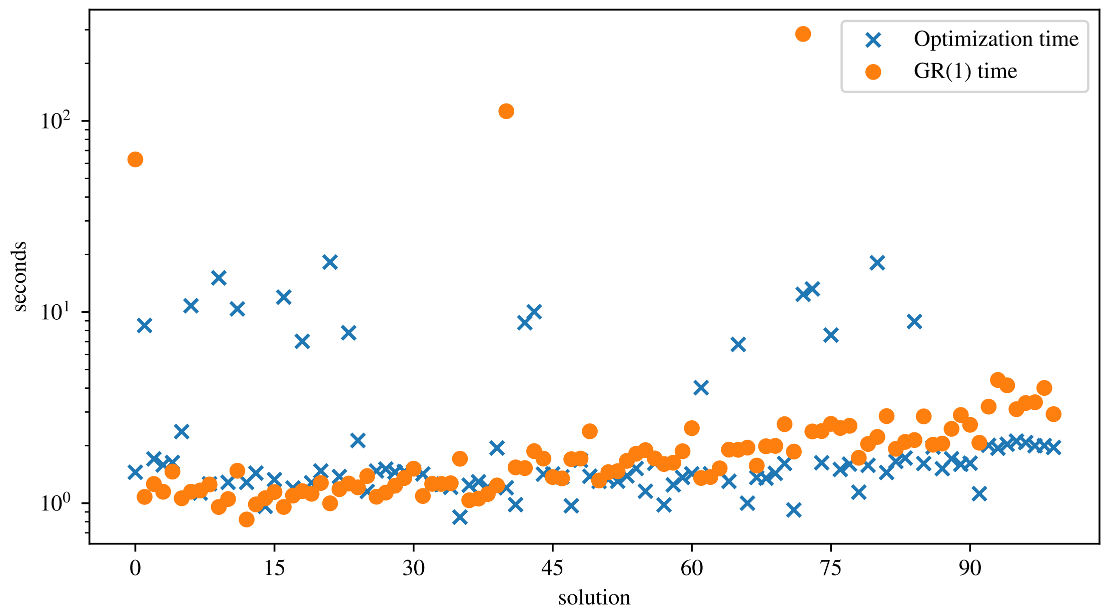


Figure 5.2.: Computation times for the highway scenario in seconds for 100 iterations.

The resulting predicate graph from $\mathcal{T}_{hw}$ in given in Figure A.1. However, since the predicate graph for a template characterizes all states all vehicles can move to, while satisfying the template, the resulting graph in this example is quite large. But one can clearly make out the individual connected components, constituting the reachable states for each vehicle. Also since the predicate graph is encoded in full into a GR(1) formula, this one is also large (cf. Listing A.1).

5.1.1. Computational Performance

From such a GR(1) formula we describe in Section 3.4 and chapter 4 how we enumerate all solutions to it and synthesize concrete scenarios those solutions. In this section we will cover the practicability of our approach by evaluating the computation times for each of the before mentioned major steps of solving the GR(1) formula and synthesizing a concrete scenario through an optimization problem.

Scaling In Figure 5.2 we give the computation times required of each generated solution to the GR(1) formula for the template $\mathcal{T}_{hw}$ (cf. Listing A.1) for the first 100 satisfying solutions to that template. From this plot it becomes immediately apparent, that computation time for solving each new GR(1) formula increases with the number of previously found solutions. This is the case, as we ensure uniqueness of our generated GR(1) solutions, by excluding each previously found solution by appending its negation to the next formula (more details in Section 3.4). This causes the number of states of each new formula to increase linearly, as each exclusion adds a new variable s_i with approximately a constant number of states[1], matching future solutions against previously seen ones. As the time required to satisfy any GR(1) formula scales as $O(N^3)$, where N is the number of states of the formula [21] we expect to see cubic scaling in our computation times for each successive GR(1) solution as well.

Outliers Also, when considering the absolute computation times for each GR(1) formula, we sometimes observe large outliers, shown in orange in Figure 5.2. Even though, we have given time on a logarithmic axis for visualization purposes, those outliers still dominate all other values. We postulate that this is due to caching employed by ourselves in Section 3.4.2 as well as caching internal to our solver `omega`, as for the first solution the required GR(1) solve time is large and is only a fraction of that for most other ones. One plausible explanation for the behavior, is that by defining the GR(1) formula for a successive solution to be an extension of the previous one (details in Section 3.4), work done to convert the previous formula to BDDs is reused and only the newly added conjunctions need to be considered. This theory also explains the rare continued outliers after the first solution, as there could plausibly be formulas where so many previous solutions were excluded, that `omega` could not reuse previous work (computed fixed points) and have to resolve large parts of the input formula.

Optimization Considering the optimization times in Figure 5.2 shown in blue, we see them clustered into roughly two groups. One greater than the GR(1) solution times and less. This is due to the optimization approach used in this work only having been able to solve 16 of the 100 generated solutions by our approach, where each infeasible optimization problem failed within a few seconds. We assume that this behavior is due to the complexity of the template $\mathcal{T}_{hw}$ and some solutions to it requiring extreme behavior by the associated vehicles to satisfy. We then assume this behavior infeasible with the provided vehicle model and given time horizon in Table 5.1.

Another possible explanation is, that overtaking maneuvers, as defined in $\mathcal{T}_{hw}$ have some inconsistency, when expressed an optimization problem as given by Klischat. That line of argument is backed up by Klischat's own synthesis approach also has trouble correctly synthesizing overtaking maneuvers [18]. However, due to the limited scope of this thesis we were unable to investigate this hypothesis and consider it possible future work.

For solutions to our GR(1) formula, which are feasible we do however, see an expected approximately constant time required to solve their associated optimization problem, as all solutions solve for similar behavior on the same lane network.

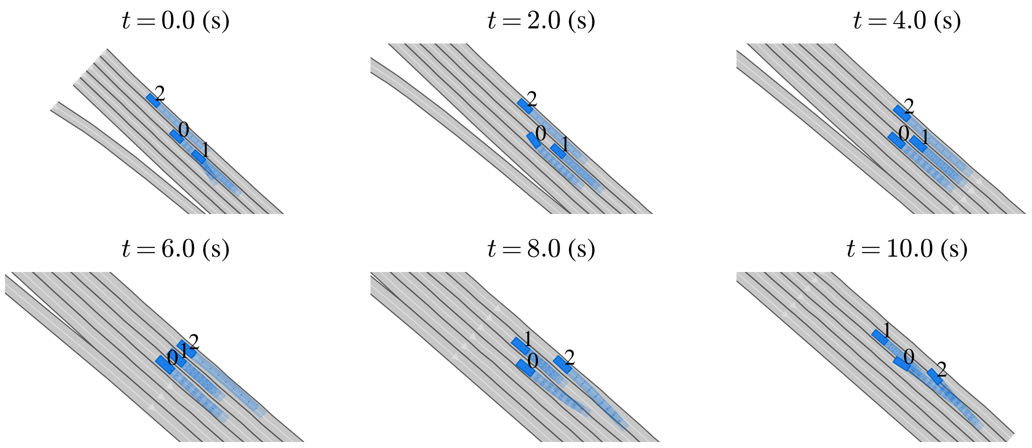


Figure 5.3.: Representative synthesized solution for the highway template $\mathcal{T}_{hw}$ at six different times $0 \leq t \leq t_h$. Light blue shows future positions of a vehicle.

5.1.2. Scenario Evaluation

In the following we consider the behavior of vehicles in the 19 solutions to the template $\mathcal{T}_{hw}$ where the optimization problem is feasible. One such solution is shown in Figure 5.3 with the full set of all solutions to the templates presented, being available alongside this work[2].

We specifically show the scenario in Figure 5.3 out of the 19 synthesized ones, as it is one of the clearest to represent in picture form and still shows behavior representable for the entire set of solutions. In it, one can see that, just as defined in the template $\mathcal{T}_{hw}$ the vehicles $0, 1$ stat on lane $(2, 3)$ with vehicle 0 behind 1 and 2 adjacent to them. With the remainder of the defined behavior in $\mathcal{T}_{hw}$ also present.

For instance, at time $t = 2.0$ one sees vehicle 0 starting its overtaking maneuver from on the right adjacent lane of 1 with vehicle 2 being unaffected by vehicles $0, 1$. This right overtaking maneuver might at first seem erroneous, as it violates traffic code in right-handed traffic systems. But when specifying the semantics of overtaking maneuvers in definition 3.3.2 we did not give a direction for doing so, allowing overtaking maneuvers to occur on both sides of the overtaken vehicle. However, a constraint preventing such right overtaking maneuvers could easily be added in the future.

