Electronic Communications of the EASST
Volume 75 (2018)

43rd International Conference on Current Trends in Theory and Practice of Computer Science

-

Student Research Forum, 2017 (SOFSEM SRF 2017)

Model-Checking-based vs. SMT-based Consistency Analysis of Industrial Embedded Systems Requirements: Application and Experience

Predrag Filipovikj, Guillermo Rodriguez-Navas and Cristina Seceleanu

20 pages

Guest Editors: Anila Mjeda
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Model-Checking-based vs. SMT-based Consistency Analysis of Industrial Embedded Systems Requirements: Application and Experience

Predrag Filipovikj[1], **Guillermo Rodriguez-Navas**[1] **and Cristina Seceleanu**[1]

predrag.filipovikj@mdh.se, guillermo.rodriguez-navas@mdh.se, cristina.seceleanu@mdh.se
[1] School of Innovation, Design and Technology
Mälardalen University, Sweden

Abstract: Industry relies predominantly on manual peer-review techniques for assessing the correctness of system specifications. However, with the ever-increasing size, complexity and intricacy of specifications, it becomes difficult to assure their correctness with respect to certain criteria such as *consistency*. To address this challenge, a technique called *sanity checking* has been proposed. The goal of the technique is to assess the quality of the system specification in a systematic and rigorous manner with respect to a formally-defined criterion. Predominantly, the sanity checking criteria, such as for instance consistency, are encoded as reachability or liveness properties which can then be verified via model checking. Recently, a complementary approach for checking the consistency of a system's specification by reducing it to a satisfiability problem that can be analyzed using Satisfiability Modulo Theories has been proposed. In this paper, we compare the two approaches for consistency analysis, by applying them on a relevant industrial use case, using the same definition for consistency and the same set of requirements. Since the bottlenecks of analyzing large systems formally are most often the construction of the model and the time needed to return a verdict, we carry out the comparison with respect to the: i) required effort for generating the analysis model and the latter's complexity, and ii) consistency analysis time. Assuming checking only invariance properties, our results show no significant difference in analysis time between the two approaches when applied on the same system specification under the same definition of consistency. As expected, the main difference between the two comes from the required time and effort of creating the analysis models.

Keywords: SMT-based consistency analysis, model-checking-based consistency analysis

1 Introduction

A system's requirements specification, often simply called *system specification*, represents one of the most important artifacts during the system's development life cycle, especially for software systems. When a traditional development process like the waterfall model (W-model) [Roy87] is followed, the system specification is created in the early stages of the development and is used as a basic input in the subsequent software development phases including the design and

architecture, implementation, and verification and validation. It is known that errors introduced in the specification propagate into the subsequent artifacts, until they are eventually detected and corrected, incurring considerable costs. For this reason, it is essential to ensure that the system specification has good quality, that is, it is free from errors like *inconsistency* (internal logical contradictions).

Assuming checking only invariance properties, the traditional, and still predominant way for assessing the correctness of a system specification is through manual peer-review. However, in areas such as embedded systems, where the system specifications are growing in size, complexity and intricacy, a number of methods for formal and automated (computer-aided) verification of system specifications have been proposed [HJL96, PHP11b, MSL16, FRNS17]. In order to analyze a system specification by using formal methods, first, the systems' behavior has to be formally encoded, that is, expressed in some formal notation. Second, one has to formally define the quality assessment criterion, and third, propose an automated technique that can be employed to assess the formally-encoded system specification with respect to the formally defined criterion. Kupferman [Kup06] coined the term *sanity checking* to denote this process. In the literature, there are many criteria for assessing the quality of a system's specification, out of which *consistency* (lack of contradicting formulas within a specification) [HL96, HJL96, PHP11b], is the one that we focus in this paper.

The sanity checking with respect to consistency is suitable for assessing the internal quality of the specification without requiring a more detailed system model. This type of sanity checking for consistency that does not require a structural or functional model of the system is called model-free checking [BBB12]. The benefits of the model-free sanity checking are mainly given by the possibility of detecting errors in specifications, at early phases of development.

Existing approaches reduce the consistency checking to a reachability or liveness property that is then assessed via model checking [CGP99]. Assuming a simple definition of requirements consistency, in our previous work [FRNS17, FRNS18] we have proposed a Satisfiability Modulo Theories (SMT)-based [DB11] approach for checking the consistency of system specifications formalized as sets of Timed Computation Tree Logic (TCTL) [ACD93] formulas. According to the assumed definition, the notion of consistency is reduced to checking whether the system specification is realizable as such, that is, whether there exists at least one consistent interpretation of the specification, consequently reducing the consistency checking to a satisfiability problem. The proposed approach is restricted to invariance properties, denoted as $AG(\varphi)$ in TCTL, where φ is a formula that contains predicates and nested operators. The decision to resort to SMT for assessing the specification's consistency is motivated by the following: i) the realizability of a set of formulas can be reduced to a constraint satisfiability problem, and ii) it might be efficient for consistency analysis of large sets of requirements. The proposed SMT-based consistency analysis procedure as reported previously [FRNS17] shows a considerable reduction of the analysis time over the model-checking-based one as reported in literature [BBB+16]. However, since the notion of consistency differs between the SMT-based and the model-checking-based one, comparing them as such is not meaningful.

In this paper, we chose to compare the model-checking-based and the SMT-based consistency analysis techniques, by applying them on the same industrial example from the automotive domain, namely the Fuel Level Display (FLD) system from Scania, using the same definition of consistency over a set of invariance ($AG(\varphi)$) properties as defined above. For the comparison of

the approaches, we consider two criteria, usually seen as the bottlenecks of formal analysis: i) the required effort to create the analysis model, and the resulting complexity of the latter, and ii) the required analysis time. Additionally, we discuss the potential for adoption of both approaches in industrial settings. The SMT-based approach is adopted as such [FRNS17], whereas for the model-checking-based one, we propose a transformation scheme for the restricted set of system specifications encoded in TCTL, into a timed automata (TA) [AD94] model suitable for analysis with the UPPAAL [LPY97] model checker. Then, we manually build the model, and analyze it with UPPAAL. Next, we compare the efficiency of the method to our findings of applying the SMT-based approach, on the same industrial system [FRNS17].

The rest of the paper continues as follows. In Section 2, we give an overview of the preliminary concepts that are used in the remainder of the paper. Next, in Section 3, we introduce the FLD system used as the running example on which both approaches are applied, after which a short overview of the SMT-based approach is given in Section 4. In Section 5, we show how to transform a set of TCTL properties of type $AG(\varphi)$ into a TA model suitable for analysis with UPPAAL. After this, we compare the results from applying both approaches of analyzing the consistency of the FLD specification, in Section 6. We present an overview and comparison with related approaches in Section 7. Finally, we summarize the paper by providing concluding remarks and setting some directions for future work, in Section 8.

2 Preliminaries

In this section, we introduce the preliminary concepts that are used in the rest of the paper. First, we present an overview of Timed Computation Tree Logic, which is the formalism that we use to formalize the system specification. Then, we present the formal definition for consistency that we use in this paper. Finally, we give an overview of the techniques and tools that we use to check the consistency of the formalized system specification, namely Satisfiability Modulo Theories (SMT) and the Z3 tool, as well as model checking and the UPPAAL tool.

