Available online at www.sciencedirect.com

ScienceDirect

IFAC PapersOnLine 53-4 (2020) 493–500

SMT-based verification of temporal properties for component-based software systems

R. Jonk [*] J. Voeten [*] M. Geilen [*] T. Basten [*,**]
R. Schiffelers [***,*]

[*] *Eindhoven University of Technology, Eindhoven, The Netherlands.*
[**] *ESI (TNO), Eindhoven, The Netherlands.*
[***] *ASML, Veldhoven, The Netherlands*

Abstract: We introduce a technique to verify temporal properties expressed in MTL on Interval Message Sequence Charts (IMSC), a model based on UML2.0 MSC that captures the timed execution of component-based software systems. We accomplish this by encoding the IMSC and the property of interest in a constraint satisfaction problem, which is then solved with an SMT solver. We demonstrate the scalability of this technique with a synthetic case study and a large-scale industrial case study.

Keywords: Verification, Component-based software system, Message Sequence Chart, Metric Temporal Logic, Modeling

1. INTRODUCTION

Component-based software systems (CBSS) are widely used, e.g. in enterprise applications (Matena et al. (2003)), consumer electronics (Van Ommering et al. (2000)), avionics applications (Sharp (1998)) and cyber-physical systems such as interventional X-ray machines (Kurtev et al. (2017)) and lithography equipment (Loose et al. (2018)). Our work is motivated by the need to verify temporal properties (expressed in MTL, first introduced by Koymans (1990)) of CBSS after implementation, e.g., on prototypes or in systems operating in the field. The execution of a component-based software system can be viewed as a timed sequence of discrete events which represent the starting or finishing of functions. Such an observed sequence models the behaviour of a single execution. Verification of temporal properties for a CBSS requires considering all possible executions.

A suitable abstraction for this purpose are Message Sequence Charts (MSC). Although primarily used for specification, the concept of lifelines representing components, the underlying partially ordered set (poset) structure of events and their dependencies, and message exchanges representing communication dependencies across components, serve as a strong basis for representing the structural concepts of component-based software systems. We capture temporal concepts of component-based software systems by introducing finite intervals on the dependencies between events. These intervals provide bounds on the minimum duration between events. The lower bound occurs naturally due to the fact that function executions and message communications take time. On modern computational platforms, consisting, e.g., of multiprocessors running Linux or RTOSs, the duration of function executions and message communication exhibits a bounded variation. This provides upper bounds on the duration between events (Wilhelm et al. (2008)).

Many approaches to the verification of (temporal) properties on posets exist. One approach is to translate a poset model to an automaton and verify it against an automaton that encodes the property (e.g. Alur and Yannakakis (1999)) by checking whether the language of the model is accepted by the automaton representing the property. Encoding a poset model by extracting global states from a partial order is a process with exponential time complexity which does not exploit the poset structure. Another approach is to translate the poset model to the language of process algebra and modal μ-calculus (e.g. Blom et al. (2003); Groote et al. (2007)). Both of these languages, while expressive, do not exploit the poset structure of the model directly, which may result in a state-space explosion. We introduce a technique to verify temporal properties by reducing the problem on the poset structure to a constraint satisfaction problem directly and show it to be effective.

The contributions of this paper are twofold. The first contribution is the introduction of Interval Message Sequence Charts (IMSC) and their semantics in terms of timed traces. The second contribution is the encoding of an IMSC and an MTL property of interest in a constraint satisfaction problem. This contribution also shows that transforming the poset to an automaton is not necessary.

The structure of the remainder of the paper is as follows. We discuss related work in Section 2. In Section 3 we recall the MTL language and provide a model of MSCs. In Section 4 we introduce IMSCs and present the MTL model checking problem on IMSCs. In Section 5 we introduce the encoding of the model checking problem as an SMT problem. In Section 6 we perform a synthetic and industrial case study to validate the method and we compare

10.1016/j.ifacol.2021.04.045

the results of the synthetic case to results obtained with Uppaal. Finally, Section 7 concludes.