Following the overtaking maneuver, all vehicles merge onto their target lane, as shown in Figure 5.3 at time $t = 10.0$, with vehicle 1 being behind vehicles $0, 2$. Since we did not specify an order between vehicles $0, 2$ for $\mathcal{S}_2$ in $\mathcal{T}_{hw}$, this is left as a degree of freedom within the corresponding GR(1) formula and was chosen in this solution as vehicle 0 being behind 2. But the other possible order, vehicle 2 behind 0 is also present

[1]The number of states depends on the number of steps required to transition from one scene to the next $\mathcal{S}_i \rightsquigarrow \mathcal{S}_{i+1}$, which is bounded by the longest satisfying path in the given lane network (i.e. constant).
[2]`https://github.com/rasaford/scenario-generation`

Parameter	Value	
Δt	0.25	(s)
t_h	18.75	(s)
max iteration	100	

Table 5.2.: Urban Scenario parameters

within the synthesized scenarios, showing how the approach presented in this work is able to handle partially specified orders and enumerate degrees of freedom left within a template.

5.2. Urban Scenario

The second considered use case of this work is one of urban scenarios. Since the approach used to synthesize satisfying solutions is the same across all of this work, we see similar patterns in the resulting scenarios here, as discussed in the previous section. Therefore, we will only point out differences to the general findings in the previous section and refer the reader to Section 5.1 for more context and details.

We consider the following template for the remainder of this section. Based on the lane network in Figure 5.4 we define four vehicles at a T-junction. Two of them approach the junction from the north going south and two different vehicles approach from the south and go west. When doing so these vehicles have to cross a common critical section, as their paths necessarily cross. For this critical section, we only allow a single vehicle at a time to be on one of the lane sections contained within. This behavior can be formally defined as follows.

Example 3 (Urban Scenario Template)**.**

$$\begin{aligned}
\mathcal{T}_{urb} =& (\mathcal{A}, \mathcal{S}_1, \mathcal{S}_2) \\
\mathcal{A} =& (0,1,2,3) \\
\mathcal{S}_1 =& \{\texttt{OnLane}(0,(6,0),[10.0,35.1]),\ \texttt{OnLane}(1,(6,0),[10.0,35.1]), \\
& \texttt{OnLane}(2,(5,0),[10.0,77.4]),\ \texttt{OnLane}(3,(5,0),[10.0,77.4]) \\
& \texttt{IsBehind}(0,1),\ \texttt{IsBehind}(2,3)\} \\
\mathcal{S}_2 =& \{\texttt{OnLane}(0,(9,0),[31.0,80.0]),\ \texttt{OnLane}(1,(9,0),[31.0,80.0]), \\
& \texttt{OnLane}(2,(1,0),[10.0,36.6]),\ \texttt{OnLane}(2,(1,0),[10.0,36.6]) \\
& \texttt{IsBehind}(0,1),\ \texttt{IsBehind}(2,3)\}
\end{aligned}$$

With the corresponding visualization in Figure 5.6 one can see that $\mathcal{T}_{urb}$ defines vehicles $0,1$ to start on lane $(6,0)$ with 0 behind 1 on that lane. Vehicles $2,3$ similarly start on lane $(5,0)$ with vehicle 2 behind 3. Both groups of vehicles $0,1$ and $2,3$ are then

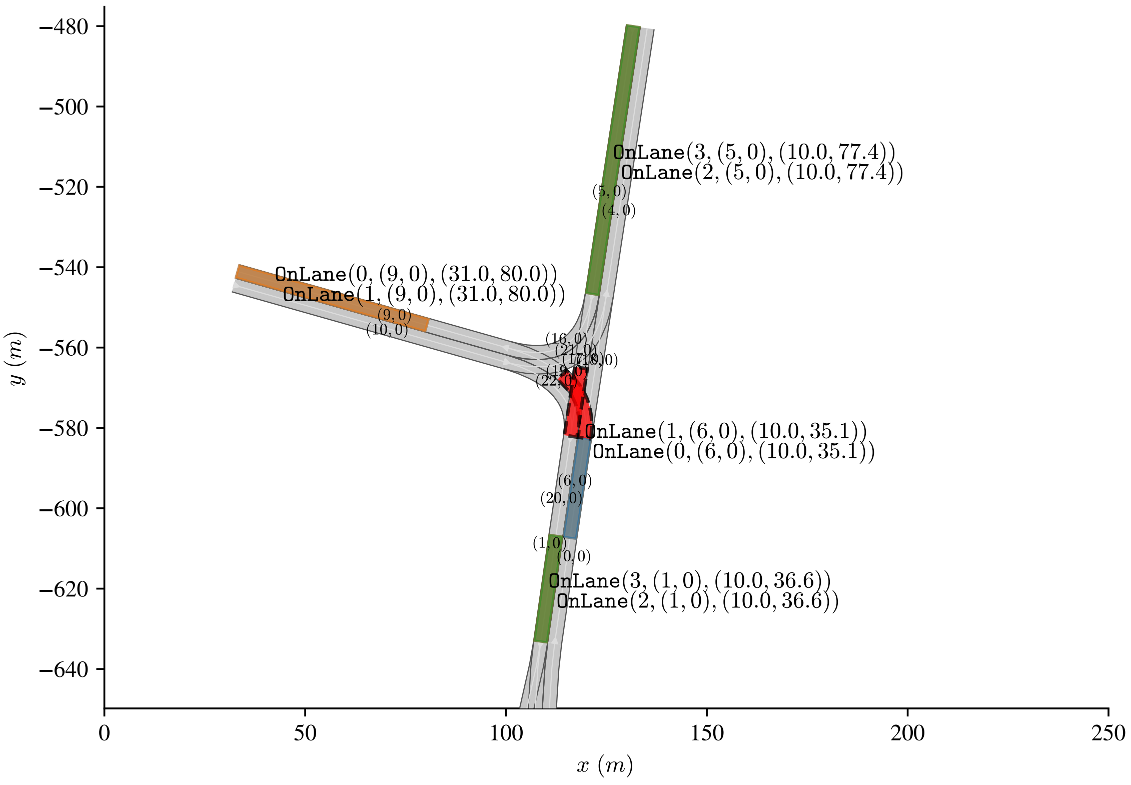

OnLane(3, (5, 0), (10.0, 77.4))
OnLane(2, (5, 0), (10.0, 77.4))
(5, 0)
(4, 0)
OnLane(0, (9, 0), (31.0, 80.0))
OnLane(1, (9, 0), (31.0, 80.0))
(9, 0)
(10, 0)
(16, 0)
(21, 0)
(17, 0)
(18, 0)
(19, 0)
(22, 0)
OnLane(1, (6, 0), (10.0, 35.1))
OnLane(0, (6, 0), (10.0, 35.1))
(6, 0)
(20, 0)
(1, 0)
(0, 0)
OnLane(3, (1, 0), (10.0, 36.6))
OnLane(2, (1, 0), (10.0, 36.6))
y (m)
x (m)
−480
−500
−520
−540
−560
−580
−600
−620
−640
0
50
100
150
200
250

Figure 5.4.: Lane network with color coded `OnLane`(V_i, l, s) specifications for each vehicle from the respective template

constrained to move to their respective end lanes $(9, 0)$ and $(1, 0)$, while maintaining their order. To satisfy these constraints, both groups have to cross the common critical section (marked in red) in serial, meaning only a single vehicle can occupy it at a given time.

Similarly to the previous example, we give the Predicate graph for $\mathcal{T}_{urb}$ in Figure A.2. Even though, the template $\mathcal{T}_{urb}$ and its associated map are more constrained than the previous one, the number of satisfying states to $\mathcal{T}_{urb}$ is still large, resulting in many nodes in the Predicate graph.

5.2.1. Computational Performance

Enumerating the solutions satisfying $\mathcal{T}_{urb}$ gives a similar picture as in the previous section. Based on the computation times given in Figure 5.5, an increase in the time required to solve each successive GR(1) formula (orange) is visible. Interestingly, the increase in time for this template is sharper than the one in the previous section. This possibly indicates that over the course of enumerating the first 100 solutions to the template $\mathcal{T}_{urb}$, its more constrained specification causes our solver `omega` to have to search longer for novel satisfying assignments.

