

Partial Consistency for Requirement Engineering with Traffic Sequence Charts

Jan Steffen Becker
OFFIS – Institute for Information Technology
Oldenburg, Germany
jan.steffen.becker@offis.de

Abstract—In the context of autonomous driving, new challenges arise also in the requirements engineering process. The specification needs to deal with a complex, open and highly dynamic context, which makes it hard to find an appropriate and still comprehensible formalism for the specification. Also, the requirements come from different sources, such as traffic regulations and consumer needs, which makes it difficult but yet important to maintain requirements consistency. This work focuses on consistency of those requirements that specify execution of driving maneuvers. In previous work, Traffic Sequence Charts have been introduced as a specification language for those requirements. Traffic Sequence Charts are a graphical formalism with a well-defined formal semantics that enables formal methods to support the development process. In this paper, a formal definition of consistency for TSCs is proposed, and its applicability in a tool-supported development process elaborated. Experimental results with a prototypical consistency analysis tool for TSCs are provided.

Index Terms—Requirements engineering, Automated driving, Formal methods, Consistency, Traffic Sequence Charts

I. INTRODUCTION

The development of a system is typically starting with a requirement elicitation process, which aims for a complete, unambiguous and conflict-free specification of the system under design. For systems like highly automated vehicles (HAV), the specification needs to deal with a complex, open and highly dynamic context, which makes it hard to find an appropriate and still comprehensible formalism for the specification. Additionally, not only system-specific requirements have to be handled, but also requirements that are induced from outside. For example, the requirements for the HAV will contain safety goals that come from hazard and risk analysis, traffic regulations, and consumer needs. This makes it hard but even more important to maintain requirement consistency, i.e. to ensure that the requirements do not conflict with each other. Although inconsistencies in the requirements will probably become visible during later development steps, finding them early saves development costs [1] and avoids implementation faults caused by conflicting specifications. This work focuses on requirements that describe the behavior of the HAV as a traffic participant, i.e. driving maneuvers by the HAV and the situations that trigger them. For that kind of requirement, Traffic Sequence Charts [2], [3] (TSCs for short) have been

This research was partly funded by the German Federal Ministry of Education and Research (BMBF), under grants “ASSUME” (01IS15031H) and “CrESr” (01IS16043).

developed as a formal specification language. The contribution of this paper is a formal but yet practical approach to find inconsistencies in TSC specifications.

Traffic Sequence Charts are a graphical specification language for traffic scenarios. As a running example, Figure 1 shows a TSC that specifies how the HAV under development (white car in the figures) shall overtake slower vehicles (black car) on a highway. The TSC is described in detail in Section II. The core idea of TSCs is to represent a scenario as a seamless sequence of traffic situations, so called snapshots. In each snapshot, the placement of symbols describes the spatial relations of traffic participants to each other and the road infrastructure. This distinguishes TSCs from other formal requirements specification languages such as ForSeL [4] which require to express spatial constraints textually. In contrast to other scenario specification formats, such as OpenSCENARIO [5], TSCs allow a high degree of freedom and abstraction. Being a graphical language with an intuitive semantics, understanding TSCs does not require deep knowledge in formal methods. Hence, a TSC can be used to display a requirement on the behavior of the driving function to different stakeholders. Furthermore, TSCs have a well-defined formal semantics that enable the use of formal methods and a wide range of tool support and automation in the development process. As such, it is possible to identify inconsistencies in TSCs automatically.

Safety standards such as ISO 26262 in the automotive industry define consistency as conflict-freeness of requirements. ISO 26262 distinguishes between internal consistency, meaning a requirement is free of contradictions within itself, and external consistency, that is a requirement does not contradict other requirements. This informal definition of consistency as conflict-freeness is too vague for the application of formal methods, though. Therefore two formal definitions of consistency, called *existential* and *partial consistency* are presented in this work. They have originally been defined and evaluated in [6], [7] for pattern-based functional requirements. In this paper, they are lifted to TSCs. In the context of TSCs, existential consistency is a form of internal consistency, and partial consistency – an extension of existential consistency – a notion of external consistency. The basic idea of existential consistency is to find at least one exemplary trajectory, i.e. a timed sequence of system states, that fulfills a TSC, or show that this is impossible. In the latter case, the TSC is called existentially inconsistent. Partial consistency does something similar for

a pair of TSCs. For partial consistency, possible cases are identified where the TSCs are concurrently active. For each of these cases it is checked whether it is possible to fulfill both TSCs. The term *partial* has been chosen, because the critical cases are found based on a heuristic (in the case of TSCs by synchronizing the start of snapshots in the different TSCs). Missing some conflicts in a specification is acceptable because experience shows that it is more important for industrial acceptance to not block the development process by reporting spurious conflicts that need to be checked manually than certifying conflict freeness [6], [8]. It is future work to extend partial consistency so that more conflicts can be found.

The paper is structured as follows. Section II introduces TSCs in detail. In Section III a generalized software architecture for consistency analysis of TSCs is proposed, together with the two concrete consistency notions for TSCs. This also includes a short report on our prototypical implementation of the approach. The paper closes with a short summary and outlook in Section IV.

II. TRAFFIC SEQUENCE CHARTS FOR REQUIREMENTS

Traffic Sequence Charts are a relatively new graphical specification language for traffic scenarios. They enable a wide range of applications in the development process, because they have an intuitive, yet formal semantics. In [9], a wide range of applications (including the use as a requirement specification language) have already been identified. Recent work [10] exploits the use of TSCs in testing. Another work [11] describes a discovery process for abstract scenarios – represented as TSCs – that exposes the violation of a safety goal. In the following, the structure and semantics for TSCs is introduced. This paper uses only a subset of the TSC language. For the complete language, refer to [2].