2. RELATED WORK

Model checking is typically done on (state and clock valuation) traces generated by timed automata. In our work, we restrict the traces of our models to those induced by partial orders specified in IMSCs, which is a proper subset of the traces generated from timed automata. Uppaal (Uppaal, 1995) is the tool most commonly used for model checking temporal properties for timed automata. Uppaal uses symbolic state exploration with zones to verify reachability and invariance properties expressed in a subset of CTL. While the logic itself is untimed, Uppaal's state formulae allow constraints on the clock values to specify timing constraints. We use MTL, a logic which explicitly uses time. For a subset of properties expressed in MTL, we can formulate equivalent properties in Uppaal's CTL logic. We compare our model checking method to Uppaal for such properties.

A method that combines timed automata and model checking using SAT solvers is presented in (Zbrzezny, 2005). This work translates a timed automaton to transition systems with discrete time while preserving reachability, i.e., a location is reachable in the real-valued timed transition system if and only if it is reachable in the discrete-time transition system. The reachability problem is translated to a bounded model checking problem which is then encoded in SAT. A similar approach that enables model checking of properties specified in MTL, albeit restricted to discrete-time automata with strongly monotonic traces, is introduced in (Woźna-Szcześniak and Zbrzezny, 2014). The discrete time automaton is augmented with additional clocks that correspond to the intervals in the MTL formula. Properties expressed in MTL are encoded in SAT to verify whether there exists a path of bounded length that satisfies the formula. While our work enables the verification of the same class of properties, the traces of IMSCs and the traces of discrete-time automata with strongly monotonic traces differ. In (Zbrzezny et al., 2019), an enhancement of the earlier SAT based method, SMT is used to verify reachability properties on timed automata using bounded model checking without discretizing time. This approach is shown to improve the efficiency of reachability verification.

To improve the model checking efficiency of certain classes of timed automata, (Malinowski and Niebert, 2010) use partial order semantics through multisteps with synchronous, semi-synchronous or relaxed time progress to reduce the state space of the model. Reachability problems are translated to bounded model checking problems, which are then encoded as SAT problems. The method is compared with Uppaal and the results show improvements.

In the work of (Alur and Yannakakis, 1999), model checking of message sequence charts (MSC) is described. The MSCs formalized in this work describe, like IMSCs, partially ordered sets, albeit without the notion of time. Model checking of MSCs is accomplished by translating an MSC to an automaton. The size of the automaton that expresses the same traces as the MSC is exponential in the number of parallel components in the MSC.

In our work, different from all the mentioned work, we encode MTL properties and timed system behaviour with real-valued time stamps captured in an IMSC, essentially a partial order, directly in SMT. In this way, we avoid the step to translate IMSCs to a more expressive language.

3. PRELIMINARIES

3.1 Metric temporal logic

For this work to be self-contained, we include the definitions of Metric Temporal Logic (MTL) (Koymans (1990)). A finite time sequence $\tau = \tau_1\tau_2\dots$ is a sequence of non-decreasing non-negative real ($\mathbb{R}_0^+$) time values that start at time 0 (i.e. $\tau_1 = 0$). A trace over a set AP of atomic propositions is a finite sequence $\sigma = \sigma_1\sigma_2\cdots\sigma_n$ of states where $\sigma_i \in 2^{AP}$ for all i $(1 \le i \le n)$.

A timed trace $\rho = (\sigma, \tau)$ is a pair consisting of a state sequence σ and a time sequence τ, both of the same length. We alternatively represent a timed trace as the sequence $(\sigma_1, \tau_1)(\sigma_2, \tau_2)\cdots$. For any $i \le |\tau|$, ρ_i refers to the state-time pair (σ_i, t_i) at position i, and (ρ, i) refers to the tail of ρ starting at position i.

The set of MTL formulas, ranged over by ϕ, is inductively defined as follows:

$$\phi = p \mid \phi \wedge \phi \mid \neg\phi \mid \phi\mathbf{U}_I\phi \qquad (1)$$

where $p \in AP$ and $I \subseteq [0, \infty)$ is an interval on $\mathbb{R}_0^+$. We use the notation (a, b), $[a, b]$, $(a, b]$, $[a, b]$, (a, ∞) or $[a, \infty)$ $(a, b \in \mathbb{Q}_0^+)$ to represent an open, closed, left-open, right-open, left-open-bounded and left-closed-bounded interval, respectively. We use rational number bounds to be able to represent the bounds in an implementation of the property. The semantics of the satisfaction relation $\models$ for MTL is given in the following definition:

