A Consistency Analysis Method for Traffic Sequence Charts

Jan Steffen Becker
DLR e.V. – Institute of Systems Engineering for Future Mobility – Oldenburg, Germany
jan.becker@dlr.de

ABSTRACT

The trend in the development of highly automated vehicles goes towards scenario-based methods. Traffic Sequence Charts are a visual but yet formal language for describing scenario-based requirements on highly automated vehicles. This work presents an approach for finding inconsistencies (conflicts) in a set of scenario-based requirements formalized with Traffic Sequence Charts. The proposed method utilizes satisfiability modulo theories solving on two-sided approximations of possible vehicle behavior. This ensures that found inconsistencies are not caused by approximations, but also occur when applying exact methods. Applicability and scalability of the analysis technique is evaluated in a case study.

keywords Scenario-based Development, Traffic Sequence Charts, Consistency, ISO 26262, Satisfiability Modulo Theories, Bounded Model Checking

1 INTRODUCTION

The development of highly automated vehicles (HAVs) has to deal with highly complex environments. In SAE Level 3 (SAE J3016_202104, 2021) and above, the automated vehicle is expected to drive safely within its design domain. The challenge of ensuring provably safe behavior starts already in the early phases of the development process. Potentially critical driving situations have to be identified, and strategies need to be defined that lower risks to an acceptable minimum (Kramer et al., 2020; Neurohr et al., 2021). The number and complexity of different situations is enormous as it includes both static and dynamic aspects, such as road and weather conditions, different traffic participants, and driving maneuvers (Koopman and Fratrik, 2019). Therefore, the trend goes towards scenario-based approaches that aim at clustering this unmanageable set of different situations and scenarios into a manageable set of scenario classes (Kalisvaart et al., 2020; Menzel et al., 2018; Riedmaier et al., 2020). During the whole development process, scenario descriptions of different granularity are used. For example, identified critical scenario classes are characterized by textual scenario descriptions, so-called *functional scenarios* during the concept phase, while testing requires *concrete scenarios* that allow an exact reproduction in a (virtual or physical) test bed (Menzel et al., 2018).

This rigorous scenario-driven approach calls for specifying also the behavioral requirements on the system, i.e., how the HAV shall behave in certain situations, in a scenario-based manner. Traffic Sequence Charts (TSCs) (Damm et al., 2018a) are a graphical formalism that enables exactly this. In a TSC, traffic situations are graphically depicted and assembled to formalized versions of functional scenarios, called *abstract scenarios* (Neurohr et al., 2021). With the same technique, TSCs also allow to formalize requirements.

As a research contribution to the field of scenario-based development of HAV, this paper reports on the author's doctoral thesis entitled "A Consistency Analysis Method for Traffic Sequence Charts" that is currently under work. In short, the thesis aims on the development and experimental evaluation of an automated consistency analysis approach for TSCs.

The remainder of the paper is structured as follows. Section 2 presents the research problem and objectives of the thesis. Section 3 presents background work, followed by an introduction to TSCs in Section 4. Section 4 also introduces the running example. In Section 5, key ideas of the methodology and expected results are named. The proposed consistency analysis method is then described in more detail in Section 6. Some experimental evaluation results are presented in Section 6.3. Finally, Section 7 concludes the paper with a short discussion and outlook to future work.

2 RESEARCH PROBLEM AND OBJECTIVES

Good systems engineering practice mandates that requirements shall comply to the three big Cs *correctness*, *completeness*, and *consistency* (Zowghi and Gervasi, 2003; Kamalrudin and Sidek, 2015). Correctness is usually defined as the combination of completeness and consistency (Zowghi and Gervasi, 2003). Normative standards – such as ISO 26262 (ISO 26262, 2018) in the automotive domain – state completeness and consistency as mandatory properties of any requirements specification. Completeness is quite hard to grasp, and only be defined relatively, for example with respect to the outcome of a preceding hazard analysis (Leveson, 2000). Consistency means that a set of requirements is free of contradictions. This includes both contradictions within itself and with respect to other requirements (ISO 26262, 2018). Detecting inconsistencies early in the development process avoids implementation faults and saves time and money (Feiler et al., 2010).

A lot of different approaches (e.g. (Jaffe et al., 1991; Ellen et al., 2014; Becker, 2018; Filipovikj et al., 2017; Aichernig et al., 2015; Post et al., 2011)) exist that formally define consistency, but none of it is directly applicable to TSCs. Therefore, the presented PhD thesis addresses two major research questions

1. How can consistency for traffic sequence charts be defined formally?
2. How can a consistency check that uses these definitions of consistency be automated?

The research objectives of the thesis can be summarized as follows.

1. Develop adequate consistency notions for TSCs
2. Define a decision procedure for the consistency notions
3. Formally prove correctness of the encoding
4. Prototypically implement the approach
5. Evaluate effectiveness and scalability of the approach

The experiments for evaluation are further described in Section 6.3.

3 STATE OF PRACTICE

Consistency analysis for TSCs bridges research from different fields.

In the automotive industry, scenario-based development (Kalisvaart et al., 2020; Menzel et al., 2018; Riedmaier et al., 2020) is the answer to the growing complexity of HAV. Menzel et. al. (Menzel et al., 2018) identified a wide range of applications of scenario descriptions in a traditional development process. As already mentioned in the introduction, scenarios are used in different abstraction levels. Traffic sequence charts (Damm et al., 2017; Damm et al., 2018a) have been developed to formally describe scenarios with a high abstraction level. TSCs have been used in a wide range of applications, e.g. scenario mining (Damm et al., 2019), test design (Damm et al., 2019) and runtime monitoring (Grundt et al., 2022). In contrast, OpenSCENARIO[1] XML has been developed as an industrial standard for describing automotive scenarios on a lower abstraction level. OpenSCENARIO mainly targets simulation of scenarios and is supported by industrial-grade driving simulators. There is also an approach to generate OpenSCENARIO XML specifications from TSCs (Becker et al., 2022). Another ASAM standard, OpenSCENARIO DSL[2], allows to specify abstract scenarios, but there is no complete tool support for these new features, yet.

In the ENABLE-S3 research project (Leitner et al., 2019), a reference process for scenario-based development has been defined. Here, consistency analysis is one step in the *requirement and scenario elicitation* activity, but is not discussed further. However, automated consistency analysis has already been developed for other types of formal requirement specifications. For example, the software cost reduction (SCR) toolset (Jaffe et al., 1991; Heitmeyer et al., 1996) provides an automated consistency analysis for automata-like specifications. More recent work (Ellen et al., 2014; Becker, 2018; Filipovikj et al., 2017; Aichernig et al., 2015; Post et al., 2011) considers pattern-based requirements.

In most of the aforementioned work, satisfiability modulo theories (SMT) solving in connection with bounded model checking (BMC) is used to find inconsistencies. All these works consider discrete transition systems. Consistency is then reduced to existence of system runs that satisfy the requirements. SMT is well suited for consistency analysis because it is capable of both synthetizing system runs (which are a witness for consistency) and proving non-existence of satisfying runs (which proves inconsistency). In

[1] https://www.asam.net/standards/detail/openscenario-xml/, accessed on 2024-02-28. OpenSCENARIO XML was formerly known as OpenSCENARIO 1.x

[2] https://www.asam.net/standards/detail/openscenario-dsl/, accessed on 2024-02-28. OpenSCENARIO DSL was formerly known as OpenSCENARIO 2.0

the authors earlier work (Becker, 2020), this idea is already applied to TSCs. Instead of discrete transition systems, the translation process to SMT is based on earlier work about duration calculus, which has some similarities to TSCs. However, it over-approximates possible vehicle behavior which means that correctness of witness trajectories cannot be guaranteed. The aforementioned generation of OpenSCENARIO from TSCs (Becker et al., 2022) is based on the same method, but extends it with a simple vehicle dynamics model. As a consequence, generated concrete scenarios are correct, but not all possible scenarios can be found. A similar technique is also applied by Eggers et al. (Eggers et al., 2018) to scenario specifications that are similar to existential TSCs, but more restrictive. To the author's knowledge, these are the only works that tackle consistency of traffic scenario specifications.