A. An Overview on TSCs

Figure 1 shows a TSC that specifies an overtaking scenario. Intuitively, the TSC specifies the following behavior of the HAV:

Whenever you are on the right of two lanes and there is another vehicle in front of you that is nearer than 50 meters and slower than you, perform the following operations within 40 seconds, as long as the other vehicle stays on the right lane and does not accelerate:

- 1) *Move to the left lane, accelerate until you are at least 20 km/h faster than the other vehicle, but stay behind*
- 2) *Pass the other vehicle on the left lane*
- 3) *When you are at least 50 meters in front of the vehicle, move back to the right lane*

As mentioned shortly in the introduction, the basic building block of TSCs are the so-called *snapshots*, which are the boxes numbered (1) to (6) in Figure 1. Every snapshot describes a traffic situation. Snapshots are assembled to so-called *snapshot-charts*. A TSC consists of three snapshot charts, the *history*, the *future* and the *consequence*. Informally, the

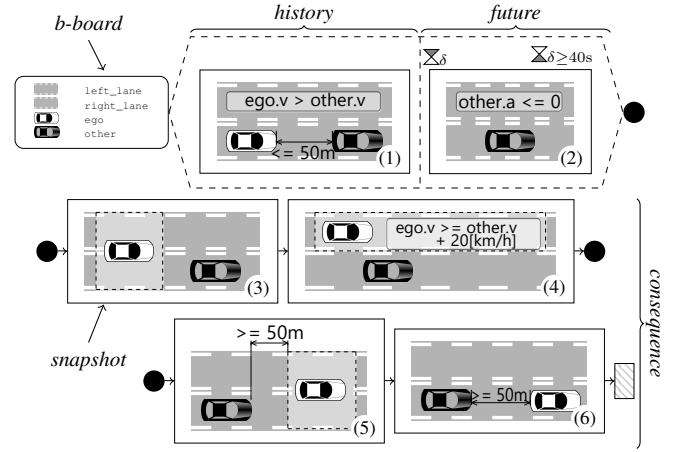


Fig. 1: TSC “Overtaking”

semantics of a TSC is as follows: Whenever the TSC’s *history* held from some point in the past until now, and the TSC’s *future* will hold from now till some point in the future, then also the *consequence* shall hold from now on. History and future form the so-called *pre-chart* of a TSC. The pre-chart is drawn as a bisected dashed hexagon, where the history is in the left, and the future is in the right part (in Figure 1 history and future consist of single snapshots). Everything that follows the pre-chart in a TSC (i.e. the second and third row of Figure 1) is part of the consequence. The rounded rectangle in Figure 1 is the *bulletin board* of the TSC and explained later.

In the following, we describe the different graphical notations that are used in snapshots. In a snapshot, symbols are placed to describe the spatial relations of their real-world counterparts. For example, the snapshots (a) and (b) in Figure 2 specify that the ego-vehicle (the HAV under development, white car) is on a lane, and that there exists a second lane with another vehicle (black car) on the left of ego. The relative placement of symbols induces somewhat strict constraints: Snapshot (a) is only satisfied in traffic situations where front and rear of the two vehicles are exactly aligned, snapshot (b) is satisfied only if the front of ego is strictly between front and rear of the other vehicle. *Somewhere-boxes* as in snapshot (c) can be used to weaken some of the constraints. The somewhere box in snapshot (c) means that there is a vehicle somewhere on the left lane, next to or in front of ego. Note that the left border of the box is aligned with the left border of the ego-symbol, so the box does not match cars on the left lane behind ego. To this end, snapshots speak only about the presence of objects at certain positions, but do not restrict the existence of other objects. For example, there could be additional cars around the ego vehicle in snapshots (a) to (c). To specify absence, *nowhere-boxes* are used. They are used analogous to somewhere-boxes, but with the opposite meaning. For example, the nowhere-box in snapshot (d) states that there must not be another vehicle on the left lane next to or in front of ego. The graphical placement of symbols in a snapshot is accomplished by textual annotations. A distance line as in snapshot (5) restricts the distance between objects and/or boxes. The distance line in snapshot (e) states that the

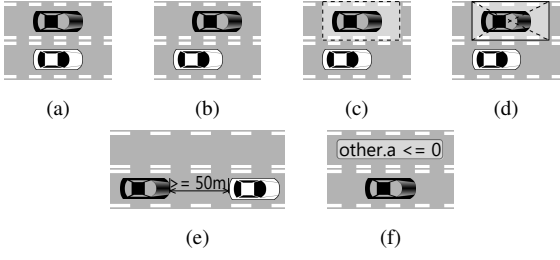


Fig. 2: Example snapshots

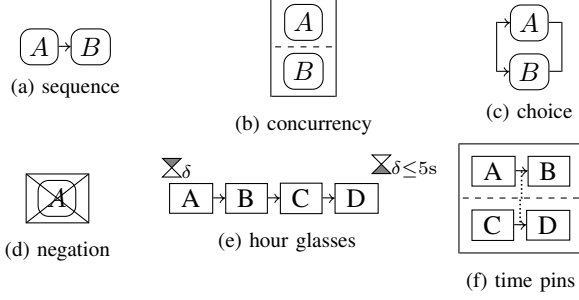


Fig. 3: Snapshot chart operations

other vehicle is at most 50 meters behind ego. Additionally, textual constraints on the objects can be placed in a snapshot, as shown in snapshot (f). Note that the empty snapshot is always satisfied.

Snapshot charts combine snapshots with the operations *sequence*, *concurrency*, *choice*, and *negation* shown in Figure 3. We use rounded rectangles (e.g.) as placeholders for arbitrary sub-charts. A sequence of two snapshots means that first the first snapshot has to hold, followed by the second. The snapshots directly follow each other; there is no time frame between the first and the second snapshot. Concurrency means that both snapshot (charts) have to hold simultaneously, choice means that at least one of the charts has to hold, respectively. Negation means the chart must *not* hold. There are two kinds of timing constraints in TSCs. *Hour glasses* constrain the duration of a snapshot chart. A full hour glass starts a clock, and the empty hour glass specifies a clock constraint. Another form of timing constraints are synchronization lines (dotted horizontal line in Figure 3 (f), also called *time-pins*). When two snapshots are connected by a synchronization line, they have to start at the same time point.