2.1 Formal System Specifications in Timed Computation Tree Logic

In this work, we use Timed Computation Tree Logic (TCTL) [ACD93] to formalize the FLD system requirements specification. TCTL is a timed extension of Computation Tree Logic (CTL) [CE82] suitable for the specification of real-time systems. TCTL is interpreted over a branching-time model (timed transition system) that consists of a non-empty set of states S, a successor relation R and a labeling function *Label* that assigns a set of atomic propositions to locations. Each state is defined as a pair (l,u), where l represents the set of valid propositions for that state, whereas u represents the valuation of the clock variables that measure the passage of time in the model.

The syntax of TCTL consists of path quantifiers (*"A", "E"*), and path-specific temporal operators. The universal path quantifier *"A"* stands for *"all paths"*, while the existential quantifier *"E"* denotes that *"there exists a path"* from the set of all future paths $P_M(s)$ starting from a given state s. A valid TCTL formula is of type $\varphi\, U_{\bowtie T}\, \psi$, where φ, ψ are state predicates, U (*"until"*) represents the basic path-specific temporal operator, which is combined with either one of the

two path quantifiers, $\bowtie \in \{<, \leq, =, \geq, >\}$, and T is a numeric bound on clock variables. The rest of the path-specific temporal operators are defined based on the U operator. The *Future* operator, denoted as $F_{\bowtie T}$ (alternative notation $\Diamond_{\bowtie T}$), is interpreted as *eventually* true along a given path, when the clock valuation satisfies $\bowtie T$ (e.g., $F_{\bowtie T}\varphi \Leftrightarrow true\, U_{\bowtie T}\, \varphi$, with φ a state predicate). The path-specific temporal operator that we use in our formal requirements, is called *globally*, represented as $G_{\bowtie T}$ (alternative notation $\Box_{\bowtie T}$) denotes *always* true in all states along a given path in which the clock valuation satisfies $\bowtie T$ (e.g., $G_{\bowtie T}\varphi \Leftrightarrow \neg F_{\bowtie T}\neg\varphi$). There exists a weaker version of the U operator called *"weak-until"* denoted as $W_{\bowtie T}$, which is used to capture formulas where the right hand side term might never be satisfied. The semantics of $W_{\bowtie T}$ is defined as: $\varphi\ W_{\bowtie T}\ \psi \equiv (\varphi\ U_{\bowtie T}\ \psi) \vee G_{\bowtie T}\varphi$. If one pairs G with the universal path quantifier A, one gets properties of the form AGφ, called invariance properties, where φ is combination of predicates and nested operators. They should be fulfilled at any state along any execution of the system model. This is the class of properties of interest in this work.

2.2 Consistency of System Specifications

To assess the consistency of a system's specification encoded as a set of logical formulas, we use the following definition:

Definition 1 (Inconsistent specification) Let $\Phi = \{\varphi_1, \varphi_2, ..., \varphi_n\}$ denote the system requirements specification, where each of the formulas $(\varphi_1, \varphi_2, ..., \varphi_n)$ encodes a requirement in the original system specification. We say that the set Φ is inconsistent if the following implication is satisfied: $\varphi_1 \wedge \varphi_2 \wedge ... \wedge \varphi_n \implies False$.

According to Definition 1, a system's requirements specification is inconsistent if there does not exist a valuation of the variables in $\varphi_1, \varphi_2, ..., \varphi_n$, such that the conjunction of all formulas evaluates to true. Conversely, a specification is consistent if there exists at least one valuation of the variables such that the conjunction of all the formulas is true. This restricted definition of consistency refers to internal consistency of requirements, which is equivalent to the lack of logical contradictions.

2.3 Satisfiability Modulo Theories and Z3

The problem of determining whether a formula expressing constraints (equality, inequality, etc.) has a solution is called *constraint satisfiability* problem. The best-known constraint satisfiability problem is the propositional satisfaction *SAT*, where the problem is to decide if a formula over Boolean variables, formed using logical connectives can become `true` by assigning `false`/`true` values to the constituent variables.

To express the requirements as constraints, in this paper we use *first-order logic*. A first-order logic formula is a logical formula formed using logical connectives, variables, quantifiers and function and predicate symbols. A solution of first-order logic formulas is a *model*, which in fact is an interpretation of the variables, function and predicate symbols that makes the given formulas `true`. For checking satisfiability of formulas that contain additional theories, as for instance theory of arithmetics, we use Satisfiability Modulo Theories (SMT) [DB11], which form an

extension of the classical SAT problem over first-order logic formulas, where the interpretation of some symbols is constrained by a background theory.

In this work, we use Z3 [DB08], which is a state-of-the-art SMT solver and theorem prover developed and maintained by the Microsoft RiSE group. The advantage of Z3 is that it has a stable group of developers who maintain the tool, as well as a broad academic community that is actively using it. The input to the tool is a set of *assertions* that can be either declarations or formulas. Originally, the assertions are specified using the SMT-LIB language [BFT15]. Declarations in Z3 can be either *constants* or *functions*. In fact, in Z3 everything is expressed as functions, with constants being functions with no input parameters. The set of predefined types in Z3 includes: `Int, Real, Bool` and `Function`. The set of supported types can be extended with user-defined types. Z3 supports two types of quantifiers: universal quantifier (`ForAll`) and existential quantifier (`Exists`). For optimizing decision procedures, the tool uses a number of tactics.

The set of formulas whose satisfiability is to be checked by the tool is kept on the internal stack. The command `assert` adds a new formula to the stack. The SMT decision procedure is invoked by executing the command `check-sat`, which checks where there is a solution for the conjunction of all the assertions on the stack. If the set of assertions is satisfiable, the Z3 tool returns the result `SAT`, which can be accompanied by the model that contains the witness assignment of the variables. The model is generated using the command `get-model`. In the opposite case, that is, when the set of assertions on the stack is not satisfied, the tool returns `UNSAT`, together with a *minimal* set of inconsistent assertions.

2.4 Model Checking

Model checking [CGP99] is an automated formal verification technique that checks, in a systematic and exhaustive manner, whether a finite-state system model satisfies a given property expressed most often in temporal logic.

The core of model checking is the verification algorithm, performed by the model checker. The input to the model checker is a system model expressed in a formal notation and a set of formally specified logical properties. For verification of qualitative properties (that admit a yes/no answer) there are two possible outcomes of the model checking procedure. If the model conforms to a given property, the model checker returns a positive answer. For reachability and some liveness properties (e.g., something good will eventually happen) the model checker returns a witness trace in case of fulfillment. When an invariance property is not satisfied, the model checker generates a counter example, which is usually a path (error trace) to the state that violates the property.

2.4.1 Timed Automata and UPPAAL

In our work, we use UPPAAL [LPY97], the state-of-the-art model checker for real-time systems. UPPAAL provides an integrated environment for modeling, simulation and verification of real-time systems. The UPPAAL model checker accepts formal system models specified as *timed automata* [AD94], and a set of properties that the formal model is checked against, specified in TCTL [ACD93].