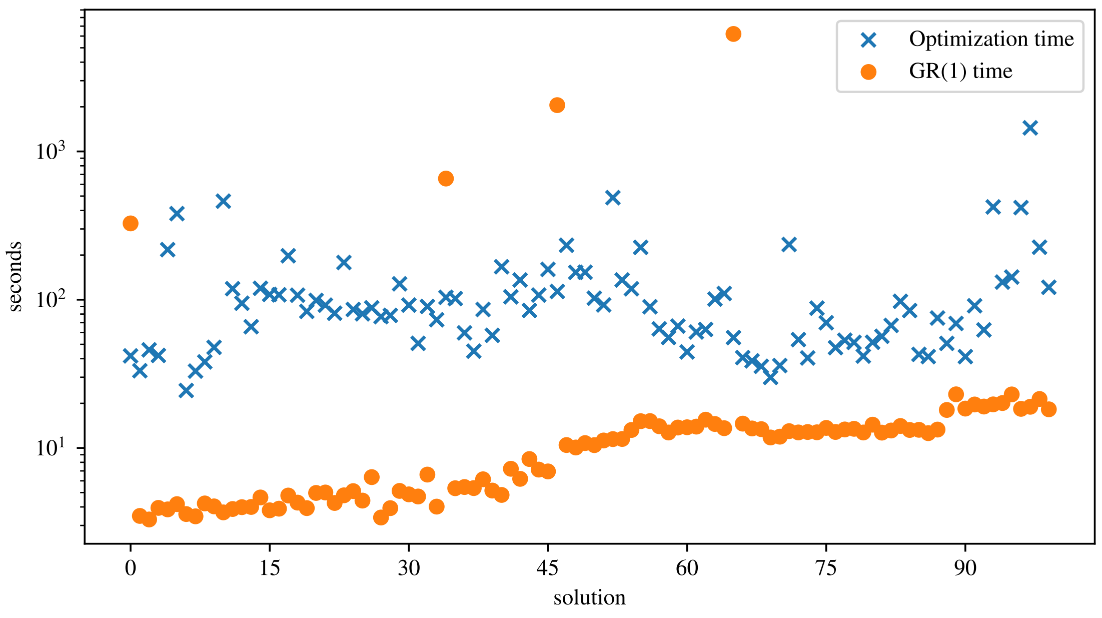


Figure 5.5.: Computation times for the urban scenario in seconds for 100 iterations.

Such a hypothesis is also supported by the relatively large number of outliers in the computation times to solve each GR(1) formula, as each new solution might be unable to reuse past work (fixed points), increasing the required solution time.

Optimization For this template $\mathcal{T}_{urb}$ the optimization approach by Klischat [18] was able to synthesize scenarios for all the 100 solutions our approach generated. Therefore, the computation times for solving each optimization problem are overall in a single cluster, in contrast to the previous section, and are approximately constant as expected, due to all problems being very similar. We assume the larger scale variations visible to be up to random behavior of the test system, independent of our approach.

On the stark difference is the number of feasible optimization problems for template $\mathcal{T}_{urb}$ when compared to $\mathcal{T}_{hw}$, we hypothesize this to be due to the lower velocity interactions between vehicles present in all solutions to $\mathcal{T}_{urb}$.

5.2.2. Scenario Evaluation

In Figure 5.6 we show one representative synthesized scenario from the set of all 100 such solved ones. We choose this specific one, as it most clearly shows the behavior of all four vehicles defined by the template $\mathcal{T}_{urb}$. However, all templates show similar behavior as the one presented in here.

From Figure 5.6 one can immediately see at time $t = 0$ that all four vehicles are on their required lanes as given by $\mathcal{T}_{urb}$. From those starting lanes, vehicles $0, 1$ first cross the intersection, having to do so in serial for all vehicles, as we require it to be a critical

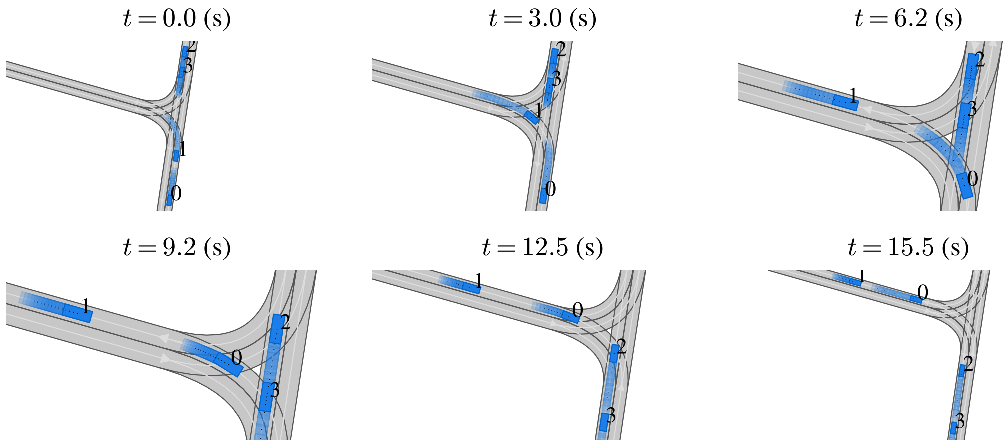


Figure 5.6.: Representative solution for the urban template $\mathcal{T}_{urb}$ at six different times t. Light blue shows future positions of a vehicle.

section to prevent vehicles colliding. Then vehicles 2, 3 cross the same critical section one after another and continue to their end lane.

As one can see from the behavior in Figure 5.6 between times $t = 3.0$ and $t = 6.2$, even though vehicles 2, 3 are just ahead of the intersection, vehicles 0, 1 cross regardless. This violates priority conditions for unregulated intersection in right-handed traffic systems. As required by law, vehicles 0, 1 would have to stop in front of the intersection and wait for vehicles 2, 3 to cross if those are sufficiently close. This proximity requirement is most definitely fulfilled in the example in Figure 5.6. Just as in the previous section, this behavior could be added to our approach, as formalizations of this rule exist [6], but were not added to this work due its limited scope.

In contrast, the set of all synthesized scenarios for $\mathcal{T}_{urb}$ also includes ones where vehicles 2, 3 traverse the critical section before 0, 1, as our approach enumerates all degrees of freedom in a template, with the traversal order of the critical section in being one of them.

5.3. Discussion

Based on the templates presented in the previous two sections (cf. Sections 5.1 to 5.2) we, in this section, give additional observations on the approach presented in this work.

Formula length scaling In the previous two sections, we only considered the computation times for generating successive solutions from a single template. However, it is also relevant to consider the scaling across different templates of increasing size. Concerning this we were already from the two templates $\mathcal{T}_{hw}, \mathcal{T}_{urb}$ able to see a large increase in the time required to generate each solution for $\mathcal{T}_{urb}$ compared to $\mathcal{T}_{hw}$. This is likely due to the larger number of vehicles, as each additional vehicle can, in the worst case, add all reachable states in a lane network to the predicate graph. Additionally, for vehicles $V_i, V_j \in \mathcal{A}$ in a template, all $\texttt{IsBehind}(V_i, V_j)$ relations with $i < j$ are added to be able to exactly specify an arbitrary order of vehicles on a given lane. As each $\texttt{IsBehind}(V_i, V_j)$ relation is encoded by a new variable (cf. paragraph 3.3.3), the number of variables scales quadratically with the number of vehicles. Since the algorithm by Piterman et al. for satisfying any GR(1) formula scales cubically with the number of states of the given input formula [21], we expect the time for satisfying a GR(1) formula generated from our approach to scale with $O(|\mathcal{A}|^6)$, where $|\mathcal{A}|$ is the size of the set of all vehicles.

A more efficient encoding would be, to instead of encoding each $\texttt{IsBehind}(V_i, V_j)$ relation directly, to encode it by an integer variable $b_i \in [\![0..|\mathcal{A}|]\!]$. Each $\texttt{IsBehind}(V_i, V_j)$ relation could then be encoded as follows.

$$\forall V_i, V_j \in \mathcal{A}, i < j : \texttt{IsBehind}(V_i, V_j) \Leftrightarrow b_i = j$$

Since integer variables are binary encoded, this encoding would have the benefit of encoding each $\texttt{IsBehind}(V_i, V_j)$ relation with a logarithmic number of variables. The overall scaling would then be in $O\left(|\mathcal{A}|^3 \log^3 |\mathcal{A}|\right)$ which is a significant improvement over the previous method. However, we were not able to explore this idea in more detail in this work.