The *bulletin board* binds symbols in the snapshots to certain objects. A symbol that is contained in the bulletin board refers to the same object in every snapshot. The bulletin board (rounded box) in Figure 1 binds the white car symbol to the ego vehicle, and the lane symbols to certain lanes of the road where ego is on. Also, the black car symbol is bound to another vehicle; if the black car would not be in the bulletin board, it could be a different vehicle in any snapshot.

A TSC is always interpreted over some *world model*. As its name suggests, the world model models the different objects that a TSC can speak about. First of all, a world model specifies a hierarchy of object types (e.g. obstacles, lanes, cars,

pedestrians, ...) and their attributes. Furthermore, the world model defines their dynamics, for example in form of hybrid automata [12]. World models can be defined on different abstraction levels. Especially for analysis of requirements it might be desirable to start the development process with a very abstract world model that does not restrict the dynamics of the system, and use more detailed world models in later steps. As discussed later in Section III-A, a simpler world model lowers the complexity of an analysis and allows complete model-checking. During test-case generation for example, a more exact world model is needed.

The link between the world model and a TSC is defined in form of a so-called *symbol dictionary*. The symbol dictionary links the graphical symbols used in the TSCs to object classes in the world model (in contrast to the bulletin board, which binds symbols to certain objects). In this paper, a very simple symbol dictionary is used that consists of only a symbol for straight lanes and a symbol for cars. The car symbol is used with different colors to distinguish the different symbols in the bulletin board.

B. TSC Formal Semantics

In [2], [3] the semantics of TSCs are formally defined. In the following, we recall the semantics in a condensed form. We refer the reader to the related literature for the complete semantics.

Every snapshot chart SC can be translated to a *multi-sorted real-time logic (MSRTL) formula* φ_{SC} that is interpreted over some *world model* WM.

Definition 1 (Multi-sorted Real-time Logic [2], [3]). A *trajectory* over an (ordered) set \mathbf{X} of variables is a piece-wise continuous differentiable function $\pi: \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}^{\mathbf{X}}$. The term π^{t_0} denotes a trajectory π time-shifted by t_0 , i.e. $\pi^{t_0}(t) = \pi(t + t_0)$.

A MSRTL formula φ over variables \mathbf{X} is of the form

$$\varphi ::= \phi \mid \exists t : \varphi \mid \neg \varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \square_{[\tau_1, \tau_2]} \varphi \mid \tau_1 \sim \tau_2$$

where ϕ is a predicate over \mathbf{X} , t a time variable, $\sim \in \{<, \leq\}$, and τ_1, τ_2 are linear expressions with free time variables. *Satisfaction* of a MSRTL formula φ by a trajectory π and a valuation μ that assigns values to free time variables in φ is defined as follows:

$$\begin{aligned} \pi, \mu \models \phi & \quad \Leftrightarrow \quad \pi(0) \models \phi \\ \pi, \mu \models \square_{[\tau_1, \tau_2]} \varphi & \quad \Leftrightarrow \quad \forall t \in [\mu(\tau_1), \mu(\tau_2)] : \pi^t \models \varphi \\ \pi, \mu \models \exists t : \varphi & \quad \Leftrightarrow \quad \exists v \in \mathbb{R}_{\geq 0} : \pi, \mu \cup \{t \mapsto v\} \models \varphi \\ \pi, \mu \models \tau_1 \sim \tau_2 & \quad \Leftrightarrow \quad \mu(\tau_1) \sim \mu(\tau_2) \end{aligned}$$

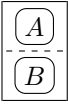
Boolean operators are defined as usual.

We write a valuation that assigns values v_1, v_2, \dots to variables t_1, t_2, \dots as $\{t_1 \mapsto v_1, t_2 \mapsto v_2, \dots\}$. The term $\mu \cup \{t \mapsto v\}$ denotes valuation μ extended by mapping $t \mapsto v$. Note that we do not define the structure of predicates ϕ explicitly here. We assume that the set \mathbf{X} in the definition is the union of the attributes of all the objects defined by the world model WM. Hence, $\pi(t)$ is the state of the world model at time t . Accordingly, the considered predicates ϕ are expressions over the world model's state.


Now, we are ready to define TSC semantics in terms of MSRTL.

Definition 2 (Snapshot Chart Translation). A snapshot chart SC is translated to a MSRTL formula φ_{SC} with free time variables b_{SC} and e_{SC} that stand for begin and end of the chart in a trajectory. In the following, $[x \setminus y]$ denotes substitution of variable x by variable y .

- If SC is a snapshot node, then $\varphi_{SC} := b_{SC} < e_{SC} \wedge \square_{[b_{SC}, e_{SC}]} \phi$, where ϕ is the translation of the snapshot content into a predicate (see [2], [3] for the translation of snapshots).

- Concurrency: If SC =  then

$$\varphi_{SC} := \varphi_A[b_A \setminus b_{SC}, e_A \setminus e_{SC}] \wedge \varphi_B[b_B \setminus b_{SC}, e_B \setminus e_{SC}]$$

- Choice: If SC =  then

$$\varphi_{SC} := \varphi_A[b_A \setminus b_{SC}, e_A \setminus e_{SC}] \vee \varphi_B[b_B \setminus b_{SC}, e_B \setminus e_{SC}]$$

- Negation: If SC =  then

$$\varphi_{SC} := \neg \varphi_A[b_A \setminus b_{SC}, e_A \setminus e_{SC}]$$

- Sequence: If SC =  then

$$\varphi_{SC} := \exists t : \varphi_A[b_A \setminus b_{SC}, e_A \setminus t] \wedge \varphi_B[b_B \setminus t, e_B \setminus e_{SC}]$$

Note, that in the above inductive definition, the substitutions $[x \setminus y]$ only replace free variables. The resulting MSRTL formula φ_{SC} has exactly two free time variables b_{SC} and e_{SC} .

Definition 3 (TSC Semantics). A TSC TSC with history H , future F , and consequence C is satisfied by some trajectory π , written $\pi \models \text{TSC}$, if

$$\forall b, m, e \in \mathbb{R}_{\geq 0} : \pi, \{b_H \mapsto b, e_H \mapsto m, b_F \mapsto m, e_F \mapsto e, b_C \mapsto m, e_C \mapsto e\} \models \neg(\varphi_H \wedge \varphi_F) \vee \varphi_C .$$

Note that the original work [2], [3] defines a more expressive MSRTL. Furthermore, [2], [3] defines the translation of timing constraints and synchronization (time pins).