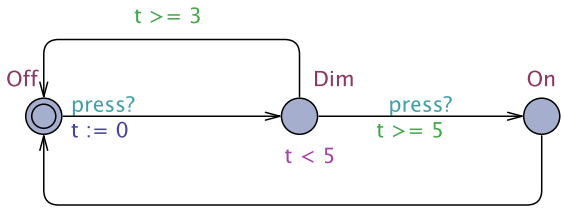

Figure 1: A timed automaton for a timer-based lamp.

Timed automata (TA) is an extension of finite-state automata with real-valued variables called *clocks*, suitable for modeling the behavior of real-time systems. The clocks are non-negative variables that grow at a fixed rate of 1, and capture the passage of time.

A timed automaton is represented by a tuple, $\langle L, l_0, X, V, A, \Sigma, I \rangle$, where: L is the set of locations, $l_0 \in L$ is the initial location, X is the set of clocks, V is the set of data variables, A represents the set of actions, $\Sigma \subseteq L \times A \times B(X,V) \times 2^X \times L$ is the set of edges, where each edge $e = (l, g, a, r, l') \in \Sigma$ is characterized by a source location l, and a sink location l', respectively, a guard g, action label a, and a set of clocks (r) that are reset when the edge is traversed, and $B(X,V)$ represents the constraints over the clock and data variables, respectively. $I : L \rightarrow B(X)$ is a function that assigns invariants to locations, which bound the allowed time in a particular location. $B(X)$ represents the set of formulas of the form: $x \bowtie c$ or $x - z \bowtie c$ called clock constrains, where $x, z \in X$, $c \in N$ and $\bowtie \in \{<, \leq, =, \geq, >\}$. A clock constraint is downwards closed if $\bowtie \in \{<, \leq, =\}$. Similarly, $B(V)$ denotes the set of non-clock constraints that are conjunctive formulas of type $i \sim j$ or $i \sim k$, where $i, j \in V$, $k \in \mathbb{Z}$ and $\sim \in \{<, \leq, =, \geq, >\}$.

The semantics of TA is defined over a *timed transition system* $(S, \rightarrow)$, where S is the set of states, and $\rightarrow$ is the transition relation that describes how the system evolves from one state to another. A state $s \in S$ is a pair $s = (l, u)$, where l is the location and u is the valuation of the clocks. A progress in the system is performed by either a *discrete* or a *delay* transition. By executing a discrete transition the automaton moves from one location to another, instantaneously, whereas by executing a delay transition the automaton remains in the same location while the time passes. A *path* of a TA is an infinite sequence of states $\sigma = s_0 a_0 s_1 a_1 s_2 a_2 \ldots$ alternated by transitions, be they delay or discrete (here they are abstracted into a generic type of transition), such that $s_i \xrightarrow{a_i} s_{i+1}$.

UPPAAL extends TA with a number of features, such as: constants, bounded data variables, arithmetic operators, urgent and committed locations, as well as urgent and broadcast channels. A committed location is used to indicate that time is not allowed to pass while the system is in that location.

The synchronization between TA in performed via synchronization channels in a hand-shake or broadcast manner. The synchronization is modeled by annotating edges with synchronization labels (a! for the sender automaton, and a? for the receiver automaton). When two or more edges are synchronized, they execute simultaneously, that is locations in all the involved automata in the synchronization are changed (provided that all involved automata are in the corresponding locations, and the guards of the annotated edges are true at the time).

A *network* of UPPAAL timed automata (NTA) is a parallel composition of and arbitrary number

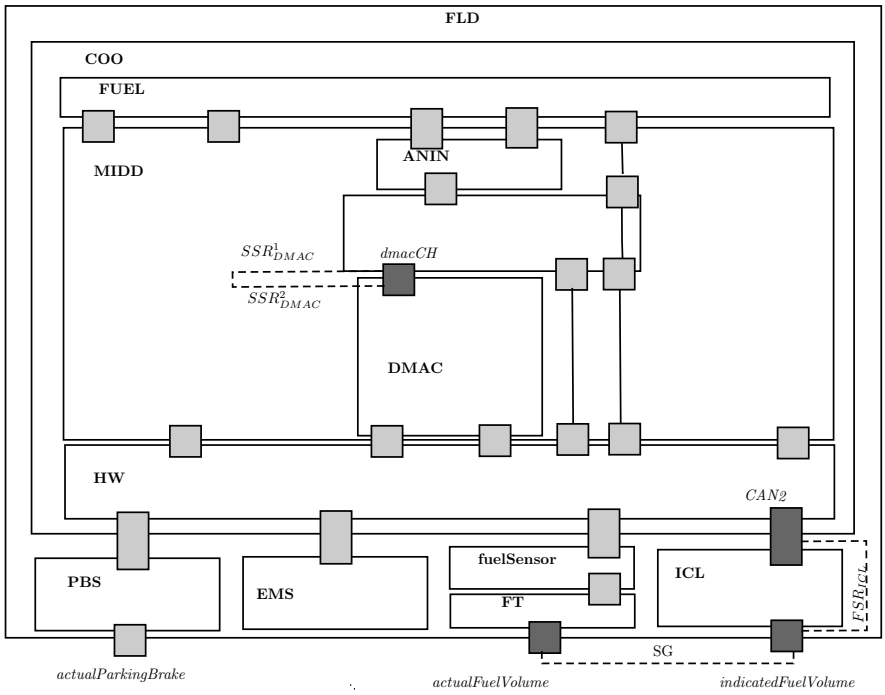

Figure 2: Excerpt of the high-level architecture of the Fuel Level Display system.

of TA over X and Σ, synchronizing on actions and using shared variables.

UPPAAL uses symbolic semantics and symbolic reachability techniques to analyze the dense-time state space of the timed automata model, with respect to a set of properties formalized in a decidable subset of (T)CTL. For a more comprehensive overview of the properties that are supported by UPPAAL, we refer the reader to the literature [LPY97].

In Figure 1 we show a timed automaton that models the behavior of a timer-based lamp. Initially, the lamp is off, represented by the location `Off`. The system remains in the given location until a synchronization message `press?` is received, which models the press of a button. Once the synchronization message is received, the system transitions into location `Dim`. During the transition, the automaton resets the system clock (denoted as `t`) via an update action `t := 0`. The lamp can stay in `Dim` location for at lest 3 but not more than 5 time units, ensured by the invariant `t < 5` and the guard `t >= 3`, after which it returns to location `Off`. If the light is dim and the button is pressed again, the light goes bright (transition to location `Bright`). Once the light is bright, the only way to turn it off is to push the turn on button again and issue `press?` synchronization message.

3 Industrial Use-Case: Fuel Level Display

In this section, we introduce the industrial system called the *Fuel Level Display* (FLD), used as a case study on which we evaluate both approaches for consistency checking.