The present work builds upon the existing work on TSC consistency analysis (Becker, 2020) and extends it with the TSC instantiation technique (Becker et al., 2022) that has already been used for OpenSCENARIO generation. The latter uses a conservative linear approximation of vehicle dynamics in conjunction with Bézier spline trajectory planning. Alternatives would be statespace exploration techniques for hybrid systems (Henzinger et al., 1997; Frehse, 2005; Frehse et al., 2011; Chen and Sankaranarayanan, 2016; Eggers et al., 2011). However, these numerical approaches focus on ordinary differential equations without a known closed form solution and unfortunately scale badly with high dimensional state spaces. Plaku et al. (Plaku et al., 2007; Plaku et al., 2013) overcome these limitations by combining searches on a discrete state spaces to guide exact simulations. The approach depends on a discrete approximation of the reachable state space which needs to be provided manually. Therefore, it is not suited for an automated consistency analysis.

4 TRAFFIC SEQUENCE CHARTS

TSCs have been developed with the goal of creating a description language that connects the intuitiveness of depicting traffic situations graphically with well-defined semantics. The core concept of a TSC is a so-called *invariant node* that graphically depicts a traffic situation (or, as we will see later, a combination of situations). Here, symbols stand for objects, and their placement indicates spatial relations. Invariant nodes are assembled to *basic charts* by combining them using the operators *sequence*, *choice*, and *concurrency* depicted in Figure 2. The operators can be arbitrarily nested. Furthermore, it is possible to add timing annotations (Figure 2d) to basic charts—time pins (🔎) with the same label synchronize time points, and hour glasses (⧖, ⧗) express duration constraints. The full TSC formalism (Damm et al., 2018b) also allows negation of basic charts, which has been omitted for this paper. Experience shows that negation is seldom needed in practice because it allows almost any (also unintended) behavior. So, this is only a minor limitation.

A *requirement TSC* resembles a typical specification pattern, sometimes called *response property*[3] (Dwyer et al., 1999; Konrad and Cheng, 2005). It consists of three parts depicted in Figure 1: a *bulletin board* declaring symbols referring to global object variables, a *pre-chart* describing a triggering condition split into history (left part) and future (right part), and the *consequence* defining a reaction to the trigger that is synced with the future. It expresses that whenever the pre-chart is observed, then also the consequence shall be observed; thereby, the consequence has to happen in parallel to the future. For simplicity, we denote a requirement TSC by a triple $\langle H, F, C \rangle$ made of history H, future F, and consequence C.

4.1 Interpretation of Spatial Views

TSCs are always interpreted with respect to a *world model* and a *symbol dictionary*. The world model defines the domain ontology for the specification. At least, it defines the *object types* that a TSC may speak about together with the attributes. There is no unique way of defining a world model. For example, one could define the world model in terms of a UML class diagram (Booch, 2005) or description logic (Baader et al., 2007). Earlier work on TSCs (Damm et al., 2018b) sees the world model as a network of communicating hybrid automata.

The *symbol dictionary* (Figure 4) defines the symbols that are used to represent objects from the world model within spatial views. This way, it provides the link between the world model and the TSC. Each object symbol has a type and a set of *anchors*. Anchors bind selected points of a symbol to positions in a 2D space. In this paper, the anchor points are always placed in the four corners of a symbol and describe the object bounding boxes in our global coordinate system. So, the bottom-left anchor binds to $(\underline{x}, \underline{y})$ and the top-right anchor to $(\bar{x}, \bar{y})$ in object attributes. The alternative symbol variants (second column of Figure 4) are used to make bulletin board symbols visually distinguishable (e.g., `carI` from `carJ` in Figure 1).

[3] With the full TSC language specified in (Damm et al., 2018b) also a wide range of other patterns can be realized.

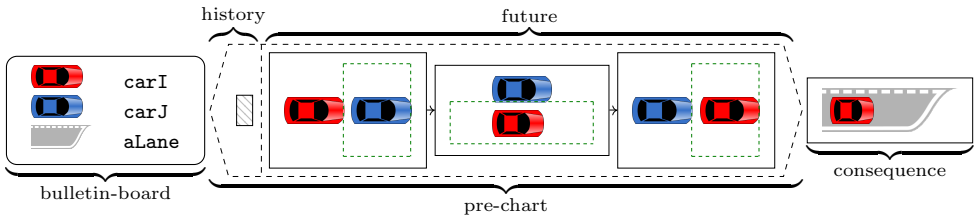
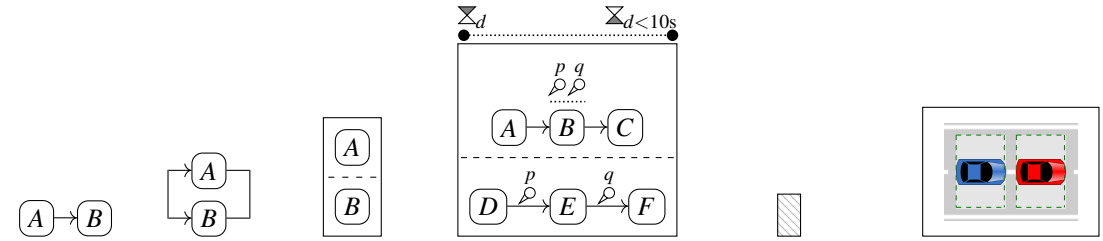


Figure 1: A TSC and its parts. The *bulletin-board* declares object symbols with global scope in the TSC; the *pre-chart* is a triggering condition for the TSC, where *history* describes past behavior; the *consequence* is the requirement obligation that shall be maintained during the *future* behavior.



(a) Sequence (b) Choice (c) Concurrency (d) Timing annotations (e) Empty invariant (f) Invariant node

Figure 2: Syntax of basic charts

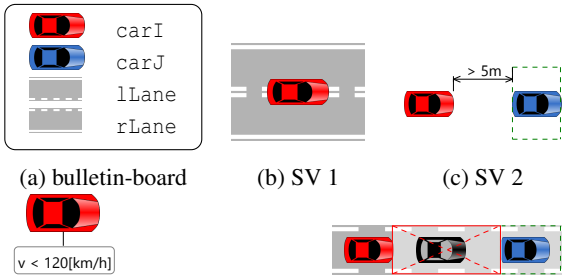
Now, we come to spatial views. As said above, symbols declared in the bulletin board stand for objects of the corresponding type. *Somewhere boxes* (green dashed rectangles) and *nowhere boxes* (red crossed rectangles) are special symbols that structure a spatial view into a hierarchy of frames—the spatial view spans the top-level frame and each somewhere or nowhere box spans an inner frame. The anchors of symbols and boxes are used to define spatial relations between the objects. In each frame, the left-to-right and bottom-to-top orderings of anchors induce an ordering of corresponding positions along x and y axes of the global coordinate system. This creates implicit spatial relations between objects directly contained in the same frame. Hence, the traffic situation depicted in a somewhere box takes place *somewhere* within the region spanned by the frame. Explicit spatial relations are defined by so-called *distance arrows* that constrain distances between anchors in x or y direction. Additional constraints over the object attributes are displayed as a textual label connected to the object symbols.

The following examples show how spatial views can be translated to mathematical formulae. The algorithm to construct the formulae can be found in (Damm et al., 2018b).

Example 1. *The following is the semantics of the spatial views in Figure 3 when using the bulletin-board in Figure 3a.*

SV 1 *expresses that* `carI` *crosses the border between the left and the right lanes.*

$$\begin{aligned} & \texttt{rLane}.\underline{x} = \texttt{lLane}.\underline{x} \\ < \;& \texttt{carI}.\underline{x} < \texttt{carI}.\bar{x} \\ < \;& \texttt{rLane}.\bar{x} = \texttt{lLane}.\bar{x} \\ \wedge \;& \texttt{rLane}.\underline{y} < \texttt{carI}.\underline{y} \\ < \;& \texttt{rLane}.\bar{y} = \texttt{lLane}.\underline{y} \\ < \;& \texttt{carI}.\bar{y} < \texttt{lLane}.\bar{y} \end{aligned}$$



(a) bulletin-board (b) SV 1 (c) SV 2 (d) SV 3 (e) SV 4

Figure 3: Spatial views from Example 1

SV 2 *expresses that* `carI` *is more than 5m behind* `carJ`. *Note that because of the somewhere box around* `carJ` *only spatial relations in x-directions are evaluated*[4].

$$\texttt{carJ}.\underline{x} - \texttt{carI}.\bar{x} > 5\ \text{m}$$

SV 3 *shows a textual annotation that constraints the speed of* `carI`*:*

$$\texttt{carI}.\text{v} < 120\ \text{km/h}$$

[4]Because the left and right borders of `carJ` and the somewhere box are aligned, `carJ` can be located on the y-axis arbitrarily within the box, but not on the x-axis.