Example 1. The consequence C of the TSC in Figure 1 is translated to the MSRTL formula

$$\varphi_C := \exists t_1, t_2, t_3 : b_C < t_1 < t_2 < e_C \wedge \square_{[b_C, t_1]} \phi_3 \wedge \square_{[t_1, t_2]} \phi_4 \wedge \square_{[t_2, t_3]} \phi_5 \wedge \square_{[t_3, e_C]} \phi_6$$

where ϕ_3, \dots, ϕ_6 are the predicates derived from snapshots (3)–(6).

III. PARTIAL CONSISTENCY FOR TSCs

A lot of previous work has been done that lead to formal definitions of requirement consistency. When lifting existing formal consistency notions to TSCs, care must be taken that they can be applied in practice. In order to integrate formal methods in a development process, easy to use tool support is essential. In the following, we start with proposing a

general architecture for a TSC consistency analysis tool. The architecture is independent from a concrete consistency notion, but takes the state of the art in the analysis of dense time and hybrid systems into account. Afterwards, based on previous work, two consistency notions for TSCs are developed that can be implemented within the architecture. The section concludes with a short evaluation based on a research prototype and the running example.

A. An Architecture for TSC Analysis

For consistency checking of formal requirements, symbolic model checking is the means of choice in most related work [6]–[8], [13]–[16]. An alternative approach is the simulation of the requirements in order to find conflicts [17]. Because TSCs by intention have a high degree of freedom (specifying a sequence of state invariants instead of concrete trajectories), simulation can only give incomplete results for TSCs. Except for some case studies [18], [19], model checking (and also simulation) has not been applied to TSCs so far. Also, decidability questions for TSCs have not been elaborated, yet. However, it is possible to transfer some results for hybrid automata and Duration Calculus (DC) [20] to TSCs.

Duration Calculus is an interval logic on dense time. Although TSCs are not intended to be an interval logic, there are many similarities between snapshot charts and DC formulas. Similar to snapshot charts, DC combines state invariants using boolean connectives and a sequence operator, called *chop*. Timing constraints are expressed in DC by bounding the duration of state invariants. Our research prototype that is shown briefly in Section III-D uses a combination of results from [21], [22] originally developed for DC. It translates snapshot charts into Satisfiability Modulo Theories (SMT) [23] problems and reduces consistency questions to the following satisfiability problem: Given a snapshot chart SC, do a trajectory $\pi \in \mathcal{T}(\text{WM})$ and time point $e \in \mathbb{R}_{\geq 0}$ exist such that $\pi, \{b_{SC} \mapsto 0, e_{SC} \mapsto e\} \models \varphi_{SC}$? Here and in the rest of the paper, $\mathcal{T}(\text{WM})$ refers to all the trajectories possible within a world model WM. Unfortunately, the translation to SMT problems based on [21], [22] does not allow dynamics in the world model or negated sequences in the snapshot chart, except for some special cases.

We see advantages in reducing consistency of TSCs to satisfiability of snapshot charts without negated sequences as above also for other model-checking approaches. For example, in [24] a translation from a subset of DC into hybrid automata is presented. Related work [25] gives a translation from negation-free DC into timed regular expressions, which in turn are expressible as timed automata. So we are optimistic that it is possible to define similar translations from negation-free snapshot charts into hybrid automata. This reduces the satisfiability problem of snapshots charts to reachability in hybrid automata, and allows also to use a network of hybrid automata as a world model. A wide range of model-checking tools for hybrid automata already exists, for example [26]–[29] to mention a few.

With this considerations on model checking, we propose the layered architecture depicted in Figure 4. As an input, the

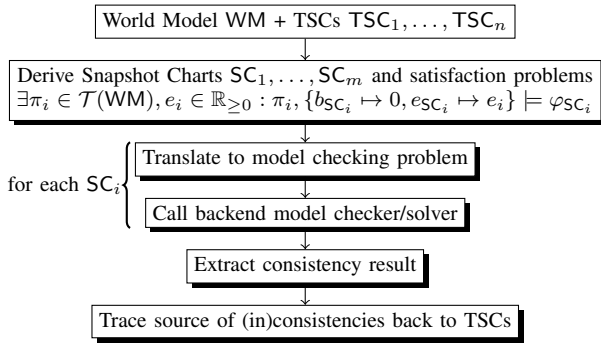


Fig. 4: Architecture for a consistency analysis

analysis takes a set of TSCs and a world model definition. The input is translated into a set SC_1, \dots, SC_n of snapshot charts and satisfaction problems of the form

$$\exists \pi_i \in \mathcal{T}(\text{WM}), e_i \in \mathbb{R}_{\geq 0} : \pi_i, \{b_{SC_i} \mapsto 0, e_{SC_i} \mapsto e_i\} \models \varphi_{SC_i}$$

for $i = 1, \dots, n$. This and the following steps are independent from the concrete consistency notion—the consistency notions defined in Sections III-B and III-C are only one possibility. The satisfaction problems are translated into an intermediate representation that is input for a model checking tool or a solver. In our research prototype, this is a translation to an SMT problem in the SMT-Lib format [30], but other formats, e.g. a network of hybrid automata, are also possible. The problem descriptions are fed into a model checker or solver backend. Finally, a consistency result is extracted from the model checker/solver results, and presented to the user. In many cases, the model checking tools give additional information on a result, that can be used to identify the cause of an inconsistency, or give a witness for consistency.