The FLD system is a function installed in all heavy-load vehicles produced by the Swedish

truck manufacturer Scania. The main functionality of the FLD is to estimate the remaining fuel in the vehicle, which is computed based on different sensor readings, and display the correct value to the driver. The fuel estimation feature is implemented as a software function deployed on the Coordinator (COO) Electronic Control Unit (ECU) system. The remaining fuel that is displayed to the driver is calculated based on the sensed fuel level obtained from the fuel tank provided by the fuel sensor (fuelSensor) placed inside the fuel tank (FT), and the current fuel consumption rate provided by the Engine Management ECU system (EMS). The system is classified as *safety critical*, meaning that its proper functioning must be ensured while the vehicle is moving, to prevent hazardous situations that can endanger human lives.

The simplified architectural break-down of the FLD system, including all aforementioned parts, is given in Figure 2. The design description is based on a set of *elements* [WN15], which are represented as rectangles in Figure 2 (ex: FLD, PBS, etc.). The elements are used to model all entities in the system's design description, including both physical and logical ones. The interface of an element is represented via one or more *ports* (ex: actualParkingBrake, actualFuelVolume, etc.), which represent the tangible entities of an element as seen by an external observer. The communication between the different elements occurs via their ports. The behavior of an element is defined through a set of constraints over its ports, which for the FLD system are specified using the contract-based approach, through assertions of type *assume-guarantee*, represented with dashed lines in Figure 2. In the following, we present some of the requirements of the FLD's system specification, which model the functional and time-bounded functional aspect of the system.

SG If `actualParkingBrake (aPB)` is false, then `indicatedFuelVolume (iFV)`, shown by the fuel gauge, is less than or equal to `actualFuelVolume (aFV)`.

FSR_{ICL} `If it has not passed more than 1s` since the last time `CAN message DashDisplay (DD)` appeared on `CAN2 CAN bus`, and the `DD` message is valid, then the `iFV`, shown by the fuel gauge, corresponds to `FuelLevel (FL)` signal value from the `DD` message.

SSR^1_{DMAC} The `Direct Memory Access (DMA)` channel that corresponds to the input value of `dmacCH` when `Dmac_enableCh()` function is called, is enabled when `Dmac_enableCh()` function finishes its execution.

SSR^2_{DMAC} The DMA channel that corresponds to the input value of `dmacCH` when `Dmac_disableCh()` function is called, is disabled when `Dmac_ disableCh()` function finishes its execution.

For the detailed functional description of the system we redirect the readers to other work [WN15].

4 SMT-based Consistency Analysis: Method and Application

In this section we give an overview of the SMT-based consistency analysis method that we have proposed in our earlier work [FRNS17]. The method is intended to be lightweight, supported

by a fast analysis procedure, suitable for application in early stages of system development, and appealing to engineers.

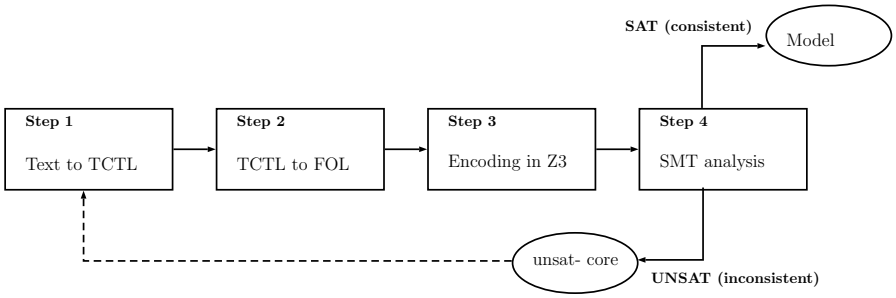

Figure 3: Our SMT-based Method for Consistency Checking.

The SMT-based method is given in Figure 3. It is composed of 5 steps, as follows: Step 1 transforms the informal free-text requirements into a formal notation suitable for the formal analysis. The step is based on Specification Patterns (SPS) [DAC98] [KC05] [AGL+15], which have been shown expressive enough for formalizing requirements in the automotive domain [FNR14] [PMHP12]. The procedure of creating the formal system specification is supported by the SeSAMM Specifier tool [FJN+16] that we have also proposed in our earlier work. The result of applying the specification patterns and the SeSAMM Specifier tool is a set of requirements that can be expressed in different notations, including Controlled Natural Language and various formal notations. The advantage of this automated step is the fact that, based on the patterns, one can automatically generate the set of temporal formulas expressed in TCTL, where each TCTL formula corresponds to a single requirement of the original system specification.

The results from analyzing the pattern-based formal system specification for FLD shows that only 5 patterns are enough to formalize the complete set of 23 requirements. The patterns used in the formalization are given as follows:

P1: Globally, Universally: $AG(\varphi)$

P2: Timed Globally, Universally: $AG(AG_{\leq T}(\varphi) \Rightarrow \psi)$

P3: Globally, Response: $AG(\varphi \Rightarrow AF_{\leq T}\psi)$

P4: After φ Until θ Universally ψ: $AG(\varphi \Rightarrow A(\psi W_{\leq T}\,\theta))$

P5: Timed After φ Until θ Universally ψ: $AG(AG_{\leq T}(\varphi) \Rightarrow A(\psi W_{\leq T}\,\theta))$

The SMT-based analysis is carried out over formulas encoded in First Order Logic (FOL). Due to the semantic gap between the temporal logics and the FOL, we propose Step 2, in which we bridge this semantic gap by instantiating the semantics of the path quantifiers and the path specific temporal operators in FOL logic. The step-by-step procedure of instantiating the operators is explained in full details in our previous work [FRNS17]. In this paper we assume that all the used SPS patterns can be transformed in FOL using the described procedure. Consequently, a set of FOL formulas is obtained, each corresponding to a TCTL formula from Step 1.

Once the FOL formulas have been obtained, in Step 3 we propose a way of encoding them into Z3 assertions using Z3Py, which is a Python API of the Z3 tool. However, using Z3 is not mandatory, as one might opt to use another SMT solver. During this step, we also propose three abstraction rules intended to simplify the original formulas without any loss of potential inconsistency in the system. The abstraction rules are suitable for system specifications expressed as invariance properties. In such cases, we show that instead of having three quantified variables (branch, location and time), each FOL formula can be reduced to a single quantified variable (time), while still preserving the information about the potential inconsistencies in the system. The application of abstraction rules is mandatory, as the SMT procedure proves not to terminate for the original set of FOL formulas. An additional advantage of the abstraction rules is the fact that their application speeds up the analysis procedure significantly [FRNS17].

Finally, in Step 4, the SMT analysis is performed to determine whether a witness assignment of variables, which satisfies the conjunction of all the encoded Z3 assertions, exists. If the set of assertions is proven satisfiable, the SMT solver returns a `Model` containing the valuation of all the atomic propositions such that the conjunction of all the assertions evaluates to true. In the opposite case, that is when there exists no solution for the conjunction of all the assertions representing the system specification, the SMT solver returns `unsat-core`. Considering the fact that during the transformation there is a consistent one-to-one mapping of natural language requirements into Z3 assertions via a unique identifier for each requirement in the specification allows us to trace the `unsat-core` assertions back to the original set of requirements.