SV 4 *requires that there is no other car* c *between* carI *and* carJ *on some lane* l*:*

$$\begin{aligned}
&\exists \mathtt{l} \in \mathtt{Lane}: \\
&\quad \mathtt{l}.\underline{\mathtt{x}} < \mathtt{carI}.\underline{\mathtt{x}} < \mathtt{carI}.\bar{\mathtt{x}} \\
&\quad\quad < \mathtt{carJ}.\underline{\mathtt{x}} < \mathtt{carJ}.\bar{\mathtt{x}} < \mathtt{l}.\bar{\mathtt{x}} \\
&\quad \wedge \mathtt{l}.\underline{\mathtt{y}} < \mathtt{carI}.\underline{\mathtt{y}} < \mathtt{carI}.\bar{\mathtt{y}} < \mathtt{l}.\bar{\mathtt{y}} \\
&\quad \wedge \mathtt{l}.\underline{\mathtt{y}} < \mathtt{carJ}.\underline{\mathtt{y}} < \mathtt{carJ}.\bar{\mathtt{y}} < \mathtt{l}.\bar{\mathtt{y}} \\
&\quad \wedge \nexists \mathtt{c} \in \mathtt{Car}: \mathtt{carI}.\bar{\mathtt{x}} < \mathtt{c}.\underline{\mathtt{x}} < \mathtt{c}.\bar{\mathtt{x}} < \mathtt{carJ}.\underline{\mathtt{x}} \\
&\quad\quad \wedge \mathtt{l}.\underline{\mathtt{y}} < \mathtt{c}.\underline{\mathtt{y}} < \mathtt{c}.\bar{\mathtt{y}} < \mathtt{l}.\bar{\mathtt{y}}
\end{aligned}$$

The object variables l *and* c *are existentially quantified because the corresponding symbols are not contained in the bulletin board.*

4.2 Chart Semantics

On top of the semantics of spatial views we define the semantics of charts. *Satisfaction* of a chart is defined with respect to a concrete *trajectory*.

Definition 1. *A* trajectory *over a set O of objects (each having some type from the world model) is a function*

$$\pi : \mathbb{R}_{\geq 0} \to \mathcal{U}^{\mathcal{A}(O)}$$

that assigns, for any point $t \in \mathbb{R}_{\geq 0}$ in time, a value to any attribute $o.a \in \mathcal{A}(O)$ of any object $o \in O$. Here, $\mathcal{U}$ is the universe of values used by the world model (e.g., reals and Booleans).

Given some trajectory, we can evaluate a spatial view at any point in time using the derived formula (given correct typing and that the global objects from the bulletin board are present in the trajectory). A spatial view is *satisfied* at time t if the formula evaluates to true under the evaluation of all object attributes (including derived ones) given by $\pi(t)$.

Satisfaction of a basic chart is always defined with respect to an interval $[b,e] \subseteq \mathbb{R}_{\geq 0}$. Furthermore, a time value t_p is selected for every time pin p.

Definition 2. *Given some trajectory and time values t_p for time pins p, a basic chart is satisfied on some interval $[b,e]$ if the following holds.*

- *Invariant nodes: $b < e$ and the spatial view holds for all $t \in [b,e)$.*
- *Empty invariant node: $b < e$*
- *Sequences (Figure 2a): there exists some $m \in [b,e]$ such that A is satisfied on $[b,m]$ and B on $[m,e]$.*
- *Choices (Figure 2b): A or B is satisfied on $[b,e]$.*
- *Concurrency (Figure 2c): both A and B are satisfied on $[b,e]$.*

For charts with timing annotations, the following has to hold additionally.

- *Sequences with a time pin p require $m = t_p$.*
- *Charts with a sequence $p_1, \ldots, p_n$ of time pins require $b \leq t_{p_1} \leq \cdots \leq t_{p_n} \leq e$.*
- *Charts with an hour glass labeled with a free variable d and a constraint $\psi(d)$ over d require that $\psi(e-b)$ evaluates to* true *(i.e., when replacing d by $e-b$).*

A requirement TSC is satisfied *on a trajectory, if for all $b \leq m \leq e \in \mathbb{R}_{\geq 0}$ holds: whenever there are time pin values such that the history is satisfied on $[b,m]$ and the future is satisfied on $[m,e]$, then there are time pin values such that the consequence is satisfied on $[m,e]$.*

Note that for invariant nodes, the end point e is explicitly excluded from the interval. This allows non-overlapping sequences of invariants.

Example 2. *The chart in Figure 2d is satisfied on an interval $[b,e]$ if $e - b < 10$s and there are time points m_1, m_2, m_3, m_4 such that subcharts A, B, C, D, E, and F are satisfied on $[b,m_1]$, $[m_1,m_2]$, $[m_2,e]$, $[b,m_3]$, $[m_3,m_4]$, and $[m_4,e]$ respectively, and*

$$\begin{aligned}
b \leq m_1 \leq t_p < t_q \leq m_2 \leq e \\
b \leq m_3 = t_p \leq m_4 = t_q \leq e \ .
\end{aligned}$$

Because time pins are existentially quantified, this is equivalent to

$$b \leq m_1 \leq m_3 \leq m_4 \leq m_2 \leq e \ .$$

In Section 6.1, consistency is reduced to satisfiablility of TSCs. Satisfiability asks whether there exists at least one trajectory that satisfies the TSC. Therefore, we assume that a world model WM, beneath a type hierarchy, defines the universe **Traj**(WM) of all possible trajectories. In other words, it describes all possible behavior. The concrete world model used throughout this paper is introduced later on in Section 4.3

Definition 3. *A basic chart* BC *is* satisfiable *in a world model* WM*, written* $\text{SAT}_{\text{WM}}(\text{BC})$*, if there exists a a trajectory $\pi \in$* **Traj**(WM)*, a time point $e > 0$ and time pin values such that* BC *is satisfied on π on $[0,e]$.*

A TSC TSC *is* satisfiable *in a world model* WM*, written* $\text{SAT}_{\text{WM}}(\text{TSC})$ *if there exists a trajectory $\pi \in$* **Traj**(WM) *such that* TSC *is satisfied on π.*

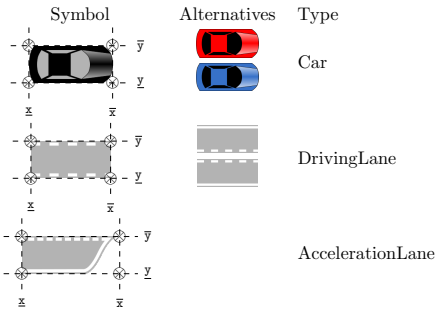


Figure 4: Symbol dictionary for the running examples

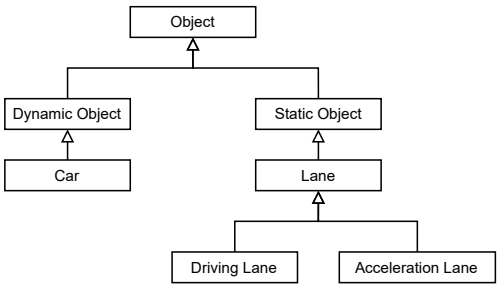


Figure 5: The world model that is used in the use case. Each box represents an object type, arrows denote inheritance

4.3 Use Case World Model

Figure 5 shows an excerpt of the world model used during this paper[5]. Every object type defines a set of attributes and invariants. The attributes include positions in form of the minimum $(\underline{x},\underline{y})$ and maximum $(\bar{x},\bar{y})$ coordinates of the object's bounding box. We use a road coordinate system where the reference line of the rightmost lane is chosen as the x-axis. Lane boundaries are then expressed in terms of `start`, `length`, and `width` of the lane. This does not mean that all roads symbolized in a TSC are straight roads in reality. Coordinate transformations such as Lanelet transformation (Bender et al., 2014) allow to interpret spatial views on curved roads, too.

The bounding box of a car is expressed by offsets `bb(x|y)(min|max)` to the reference point as denoted in Figure 6.

Cars move according to a single track model given by

$$\dot{x} = v\cos\theta \quad \dot{y} = v\sin\theta \quad \dot{\theta} = \frac{v}{r} = \frac{\tan\delta}{L}v \quad \dot{v} = a$$

subject to the constraints $|\delta| \leq \delta_{\max}$ and lateral acceleration $|a_{lat}| = |v\dot{\theta}| \leq 0.4g$. The lateral acceleration bound of $0.4g \approx 3.92\,\mathrm{m/s^2}$ is stated in the literature (Schramm et al., 2014) as a validity constraint for the single track model.