Scene synchronization When parsing a solution to one of the GR(1) formulas presented in this work (cf. Section 3.5), we create a new scenario template from $\mathcal{T}^{sol}$ from it, where the behavior of all vehicles is fully specified. One such template could abstractly be the following.

$$\begin{aligned}\mathcal{T}^{sol} &= (\mathcal{A}, \mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_{n_{sol}}) \\ &\vdots \\ \mathcal{S}_k &= \{\ldots, \texttt{OnLane}(V_i, l_i, s_i), \texttt{OnLane}(V_j, l_j, s_j), \ldots\} \\ &\vdots\end{aligned}$$

Within $\mathcal{S}_k$ there are at least two on-lane relations specifying the behavior of vehicles $V_i, V_j \in \mathcal{A}$ to be on their respective lanes l_i, l_j and within those lanes in the longitudinal intervals s_i, s_j. However, lets consider the case where $\max s_i - \min s_i \gg \max s_j - \min s_j$, so the range of the interval s_i is a lot larger than the one of s_j. Since in the associated optimization problem for this solved template (cf. chapter 4), we require all constraints in $\mathcal{S}_k$ to hold for the entire duration of that scene, we have implicitly created coupled the behavior of vehicles V_i and V_j because in the same time as V_i has to traverse the section s_i on l_i, V_j has to traverse its, a lot smaller, section s_j on l_j.

This rather theoretical example, could however lead to invalid synthesized scenarios, because to satisfy the constraints in $\mathcal{S}_k$ in the above example, either the vehicle V_i would have to traverse its section s_i slower, or vehicle V_j do so faster. Since such a coupling was most likely not intended in the template $\mathcal{T}^{sol}$, we consider it invalid behavior.

Similar optimal solutions Lastly, since we synthesize a concrete scenario via an optimization problem for each solution generated by our approach, but enumerate GR(1) solutions, not synthesized scenarios and see many similar synthesized scenarios in the set of all solutions to a template. This likely due to a minimal reordering of relations in the solution to a GR(1) formula, already classifying as an unseen solution to the template. However, such a reordering can be arbitrarily small. Therefore, when solving the associated optimization problem with this slightly changed solution, we see a similar result. Because the optimal solution for two such slightly different GR(1) solutions might be very similar under the same cost function $J_s(x_s(k), u_s(k), w)$.

To encourage more different solutions and therefore provide more meaningful test cases, one could change the enumeration procedure described in Section 3.4 to not only exclude the exact versions of previously given solutions, but also minor reorderings. Such a more general exclusion, however still needs to make sure all sufficiently different solutions to a GR(1) formula are still enumerated, taking care not to exclude ones with too much interesting behavior.

6. Conclusion & Future Work

We presented an approach for synthesizing concrete scenarios from a simple template specifying behavior over time for a number of vehicles on a given lane network. By representing this problem in LTL we were able to model complex behavior of all vehicles in the template as well as give specificity for defining templates. This allows us to define very tightly constrained scenarios as well as ones with many degrees of freedom within the same system. Additionally, the LTL representation affords for a full coupling between vehicles at all time steps, allowing for complex interactions between them to emerge.

By viewing this approach as an extension of the work by Klischat [18], we are in a similar vain able to generate scenarios efficiently with a high degree of specificity, removing the need to search large databases for desired behavior. Also, once interesting desired behavior has been identified, our approach is able to generate large amounts of scenarios satisfying that template. Therefore, we deem this work is highly applicable to the domain of falsification testing.

6.1. Lessons Learned

Working on this thesis, a primary takeaway was that using different solvers can make a large difference with respect to the concepts one can express in any given formula and the time required to find a solution to that formula. In the process of creating this thesis we worked with a number of different ones and came to the following conclusions.

Spot [34] is a very powerful solver, as it supports the entire LTL language but therefore scales poorly with formula size due to the lower doubly exponential bound proven by Pnueli and Rosner [20]. But Spot was still useful, both as a starting point, and to verify the transformation of an LTL formula to GR(1) for modelling overtaking and solution enumeration. when satisfying a formula, but only supports GR(1) and therefore comes with the limitations discussed throughout this work. Also, documentation for it is lacking. Lastly, `omega`[1] was instrumental for the results in this work, as it supports a richer syntax than `gr1c`, namely comparisons between integer valued variables, which allowed for a much conciser and performant expression of `IsBehind`(V_i, V_j) and `OnLane`(V_i, l, s) rules.

[1]`https://github.com/tulip-control/omega`

6.2. Future Work

This includes some minor improvements, such as parallelizing the BDD conversion our used solver `omega`. There already exists a parallel BDD framework by van Dijk and de Pol [42] which is integrated in `omega`'s underlying BDD library `dd`, but our preliminary testing showed that its use in `omega` still has major problems. Additionally, traffic rules could be integrated to limit our synthesized scenarios to more realistic ones. Work by Maierhofer et al. [6] already does this for a more general set of Temporal Logic, Signal Temporal Logic (STL), but expression of similar rules in GR(1) seems feasible. Also, the ordering of generated solutions in our synthesis process could be improved, as in this work we did not concern ourselves with comparing synthesized solutions. By defining preference relations and integrating them into directly into the solver, search for desired solutions could be performed very efficiently.

Most importantly however, is the before mentioned falsification testing. The approach presented in this work could be used in a closed loop testing framework for an autonomous vehicle motion planning algorithm, by interacting with that algorithm to identify critical scenarios. Then using the templating system detailed in this work, one could employ a sampling technique to generate new scenarios for interactions the motion planner finds particularly challenging. If our approach is able to generate scenarios which are realistic, as validated by e.g. [5], but which the motion planner exhibits unsafe behavior for, we have falsified it. If this is not the case, more sampling could be used or our approach extended to generate more critical scenarios. From the amount of passed tests and the criticality as well as realism of the synthesized scenarios, a confidence score could then be computed, indicating the safety of the motion planning algorithm.

A. Appendix

Parameter	Value	
l	5	(m)
w	2	(m)
a_{min}	-3	(m/s^2)
a_{max}	9.81	(m/s^2)
$\|a_{max}\|$	9.81	(m/s^2)
$v_{max,lat}$	5	(m/s)

Table A.1.: Vehicle Dynamics

Parameter	Value	
$min_{d,lng}$	5	(m)
j_{min}	$-0.15 \cdot 10^{-15}$	(m/s^3)
j_{max}	$0.15 \cdot 10^{-15}$	(m/s^3)
v_{min}	0	(m/s)
v_{max}	20	(m/s)

Table A.2.: Longitudinal Planning Paramters

Parameter	Value	
$j_{max,lat}$	$1 \cdot 10^4$	(m/s^3)
Q_s, Q_d	$\text{diag}([5,1,1])$	
R_s, R_d	0.5	