The SMT analysis procedure is applied over the complete set of FLD requirements that comprises of 36 Z3 assertions. It is executed on a Linux machine with 2.4 GHz Dual Core processor and 4GB RAM. Initially not all assertions could be solved by the Z3 procedure, but after a series of conservative mitigation strategies the SMT analysis over the complete set of FLD system requirements terminates within seconds [FRNS17].

5 Consistency Analysis by Model checking: Method and Application

In this section, we provide details on how we build the model that encodes the requirements specification of a system, with application on FLD, as well as on how to asses the consistency of the system requirements encoding via model checking.

According to Definition 1, in order to disprove the inconsistency of a system's requirements specification represented as a set of logical formulas, it is enough to find a state in which all the requirements in the specification are non-vacuously satisfied. To explain the non-vacuous satisfaction, let us consider the following property: AG($\varphi \Rightarrow \psi$). The given property can be satisfied in two ways: i) the property is *vacuously* satisfied if φ is *false* in all states, or ii) *non-vacuously* if there exists at least one state in which φ is true, and whenever φ is true ψ is also true.

Next, we give an overview of our approach for building the network of TA from the system requirements specification as a set of TCTL formulas (Section 5.1), after which we show the properties of the FLD model that we have checked for consistency (Section 5.2).

5.1 Building the System Model

In order to be able to analyze the consistency of requirements specifications via model checking, first one has to build a formal model of the system specification. As an input, we use the system specification expressed as a set of TCTL formulas, as presented in Section 2.3. For the analysis, we use UPPAAL, which means that the set of TCTL formulas needs to be transformed into an NTA.

To be able to compare the SMT and model checking for analyzing consistency of requirements as invariance properties, we apply common modeling principles in both methods, such that the comparison becomes feasible. The basic modeling principle applied in the process of transforming the TCTL requirements into an NTA relies on mapping each TCTL formula onto a single timed automaton. The goal is to transform each individual formula (requirement) in isolation, by using only minimal contextual information. Additionally, the one-to-one mapping between the TCTL formula and the respective automaton ensures the traceability between the requirements in the system specification and the TA model, which is fundamental for error localization and its further mitigation. All system variables (usually ports of components in Figure 2) are modeled as global data variables in UPPAAL.

All the formalized requirements in the specification are *invariance* properties (AG(φ)), meaning that the properties must hold in all system states. However, some of the TCTL formulas contain implications, thus can be vacuously satisfied. To make sure that we can clearly distinguish between the vacuous and non-vacuous satisfaction for each formula, we propose two basic templates for encoding requirements as TA that eliminate vacuous satisfaction by design, shown in Figure 4.

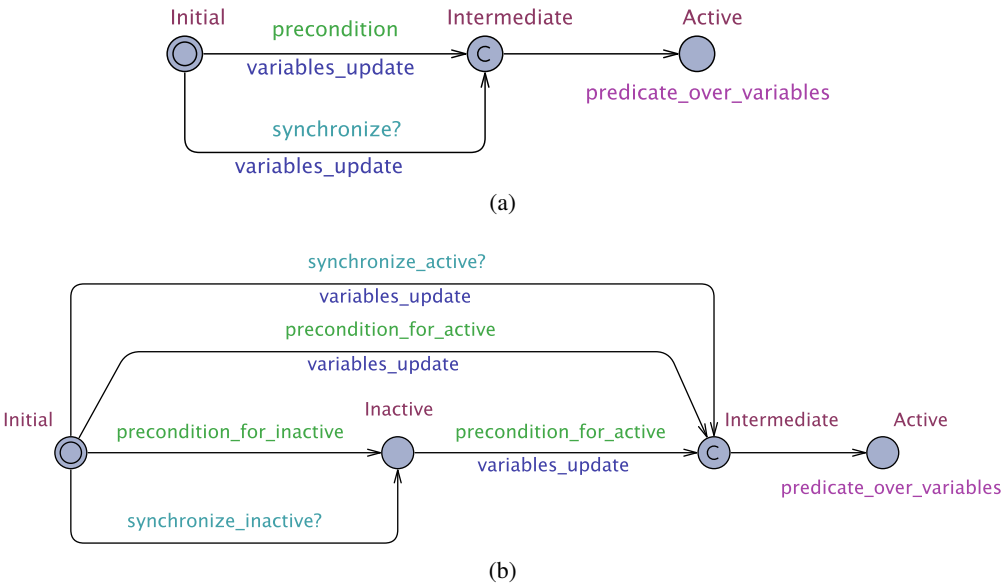

Figure 4: TA templates for transforming TCTL formulas into TA.

The template given in Figure 4a is used for modeling the TCTL formulas that do not contain

implication. The automaton is composed of three locations, called `Initial`, `Intermediate` and `Active`. The behavior of the automaton is defined as follows: initially, the automaton is in the `Initial` location, and remains there until either the precondition for execution is satisfied or its execution is triggered by some event. If the precondition is a predicate over system variables, then it is modeled as a guard decorating the outgoing edge from the `Initial` location. For modeling the events in the system, we use `broadcast channels`. Given the fact that the requirements are specified using the assume-guarantee contract-based approach [WN15], a requirement is guaranteed to be fulfilled provided that the set of assumptions for that requirement holds. In the context of FLD, it means that the postcondition of a given requirement can be guaranteed if the requirements that have been annotated as assumptions have already executed and their postconditions established. By using this mechanism we assure the correct ordering of execution of the automata, that is, it is guaranteed that a given requirement TA is executed (can transition to the `Active` location) if and only if its precondition is established.

Once the automaton is triggered, it performs a transition from `Initial` location into `Active` location, which is decorated with an invariant corresponding to the postcondition that has to be established when a given requirement is non-vacuously fulfilled. Since the update transition action and the invariant of the destination location are over the same data variables, the transition is divided into two parts by a committed location denoted as `Intermediate`, otherwise a deadlock occurs. The first part, made of two edges, one for the precondition over variables, and one for triggering via a broadcast channel, is additionally decorated with an update action intended to update the values of variables such that the requirement is actively fulfilled. The choice of two edges is motivated by the fact that the invariant on the `Active` location is over data variables. Once location `Active` is reached, the automaton remains in that location indefinitely, as we do not model any release condition.

The automaton shown in Figure 4b is used as a basis for representing the TCTL formulas that contain implication. This is a most basic form of an automaton model meant to encode the TCTL formulas obtained by using patterns P2 to P5 as presented in Section 4. It represents an extension of the automaton given in Figure 4a, with an additional location called `Inactive` that denotes the mode in which the antecedent of the formula has not yet been satisfied. The template automaton in general models two distinct branches of execution. The first branch starts with location `Initial`, and then reaches location `Active`, via locations `Inactive` and `Intermediate`. This branch models the behavior corresponding to the antecedent not being satisfied. The second branch of execution involves locations `Initial`, `Intermediate` and finally `Active`, and models the behavior in which the antecedent holds at the time the execution of the automaton is started.

As already mentioned, the template given in Figure 4b represents a basic template, which, in order to correctly capture the behavior of the TCTL formulas obtained by using patterns P2 - P5, has to be modified accordingly. Relying on the explanation that we have given for the basic behavior, the extension to accommodate the behavior of the aforementioned patterns is trivial, hence we do not discuss it further.