[5]It is used by both the examples and the evaulation case.

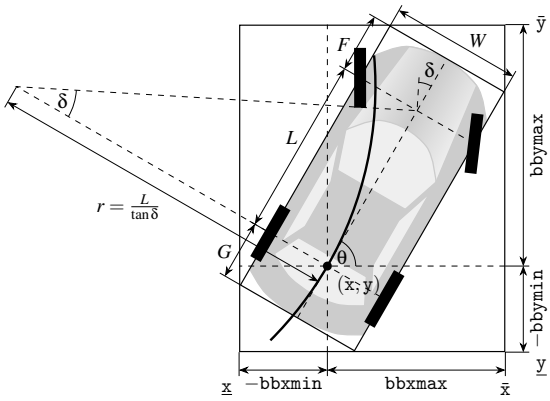


Figure 6: Bounding box for a car and illustration of the single track model. Constants F, G, L and W together describe the vehicle dimensions. The turning radius r depends on wheel base L and Ackermann steering angle δ.

4.4 Satisfiability Modulo Theories and Bounded Model Checking

Satisfiability modulo theories (SMT) solving is an extension of classical satisfiability (SAT) solving that allows to combine different theories, e.g., linear arithmetics and bit vectors. An overview on SMT solving can be found in (Barrett et al., 2009). An SMT solver (supporting a set of theories) takes as an input a problem description in form of a set Φ of constraints. The solver then tries to either find a satisfying assignment (returning `sat`) or prove unsatisfiability (returning `unsat`) of Φ within the used theories. If the solver cannot do either (e.g., because of a timeout or incompleteness of the implemented decision procedure), it returns `unknown`.

With bounded model checking (BMC) (Armando et al., 2009), SMT solving is utilized to check bounded reachability in symbolic transition systems. In BMC, the state vector is encoded in a set $\mathbf{X}$ of variables. A BMC problem is a triple (I,T,F) of constraints over $\mathbf{X}$. The constraints $I(\mathbf{X})$ and $F(\mathbf{X})$ characterize the initial and final states of the system. The possible transitions are encoded in the constraint $T(\mathbf{X},\mathbf{X}')$ that evaluates to true whenever there is a transition between the current state $\mathbf{X}$ and the next state $\mathbf{X}'$. Introducing an instance $\mathbf{X}_i$ of $\mathbf{X}$ for every step $i = 0,\ldots,n$, the constraint

$$I(\mathbf{X}_0) \wedge \bigwedge_{i=1}^{n} T(\mathbf{X}_{i-1},\mathbf{X}_i) \wedge F(\mathbf{X}_n)$$

characterizes all accepting runs with n steps.

5 METHODOLOGY AND EXPECTED RESULTS

The consistency analysis shall be based on a rigorous formal method. Due to the incompleteness of scenario descriptions and domain models in early phases of the development process, validity of the found inconsistencies is mote important than completeness of the findings. Following some of the existing approaches (Ellen et al., 2014; Becker, 2018; Filipovikj et al., 2017), it seems a good idea to base a consistency notion for TSCs on the existence of satisfying trajectories. Because TSCs describe requirements as sequences of invariants, SMT solving seems a promising approach to tackle the problem. Some of the related work (Ellen et al., 2014; Becker, 2018; Filipovikj et al., 2017; Eggers et al., 2018) also uses SMT solving for the generation of trajectories.

The consistency analysis shall provide feedback about a scenario specification already in early phases of the development process. Here, scenario specifications may be still incomplete. Furthermore, the operation environment and physical constraints of the HAV may be under-specified. Because the consistency analysis has to deal with large sets of requirements in a potentially underspecified context, performance shall be favored over completeness of the results. TSCs describe scenarios in a dense time domain, and require to consider non-linear movement given usually as differential equations or hybrid automata (Damm et al., 2018a). Numerical approaches to explore this kind of continuous-time hybrid systems exist (Henzinger et al., 1997; Frehse, 2005; Frehse et al., 2011; Chen and Sankaranarayanan, 2016; Eggers et al., 2011), but scalability of these methods is an issue. The TSC semantics adds another layer of complexity. Results on Duration Calculus (that has similar operators to TSCs) (Chaochen et al., 1993; Bouajjani et al., 1995) show that this might easily lead to state explosion. Therefore, the consistency analysis uses two-sided approximations. As explained in detail in Section 6.2, both a necessary and a sufficient condition for the existence of trajectories is developed. The former produces discrete approximations that don't ensure validity (except from continuity). The latter is incomplete, but produces trajectories that are valid with respect to the TSCs and a realistic vehicle dynamics model. Because the considered requirement specification mainly address maneuver and trajectory planning, a single-track model (explained in Section 4.3) is sufficient. For reasons of efficiency, the methods are designed to be usable with solvers for mixed Boolean and linear real arithmetic.

The developed prototype shall be applied to medium sized (sets of) TSCs, thereby evaluating scalability (i.e., in terms of runtime) and completeness of the approach. It is expected that the experimental evaluation shows that an SMT-based automated consistency analysis for TSCs is practicable. By the nature of SMT solving, it must be expected that the execution time for solving the generated SMT problems is expnential to the size of the TSCs. However, it shall be possible to design the analysis method in a way the overall number of SMT problems to be solved in practice grows only polynomial to the size of the specification, and that each SMT problem is small enough to produce a result in acceptable time. Furthermore, it is expected that the experiments show some limitations of the approach wrt. completeness of found inconsistencies, but that the chosen approximations are sufficient to find common specification faults.

6 STATE OF WORK

The following is a summary about the developed consistency analysis method, including the translation of TSCs into SMT problems. For reasons of space, most of the constructions can only be presented by examples. The complete constructions including correctness proofs will be given in the author's PhD thesis.

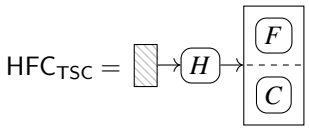
6.1 Defining Consistency

The idea behind the formal definition of a set of TSCs being consistent is not totally new. The following consistency notions adopt ideas from related work (Becker, 2018; Becker, 2020; Filipovikj et al., 2017; Ellen et al., 2014).

In general, consistency asks the question: "Is it possible to build a system that satisfies all my requirements?" Usually, this question cannot be answered unless you build the system itself. Therefore, some weaker question is used: "Does there exist at least one trajectory (called a witness trajectory), i.e., a system observation, that does not violate one of my requirements?" Obviously, if the answer to the second question is "No" then it is impossible to build a system that implements all the requirements together. In other words, there is some conflict in the requirements specification that makes implementation impossible and needs to be resolved.

As a starting point for formally defining consistency of requirement TSCs we take the second question. For specifications consisting of a single TSC, we can write it as $\mathrm{SAT}_{\mathsf{WM}}(\mathsf{TSC})$. However, this can be trivially answered with true for may TSCs, simply by providing a trajectory where the pre-chart is

not satisfied. So, $\text{SAT}_{\mathsf{WM}}(\mathsf{TSC})$ alone is not an appropriate consistency criterion. Similar observations have been made in all of the related work (Becker, 2018; Becker, 2020; Filipovikj et al., 2017; Ellen et al., 2014) that considers requirements in implication form. It is solved typically by requiring that the premise of the requirement – in TSC terms the pre-chart – occurs at least once on the trajectory. For TSCs $\langle H, F, C \rangle$, we realize this by asking for satisfiability of the basic chart



with H, F, C being history, future and consequence of the original TSC. Formally, it expresses the following

- The pre-chart, H followed by F, occurs at least once.
- The consequence C is satisfied at least once in parallel to the future F.

This includes only a weak approximation of the formal TSC semantics (because the consequence is checked only in parallel to *one* occurrence of the future, but there may be more occurrences) but is indeed a necessary condition for what we would achieve and turns out to be sufficient to find typical specification faults[6]. As a side effect, this removes the implicit implication between pre-chart and consequence. In related work (Becker, 2020; Ellen et al., 2014), this form of consistency is called *existential consistency*.