Table A.3.: Lateral Planning Parameters

```
ENV:;
SYS:0behind1 0behind2 1behind0 1behind2 2behind0 2behind1 v0 v1 v2;
ENVINIT:;
ENVTRANS:;
ENVGOAL:;
SYSINIT:(v0 = 17)
 & (v1 = 4)
 & (v2 = 15);
SYSTRANS: G!((0behind1 & 1behind0))
 & G!((0behind2 & 2behind0))
 & G!((1behind0 & 0behind1))
 & G!((1behind2 & 2behind1))
 & G!((2behind0 & 0behind2))
 & G!((2behind1 & 1behind2))
 & G((v0 = 1) -> ((!((0behind1 | 0behind2)) & X(((((v0 = 1) | (v0 = 5)) | (v0 = 9)) | (v0 = 12)))) | X((((v0 = 1) | (v0 = 5)) | (v0
↪  = 9)))))
 & G((v0 = 10) -> ((!((0behind1 | 0behind2)) & X(((((v0 = 3) | (v0 = 5)) | (v0 = 10)) | (v0 = 14)))) | X((((v0 = 3) | (v0 = 10)) |
↪  (v0 = 14)))))
 & G((v0 = 11) -> X((((v0 = 5) | (v0 = 7)) | (v0 = 11))))
 & G((v0 = 12) -> X((((v0 = 5) | (v0 = 9)) | (v0 = 12))))
 & G((v0 = 13) -> ((!((0behind1 | 0behind2)) & X(((((v0 = 1) | (v0 = 5)) | (v0 = 9)) | (v0 = 13)))) | X((((v0 = 5) | (v0 = 9)) | (v0
↪  = 13)))))
 & G((v0 = 14) -> ((!((0behind1 | 0behind2)) & X((((((v0 = 10) | (v0 = 13)) | (v0 = 14)) | (v0 = 17)) | (v0 = 18)))) | X(((((v0 =
↪  10) | (v0 = 14)) | (v0 = 17)) | (v0 = 18)))))
 & G((v0 = 17) -> X((((v0 = 14) | (v0 = 17)) | (v0 = 18))))
 & G((v0 = 18) -> X((((v0 = 9) | (v0 = 14)) | (v0 = 18))))
 & G((v0 = 2) -> X(((v0 = 2) | (v0 = 7))))
 & G((v0 = 3) -> ((!((0behind1 | 0behind2)) & X(((((v0 = 3) | (v0 = 6)) | (v0 = 10)) | (v0 = 11)))) | X((((v0 = 3) | (v0 = 6)) | (v0
↪  = 10)))))
 & G((v0 = 5) -> X((((((v0 = 1) | (v0 = 5)) | (v0 = 11)) | (v0 = 12)) | (v0 = 13))))
 & G((v0 = 6) -> ((!((0behind1 | 0behind2)) & X((((v0 = 3) | (v0 = 6)) | (v0 = 7)))) | X(((v0 = 3) | (v0 = 6)))))
 & G((v0 = 7) -> X((((v0 = 2) | (v0 = 7)) | (v0 = 11))))
 & G((v0 = 9) -> X(((((v0 = 1) | (v0 = 9)) | (v0 = 12)) | (v0 = 13))))
 & G((v0 = v1) -> (0behind1 | 1behind0))
 & G((v0 = v2) -> (0behind2 | 2behind0))
 & G((v1 = 0) -> ((!((1behind0 | 1behind2)) & X((((v1 = 0) | (v1 = 4)) | (v1 = 8)) | (v1 = 9)))) | X(((v1 = 0) | (v1 = 4)) | (v1 =
↪  8)))))
 & G((v1 = 1) -> ((!((1behind0 | 1behind2)) & X((((v1 = 1) | (v1 = 5)) | (v1 = 9)) | (v1 = 12)))) | X(((v1 = 1) | (v1 = 5)) | (v1
↪  = 9)))))
 & G((v1 = 10) -> ((!((1behind0 | 1behind2)) & X(((((v1 = 3) | (v1 = 4)) | (v1 = 5)) | (v1 = 8)) | (v1 = 10)))) | X((((v1 = 3) |
↪  (v1 = 4)) | (v1 = 8)) | (v1 = 10)))))
 & G((v1 = 11) -> X(((v1 = 5) | (v1 = 7)) | (v1 = 11))))
 & G((v1 = 12) -> X(((v1 = 5) | (v1 = 9)) | (v1 = 12))))
 & G((v1 = 13) -> ((!((1behind0 | 1behind2)) & X((((v1 = 1) | (v1 = 5)) | (v1 = 9)) | (v1 = 13)))) | X(((v1 = 5) | (v1 = 9)) | (v1
↪  = 13)))))
 & G((v1 = 2) -> X(((v1 = 2) | (v1 = 7))))
 & G((v1 = 3) -> ((!((1behind0 | 1behind2)) & X((((v1 = 3) | (v1 = 6)) | (v1 = 10)) | (v1 = 11)))) | X(((v1 = 3) | (v1 = 6)) | (v1
↪  = 10)))))
 & G((v1 = 4) -> X((((v1 = 0) | (v1 = 4)) | (v1 = 8)) | (v1 = 10))))
 & G((v1 = 5) -> X(((((v1 = 1) | (v1 = 5)) | (v1 = 11)) | (v1 = 12)) | (v1 = 13))))
 & G((v1 = 6) -> ((!((1behind0 | 1behind2)) & X(((v1 = 3) | (v1 = 6)) | (v1 = 7)))) | X(((v1 = 3) | (v1 = 6)))))
 & G((v1 = 7) -> X(((v1 = 2) | (v1 = 7)) | (v1 = 11))))
 & G((v1 = 8) -> X((((v1 = 0) | (v1 = 8)) | (v1 = 10)) | (v1 = 13))))
 & G((v1 = 9) -> X((((v1 = 1) | (v1 = 9)) | (v1 = 12)) | (v1 = 13))))
 & G((v1 = v2) -> (1behind2 | 2behind1))
 & G((v2 = 0) -> ((!((2behind0 | 2behind1)) & X(((v2 = 0) | (v2 = 9)) | (v2 = 14)))) | X(((v2 = 0) | (v2 = 14)))))
 & G((v2 = 1) -> ((!((2behind0 | 2behind1)) & X((((v2 = 1) | (v2 = 5)) | (v2 = 9)) | (v2 = 12)))) | X(((v2 = 1) | (v2 = 5)) | (v2
↪  = 9)))))
 & G((v2 = 11) -> X(((v2 = 5) | (v2 = 7)) | (v2 = 11))))
 & G((v2 = 12) -> X(((v2 = 5) | (v2 = 9)) | (v2 = 12))))
 & G((v2 = 13) -> ((!((2behind0 | 2behind1)) & X((((v2 = 1) | (v2 = 5)) | (v2 = 9)) | (v2 = 13)))) | X(((v2 = 5) | (v2 = 9)) | (v2
↪  = 13)))))
 & G((v2 = 14) -> ((!((2behind0 | 2behind1)) & X(((((v2 = 0) | (v2 = 13)) | (v2 = 14)) | (v2 = 15)) | (v2 = 16)))) | X((((v2 = 0)
↪  | (v2 = 14)) | (v2 = 15)) | (v2 = 16)))))
 & G((v2 = 15) -> X((((v2 = 3) | (v2 = 14)) | (v2 = 15)) | (v2 = 16))))
 & G((v2 = 16) -> X((((v2 = 3) | (v2 = 5)) | (v2 = 14)) | (v2 = 16))))
 & G((v2 = 2) -> X(((v2 = 2) | (v2 = 7))))
 & G((v2 = 3) -> ((!((2behind0 | 2behind1)) & X(((((v2 = 3) | (v2 = 6)) | (v2 = 11)) | (v2 = 15)) | (v2 = 16)))) | X((((v2 = 3) |
↪  (v2 = 6)) | (v2 = 15)) | (v2 = 16)))))
 & G((v2 = 5) -> X(((((v2 = 1) | (v2 = 5)) | (v2 = 11)) | (v2 = 12)) | (v2 = 13))))
 & G((v2 = 6) -> ((!((2behind0 | 2behind1)) & X(((v2 = 3) | (v2 = 6)) | (v2 = 7)))) | X(((v2 = 3) | (v2 = 6)))))
 & G((v2 = 7) -> X(((v2 = 2) | (v2 = 7)) | (v2 = 11))))
 & G((v2 = 9) -> X((((v2 = 1) | (v2 = 9)) | (v2 = 12)) | (v2 = 13))))
 & G((((v0 != v1) & (v1 = X(v1))) & X((v0 = v1))) -> X(0behind1))
 & G((((v0 != v2) & (v2 = X(v2))) & X((v0 = v2))) -> X(0behind2))
 & G((((v0 = 1) & (v1 = 1)) & (v2 = 1)) -> X((((v0 = 1) & (v1 = 1)) & (v2 = 1))))
 & G((((v0 = v1) & 0behind1) & X((v0 = v1))) -> X(0behind1))
 & G((((v0 = v1) & 1behind0) & X((v0 = v1))) -> X(1behind0))
 & G((((v0 = v2) & 0behind2) & X((v0 = v2))) -> X(0behind2))
 & G((((v0 = v2) & 2behind0) & X((v0 = v2))) -> X(2behind0))
 & G((((v1 != v0) & (v0 = X(v0))) & X((v1 = v0))) -> X(1behind0))
 & G((((v1 != v2) & (v2 = X(v2))) & X((v1 = v2))) -> X(1behind2))
 & G((((v1 = v2) & 1behind2) & X((v1 = v2))) -> X(1behind2))
 & G((((v1 = v2) & 2behind1) & X((v1 = v2))) -> X(2behind1))
 & G((((v2 != v0) & (v0 = X(v0))) & X((v2 = v0))) -> X(2behind0))
 & G((((v2 != v1) & (v1 = X(v1))) & X((v2 = v1))) -> X(2behind1));
SYSGOAL:G(F(((v0 = 1) & (v1 = 1)) & (v2 = 1)));
```