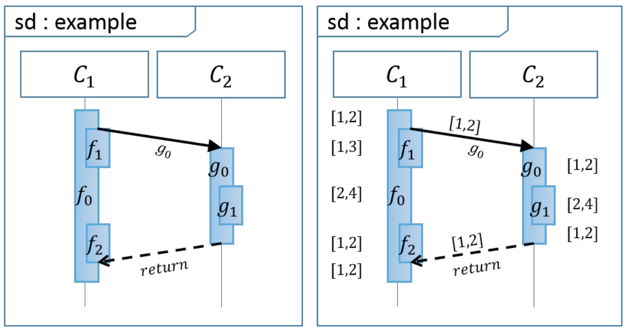
Definition 1. (Trace semantics of MTL). Let $\rho = (\sigma_1, \tau_1)$, (σ_2, τ_2), $\cdots$, (σ_n, τ_n) be a finite timed trace of length $n \ge 1$. The satisfaction relation on ρ starting at position $1 \le i \le n$ is defined inductively as follows:

- $(\rho, i) \models p$ iff $p \in \sigma_i$,
- $(\rho, i) \models \phi \wedge \psi$ iff $(\rho, i) \models \phi$ and $(\rho, i) \models \psi$,
- $(\rho, i) \models \neg\phi$ iff $(\rho, i) \not\models \phi$, and
- $(\rho, i) \models \phi\mathbf{U}_I\psi$ iff there exists $j \ge i$ s.t. $(\rho, j) \models \psi$, $\tau_j - \tau_i \in I$ and for all $i \le k < j$, $(\rho, k) \models \phi$

The satisfaction of $\rho \models \phi$ is equivalent to $(\rho, 1) \models \phi$. We use the following shorthand notations: $\mathit{false} \equiv p \wedge \neg p$, $\mathit{true} \equiv \neg\mathit{false}$, $\phi \vee \psi \equiv \neg(\neg\phi \wedge \neg\psi)$, $\phi \Rightarrow \psi \equiv \neg\phi \vee \psi$, $\phi \Leftrightarrow \psi \equiv \phi \Rightarrow \psi \wedge \psi \Rightarrow \phi$, $\mathbf{F}_I\phi \equiv \mathit{true}\mathbf{U}_I\phi$, and $\mathbf{G}_I\phi \equiv \neg\mathbf{F}_I\neg\phi$. In the shorthand for *false* $(p \wedge \neg p)$, p denotes an arbitrary element of AP, which is assumed to be a non-empty set. The operators $\mathbf{F}_I\phi$ and $\mathbf{G}_I\phi$ denote that ϕ must sometime be satisfied and must always be satisfied on the interval I, respectively.

3.2 Untimed Message Sequence Charts

We formalize a subset of the UML2.0 standard (UML (2005)) which, in this work, we call Untimed Message Sequence Charts (UMSCs). An example illustrating the concepts included in a UMSC is shown in Figure 1a. Vertical lifelines depict the components that execute functions.



(a) An Untimed Message Sequence Chart (UMSC).

(b) An Interval Message Sequence Chart (IMSC).

Fig. 1. A schematic view of UMSCs and IMSCs.

Each lifeline contains a component label from the set $\mathcal{C}$ of component labels. Function executions are depicted by rectangles drawn on the component's lifelines, annotated with a function label from the set $\mathcal{F}$ of function labels. Functions can be executed within the context of another function representing nested local function calls. In the figure, nesting is represented by positioning function executions on top of each other in a call stack. Finally, components can exchange messages. These are depicted by arrows going from the start/end of one function execution's rectangle to the start/end of another's, with message labels from the set $\mathcal{M}$ of message labels. We draw solid arrows for messages indicating remote function calls, and dashed arrows for the reply of such a remote function call.

The formalization of UMSCs is done in terms of events and dependencies between events. We assume a set $\mathcal{E} = \{\{c, f, i, s\} \mid c \in \mathcal{C}, f \in \mathcal{F}, i \in \mathbb{N}^+, s \in \mathcal{S}\}$ of events, where $\mathcal{S} = \{\uparrow, \downarrow\}$ ($\mathcal{C}$, $\mathcal{F}$, $\mathbb{N}$ and $\mathcal{S}$ are assumed to be disjoint). If $e = \{c, f, i, s\} \in \mathcal{E}$ and $s = \uparrow$, then e denotes the start event of the i^{th} execution of function f on component c. Similarly, if $s = \downarrow$, then e denotes the end event of the i^{th} execution of function f on component c. An event is uniquely identified by its component name (c), function name (f), index (i) and event type ($\uparrow$ or $\downarrow$), and therefore will be referred to by expression $c.f(i)^s$. For event $e \in \mathcal{E}$, we use $c(e)$, $f(e)$, $i(e)$, $s(e)$ to refer to the component label, function label, index and start/end of e, respectively.