Now we lift this idea to sets (with size > 1) of TSCs. For sets of TSCs, the witness trajectory shall show that *all* TSCs are satisfied together. Recall that TSCs only constrain those time intervals on a trajectory where history and future hold. Hence, if we have witnesses for existential consistency of each TSC in a set, a witness for the whole set may be easily constructed by putting the individual witness trajectories in sequence, with some glue parts between them that are not covered by the specification. Hence, this does not give more insight on consistency of the specification than consistency of each TSC alone (Becker, 2020). The interesting case in a set of TSCs, however, are those cases where TSCs are active in parallel. Here, consequences from different TSCs must be satisfied in parallel, which may cause actual conflicts. Of course, a specification may contain sub-sets where the pre-charts are mutual exclusive such that they, by intention cannot be active together. Therefore, the consistency notion proposed for sets of TSCs does the following.

- Each sub-set of TSCs is checked separately.
- For each set, check if the TSCs can be active in parallel.
- If so, check whether also the consequences can be satisfied in parallel to the futures.

The term *active in parallel* is realized by choosing one TSC from each subset as the "innermost" TSC. For other TSCs, let the future start before the innermost TSC's future, and end afterwards. This way, all the TSCs are active in parallel. For some innermost TSC $\mathsf{TSC} = \langle H, F, C \rangle$ and a context $T = \{\langle H_1, F_1, C_1 \rangle, \ldots, \langle H_n, F_n, C_n \rangle\}$ this is encoded in the basic charts $\mathsf{BC}_1^{\mathsf{TSC},T}$ and $\mathsf{BC}_2^{\mathsf{TSC},T}$ shown in Figure 7, where $\mathsf{BC}_1^{\mathsf{TSC},T}$ does include the consequences and $\mathsf{BC}_2^{\mathsf{TSC},T}$ does not. Because timing constraints may restrict the duration of the future, it is for some TSCs not possible to be the innermost TSC. Therefore, every TSC is tried as the innermost one.

As already mentioned before, we don't try to solve the satisfiability problem exactly. Instead, we approach it from both directions building two semi-decision procedures $\text{CHECKSATN}_{\mathsf{WM}}$ and $\text{CHECKSATS}_{\mathsf{WM}}$. They check a necessary respectively sufficient condition for satisfiability of a basic chart in context of our world model WM. Taking some basic chart BC as input, the two functions each do the following.

1. Derive a BMC problem from BC and WM, and an unrolling depth n.
2. Unroll the BMC problem for n steps yielding a constraint Φ
3. Send Φ to an SMT solver and return the result.

The functions differ in the constructed BMC problem and chosen unrolling depth and are explained in more detail in the following sections. $\text{CHECKSATN}_{\mathsf{WM}}(\mathsf{BC})$ returns `unsat` only if BC is *unsatisfiable* and $\text{CHECKSATS}_{\mathsf{WM}}(\mathsf{BC})$ returns `sat` only if BC is *satisfiable* within WM.

Algorithm 1 lists the procedure for finding inconsistencies in a set TSC using both approximations. Recall that a single TSC is existentially inconsistent if $\text{CHECKSATN}_{\mathsf{WM}}(\mathsf{HFC}_{\mathsf{TSC}}) = \texttt{unsat}$. In the case $T = \emptyset$ is $\mathsf{BC}_2^{\mathsf{TSC},T}$ equivalent to $\mathsf{HFC}_{\mathsf{TSC}}$ above. Hence, the algorithm reports also existetial inconsistencies of single TSCs in the set. Note also that CHECKSATN and CHECKSATS are chained in a way that an inconsistency is only reported if an exact decision procedure would yield the same result. Provided

[6]The specification fault that is observed most often is a discontinuity between consecutive spatial views in a sequence that is only satisfiable when vehicles are teleported.

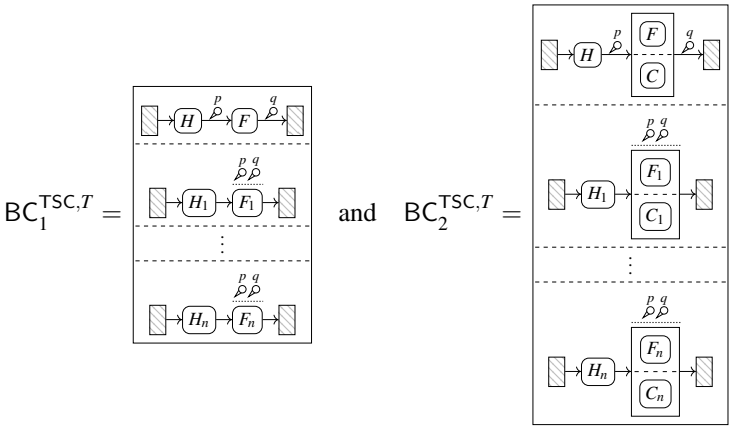


Figure 7: Definitions of the charts $\mathsf{BC}_1^{\mathsf{TSC},T}$ and $\mathsf{BC}_2^{\mathsf{TSC},T}$

```
1 foreach TSC ∈ 𝕋𝕊ℂ do
2     foreach T ⊆ 𝕋𝕊ℂ \ {TSC} do
3         if T = ∅ or
            CheckSatS_WM(BC_1^{TSC,T}) = sat
            then
4             if
                CheckSatN_WM(BC_2^{TSC,T}) =
                unsat then
5                 Report inconsistency of
                    {TSC} ∪ T;
6             end
7         end
8     end
9 end
```

Algorithm 1: Finding inconsistencies in TSCs

that the implementation is sound, the approximations cannot result in spuriously reported inconsistencies.

The algorithm has much room for optimization.

- Usually, the user is interested in the *minimum inconsistent subsets*, i.e., inconsistent sets that become consistent when removing one TSC. So we can skip any subset where we know that it contains an inconsistent subset.
- For sets $S \supseteq T$, unsatisfiability of $\mathsf{BC}_1^{\mathsf{TSC},T}$ implies unsatisfiablility of $\mathsf{BC}_1^{\mathsf{TSC},S}$ (the same holds for the corresponding BMC problems). So we can skip any superset of T if $\textsc{CheckSatS}_{\mathsf{WM}}\left(\mathsf{BC}_1^{\mathsf{TSC},T}\right) = \texttt{unsat}$ in line 3 of the algorithm.
- The BMC problem generated by CheckSatS is usually harder to solve than the one from CheckSatN. so we check it only if $\textsc{CheckSatN}_{\mathsf{WM}}\left(\mathsf{BC}_2^{\mathsf{TSC},T}\right) = \texttt{unsat}$.

The prototype used for evaluation implements these optimizations, and experiments (see Section 6.3) show that most of the cases can be skipped.

6.2 Semi-deciding Satisfiability of Basic Charts

Both CheckSatS and CheckSatN work by encoding a basic chart as a BMC problem, unrolling it for n steps, and running an SMT solver on it. As a necessary condition for satisfiability, we use SMT solving to find a satisfying witness trajectory. The same technique has been used in related work (Becker, 2020) for simulation of TSCs. The same idea is also followed for the necessary condition. The differences are described later on in Section 6.2.3. In both cases, the BMC problem (I,T,F) is of the form

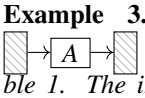
$$I \equiv I_{\text{chart}}$$
$$T \equiv T_{\text{chart}} \wedge T_{\mathsf{WM}} \wedge \bigwedge_{sv \in SV} (b_{sv} \leftrightarrow T_{sv})$$
$$F \equiv F_{\text{chart}}$$

The triple $(I_{\text{chart}}, T_{\text{chart}}, F_{\text{chart}})$ encodes the chart structure, where satisfaction of a spatial view sv in the current step is substituted by some variable b_{sv}. The actual formula for sv (ranging over all spatial views in the chart) is encoded in the constraint T_{sv}. Finally, T_{WM} encodes world model constraints, i.e. it restricts solutions to trajectories from the world model.

6.2.1 Encoding the Chart Structure

Encoding of the chart structure is the same for both sufficient and necessary conditions, except

that CHECKSATS uses a fixed time step[7] and CHECKSATN variable step size. Start and end of sub-charts (with index i) is encoded with Boolean variables $\mathsf{started}_i$ and $\mathsf{complete}_i$.

Example 3. *Encoding the chart structure of*  *leads to the constraints shown in Table 1. The invariant nodes are numbered from left to right. Note that no variable* $\mathsf{started}_1$ *is needed for the first sub-chart, because it always starts with the trajectory. The variable* ok_2 *keeps track of the second invariant after start of the sub-chart (for empty spatial views this is not needed because they cannot be violated).*

Another example can be found in the related work (Becker, 2020).