Listing A.1.: GR(1) formula for the highway scenario derived from $\mathcal{T}_{hw}$

```
ENV:;
SYS:0behind1 0behind2 0behind3 1behind0 1behind2 1behind3 2behind0 2behind1 2behind3 3behind0 3behind1 3behind2 v0 v1 v2 v3;
ENVINIT:;
ENVTRANS:;
ENVGOAL:;
SYSINIT:0behind1
 & 2behind3
 & (v0 = 1)
 & (v1 = 1)
 & (v2 = 6)
 & (v3 = 6);
SYSTRANS: G!((0behind1 & 1behind0))
 & G!((0behind2 & 2behind0))
 & G!((0behind3 & 3behind0))
 & G!((1behind0 & 0behind1))
 & G!((1behind2 & 2behind1))
 & G!((1behind3 & 3behind1))
 & G!((2behind0 & 0behind2))
 & G!((2behind1 & 1behind2))
 & G!((2behind3 & 3behind2))
 & G!((3behind0 & 0behind3))
 & G!((3behind1 & 1behind3))
 & G!((3behind2 & 2behind3))
 & G((v0 = 0) -> X((v0 = 0)))
 & G((v0 = 1) -> ((((v0 != 4) & !(((0behind1 | 0behind2) | 0behind3))) & X(((v0 = 1) | (v0 = 4)))) | X((v0 = 1))))
 & G((v0 = 2) -> ((!(((0behind1 | 0behind2) | 0behind3)) & X(((v0 = 0) | (v0 = 2)))) | X((v0 = 2))))
 & G((v0 = 3) -> ((!(((0behind1 | 0behind2) | 0behind3)) & X(((v0 = 2) | (v0 = 3)))) | X((v0 = 3))))
 & G((v0 = 4) -> ((!(((0behind1 | 0behind2) | 0behind3)) & X(((v0 = 3) | (v0 = 4)))) | X((v0 = 4))))
 & G((v0 = v1) -> (0behind1 | 1behind0))
 & G((v0 = v2) -> (0behind2 | 2behind0))
 & G((v0 = v3) -> (0behind3 | 3behind0))
 & G((v1 = 0) -> X((v1 = 0)))
 & G((v1 = 1) -> ((((v1 != 4) & !(((1behind0 | 1behind2) | 1behind3))) & X(((v1 = 1) | (v1 = 4)))) | X((v1 = 1))))
 & G((v1 = 2) -> ((!(((1behind0 | 1behind2) | 1behind3)) & X(((v1 = 0) | (v1 = 2)))) | X((v1 = 2))))
 & G((v1 = 3) -> ((!(((1behind0 | 1behind2) | 1behind3)) & X(((v1 = 2) | (v1 = 3)))) | X((v1 = 3))))
 & G((v1 = 4) -> ((!(((1behind0 | 1behind2) | 1behind3)) & X(((v1 = 3) | (v1 = 4)))) | X((v1 = 4))))
 & G((v1 = v2) -> (1behind2 | 2behind1))
 & G((v1 = v3) -> (1behind3 | 3behind1))
 & G((v2 = 5) -> ((((v2 != 7) & !(((2behind0 | 2behind1) | 2behind3))) & X(((v2 = 5) | (v2 = 7)))) | X((v2 = 5))))
 & G((v2 = 6) -> ((!(((2behind0 | 2behind1) | 2behind3)) & X(((v2 = 5) | (v2 = 6)))) | X((v2 = 6))))
 & G((v2 = 7) -> ((!(((2behind0 | 2behind1) | 2behind3)) & X(((v2 = 7) | (v2 = 8)))) | X((v2 = 7))))
 & G((v2 = 8) -> ((!(((2behind0 | 2behind1) | 2behind3)) & X(((v2 = 8) | (v2 = 9)))) | X((v2 = 8))))
 & G((v2 = 9) -> X((v2 = 9)))
 & G((v2 = v3) -> (2behind3 | 3behind2))
 & G((v3 = 5) -> ((((v3 != 7) & !(((3behind0 | 3behind1) | 3behind2))) & X(((v3 = 5) | (v3 = 7)))) | X((v3 = 5))))
 & G((v3 = 6) -> ((!(((3behind0 | 3behind1) | 3behind2)) & X(((v3 = 5) | (v3 = 6)))) | X((v3 = 6))))
 & G((v3 = 7) -> ((!(((3behind0 | 3behind1) | 3behind2)) & X(((v3 = 7) | (v3 = 8)))) | X((v3 = 7))))
 & G((v3 = 8) -> ((!(((3behind0 | 3behind1) | 3behind2)) & X(((v3 = 8) | (v3 = 9)))) | X((v3 = 8))))
 & G((v3 = 9) -> X((v3 = 9)))
 & G((((v0 != v1) & (v1 = X(v1))) & X((v0 = v1))) -> X(0behind1))
 & G((((v0 != v2) & (v2 = X(v2))) & X((v0 = v2))) -> X(0behind2))
 & G((((v0 != v3) & (v3 = X(v3))) & X((v0 = v3))) -> X(0behind3))
 & G((((v0 = v1) & 0behind1) & X((v0 = v1))) -> X(0behind1))
 & G((((v0 = v1) & 1behind0) & X((v0 = v1))) -> X(1behind0))
 & G((((v0 = v2) & 0behind2) & X((v0 = v2))) -> X(0behind2))
 & G((((v0 = v2) & 2behind0) & X((v0 = v2))) -> X(2behind0))
 & G((((v0 = v3) & 0behind3) & X((v0 = v3))) -> X(0behind3))
 & G((((v0 = v3) & 3behind0) & X((v0 = v3))) -> X(3behind0))
 & G((((v1 != v0) & (v0 = X(v0))) & X((v1 = v0))) -> X(1behind0))
 & G((((v1 != v2) & (v2 = X(v2))) & X((v1 = v2))) -> X(1behind2))
 & G((((v1 != v3) & (v3 = X(v3))) & X((v1 = v3))) -> X(1behind3))
 & G((((v1 = v2) & 1behind2) & X((v1 = v2))) -> X(1behind2))
 & G((((v1 = v2) & 2behind1) & X((v1 = v2))) -> X(2behind1))
 & G((((v1 = v3) & 1behind3) & X((v1 = v3))) -> X(1behind3))
 & G((((v1 = v3) & 3behind1) & X((v1 = v3))) -> X(3behind1))
 & G((((v2 != v0) & (v0 = X(v0))) & X((v2 = v0))) -> X(2behind0))
 & G((((v2 != v1) & (v1 = X(v1))) & X((v2 = v1))) -> X(2behind1))
 & G((((v2 != v3) & (v3 = X(v3))) & X((v2 = v3))) -> X(2behind3))
 & G((((v2 = v3) & 2behind3) & X((v2 = v3))) -> X(2behind3))
 & G((((v2 = v3) & 3behind2) & X((v2 = v3))) -> X(3behind2))
 & G((((v3 != v0) & (v0 = X(v0))) & X((v3 = v0))) -> X(3behind0))
 & G((((v3 != v1) & (v1 = X(v1))) & X((v3 = v1))) -> X(3behind1))
 & G((((v3 != v2) & (v2 = X(v2))) & X((v3 = v2))) -> X(3behind2))
 & G((((((0behind1 & 2behind3) & (v1 = 0)) & (v3 = 9)) & (v2 = 9)) & (v0 = 0)) -> X((((((0behind1 & 2behind3) & (v1 = 0)) & (v3 =
↪  9)) & (v2 = 9)) & (v0 = 0))))
 & G((((((v3 = 7) & !((((v0 = 4) | (v2 = 7)) | (v1 = 4)))) | ((v0 = 4) & !((((v3 = 7) | (v2 = 7)) | (v1 = 4))))) | ((v2 = 7) &
↪  !((((v3 = 7) | (v0 = 4)) | (v1 = 4))))) | ((v1 = 4) & !((((v3 = 7) | (v0 = 4)) | (v2 = 7))))) | (((!((v3 = 7)) & !((v0 = 4))) &
↪  !((v2 = 7))) & !((v1 = 4))));
SYSGOAL: G(F(((((0behind1 & 2behind3) & (v1 = 0)) & (v3 = 9)) & (v2 = 9)) & (v0 = 0)));
```

Listing A.2.: GR(1) formula for the urban scenario derived from $\mathcal{T}_{urb}$