Definition 2. (Untimed Message Sequence Charts). A UMSC $\mathcal{U}$ is a three-tuple $(E, \rightarrow, m)$, where

- $E \subseteq \mathcal{E}$ is a finite set of events,
- $\rightarrow \subseteq E \times E$ defines an ordering relation between events, and
- $m : \rightarrow \nrightarrow \mathcal{M}$ is a partial function that maps the ordering relation to messages.

We write $e_1 \rightarrow e_2$ to denote that $(e_1, e_2) \in \rightarrow$ and say there is an edge from e_1 to e_2. Furthermore, we use the functions $pred(e) = \{e' \in E \mid e' \rightarrow e\}$ and $succ(e) = \{e' \in E \mid e \rightarrow e'\}$ to denote all the events which have edges to e and from e, respectively. We require that the transitive closure $\rightarrow^*$ of the ordering relation $\rightarrow$ restricted to the events of a component is a total order.

4. PROBLEM STATEMENT

We introduce IMSCs to add temporal concepts to UMSCs and give semantics to IMSCs in terms of timed traces. We then introduce the model-checking problem on IMSCs.

4.1 Interval Message Sequence Charts

Figure 1b depicts an Interval Message Sequence Chart. Edges between events are annotated with an interval.

Definition 3. (Interval Message Sequence Charts). An IMSC $\mathcal{I}$ is a tuple $(E, \rightarrow, m, I)$, where

- $(E, \rightarrow, m)$ is a UMSC, and
- I is a function that maps each edge in $\rightarrow$ to an open/closed/half-open interval of real numbers with rational bounds.

We use the notation $I_{e',e}$ to denote the interval on the edge $e' \rightarrow e$. The interpretation of IMSCs is a set $P(\mathcal{I})$ of timed traces. Figure 2 shows an example time valuation of the events, depicted by the numbers next to the start/finish of function executions. The values depicted by the numbers between brackets between start/finish of function executions represent the minimal time that must elapse before the subsequent event may occur. These values are taken from the intervals depicted in Figure 1b. The timed traces that correspond to this time valuation are shown on the right of the figure.

The set $P(\mathcal{I})$ of timed traces is constructed as follows. The set AP of atomic propositions of an IMSC is the union of the set of all component labels, function labels, used indices, and the start/finish labels. In the example, the set AP is $\{C_1, C_2, f_0, f_1, f_2, g_0, g_1, 1, \uparrow, \downarrow\}$. Every $e = \{c(e), f(e), i(e), s(e)\} \in E$ is a state. For example, event $C_1.f_0(1)^\uparrow$ corresponds to the state $\{C_1, f_0, 1, \uparrow\}$. For every edge $e' \rightarrow e$, a value is chosen from the interval, representing the minimum duration between events e' and e. For instance, in Figure 2, the value 1 ($1 \in [1, 2]$) is chosen for the edge between $C_1.f_0(1)^\uparrow$ and $C_1.f_1(1)^\uparrow$. These chosen values determine a time sequence τ. The time value τ_{i_e} for a state σ_{i_e} is determined to be 0 if the event has no predecessors, and otherwise the maximum of the time values of each predecessor plus the value chosen from the interval on the edge from the predecessor. These time values correspond to the urgent execution of events, i.e., an event occurs at the earliest moment in time at which all incoming edges are ready. This is illustrated in Figure 2 at event $C_1.f_2(1)^\downarrow$. Here, the time value of $C_1.f_2(1)^\downarrow$ is the maximal value of $5 + 1 = 6$ (from the predecessor $C_1.f_1(1)^\uparrow$) and $7+1 = 8$ (from the predecessor $C_1.f_2(1)^\downarrow$), $max(6, 8) = 8$. This is the smallest time value at which $C_1.f_2(1)^\downarrow$ can occur given the chosen time values in each interval. The chosen values from each interval determine a set of timed traces. This set of timed traces consist of the linearisations σ of the poset $(E, \rightarrow)$ such that the time sequences are monotonically ascending. We use the notation i_e to refer to the index at which event e occurs in σ. Note that $\sigma_{i_e} = e$.