6.2.2 Sufficient Conditions for Invariants

Now, we have a closer look to the spatial view constraints Φ_{sv} and world model constraints Φ_{WM}. Attribute values of non-dynamic objects (such as lanes) are represented by free variables in the BMC problem. So, for some lane `l` we'd introduce four variables $\mathtt{y}_1$, $\mathtt{start}_1$, and $\mathtt{end}_1$ – derived attributes can be inlined and don't need variables. The modeling of cars is more complicated and differs for CHECKSATS and CHECKSATN. In the following, CHECKSATS is described.

The trajectories of cars are described as quadratic Bézier splines. Each segment of the spline is described by control points

$$\mathbf{p}_0 = (x_0, y_0), \qquad \mathbf{p}_1 = (x_1, y_1), \qquad \mathbf{p}_2 = \mathbf{p}'_0 = (x'_0, y'_0) \ .$$

Start and end point of consecutive segments are shared. Each segment describes the movement within one unrolling step of the BMC problem. The position at time t after start of the current step (with step size Δ) is given by the function

$$\mathbf{p}(t) = \mathbf{p}_0\left(1-\frac{t}{\Delta}\right)^2 + 2\mathbf{p}_1\left(1-\frac{t}{\Delta}\right)\frac{t}{\Delta} + \mathbf{p}_2\frac{t^2}{\Delta^2}$$

and the velocity vector is determined by its derivative. Other attributes, such as the bounding box extends, are encoded by two variables standing for safe upper and lower bounds in the current step. The special properties of Bézier splines (see (Prautzsch et al., 2013) for an overview) allow a quite convenient encoding of spatial views. For example, distances between Bézier splines can be expressed as pairwise distances between their control points. Furthermore, every Bézier spline segment lies within the convex polygon spanned by its control points. These properties make it possible to conservatively approximate spatial views by linear constraints over the control points.

Example 4. *In* CHECKSATS*, the spatial view in Figure 3c is encoded as*

$$\bigwedge_{i\in\{0,1\}} \begin{pmatrix} x_{\mathtt{carJ},i} + \mathtt{bbxmin}^l_{\mathtt{carJ}} \\ \qquad -x_{\mathtt{carI},i} - \mathtt{bbxmax}^u_{\mathtt{carI}} > 5m \end{pmatrix}$$
$$\wedge \begin{pmatrix} x'_{\mathtt{carJ},0} + \mathtt{bbxmin}^l_{\mathtt{carJ}} \\ \qquad -x'_{\mathtt{carI},0} - \mathtt{bbxmax}^u_{\mathtt{carI}} \geq 5m \end{pmatrix}$$

If this formula holds, then the spatial view is satisfied on the whole spline segment.

The world model constraints Φ_{WM} ensure that $\mathtt{bbxmin}^l$ and $\mathtt{bbxmax}^u$ are safe upper and lower bounds. Because they depend on the heading angle θ, we choose a piece-wise approximation where different heading angles are approximated by a finite set I of intervals $I \in \mathcal{I}$. E.g., `bbxmax` of `carI` is characterized by a constraint

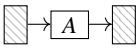
$$\bigvee_{I\in\mathcal{I}} \phi_{\mathtt{carI}.\theta\in I} \wedge \Big(\mathtt{bbxmax}^l \leq \inf_{\theta\in I}\{\mathtt{bbxmax}(\theta)\} \wedge \mathtt{bbxmax}^u \geq \sup_{\theta\in I}\{\mathtt{bbxmax}(\theta)\}\Big)$$

where $\phi_{\mathtt{carI}.\theta\in I}$ is a linear constraint describing that the heading is within I. The infimum and supremum of the heading-dependent bounding box extend $\mathtt{bbxmax}(\theta)$ can be calculated numerically when generating the BMC problem. Furthermore, constrains are added that ensure that the trajectory is continuously differentiable with curvature $|\kappa| \leq \frac{\tan\delta_{\max}}{L}$ and lateral acceleration $|a_{lat}| = |v^2\kappa| = |(\dot{x}^2+\dot{y}^2)\kappa| \leq 0.4g$. These ensure that generated trajectories are solutions of the single track model specified in Section 4.3.

6.2.3 Necessary Conditions for Invariants

For CHECKSATN, the chart structure is encoded the same way in the BMC problem, but the constraints T_{WM} and T_{sv} are relaxed. Following a result by Fränzle and Hansen about positive Duration Calculus (Fränzle and Hansen, 2007), we know that the chart structure BMC problem is either unsatisfiable or has a solution after $m+1$ unrolling steps, with m being the number of sequence operation in the TSC. However, in a solution with minimum unrolling depth, a single BMC step may cover an arbitrarily long time interval. For example, during one step, a whole overtaking maneuver may take place. Therefore, instead of describing and restricting the exact vehicle movement, invariants are checked at discrete time points only.

[7] Because of efficiency and decidability, we restrict ourselves to linear inequalities only. With variable time steps, the relation between velocity and position becomes nonlinear.

Table 1: The chart structure of  encoded in a BMC problem.

Initial	Transition	Final
$\neg\mathsf{complete}_1$	$\mathsf{started}'_2 \rightarrow (\mathsf{complete}_1 \vee \mathsf{started}_2)$	$\mathsf{complete}_3$
$\neg\mathsf{complete}_2$	$\mathsf{complete}'_2 \leftrightarrow (\mathsf{started}_2 \wedge \mathsf{ok}'_2)$	
ok_2	$\mathsf{ok}'_2 \leftrightarrow (\mathsf{started}_2 \rightarrow \mathsf{ok}_2 \wedge b_A)$	
$\neg\mathsf{complete}_3$	$\mathsf{started}'_3 \rightarrow (\mathsf{complete}'_2 \vee \mathsf{started}_3)$	
	$\mathsf{complete}'_3 \leftrightarrow \mathsf{started}_3$	

Example 5. *In* CHECKSATN*, the spatial view in Figure 3c is encoded as*

$$x_{\texttt{carJ}} + \texttt{bbxmin}_{\texttt{carJ}} - x_{\texttt{carI}} - \texttt{bbxmax}_{\texttt{carI}} > 5m$$
$$\wedge\, x'_{\texttt{carJ}} + \texttt{bbxmin}'_{\texttt{carJ}} - x'_{\texttt{carI}} - \texttt{bbxmax}'_{\texttt{carI}} \geq 5m$$

where variables stand for object attribute values in the current, respective next, step.

By checking spatial views (as in the above formula) both at beginning and end of a step, we catch both contradictions between parallel as well as consecutive invariant nodes. Note that, because the end time e is excluded in Definition 2 when evaluating invariant nodes, we must only enforce non-strict spatial relations for the end of a step. Because we don't need to characterize the evaluation of the bounding box for the whole unrolling step anymore, the upper and lower bounds can be replaced by single variables.

6.3 Evaluation

To avoid that the results are biased by the author's intention when designing the use case, the evaluation is based on an earlier case study (Esterle, 2021). In this work, common highway traffic rules have been formalized in LTL. In the original work (Esterle et al., 2020), they have been used to examine databases of recorded traffic data for rule violations. In order to translate them to TSCs, first spatial views have been designed that match the atomic propositions in the LTL rules. An example is the proposition $\mathit{succ}^{i\rightarrowtail j}$ saying `carI` is the successor of `carJ` shown in Figure 3d. Then, the LTL expressions have been translated to TSCs using a set of pre-defined patterns. The result is a set of nine TSCs – rules that do not apply to two-lane highways have been omitted.

The analysis finds that the TSCs shown in Figure 8 are pairwise inconsistent. Taking a closer look at the TSCs shows that TSC (a) forbids to use any other lane than the right one, making lane-change rules (TSCs (b) and (c)) inapplicable. The conflict between (b) and (c) comes from the chosen formalization of (b) as a TSC: the forbidden behavior (right overtaking) is in the pre-chart, and the exception (being on an acceleration lane) to this rule forms the consequence. From the TSC semantics point of view, this is correct, but breaks the intuition behind premise-consequence charts.