The FLD system specification includes a number of events that trigger the behavior of some of the requirements. The problem is that the existence of such events in the system specification is assumed, and there is no detailed information about their occurrence, duration, etc. In order to enable the execution of requirements that depend on the occurrence of events, we have

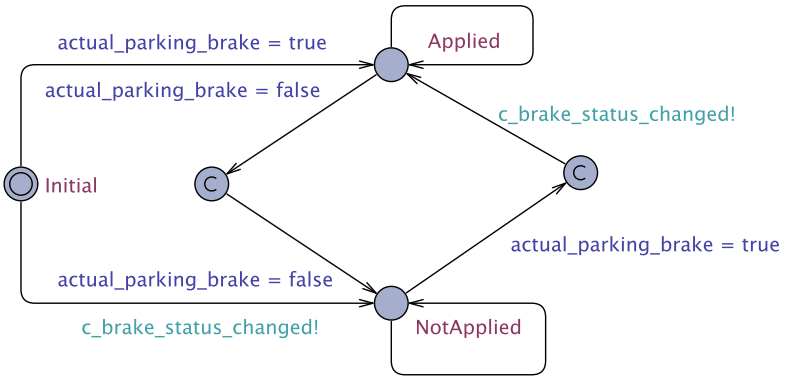

Figure 5: Controller automaton for the *Actual Parking Brake* event.

to additionally include components in the NTA model, which are responsible for generating those events. We call such components *Controllers*, and for the FLD system model we include nine controllers; the latter are responsible for triggering individual events such as function calls (ex: dmac_enableCh, dmac_disableCh, etc.), or modeling the dynamics of some of the system's variables' evolution (ex: actualFuelVolume, sensedFuelVolume, actualParkingBrake, etc). Since there is no information about the frequency of their occurrence or the order at which they are allowed to occur, the behavior of the controller automata is completely non-deterministic.

One example of a controller is given in Figure 5, and its responsibility is to model the behavior of the parking brake sensor. The controller models the two operational modes of the sensor, which in the NTA model is represented by the `actual_parking_brake` global Boolean variable, which is considered applied (engaged) if set to true, and disengaged if set to false. The controller starts the execution in the `Initial` location and non-deterministically decides whether the parking brake is initially applied or not. Since the execution of the controller starts before any of the requirements is enabled, no event is raised on either of the edges. Once initialized, the controller then non-deterministically switches between the two modes. If the controller decides to switch mode, then it will first update the value of the sensor variable and issue the `c_brake_status_changed!` broadcast event to notify the rest of the automata in the network.

5.2 Analyzing the Consistency of the Fuel-Level-Display System Specification

As already presented in Section 2.1, in order to prove the consistency of requirements it is sufficient to show that all the requirements in the specification can be (non-vacuously) satisfied simultaneously at least by one witness trace. In model checking terms, this means that it is enough to show that there exists a run of the model, such that all the automata (excluding the Controllers) simultaneously reach their `Active` location, respectively. Additionally, special care has to be taken when there are requirements in the specification that describe complementary behaviors, as they cannot be actively fulfilled at the same time.

Consequently, the consistency of the system specification consisting of m temporal logic formulas (requirements), encoded as an NTA composed of automata $A_1, A_2, \ldots, A_n$, $n \in \mathbf{N}$, can be

proved if the following reachability property of the model is satisfied:

$$E\Diamond(A_1.Active \text{ and } A_2.Active \text{ and } ... \text{ and } A_n.Active), \text{ where } i \in [1,n] \text{ and } n < m \text{ if there are complementary requirements in the specification, otherwise } n = m.$$

Provided that each location `Active` in the model is decorated by an invariant, respectively, the given property can never be satisfied if two postconditions contradict each other.

6 Comparison and Discussion

In this section, we present and discuss the results of applying the model-checking-based approach for checking the consistency of the FLD system requirements, in terms of modeling effort and time required for the procedure to complete, and compare the approach to the SMT-based one.

Modeling. For validation, we have applied the model-checking-based approach on a set of requirements of the FLD display system, responsible for measuring, estimating and displaying the correct information about the remaining fuel in the vehicle. The resulting NTA that models the behavior of the FLD system obtained by applying the TA templates given in Figures 4a and 4b in Section 5.1 consists of 32 automata. In the model, 23 automata correspond to the requirements from the system specification, while the remaining 9 correspond to the different controllers that model the behavior of the various system variables and events.

As already defined in Section 5.2, the consistency checking for the FLD system specification is reduced to a reachability problem, according to which the specification is consistent if a system state where all of the 20 operational FLD requirements are actively fulfilled is reachable. The defined property does not include the requirements that describe complementary behavior, and in case of FLD only thee of the total requirements are of this type.

When the two presented methods are compared in terms of time and effort for creating the system model that is then used for consistency analysis, we observe that: the mapping between the intermediate first-order-logic(FOL)-based system requirements specification and the SMT-LIB assertions is more intuitive compared to building the NTA model of formal system specification expressed in TCTL. The reason for this is that, in the first case, one can easily define the mapping between the different formulas, whereas the mapping between a temporal logic formula and an automata template is not as straightforward. This is primarily due to the different modeling concepts and constructs that are available in the different formalisms. For instance, in Z3's SMT-LIB representations, there is no special encoding of the events in the system, whereas in the timed automata world, the events can be modeled via different mechanisms, depending on whether the event has the purpose of synchronizing the execution of automata, or just triggering the execution of some of them. However, the encoding of the FOL formulas in the SMT-based consistency analysis approach, relies on abstractions defined in advance, so the encoding is an approximation rather than an exact one.

Another difference is the fact that the SMT-LIB assertions describe the static structure of the system in which there is no notion on how the system evolves in time. The Z3 model (encoded as SMT-LIB script) is represented through a set of constraints of the system's variables, which the solver tries to solve such that that they become satisfiable. In contrast, when building a TA

model for the system specification, one has to correctly capture the system's evolution in time. This means that additional effort has to be invested in order to model the NTA such that the system's execution and evolution are correct. This includes the modeling of poorly documented entities of the system, encoded by the controller automata, which have to be modeled based on contextual information and some assumptions from the system designer. Ideally, there should exist a detailed specification on how the events that trigger the requirements occur, which sometimes can be part of the specification of another system.

From the above observations, it follows that the transformation of TCTL formulas into NTA can be more challenging compared to their transformation in SMT-LIB script via patterns. Even though these findings are somewhat expected, the contribution of our modeling exercise over the FLD system shows them in practice on an industrial use-case.