B. Existential Consistency

We propose a notion for consistency of TSCs based on previous work for pattern-based requirements [6]–[8] due to the following reasons:

- The considered pattern languages define the semantics of requirements in terms of unbounded trajectories (system runs)
- They allow “don’t care”s in the requirements
- The considered requirements are structured in a premise/consequence scheme

Here, “don’t care” means that a requirement only restricts some variables of the system, and even these restrictions may allow a high degree of freedom. For example, a TSC typically specifies the behavior of traffic participants in terms of distance intervals. Within the interval, any behavior is valid. Also, the TSCs that are considered in this paper consist of two parts: the pre-chart (history and future) and the consequence. On any sub-trajectory, where the history is followed by the future, the consequence must hold in parallel to the future. The requirements tackled in [6]–[8] are structured similar: They specify some consequence (called action in [6], [7]) that shall occur in response to some premise (called trigger in [6], [7]).

The authors of [6], [8] define consistency as follows: A set of requirements is consistent, if there exists at least one trajectory that satisfies all the requirements in the set. This is called *existential consistency* in [6]. Lifting existential consistency to TSCs means the following: A set of TSCs is existentially consistent, if there exists at least one trajectory that satisfies all the TSCs. In the following, we restrict ourselves to a set consisting of a single TSC and consider the case of multiple TSCs later. By Definition 3, a trajectory satisfies a TSC if, for any time points $b < m < e$ where the history holds from b to m and the future holds from m to e , the consequence holds from t to e . This is problematic for two reasons: On the one hand, the property is difficult to check, because there are typically infinitely many triples (b, t, e) on a trajectory where history and future hold. On the other hand, we may choose a trajectory for witness, where history and future are never satisfied at all, in which case the consistency argument is not of practical use. The authors of [6], [8] had to deal at least with the second problem. They solved this by requiring that the premise of the requirements shall be satisfied at least once on the witness trajectory. When speaking about TSCs, this means that we search for a trajectory $\pi \in \mathcal{T}(\text{WM})$ where the history occurs, directly followed by the future. Formally

$$\exists b, m, e \in \mathbb{R}_{\geq 0} : \pi, \{b_H \mapsto b, e_H \mapsto m, b_F \mapsto m, e_F \mapsto e\} \models 0 < b < m < e \wedge \varphi_H \wedge \varphi_F \quad (1)$$

where H and F are history and future of the TSC. The inequality $0 < b$ in that formula allows us to prefix the sequence of H and F with an empty snapshot and to encode it as a single chart $\text{HF} = \boxed{} \rightarrow \boxed{H} \rightarrow \boxed{F}$. Because of the empty snapshot, we can fix the start of HF to 0, while the sub-chart H still starts at a time point $b_H > 0$. Because a snapshot by definition has a non-zero duration, $b_H = 0$ is excluded. As a result we can rewrite (1) as

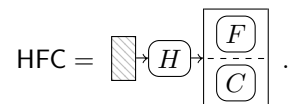
$$\exists e \in \mathbb{R}_{\geq 0} : \pi, \{b_{\text{HF}} \mapsto 0, e_{\text{HF}} \mapsto e\} \models \varphi_{\text{HF}} \quad (2)$$

As a consistency argument, we would like to see that it is at least possible to satisfy the consequence C of the TSC in parallel to the future. This brings us to our first consistency notion for TSCs that we call *existential consistency for TSCs*.

Definition 4 (Existential Consistency). A TSC with history H , future F , and consequence C is *existentially consistent* wrt. the world model WM if there exists a trajectory $\pi \in \mathcal{T}(\text{WM})$ such that

$$\exists e \in \mathbb{R}_{\geq 0} : \pi, \{b_{\text{HFC}} \mapsto 0, e_{\text{HFC}} \mapsto e\} \models \varphi_{\text{HFC}} \quad (3)$$

with the snapshot chart



Fact 1. Assume some TSC TSC with history H , future F and consequence C that is interpreted over some world model WM. A trajectory $\pi \in \mathcal{T}(\text{WM})$ that satisfies both $\pi \models \text{TSC}$ and equation (2), also satisfies equation (3).

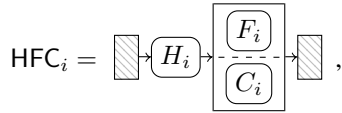
This fact follows with some basic algebra from Definitions 1 to 3. The consequence of the fact is that existential consistency does not produce false warnings, in the sense that if a TSC is existentially inconsistent, then it is indeed impossible within the world model to find a trajectory that both satisfies the TSC and where history and future occur. Note that the opposite is not true. A trajectory that satisfies equation (3) does not necessarily satisfy the TSC. However, using this under-approximation of a satisfying trajectory we get rid of the all-quantifier in the formal definition of satisfaction of a trajectory by a TSC. Also, we can restrict ourselves to search for a prefix of a trajectory, instead of a complete trajectory, which would require some kind of unbounded model checking techniques in a consistency analysis.

C. From Existential to Partial Consistency

Existential consistency as defined in the last section does not give meaningful results for sets of TSCs in most cases. If we build the conjunction of equation (3) over multiple TSCs, e.g. searching for a trajectory $\pi \in \mathcal{T}(\text{WM})$ that satisfies

$$\exists v_e \in \mathbb{R}_{\geq 0}: \pi, \{b \mapsto 0, e \mapsto v_e\} \models \bigwedge_{i=1}^n \varphi_{\text{HFC}_i}[b_{\text{HFC}_i} \setminus b, e_{\text{HFC}_i} \setminus e]$$

with



we end up in most cases with a trajectory where the futures of the different TSCs do not overlap. The result is not different than checking existential consistency of each TSC separately. In practice, the future of a TSC usually speaks about some behavior of the environment that is not under control of the ego vehicle. At least it is not the intended strategy for a controller to satisfy its specification by forcing a violation of history or future of the TSCs in the specification. As a consequence, a consistency analysis is needed that considers the possible overlappings between history and/or future of different TSCs. Therefor we develop a second consistency notion that is called *partial consistency for TSCs*. It is inspired by the partial consistency for pattern-based requirements defined in [7]. The basic ideas are the following:

- 1) Consider pairs of TSCs.
- 2) For every pair, determine the possible ways how the futures of the two TSCs can overlap. We consider cases where the future of the second TSC starts during the consequence of the first one.
- 3) For every possible overlapping, check if it is possible to satisfy both consequences.