The BMC problems for CHECKSATS have been configured with a step size of 3 s and unrolled for 10 steps[8]. From the 2304 cases that need to be checked in the worst case when analyzing consistency of nine TSCs, only 467 (21 %) must be handed over to the SMT solver; the rest could be inferred from existing results using the optimizations mentioned at the end of Section 6.1. In total, the analysis takes about 1 min and 20 s on a Windows 10 notebook with an Intel Core i7-4700MQ CPU @ 2.39 GHz and 8 GB RAM, using version 4.6.0 of the Z3 solver (de Moura and Bjørner, 2008) with default settings. The results show that in practice only small subsets (up to size 3 in the experiment) of TSCs need to be checked, so there is only a polynomial growth in the number of cases, and the individual SMT problems remain manageably small. In (Becker et al., 2022), it has been shown that also more complex TSCs can be checked for satisfiability in reasonable time.

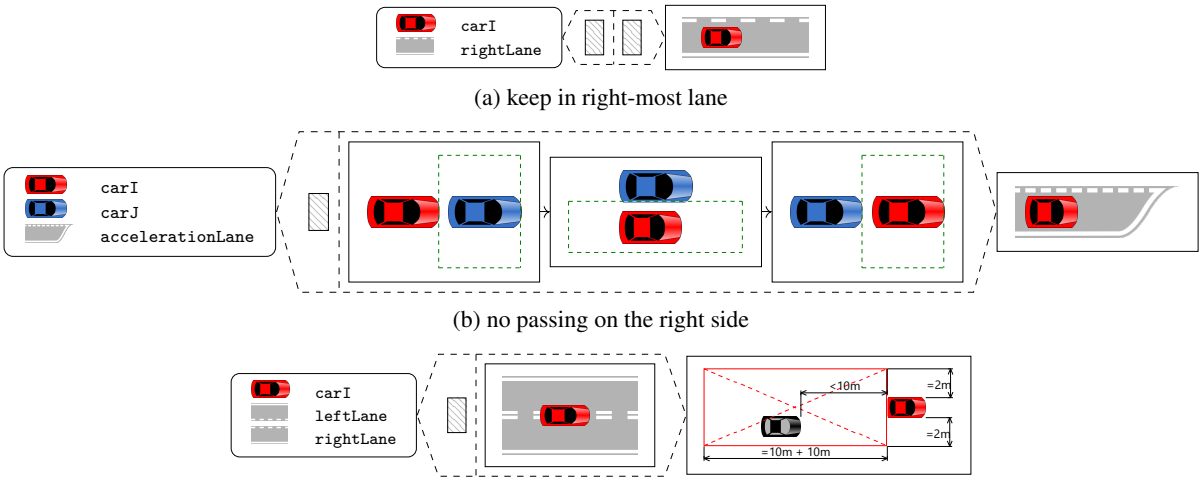
A similar experiment has been carried out on a specification done by a non-expert in formal specification[9]. This second experiment shows that the consistency analysis is especially strong in finding formalization errors such as unsatisfiable spatial constraints or impossible sequences. The scalability of the TSC encoding has been evaluated in the context of already published work (Becker et al., 2022).

6.4 Practical Application

The consistency analysis prototype has been integrated into a graphical specification tool for TSCs (Figure 9). It is part of a tool chain for scenario-based development that is under ongoing development at the DLR SE institute. Other parts of the tool chain include generation of OpenSCENARIO files (Becker et al., 2022) and scenario monitors (Grundt et al.,

[8]The unrolling depth for CHECKSATN is calculated automatically

[9]A student employee at the author's institute who was not under the author's supervision.



(a) keep in right-most lane

(b) no passing on the right side

(c) safe lane change

Figure 8: Conflicting traffic rules found in the evaluation

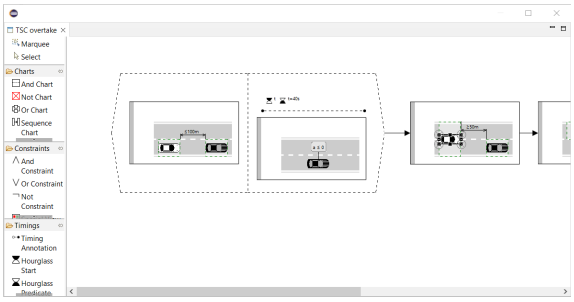


Figure 9: Screenshot of the graphical specification tool for TSCs

2022). The tooling is actively used both internally and in cooperation with industry partners. Furthermore, tooling and methods are extended to the railway and maritime transport domains as well. The future use of the specification tooling will yield more insight about practical suitability of the consistency analysis.

7 CONCLUSION AND OUTLOOK

This work proposes a consistency analysis technique for TSCs. Although being grounded on the formal semantics of TSCs, the goal is not to produce a certificate of formal requirements consistency, but to find real conflicts in the specification on a well-defined basis. Consistency is reduced to satisfiability of existential TSCs, which in turn is encoded as a BMC problem. For reason of space, the latter has only been sketched here with some examples. The full construction, including correctness proofs, will be given in the author's PhD thesis. It shall be highlighted that the implementation (that has been used to carry out the presented evaluation) assumes the shown parameterized vehicle model, but otherwise is not bound to a specific domain ontology.

The proposed consistency analysis re-uses basic ideas from earlier work (Becker, 2020). The consistency notion for single TSCs remains the same. The idea for consistency of non-singleton sets is also similar. Both approaches split analysis cases each into a premise (can the TSCs be active in parallel?) and a consequence (is there a trajectory where TSCs are active in parallel?). However, the technical realization of the term "active in parallel" has been reworked. The earlier work synchronizes invariant nodes from pairs of TSCs. Although it is possible to extend this to larger subsets, these number of analysis cases per subset explodes. In contrast, the current approach work only requires n cases per subset of size n, and it is possible to re-use information and skip some of the cases. Furthermore, the earlier work uses a necessary condition for both the premise and the consequence. So, there is no guarantee that false inconsistencies are ruled out. In the present work, this is solved by generating a satisfying trajectory (Becker et al., 2022) instead of an approximation thereof. The analysis tool then creates a trajectory for every (minimal) inconsistent subset that shows how a conflicting situation arises. By the use of a Bézier spline encoding technique with additional constraints (see Section 6.2.2), the generated trajectories are solutions of a single-track vehicle dynamics model. Simpler (e.g., piece-wise linear) trajectories would not have this property, while alternative trajectory encodings (such as arc segments or clothoids) can be hardly expressed by linear constraint systems. Related work (Becker, 2020) shows that the generated trajectories can be simulated.

The experimental results demonstrate practical applicability of the approach. Furthermore, it turns out that it scales in practice because constraint solving is only needed for small subsets. However, it would be worthwhile to support the experimental results with further case studies. So far, the present case study shows that the consistency analysis catches both contradicting as well as ill-structured (violating specification patterns) requirements. Therefore, it is future work to develop explicit specification guidelines for the work with TSCs, and to extend the consistency analysis to handle a broader set of specification patterns.

Besides the application in the presented consistency analysis method, parts of the used techniques (and their implementation) are planed to be re-used in other applications of TSCs. For example, trajectory generation can be used for simulation in the context of criticality analysis, as well as test case generation. Translation of spatial invariants to constraint systems is also required for monitoring.

REFERENCES

Aichernig, B. K., Hörmaier, K., Lorber, F., Ničković, D., and Tiran, S. (2015). Require, test and trace IT. In *Formal Methods for Industrial Critical Systems - 20th International Workshop, FMICS 2015, Proceedings*, volume 9128 of *LNCS*, pages 113–127. Springer.

Armando, A., Mantovani, J., and Platania, L. (2009). Bounded model checking of software using smt solvers instead of sat solvers. *International Journal on Software Tools for Technology Transfer*, 11:69–83.

Baader, F., Calvanese, D., Mcguinness, D., Nardi, D., and Patel-Schneider, P. (2007). *The Description Logic Handbook: Theory, Implementation, and Applications*.

Barrett, C. W., Sebastiani, R., Seshia, S. A., and Tinelli, C. (2009). Satisfiability modulo theories. *Handbook of satisfiability*, 185:825–885.

Becker, J., Koopmann, T., Neurohr, B., Neurohr, C., Westhofen, L., Wirtz, B., Böde, E., and Damm, W. (2022). Simulation of Abstract Scenarios: Towards Automated Tooling in Criticality Analysis. pages 42–51.

Becker, J. S. (2018). Analyzing consistency of formal requirements. In *18th International Workshop on Automated Verification of Critical Systems*.

Becker, J. S. (2020). Partial consistency for requirement engineering with traffic sequence charts. In *Automotive Software Engineering (ASE2020)*.