Analysis time. The second aspect in which we compare the two approaches is the analysis time required by both the SMT solver and the model checker to complete the consistency analysis and come up with an answer. The procedure for analyzing the consistency of the 23 FLD requirements via reachability analysis as described in Section 5.2 terminates in less than 60 seconds on a standard workstation personal computer. The model checker provides a witness execution trace which shows how the set of requirements has been actively satisfied, which disproves the inconsistency of the FLD system requirements specification. As mentioned in Section 4, and reported in detail in our previous work on the SMT-based consistency analysis approach [FRNS17], the consistency analysis of the FLD requirements specification terminates within seconds (usually less than 10, depending on other processes executing on the machine). Therefore, we can conclude that there is no considerable difference in the analysis time in both approaches when applied to the FLD case study, which renders the two methods almost equally efficient. However, in order to draw more accurate conclusions on the analysis time between the two techniques, we need to setup evaluations on a larger scale, including larger and more complex models.

7 Related Work

There is a number of consistency analysis approaches proposed in the literature. In this section, we try to relate the most prominent to the consistency analysis approaches presented in this paper.

Barnat et al. [BBB12] propose a model-free sanity checking procedure for consistency analysis of system requirements specification in Linear Temporal Logic (LTL) [Pnu77] by means of model checking. The approach has later been extended [BBB+16] to support the generation of a minimal set of inconsistent requirements. Despite the exhaustiveness, the approach suffers from the inherent complexity of transforming the LTL formulas into automata, especially for complex systems, rendering the method unusable in industrial settings. A similar approach for consistency checking of requirements specified in LTL is proposed by Ellen et al. [ESH14]. The authors introduce a definition for the so-called existential consistency, that is, the existence of at least one system run that satisfies the complete set of requirements. Similar to what we propose, the analysis procedure has been integrated into an industrially-relevant tool, aiming at industrial application. The work by Post et al. [PHP11b] defines the notion of rt-(in)consistency of real-time requirements. The definition covers cases where the requirements in the systems requirements

specification can be inconsistent due to timing constrains. The checking for rt-inconsistency is reduced to model checking, where a deadlock situation implies the rt-inconsistency of requirements.

Despite the exhaustiveness of the consistency checking approaches mentioned above, all of them suffer from two major limitations: the long time needed to create the formal requirements model, as well as the time for analysis that grows exponentially with the number of requirements that are analyzed. The model-checking-based consistency analysis approach applied in this paper shows some intricacy of building the TA model, however, compared to the aforementioned approaches, our approach relies on transformation patterns, which can be used to partially or fully automate the model generation, thus potentially reducing the modeling time.

Regarding the consistency analysis efficiency, the model-checking-based approach applied in this paper takes less time if compared to the checking efficiency, reported in related work [BBB+16, PHP11a]. The reason for this difference should be investigated in more detail, but at this point it seems that the main reason stays in the different definition of consistency (ours is weaker), and the different modeling principles applied to build the models. Our definition of consistency is closest to the one proposed by Ellen et al. [ESH14], however, the latter approach is yet to be applied on industrial use cases, in order to have a basis for a meaningful comparison.

8 Conclusions and Future Work

In this paper, we have evaluated two methods of checking the consistency of system requirements specifications based on model checking and SMT analysis techniques, respectively, on an industrial use-case from the automotive domain, namely Fuel Level Display system. The input of both methods is a set of invariance properties encoded as $AG(\varphi)$ in TCTL, where φ is a formula that contains predicates and nested operators. The aim of the evaluation is to compare the two different methods for consistency analysis, with respect to: i) modeling effort, and ii) analysis time. The SMT-based consistency analysis technique is completely based on our earlier work [FRNS17], whereas for the model-checking-based consistency analysis, we have proposed an approach for transforming the supported TCTL requirements into NTA, which we have shown how to analyze for existential consistency, as defined in our earlier work [FRNS17]. In order for our comparison to be meaningful, we have formalized the same set of requirements [FRNS17] that describe the Fuel Level Display system, which is an operational function installed in all heavy load vehicles produced by Scania.

After applying and analyzing both approaches, our results show that there is no significant difference in the efficiency of the two consistency analysis procedures, however the model-checking-based method requires a more intricate modeling effort. Despite the fact that such observation is somewhat expected, we present the actual reasons on why building timed automata models is more challenging than generating a set of SMT constraints. Basically, the main modeling challenge is the encoding of the dynamics of the system behavior in TA model (for instance, the existence of locations `Intermediate`), whereas in the SMT-based approach one represents the model just as a set of constraints.

Given the fact that the required analysis time for both methods is measured in seconds, the difference in the modeling effort becomes the advantage to consider when it comes to industrial

adoption. The generation of SMT-LIB script can be fully automated, thus requiring no manual intervention during the complete process [FRNS17, FRNS18], yet based on abstraction rules that eliminate some of the initial behavioral information (without losing the potential sources of inconsistency). On the other hand, even though we have shown that the transformation of the system specification expressed as TCTL properties into NTA can be automated to a large extent via patterns, complete automation of the same seems quite unlikely primarily due to the inability to automate the modeling of the synchronization mechanism for the execution of the requirements and the environment.

The future work goes in several directions. First, we aim at applying the two techniques on more industrial examples in order to extend the boundaries of their understanding, applicability and usability. Knowing that model checking suffers from the infamous state-space-explosion problem, it would be beneficial to investigate the scalability of each approach on larger industrial case studies. As a second direction for future work, we target developing appropriate tool support that will automate (partially or completely) the process of generation of the formal model, in both approaches. This should reduce the time required for creating the models, allowing us to test the approaches on more systems. The last direction of future work envisions the extension of the definition of consistency, in both methods, such that it can be applied for full consistency of requirements: instead of checking for the existence of a consistent valuation of the system specification variables, one could check for the existence of any inconsistent interpretation of the requirements. Our initial work on the subject shows that both approaches can be extended for checking such property of an industrial system's specification.

Acknowledgements: This work has been funded by the Swedish Governmental Agency for Innovation Systems (VINNOVA) under the VeriSpec project 2013-01299.

Bibliography

[ACD93] R. Alur, C. Courcoubetis, D. Dill. Model-Checking in Dense Real-Time. *Information and Computation* 104(1):2 – 34, 1993.
doi:http://dx.doi.org/10.1006/inco.1993.1024
http://www.sciencedirect.com/science/article/pii/S0890540183710242

[AD94] R. Alur, D. L. Dill. A Theory of Timed Automata. *Journal of Theoretical Computer Science* 126(2):183–235, Apr. 1994.
doi:10.1016/0304-3975(94)90010-8
http://dx.doi.org/10.1016/0304-3975(94)90010-8

[AGL+15] M. Autili, L. Grunske, M. Lumpe, P. Pelliccione, A. Tang. Aligning Qualitative, Real-Time, and Probabilistic Property Specification Patterns Using a Structured English Grammar. *IEEE Transactions on Software Engineering* 41(7):620–638, 2015.

[BBB12] J. Barnat, P. Bauch, L. Brim. Checking Sanity of Software Requirements. In *Proceedings of the 10th International Conference on Software Engineering and Formal Methods*. SEFM'12, pp. 48–62. Springer-Verlag, Berlin, Heidelberg, 2012.