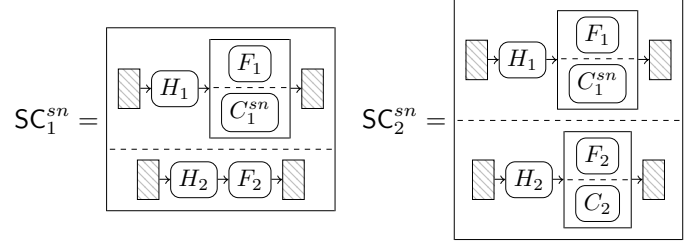
Obviously, not all inconsistencies within a set of TSCs can be reduced to conflicts between pairs, therefor a possible generalization is briefly sketched at the end of this section.

Definition 5 (Partial Consistency). A pair TSC₁, TSC₂ of TSCs is *partially consistent* wrt. some world model WM if

the following implication holds for every snapshot sn in the consequence C_1 of the first TSC:

$$(\exists \pi_1 \in \mathcal{T}(\text{WM}), e_1 \in \mathbb{R}_{\geq 0}: \pi_1, \{b_{\text{SC}_1} \mapsto 0, e_{\text{SC}_1} \mapsto e_1\} \models \varphi_{\text{SC}_1}) \Rightarrow (\exists \pi_2 \in \mathcal{T}(\text{WM}), e_2 \in \mathbb{R}_{\geq 0}: \pi_2, \{b_{\text{SC}_2} \mapsto 0, e_{\text{SC}_2} \mapsto e_2\} \models \varphi_{\text{SC}_2})$$

with the snapshot charts



where H_i , F_i and C_i are history, future and consequence of TSC _{i} ($i \in \{1, 2\}$). The snapshot chart C_1^{sn} is derived from C_1 by removing all branches of choice operations that bypass sn .¹ In both SC_1^{sn} and SC_2^{sn} , the start of F_2 is synchronized with the start of sn in C_1^{sn} . Removing some choices from C_1 is necessary because it is neither allowed in TSCs nor makes sense to synchronize sub-charts if one of them is part of a choice.

Example 2. Consider the TSCs from Figure 1 and Figure 5. The latter says, that the ego vehicle must stay on the right lane and must not accelerate, as long as there is another vehicle on the left lane, near ego. They are partially inconsistent. Intuitively, the conflict arises when the ego vehicle approaches another car while a third vehicle is on the left lane. The first TSC says ego shall overtake, while the second says ego must keep right. The partial consistency check for these two TSCs consists of five cases (four cases where the future of the second TSC is synchronized with one of the snapshots (3)–(6) from the first TSC's consequence, and one where the future of the first TSC is synchronized with the consequence of the second). The case where the future of the TSC in Figure 5 is synchronized with snapshot (4) from the TSC in Figure 1 reveals the inconsistency. The consequence SC_2^4 of that case is shown in Figure 6. The premise SC_1^4 for that case is identical to SC_2^4 , except that the snapshot (7) is omitted. The premise SC_1^4 is satisfiable, which means it is possible to satisfy all parts of the first TSC (Figure 1) while the future of the second TSC (snapshot (7) in Figure 5) starts together with snapshot (4) from the first TSC. An example of a satisfying trajectory is depicted in Figure 7 as a sequence of snapshots. However, the consequence SC_2^4 is not satisfiable, which means it is not possible to find such a trajectory that also satisfies the consequence of the second TSC. In SC_2^4 , snapshots (4) and (8) must be fulfilled concurrently, but are clearly conflicting: In (4), ego is on the left lane, while in (8) it is on the right lane.

The case construction for partial consistency of two TSCs can be generalized to three TSCs by combining SC_1^{sn} and SC_2^{sn}

¹For example, if $C_1 = \begin{matrix} \boxed{a} \\ \boxed{b} \end{matrix} \rightarrow \boxed{c}$ then $C_1^a = \boxed{a} \rightarrow \boxed{c}$ and $C_1^b = \boxed{b} \rightarrow \boxed{c}$ and $C_1^c = C_1$.

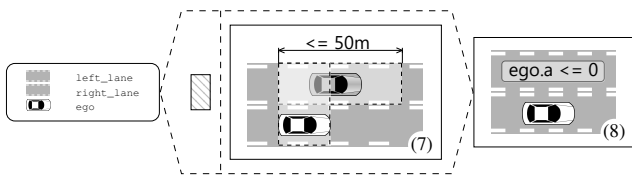
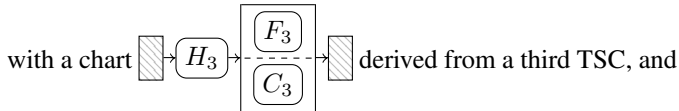


Fig. 5: A TSC that prohibits lane changes and acceleration while there is another vehicle around ego. Note the overlapping somewhere-boxes.



synchronizing the start of F_1 in SC_1^{sn} (resp. SC_2^{sn}) with one snapshot from C_3 , or synchronizing F_3 with a snapshot from C_1^{sn} . This way, the consistency cases for a group of n TSCs can inductively be derived from the cases for $n - 1$ TSCs.

D. Prototypical Implementation

We evaluated the above consistency notions in a prototypical implementation that has been partially already described in Section III-A. The consistency analysis uses the Z3 SMT solver [31] in version 4.6.0. For the existential and partial consistency analysis, a translation from snapshot charts to SMT is used that is based on a combination of results from [21], [22]. The layered architecture described in Section III-A made it easy to implement both consistency notions and a few similar ones using a common analysis code base. The prototype needs less than 12 seconds to analyze the partial consistency from Example 2 running on a Windows 7 desktop PC with an Intel Core i5-2400 CPU @ 3.1 GHz and 4GB RAM². The translation procedure derived from [21], [22] cannot handle differential equations currently, and does not distinguish between continuous and discrete variables.