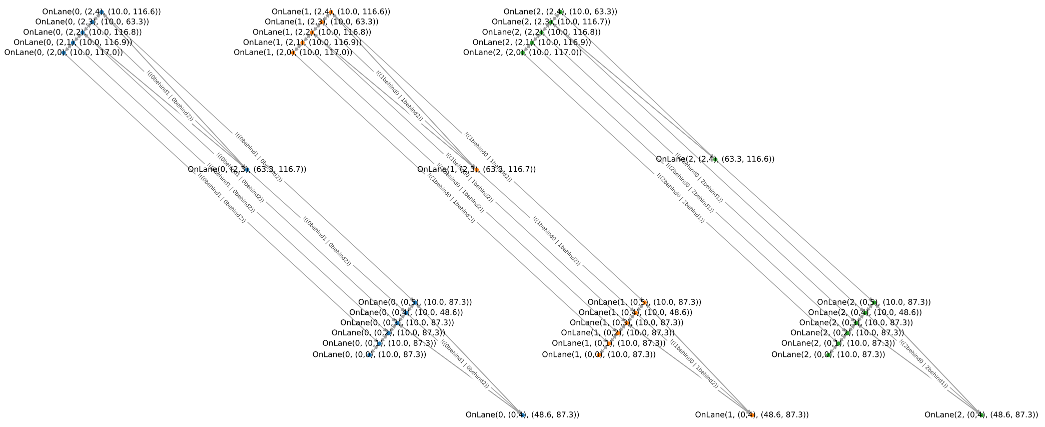

OnLane(0, (2,4), (10.0, 116.6))
OnLane(0, (2,3), (10.0, 63.3))
OnLane(0, (2,2), (10.0, 116.8))
OnLane(0, (2,1), (10.0, 116.9))
OnLane(0, (2,0), (10.0, 117.0))
OnLane(1, (2,4), (10.0, 116.6))
OnLane(1, (2,3), (10.0, 63.3))
OnLane(1, (2,2), (10.0, 116.8))
OnLane(1, (2,1), (10.0, 116.9))
OnLane(1, (2,0), (10.0, 117.0))
OnLane(2, (2,4), (10.0, 63.3))
OnLane(2, (2,3), (10.0, 116.7))
OnLane(2, (2,2), (10.0, 116.8))
OnLane(2, (2,1), (10.0, 116.9))
OnLane(2, (2,0), (10.0, 117.0))
OnLane(0, (2,3), (63.3, 116.7))
OnLane(1, (2,3), (63.3, 116.7))
OnLane(2, (2,4), (63.3, 116.6))
OnLane(0, (0,5), (10.0, 87.3))
OnLane(0, (0,4), (10.0, 48.6))
OnLane(0, (0,3), (10.0, 87.3))
OnLane(0, (0,2), (10.0, 87.3))
OnLane(0, (0,1), (10.0, 87.3))
OnLane(0, (0,0), (10.0, 87.3))
OnLane(1, (0,5), (10.0, 87.3))
OnLane(1, (0,4), (10.0, 48.6))
OnLane(1, (0,3), (10.0, 87.3))
OnLane(1, (0,2), (10.0, 87.3))
OnLane(1, (0,1), (10.0, 87.3))
OnLane(1, (0,0), (10.0, 87.3))
OnLane(2, (0,5), (10.0, 87.3))
OnLane(2, (0,4), (10.0, 48.6))
OnLane(2, (0,3), (10.0, 87.3))
OnLane(2, (0,2), (10.0, 87.3))
OnLane(2, (0,1), (10.0, 87.3))
OnLane(2, (0,0), (10.0, 87.3))
OnLane(0, (0,4), (48.6, 87.3))
OnLane(1, (0,4), (48.6, 87.3))
OnLane(2, (0,4), (48.6, 87.3))

Figure A.1.: Predicate Graph for the highway scenario template $\mathcal{T}_{hw}$

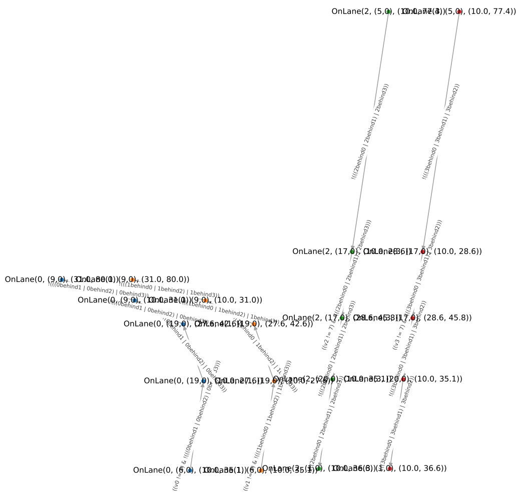

OnLane(2, (5,0), (10.0, 77.4))
OnLane(3, (5,0), (10.0, 77.4))
OnLane(2, (17,0), (10.0, 28.6))
OnLane(3, (17,0), (10.0, 28.6))
OnLane(0, (9,0), (31.0, 80.0))
OnLane(1, (9,0), (31.0, 80.0))
OnLane(0, (9,0), (10.0, 31.0))
OnLane(1, (9,0), (10.0, 31.0))
OnLane(0, (19,0), (27.6, 42.6))
OnLane(1, (19,0), (27.6, 42.6))
OnLane(2, (17,0), (28.6, 45.8))
OnLane(3, (17,0), (28.6, 45.8))
OnLane(0, (19,0), (10.0, 27.6))
OnLane(1, (19,0), (10.0, 27.6))
OnLane(2, (20,0), (10.0, 35.1))
OnLane(3, (20,0), (10.0, 35.1))
OnLane(0, (6,0), (10.0, 35.1))
OnLane(1, (6,0), (10.0, 35.1))
OnLane(2, (3,0), (10.0, 36.6))
OnLane(3, (3,0), (10.0, 36.6))

Figure A.2.: Predicate Graph for the urban scenario template $\mathcal{T}_{urb}$

List of Figures

2.1. Lane network showing lanes and lanelets. From [18] 5
2.2. Lane Network for example 1. Brown shows start- and end lanes for both vehicles. The critical section between intersecting lanes is given in red (more in Section 3.2.1). 10

3.1. High level overview the pipeline synthesizing all satisfying CommonRoad scenarios from a template 11
3.2. Predicate graph for example 1. 15
3.3. Node ID encoding for all Vehicles V with `v0` and `v1` representing ν in Equation 3.4 18
3.4. Deterministic Büchi automata for Equation 3.13 22
3.5. Edge based deterministic Büchi automaton for excluding of a previously seen word $w^{(i)}$ 25
3.6. Accepting Büchi automaton for example 1. The split of $w = w_p w_c w_c \dots$ is given by the respective boxes. 28

5.1. Lane network with color coded $\texttt{OnLane}(V_i, l, s)$ specifications for each vehicle from the respective template 34
5.2. Computation times for the highway scenario in seconds for 100 iterations. 35
5.3. Representative synthesized solution for the highway template $\mathcal{T}_{hw}$ at six different times $0 \leq t \leq t_h$. Light blue shows future positions of a vehicle. 37
5.4. Lane network with color coded $\texttt{OnLane}(V_i, l, s)$ specifications for each vehicle from the respective template 39
5.5. Computation times for the urban scenario in seconds for 100 iterations. . 40
5.6. Representative solution for the urban template $\mathcal{T}_{urb}$ at six different times t. Light blue shows future positions of a vehicle. 41

A.1. Predicate Graph for the highway scenario template $\mathcal{T}_{hw}$ 50
A.2. Predicate Graph for the urban scenario template $\mathcal{T}_{urb}$ 50

List of Tables

2.1. Available relations with a definition in natural language 10

5.1. Highway Scenario parameters 33
5.2. Urban Scenario parameters 38

A.1. Vehicle Dynamics 47
A.2. Longitudinal Planning Paramters 47
A.3. Lateral Planning Parameters 47

Bibliography

[1] B. W. Smith. *Human error as a cause of vehicle crashes*. en. Dec. 2013.

[2] N. Kalra and S. M. Paddock. "Driving to Safety: How Many Miles of Driving Would It Take to Demonstrate Autonomous Vehicle Reliability?" en. In: (Dec. 2016), p. 15.

[3] P. Koopman and M. Wagner. "Challenges in Autonomous Vehicle Testing and Validation." In: *SAE International Journal of Transportation Safety* 4.1 (2016). Publisher: SAE International, pp. 15–24. ISSN: 2327-5626.

[4] P. Koopman and M. Wagner. "Autonomous Vehicle Safety: An Interdisciplinary Challenge." In: *IEEE Intelligent Transportation Systems Magazine* 9.1 (2017). Conference Name: IEEE Intelligent Transportation Systems Magazine, pp. 90–96. ISSN: 1941-1197.

[5] C. Pek, V. Rusinov, S. Manzinger, M. C. Uste, and M. Althoff. "CommonRoad Drivability Checker: Simplifying the Development and Validation of Motion Planning Algorithms." en. In: *2020 IEEE Intelligent Vehicles Symposium (IV)*. Las Vegas, NV, USA: IEEE, Oct. 2020, pp. 1013–1020. ISBN: 978-1-72816-673-5.

[6] S. Maierhofer, A.-K. Rettinger, E. C. Mayer, and M. Althoff. "Formalization of Interstate Trafc Rules in Temporal Logic." en. In: (), p. 8.

[7] E. C. Mayer. *Formalization of Traffic Rules using Temporal Logic*. en. Tech. rep., p. 46.

[8] D. J. Fremont, T. Dreossi, S. Ghosh, X. Yue, A. L. Sangiovanni-Vincentelli, and S. A. Seshia. "Scenic: a language for scenario specification and scene generation." en. In: *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation - PLDI 2019*. Phoenix, AZ, USA: ACM Press, 2019, pp. 63–78. ISBN: 978-1-4503-6712-7.

[9] R. Majumdar, A. Mathur, M. Pirron, L. Stegner, and D. Zufferey. "Paracosm: A Language and Tool for Testing Autonomous Driving Systems." en. In: *arXiv:1902.01084 [cs]* (Feb. 2019). arXiv: 1902.01084.