[BBB+16] J. Barnat, P. Bauch, N. Beneš, L. Brim, J. Beran, T. Kratochvíla. Analyzing Sanity of Requirements for Avionics Systems. *Form. Asp. Comput.* 28(1):45–63, Mar. 2016. doi:10.1007/s00165-015-0348-9 http://dx.doi.org/10.1007/s00165-015-0348-9

[BFT15] C. Barrett, P. Fontaine, C. Tinelli. The SMT-LIB Standard: Version 2.5. Technical report, Department of Computer Science, The University of Iowa, 2015. Available at `www.SMT-LIB.org`.

[CE82] E. M. Clarke, E. A. Emerson. Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic. In *Logic of Programs, Workshop*. Pp. 52–71. Springer-Verlag, London, UK, UK, 1982. http://dl.acm.org/citation.cfm?id=648063.747438

[CGP99] E. M. Clarke, O. Grumberg, D. Peled. *Model checking*. MIT press, 1999.

[DAC98] M. B. Dwyer, G. S. Avrunin, J. C. Corbett. Property Specification Patterns for Finite-state Verification. In *Proceedings of the Second Workshop on Formal Methods in Software Practice*. FMSP '98, pp. 7–15. ACM, New York, NY, USA, 1998. doi:10.1145/298595.298598 http://doi.acm.org/10.1145/298595.298598

[DB08] L. De Moura, N. Bjørner. Z3: An Efficient SMT Solver. In *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. TACAS'08/ETAPS'08, pp. 337–340. Springer-Verlag, Berlin, Heidelberg, 2008. http://dl.acm.org/citation.cfm?id=1792734.1792766

[DB11] L. De Moura, N. Bjørner. Satisfiability Modulo Theories: Introduction and Applications. *Commun. ACM* 54(9):69–77, Sept. 2011. doi:10.1145/1995376.1995394 http://doi.acm.org/10.1145/1995376.1995394

[ESH14] C. Ellen, S. Sieverding, H. Hungar. *Detecting Consistencies and Inconsistencies of Pattern-Based Functional Requirements*. Pp. 155–169. Springer International Publishing, Cham, 2014.

[FJN+16] P. Filipovikj, T. Jagerfield, M. Nyberg, G. Rodríguez-Navas, C. Seceleanu. Integrating Pattern-Based Formal Requirements Specification in an Industrial Tool-Chain. In *40th IEEE Annual Computer Software and Applications Conference, COMPSAC Workshops 2016, Atlanta, GA, USA, June 10-14, 2016*. Pp. 167–173. IEEE Computer Society, 2016. doi:10.1109/COMPSAC.2016.140 http://dx.doi.org/10.1109/COMPSAC.2016.140

[FNR14] P. Filipovikj, M. Nyberg, G. Rodriguez-Navas. Reassessing the pattern-based approach for formalizing requirements in the automotive domain. In *Proceedings of the*

22nd IEEE International Requirements Engineering Conference (RE). Volume 00, pp. 444–450. IEEE Computer Society, Los Alamitos, CA, USA, 2014.

[FRNS17] P. Filipovikj, G. Rodriguez-Navas, M. Nyberg, C. Seceleanu. SMT-based Consistency Analysis of Industrial Systems Requirements. In *The proceedings of the 32nd ACM Symposium on Applied Computing (SAC). Marrakech, Morocco.* ACM, April 2017.

[FRNS18] P. Filipovikj, G. Rodriguez-Navas, M. Nyberg, C. Seceleanu. Automated SMT-based Consistency Checking of Industrial Critical Requirements. *SIGAPP Appl. Comput. Rev.* 17(4):15–28, Jan. 2018.
doi:10.1145/3183628.3183630
http://doi.acm.org/10.1145/3183628.3183630

[HJL96] C. L. Heitmeyer, R. D. Jeffords, B. G. Labaw. Automated Consistency Checking of Requirements Specifications. *ACM Transactions Software Engineering Methodology* 5(3):231–261, July 1996.
doi:10.1145/234426.234431
http://doi.acm.org/10.1145/234426.234431

[HL96] M. P. E. Heimdahl, N. G. Leveson. Completeness and Consistency in Hierarchical State-Based Requirements. *IEEE Trans. Softw. Eng.* 22(6):363–377, June 1996.
doi:10.1109/32.508311
http://dx.doi.org/10.1109/32.508311

[KC05] S. Konrad, B. H. C. Cheng. Real-time Specification Patterns. In *Proceedings of the 27th International Conference on Software Engineering*. ICSE '05, pp. 372–381. ACM, New York, NY, USA, 2005.
doi:10.1145/1062455.1062526
http://doi.acm.org/10.1145/1062455.1062526

[Kup06] O. Kupferman. Sanity Checks in Formal Verification. In *Proceedings of the 17th International Conference on Concurrency Theory*. CONCUR'06, pp. 37–51. Springer-Verlag, Berlin, Heidelberg, 2006.

[LPY97] K. G. Larsen, P. Pettersson, W. Yi. UPPAAL in a nutshell. *Int. Journal on Software Tools for Technology Transfer* 1:134–152, 1997.

[MSL16] N. Mahmud, C. Seceleanu, O. Ljungkrantz. ReSA Tool: Structured Requirements Specification and SAT-based Consistency-checking. In Ganzha et al. (eds.), *Proceedings of the 2016 Federated Conference on Computer Science and Information Systems, FedCSIS 2016, Gdańsk, Poland, September 11-14, 2016.* Pp. 1737–1746. 2016.
doi:10.15439/2016F404
http://dx.doi.org/10.15439/2016F404

[PHP11a] A. Post, J. Hoenicke, A. Podelski. Vacuous Real-time Requirements. In *Proceedings of the 2011 IEEE 19th International Requirements Engineering Conference*. RE '11, pp. 153–162. IEEE Computer Society, Washington, DC, USA, 2011. doi:10.1109/RE.2011.6051657 http://dx.doi.org/10.1109/RE.2011.6051657

[PHP11b] A. Post, J. Hoenicke, A. Podelski. Rt-inconsistency: A New Property for Real-time Requirements. In *Proceedings of the 14th International Conference on Fundamental Approaches to Software Engineering: Part of the Joint European Conferences on Theory and Practice of Software*. FASE'11/ETAPS'11, pp. 34–49. Springer-Verlag, Berlin, Heidelberg, 2011. http://dl.acm.org/citation.cfm?id=1987434.1987440

[PMHP12] A. Post, I. Menzel, J. Hoenicke, A. Podelski. Automotive Behavioral Requirements Expressed in a Specification Pattern System: A Case Study at BOSCH. *Requir. Eng.*, pp. 19–33, 2012.

[Pnu77] A. Pnueli. The Temporal Logic of Programs. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*. SFCS '77, pp. 46–57. IEEE Computer Society, Washington, DC, USA, 1977. doi:10.1109/SFCS.1977.32 http://dx.doi.org/10.1109/SFCS.1977.32

[Roy87] W. W. Royce. Managing the development of large software systems: concepts and techniques. In *Proceedings of the 9th international conference on Software Engineering*. Pp. 328–338. 1987.

[WN15] J. Westman, M. Nyberg. Contracts for Specifying and Structuring Requirements on Cyber-Physical Systems. In Rawat et al. (eds.), *Cyber Physical Systems: From Theory to Practice*. CRC Press, 2015.