IV. CONCLUSION & FUTURE WORK

In this paper, two formal consistency notions for TSCs have been presented. Beneath TSCs, a bunch of other scenario specification languages exist [32]–[34], the most popular one is probably the standardized exchange format OpenSCENARIO [5]. In contrast to other approaches, TSCs are both a graphical language and have a well-defined formal semantics. A TSCs define scenarios in an abstract way, i.e. one TSC captures infinitely many concrete scenarios. Therefore, TSCs are suitable as a requirement specification language for HAV. By providing formal consistency notions for TSCs, this work is a first step towards tool supported formal methods for requirements engineering with TSCs. In [19] a somewhat simpler form of TSCs are used to generate concrete scenarios for testing. The authors of [19] observe that solving constraint systems for abstract scenarios in a hybrid world model is a challenging task, even for simple scenarios. In contrast to test case generation, where the concrete scenarios need to be as close to real world behavior

²The SMT scripts for the example are provided for download under <https://vhome.offis.de/jbecker/ase2020.zip>

as possible, consistency of TSCs can be checked on different abstraction levels. The consistency notions defined here are independent from a concrete world model. The examples presented in this paper show that an analysis implementation that abstracts from any differential equations in a hybrid world model already gives meaningful consistency results and finds conflicts in a specification. Performing consistency analysis on an abstract world model is not only caused by the limitations of current model checkers, but also driven by the fact that detailed world models may not be present in early development phases, where a consistency analysis is typically performed. Nonetheless our first experiments with the consistency analysis showed that a consistency analysis on an abstract level cannot find all the conflicts in a specification. In the running example (see Figure 1), it might be impossible for the ego vehicle to accelerate and to be at least 20km/h faster before passing the other vehicle. This conflict is not found because it depends both on the vehicle speeds at the beginning of the scenario and the possible vehicle dynamics in the world model. Therefore future research directions are to (1) find more sophisticated consistency notions that find more conflicts even on a high abstract level (2) investigate into analysis methods for hybrid systems and how they can be used for consistency analysis of TSCs.

REFERENCES

- [1] P. Feiler, L. Wrage, and J. Hansson, “System architecture virtual integration: A case study,” in *Embedded Real-time Software and Systems Conference*, 2010.
- [2] W. Damm, S. Kemper, E. Möhlmann, T. Peikenkamp, and A. Rakow, “Traffic sequence charts - from visualization to semantics,” SFB/TR 14 AVACS, Reports of SFB/TR 14 AVACS 117, 10 2017.
- [3] W. Damm, E. Möhlmann, T. Peikenkamp, and A. Rakow, *A Formal Semantics for Traffic Sequence Charts*. Cham: Springer International Publishing, 2018, pp. 182–205.
- [4] J. Hartmann, S. Rittmann, D. Wild, and P. Scholz, “Formal incremental requirements specification of service-oriented automotive software systems,” in *2006 Second IEEE International Symposium on Service-Oriented System Engineering (SOSE’06)*. IEEE, 2006, pp. 130–133.
- [5] VIRES Simulationstechnologie GmbH, “OpenSCENARIO,” online, 2018, last visited on 2019-07-01. [Online]. Available: <http://openscenario.org>
- [6] C. Ellen, S. Sieverding, and H. Hungar, “Detecting consistencies and inconsistencies of pattern-based functional requirements,” in *Formal Methods for Industrial Critical Systems, FMICS 2014, Proceedings*, ser. LNCS, vol. 8718. Springer, 2014, pp. 155–169.
- [7] J. S. Becker, “Analyzing consistency of formal requirements,” in *18th International Workshop on Automated Verification of Critical Systems*, 2018.
- [8] P. Filipovikj, G. Rodriguez-Navas, M. Nyberg, and C. Seceleanu, “SMT-based consistency analysis of industrial systems requirements,” in *Proceedings of the Symposium on Applied Computing*. ACM, 2017, pp. 1272–1279.
- [9] W. Damm, S. Kemper, E. Möhlmann, T. Peikenkamp, and A. Rakow, “Traffic sequence charts: A visual language for capturing traffic scenarios,” in *Embedded Real Time Software and Systems - ERTS2018*, February 2018.
- [10] W. Damm, E. Möhlmann, and A. Rakow, “Traffic sequence charts for the ENABLE-S3 test architecture,” in *Validation & Verification of Automated Systems – Results of the ENABLE-S3 Project*, 2019.
- [11] —, “A scenario discovery process based on traffic sequence charts,” in *Validation & Verification of Automated Systems – Results of the ENABLE-S3 Project*, 2019.
- [12] R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho, “Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems,” in *Hybrid systems*. Springer, 1992, pp. 209–229.