[10] M. O'Kelly, A. Sinha, H. Namkoong, J. Duchi, and R. Tedrake. "Scalable End-to-End Autonomous Vehicle Testing via Rare-event Simulation." In: *arXiv:1811.00145 [cs, stat]* (Jan. 2019). arXiv: 1811.00145.

[11] M. Klischat and M. Althoff. "Generating Critical Test Scenarios for Automated Vehicles with Evolutionary Algorithms." en. In: *2019 IEEE Intelligent Vehicles Symposium (IV)*. Paris, France: IEEE, June 2019, pp. 2352–2358. ISBN: 978-1-72810-560-4.

[12] G. Bagschik, T. Menzel, and M. Maurer. "Ontology based Scene Creation for the Development of Automated Vehicles." en. In: *arXiv:1704.01006 [cs]* (Apr. 2018). arXiv: 1704.01006.

[13] D. Zhao, X. Huang, H. Peng, H. Lam, and D. J. LeBlanc. "Accelerated Evaluation of Automated Vehicles in Car-Following Maneuvers." en. In: *IEEE Trans. Intell. Transport. Syst.* 19.3 (Mar. 2018), pp. 733–744. ISSN: 1524-9050, 1558-0016.

[14] M. Althoff and J. M. Dolan. "Online Verification of Automated Road Vehicles Using Reachability Analysis." en. In: *IEEE Trans. Robot.* 30.4 (Aug. 2014), pp. 903–918. ISSN: 1552-3098, 1941-0468.

[15] S. Shalev-Shwartz, S. Shammah, and A. Shashua. "On a Formal Model of Safe and Scalable Self-driving Cars." In: *arXiv:1708.06374 [cs, stat]* (Oct. 2018). arXiv: 1708.06374.

[16] S. A. Seshia, D. Sadigh, and S. S. Sastry. "Formal methods for semi-autonomous driving." en. In: *Proceedings of the 52nd Annual Design Automation Conference*. San Francisco California: ACM, June 2015, pp. 1–5. ISBN: 978-1-4503-3520-1.

[17] E. Rocklage, H. Kraft, A. Karatas, and J. Seewig. "Automated scenario generation for regression testing of autonomous vehicles." In: *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*. ISSN: 2153-0017. Oct. 2017, pp. 476–483.

[18] M. Klischat and M. Althoff. "Synthesizing Trafc Scenarios from Formal Specications for Testing Automated Vehicles." en. In: *Proc. of the IEEE Intelligent Vehicles Symposium* (2020), p. 8.

[19] C. Miller, C. Pek, and M. Althoff. "Efficient Mixed-Integer Programming for Longitudinal and Lateral Motion Planning of Autonomous Vehicles." en. In: *2018 IEEE Intelligent Vehicles Symposium (IV)*. Changshu: IEEE, June 2018, pp. 1954–1961. ISBN: 978-1-5386-4452-2.

[20] A. Pnueli and R. Rosner. "On the synthesis of a reactive module." en. In: *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages - POPL '89*. Austin, Texas, United States: ACM Press, 1989, pp. 179–190. ISBN: 978-0-89791-294-5.

[21] N. Piterman, A. Pnueli, and Y. Saar. "Synthesis of Reactive(1) Designs." en. In: *Verification, Model Checking, and Abstract Interpretation*. Ed. by E. A. Emerson and K. S. Namjoshi. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2006, pp. 364–380. ISBN: 978-3-540-31622-0.

[22] I. Filippidis, S. Dathathri, S. C. Livingston, N. Ozay, and R. M. Murray. "Control design for hybrid systems with TuLiP: The Temporal Logic Planning toolbox." In: *2016 IEEE Conference on Control Applications (CCA)*. Sept. 2016, pp. 1030–1041.

[23] M. Althoff and R. Muller. "Automatic Synthesis of Human Motion from Maneuver Primitives." en.

[24] E. M. Wolff, U. Topcu, and R. M. Murray. “Efficient reactive controller synthesis for a fragment of linear temporal logic.” en. In: *2013 IEEE International Conference on Robotics and Automation*. Karlsruhe, Germany: IEEE, May 2013, pp. 5033–5040. ISBN: 978-1-4673-5643-5 978-1-4673-5641-1.

[25] E. M. Wolff, U. Topcu, and R. M. Murray. “Optimization-based trajectory generation with linear temporal logic specifications.” en. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. Hong Kong, China: IEEE, May 2014, pp. 5319–5325. ISBN: 978-1-4799-3685-4.

[26] S. Karaman, R. G. Sanfelice, and E. Frazzoli. “Optimal control of Mixed Logical Dynamical systems with Linear Temporal Logic specifications.” en. In: *2008 47th IEEE Conference on Decision and Control*. Cancun, Mexico: IEEE, 2008, pp. 2117–2122. ISBN: 978-1-4244-3123-6.

[27] Y. Kwon and G. Agha. “LTLC: Linear Temporal Logic for Control.” en. In: *Hybrid Systems: Computation and Control*. Ed. by M. Egerstedt and B. Mishra. Vol. 4981. ISSN: 0302-9743, 1611-3349 Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 316–329. ISBN: 978-3-540-78928-4 978-3-540-78929-1.

[28] K. S. Kuhlmann. “LTL specifications for Highway Lane Changing Maneuvers of Highly Automated Vehicles.” en. PhD thesis. 2014.

[29] S. Sadraddini and C. Belta. “Robust Temporal Logic Model Predictive Control.” en. In: *2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton)* (Sept. 2015). arXiv: 1511.00347, pp. 772–779.

[30] M. Althoff, M. Koschi, and S. Manzinger. “CommonRoad: Composable benchmarks for motion planning on roads.” en. In: *2017 IEEE Intelligent Vehicles Symposium (IV)*. Los Angeles, CA, USA: IEEE, June 2017, pp. 719–726. ISBN: 978-1-5090-4804-5.

[31] P. Bender, J. Ziegler, and C. Stiller. “Lanelets: Efficient map representation for autonomous driving.” en. In: *2014 IEEE Intelligent Vehicles Symposium Proceedings*. MI, USA: IEEE, June 2014, pp. 420–425. ISBN: 978-1-4799-3638-0.

[32] C. Pek and M. Althoff. “Computationally Efficient Fail-safe Trajectory Planning for Self-driving Vehicles Using Convex Optimization.” en. In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. Maui, HI: IEEE, Nov. 2018, pp. 1447–1454. ISBN: 978-1-72810-321-1 978-1-72810-323-5.

[33] J. Esparza, J. Ketínský, and S. Sickert. “A Unified Translation of Linear Temporal Logic to omega-Automata.” In: *J. ACM* 67.6 (Oct. 2020), 33:1–33:61. ISSN: 0004-5411.

[34] A. Duret-Lutz, A. Lewkowicz, A. Fauchille, and T. Michaud. “Spot 2.0 a framework for LTL and -automata manipulation.” en. In: (), p. 7.

[35] W. Li, S. Kan, and Z. Huang. “A Better Translation From LTL to Transition-Based Generalized Büchi Automata.” In: *IEEE Access* 5 (2017). Conference Name: IEEE Access, pp. 27081–27090. ISSN: 2169-3536.

[36] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. “Patterns in property specifications for finite-state verification.” In: *Proceedings of the 1999 International Conference on Software Engineering (IEEE Cat. No.99CB37002)*. ISSN: 0270-5257. May 1999, pp. 411–420.

[37] S. Maoz and J. O. Ringert. “GR(1) synthesis for LTL specification patterns.” en. In: *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2015*. Bergamo, Italy: ACM Press, 2015, pp. 96–106. ISBN: 978-1-4503-3675-8.

[38] D. Kroening and O. Strichman. *Decision procedures: an algorithmic point of view*. en. Texts in theoretical computer science. OCLC: 244022161. Berlin: Springer, 2008. ISBN: 978-3-540-74104-6 978-3-540-74105-3.

[39] S. Jacobs. *ReactiveSynthesis*. English. Nancy, France, 2013.

[40] R. E. Bryant. “Graph-Based Algorithms for Boolean Function Manipulation.” In: *IEEE Transactions on Computers* C-35.8 (Aug. 1986). Conference Name: IEEE Transactions on Computers, pp. 677–691. ISSN: 1557-9956.

[41] Y. Kesten, N. Piterman, and A. Pnueli. “Bridging the gap between fair simulation and trace inclusion.” en. In: *Information and Computation* 200.1 (July 2005), pp. 35–61. ISSN: 0890-5401.

[42] T. van Dijk and J. van de Pol. “Sylvan: multi-core framework for decision diagrams.” en. In: *Int J Softw Tools Technol Transfer* 19.6 (Nov. 2017), pp. 675–696. ISSN: 1433-2779, 1433-2787.