Bender, P., Ziegler, J., and Stiller, C. (2014). Lanelets: Efficient map representation for autonomous driving. In *2014 IEEE Intelligent Vehicles Symposium Proceedings*, pages 420–425. IEEE.

Booch, G. (2005). *The unified modeling language user guide*. Pearson Education India.

Bouajjani, A., Lakhnech, Y., and Robbana, R. (1995). From duration calculus to linear hybrid automata. In *LNCS*, volume 939, pages 196–210.

Chaochen, Z., Hansen, M. R., and Sestoft, P. (1993). Decidability and undecidability results for duration calculus. In *STACS 93, 10th Annual Symposium on Theoretical Aspects of Computer Science, Würzburg, Germany, February 25-27, 1993, Proceedings*, pages 58–68.

Chen, X. and Sankaranarayanan, S. (2016). Decomposed reachability analysis for nonlinear systems. In *Real-Time Systems Symposium (RTSS), 2016 IEEE*, pages 13–24. IEEE.

Damm, W., Kemper, S., Möhlmann, E., Peikenkamp, T., and Rakow, A. (2017). Traffic sequence charts - from visualization to semantics. Reports of SFB/TR 14 AVACS 117, SFB/TR 14 AVACS.

Damm, W., Kemper, S., Möhlmann, E., Peikenkamp, T., and Rakow, A. (2018a). Using Traffic Sequence Charts for the Development of HAVs. In *ERTS 2018*, 9th European Congress on Embedded Real Time Software and Systems (ERTS 2018), Toulouse, France.

Damm, W., Möhlmann, E., Peikenkamp, T., and Rakow, A. (2018b). *A Formal Semantics for Traffic Sequence Charts*, pages 182–205. Springer International Publishing, Cham.

Damm, W., Möhlmann, E., and Rakow, A. (2019). A scenario discovery process based on traffic sequence charts. In *Validation & Verification of Automated Systems – Results of the ENABLE-S3 Project*.

de Moura, L. and Bjørner, N. (2008). Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems: 14th International Conference, TACAS 2008. Proceedings*, pages 337–340. Springer.

Dwyer, M. B., Avrunin, G. S., and Corbett, J. C. (1999). Patterns in property specifications for finite-state verification. In *Proceedings of the 21st international conference on Software engineering*, pages 411–420.

Eggers, A., Ramdani, N., Nedialkov, N., and Fränzle, M. (2011). Improving sat modulo ode for hybrid systems analysis by combining different enclosure methods. In *International Conference on Software Engineering and Formal Methods*, pages 172–187. Springer.

Eggers, A., Stasch, M., Teige, T., Bienmüller, T., and Brockmeyer, U. (2018). Constraint systems from traffic scenarios for the validation of autonomous driving. In *Third International Workshop on Satisfiability Checking and Symbolic Computation, Part of FLOC 2018*.

Ellen, C., Sieverding, S., and Hungar, H. (2014). Detecting consistencies and inconsistencies of pattern-based functional requirements. In Lang, F. and Flammini, F., editors, *Formal Methods for Industrial Critical Systems - 19th International Conference, FMICS 2014, Florence, Italy, September 11-12, 2014. Proceedings*, volume 8718 of *Lecture Notes in Computer Science*, pages 155–169. Springer.

Esterle, K. (2021). *Formalizing and Modeling Traffic Rules Within Interactive Behavior Planning*. PhD thesis, Universität München.

Esterle, K., Gressenbuch, L., and Knoll, A. (2020). Formalizing traffic rules for machine interpretability. In *2020 IEEE 3rd Connected and Automated Vehicles Symposium (CAVS)*, pages 1–7. IEEE.

Feiler, P., Wrage, L., and Hansson, J. (2010). System architecture virtual integration: A case study. In *Embedded Real-time Software and Systems Conference*.

Filipovikj, P., Rodriguez-Navas, G., Nyberg, M., and Seceleanu, C. (2017). Smt-based consistency analysis of industrial systems requirements. In *Proceedings of the Symposium on Applied Computing*, pages 1272–1279. ACM.

Fränzle, M. and Hansen, M. R. (2007). Deciding an interval logic with accumulated durations. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 4424 of *LNCS*, pages 201–215. Springer.

Frehse, G. (2005). Phaver: Algorithmic verification of hybrid systems past hytech. In *International workshop on hybrid systems: computation and control*, pages 258–273. Springer.

Frehse, G., Le Guernic, C., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., and Maler, O. (2011). Spaceex: Scalable verification of hybrid systems. In *International Conference on Computer Aided Verification*, pages 379–395. Springer.

Grundt, D., Köhne, A., Saxena, I., Stemmer, R., Westphal, B., and Möhlmann, E. (2022). Towards runtime monitoring of complex system requirements for autonomous driving functions.

Heitmeyer, C. L., Jeffords, R. D., and Labaw, B. G. (1996). Automated consistency checking of requirements specifications. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 5(3):231–261.

Henzinger, T. A., Ho, P.-H., and Wong-Toi, H. (1997). Hytech: A model checker for hybrid systems. *International Journal on Software Tools for Technology Transfer*, 1(1-2):110–122.

ISO 26262 (2018). Road vhicles—functional safety. Technical report, International Organisation for Standardization.

Jaffe, M. S., Leveson, N. G., Heimdahl, M. P. E., and Melhart, B. E. (1991). Software requirements analysis for real-time process-control systems. *IEEE transactions on software engineering*, 17(3):241–258.

Kalisvaart, S., Slavik, Z., and Op den Camp, O. (2020). Using scenarios in safety validation of automated systems. In Leitner, A., Watzenig, D., and Ibanez-Guzman, J., editors, *Validation and Verification of Automated Systems*, pages 27–44. Springer International Publishing, Cham.

Kamalrudin, M. and Sidek, S. (2015). A review on software requirements validation and consistency management. *International journal of software engineering and its applications*, 9(10):39–58.

Konrad, S. and Cheng, B. H. (2005). Real-time specification patterns. In *Proceedings of the 27th international conference on Software engineering*, pages 372–381.

Koopman, P. and Fratrik, F. (2019). How many operational design domains, objects, and events? In *Safeai@ aaai*.

Kramer, B., Neurohr, C., Büker, M., Böde, E., Fränzle, M., and Damm, W. (2020). Identification and quantification of hazardous scenarios for automated driving. In Zeller, M. and Höfig, K., editors, *Model-Based Safety and Assessment*, pages 163–178, Cham. Springer International Publishing.

Leitner, A. et al. (2019). ENABLE-S3 Summary of Results. Technical report.

Leveson, N. (2000). Completeness in formal specification language design for process-control systems. In *Proceedings of the third workshop on Formal methods in software practice*, pages 75–87.

Menzel, T., Bagschik, G., and Maurer, M. (2018). Scenarios for development, test and validation of automated vehicles. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1821–1827. IEEE.

Neurohr, C., Westhofen, L., Butz, M., Bollmann, M. H., Eberle, U., and Galbas, R. (2021). Criticality analysis for the verification and validation of automated vehicles. *IEEE Access*, 9:18016–18041.

Plaku, E., Kavraki, L. E., and Vardi, M. Y. (2007). Hybrid systems: From verification to falsification. In *International Conference on Computer Aided Verification*, pages 463–476. Springer.

Plaku, E., Kavraki, L. E., and Vardi, M. Y. (2013). Falsification of ltl safety properties in hybrid systems. *International Journal on Software Tools for Technology Transfer*, 15(4):305–320.

Post, A., Hoenicke, J., and Podelski, A. (2011). rt-inconsistency: A new property for real-time requirements. In *Fundamental Approaches to Software Engineering - 14th International Conference, FASE 2011. Proceedings*, pages 34–49.

Prautzsch, H., Boehm, W., and Paluszny, M. (2013). *Bézier and B-spline techniques*. Springer Science & Business Media.

Riedmaier, S., Ponn, T., Ludwig, D., Schick, B., and Diermeyer, F. (2020). Survey on scenario-based safety assessment of automated vehicles. *IEEE Access*, 8:87456–87477.

SAE J3016_202104 (2021). Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles. Technical report, International Organisation for Standardization.

Schramm, D., Hiller, M., and Bardini, R. (2014). Single track models. In *Vehicle Dynamics*, pages 223–253. Springer.

Zowghi, D. and Gervasi, V. (2003). On the interplay between consistency, completeness, and correctness in requirements evolution. *Information & Software Technology*, 45(14):993–1009.