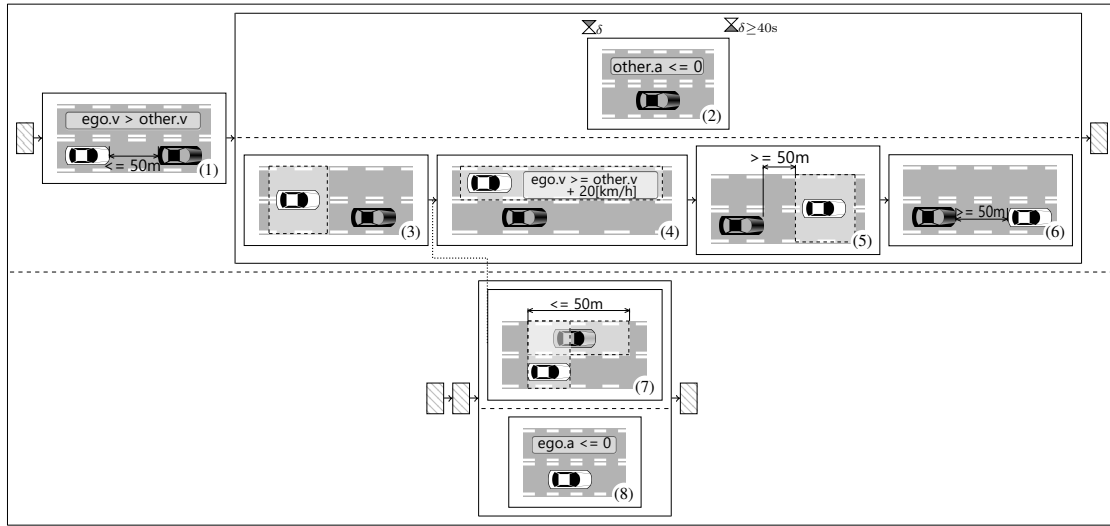


Fig. 6: Consequence (SC_2) for the partial inconsistency

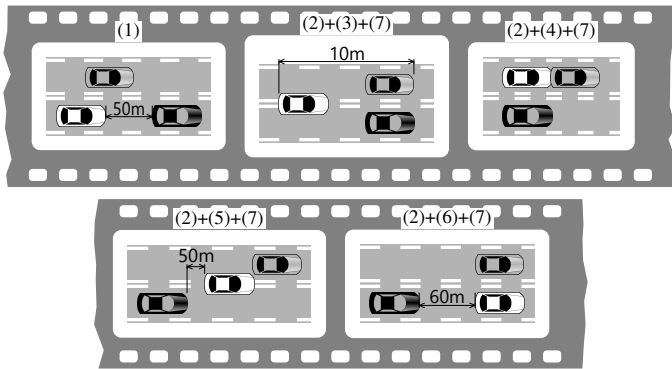


Fig. 7: Illustration of a trajectory that satisfies SC_1 . The numbers above the snapshots show which snapshot nodes from Figure 6 are satisfied in each step.

[13] A. Post, I. Menzel, J. Hoenicke, and A. Podelski, "Automotive behavioral requirements expressed in a specification pattern system: a case study at BOSCH," *Requirements Engineering*, vol. 17, no. 1, pp. 19–33, 2012.

[14] A. Post, J. Hoenicke, and A. Podelski, "rt-inconsistency: A new property for real-time requirements," in *Fundamental Approaches to Software Engineering - 14th International Conference, FASE 2011. Proceedings*, 2011, pp. 34–49.

[15] T. Bienmüller, T. Teige, A. Eggers, and M. Stasch, "Modeling requirements for quantitative consistency analysis and automatic test case generation," in *FM&MDD 2016*, ser. Computing Science Technical Report Series, vol. CS-TR-1503. Newcastle University, 2016.

[16] B. K. Aichernig, K. Hörmaier, F. Lorber, D. Ničković, and S. Tiran, "Require, test and trace IT," in *Formal Methods for Industrial Critical Systems - 20th International Workshop, FMICS 2015, Proceedings*, ser. LNCS, vol. 9128. Springer, 2015, pp. 113–127.

[17] B. Jeannot and F. Gaucher, "Debugging embedded systems requirements with STIMULUS: an automotive case-study," in *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*, 2016.

[18] S. Gerwin, E. Möhlmann, and A. Sieper, "Statistical model checking for scenario-based verification of ADAS," in *Control Strategies for Advanced Driver Assistance Systems and Autonomous Driving Functions*. Springer, 2019, pp. 67–87.

[19] A. Eggers, M. Stasch, T. Teige, T. Bienmüller, and U. Brockmeyer, "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*, 2018.

[20] Z. Chaochen, C. Hoare, and A. Ravn, "A calculus of durations," *Information Processing Letters*, vol. 40, pp. 269–276, 12 1991.

[21] M. Fränzle and M. R. Hansen, "Deciding an interval logic with accumulated durations," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, ser. LNCS, vol. 4424. Springer, 2007, pp. 201–215.

[22] B. Sharma, P. K. Pandya, and S. Chakraborty, "Bounded validity checking of interval duration logic," in *TACAS*, vol. 5. Springer, 2005, pp. 301–316.

[23] C. W. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli, "Satisfiability modulo theories," *Handbook of satisfiability*, vol. 185, pp. 825–885, 2009.

[24] A. Bouajjani, Y. Lakhnech, and R. Robbana, "From duration calculus to linear hybrid automata," in *LNCS*, vol. 939, 07 1995, pp. 196–210.

[25] M. Fränzle, "Model-checking dense-time duration calculus," *Formal Asp. Comput.*, vol. 16, no. 2, pp. 121–139, 2004.

[26] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi, "Hytech: A model checker for hybrid systems," *International Journal on Software Tools for Technology Transfer*, vol. 1, no. 1-2, pp. 110–122, 1997.

[27] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler, "SpaceEx: Scalable verification of hybrid systems," in *International Conference on Computer Aided Verification*. Springer, 2011, pp. 379–395.

[28] A. Eggers, N. Ramdani, N. Nedialkov, and M. Fränzle, "Improving SAT modulo ODE for hybrid systems analysis by combining different enclosure methods," in *International Conference on Software Engineering and Formal Methods*. Springer, 2011, pp. 172–187.

[29] E. Althaus, B. Beber, W. Damm, S. Disch, W. Hagemann, A. Rakow, C. Scholl, U. Waldmann, and B. Wirtz, "Verification of linear hybrid systems with large discrete state spaces using counterexample-guided abstraction refinement," *Science of Computer Programming*, vol. 148, pp. 123–160, 2017.

[30] D. R. Cok, *The SMT-LIBv2 Language and Tools: A Tutorial*, 1st ed., 11 2013. [Online]. Available: <http://smtlib.github.io/jSMTLIB/SMTLIBTutorial.pdf>

[31] L. de Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Tools and Algorithms for the Construction and Analysis of Systems: 14th International Conference, TACAS 2008. Proceedings*. Springer, 2008, pp. 337–340.

[32] P. Suresh and R. Mourant, "A tile manager for deploying scenarios in virtual driving environments," in *Driving Simulation Conference*, 2005, pp. 21–29.

[33] I. H. C. Wassink, E. M. A. G. van Dijk, J. Zwiers, A. Nijholt, J. Kuipers, and A. O. Brugman, *Bringing Hollywood to the Driving School: Dynamic Scenario Generation in Simulations and Games*. Springer, 2005, pp. 288–292.

[34] J. Kearney, P. Willemsen, S. Donikian, and F. Devillers, "Scenario languages for driving simulation," in *Driving Simulation Conference, DSC'99*, 1999, pp. 377–393, [Online]. Available: www.cs.uiowa.edu/~kearney/pubs/dsc99_kearney.pdf.