Definition 4. (Timed traces of IMSCs). Let $\mathcal{I}$ be an IMSC. The set $P(\mathcal{I})$ contains all timed traces $\rho = (\sigma, \tau)$ where σ is a linearisation of the poset $(E, \rightarrow)$ and where τ is a non-decreasing time sequence constructed inductively according to the following rules for the timing of events:

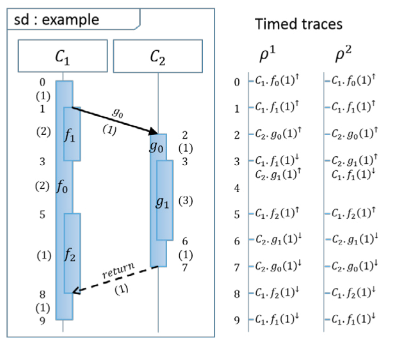

sd : example
C_1
C_2
f_0
f_1
f_2
g_0
g_1
g_0 (1)
return (1)
0 (1) 1 (2) 3 (2) 5 (1) 8 (1) 9
2 (1) 3 (3) 6 (1) 7
Timed traces
ρ^1
ρ^2
0 $C_1.f_0(1)^\uparrow$ $C_1.f_0(1)^\uparrow$
1 $C_1.f_1(1)^\uparrow$ $C_1.f_1(1)^\uparrow$
2 $C_2.g_0(1)^\uparrow$ $C_2.g_0(1)^\uparrow$
3 $C_1.f_1(1)^\downarrow$ $C_2.g_1(1)^\uparrow$ $C_2.g_1(1)^\uparrow$ $C_1.f_1(1)^\downarrow$
4
5 $C_1.f_2(1)^\uparrow$ $C_1.f_2(1)^\uparrow$
6 $C_2.g_1(1)^\downarrow$ $C_2.g_1(1)^\downarrow$
7 $C_2.g_0(1)^\downarrow$ $C_2.g_0(1)^\downarrow$
8 $C_1.f_2(1)^\downarrow$ $C_1.f_2(1)^\downarrow$
9 $C_1.f_1(1)^\downarrow$ $C_1.f_1(1)^\downarrow$

Fig. 2. On the left: a visualization of a time value assignment of the IMSC in Figure 1b. On the right: the two timed traces derived from the time sequence.

- if $pred(e) = \emptyset$, then $\tau_{i_e} = 0$,
- if $pred(e) = \{e_1, e_2, \ldots, e_n\}$, then there exist $d_{e_1,e} \in I(e_1 \to e), d_{e_2,e} \in I(e_2 \to e), \ldots, d_{e_n,e} \in I(e_n \to e)$ such that $\tau_{i_e} = max_{k\in\{1..n\}}\ \tau_{i_{e_k}} + d_{e_k,e}$.

4.2 Model checking problem

Our goal is to determine $\mathcal{I} \vDash \phi$ which expresses that ϕ is satisfied for all traces in $P(\mathcal{I})$.

Definition 5. (Satisfaction semantics of $\mathcal{I} \vDash \phi$). Let $\mathcal{I}$ be an IMSC and ϕ be an MTL formula. Then $\mathcal{I} \vDash \phi$ holds if and only if for all $\rho \in P(\mathcal{I})$, $\rho \vDash \phi$.

Since $P(\mathcal{I})$ can be (uncountably) infinite, we cannot enumerate all timed traces to solve the model-checking problem. Therefore, we need an alternative method.

5. MODEL CHECKING OF INTERVAL MSCS

In this section, we encode the IMSC model-checking problem in a constraint satisfaction problem (CSP). We use Satisfiability Modulo Theories (SMT) as a framework. In Section 5.1 we define the constraint language. Section 5.2 encodes an IMSC $\mathcal{I}$ as a constraint $C^{\mathcal{I}}$ that models the semantics of IMSCs given in Definition 4. Then, in Section 5.3, we phrase a constraint $C^{\mathcal{I},\phi}$ that encodes whether or not a timed trace of $\mathcal{I}$ exists that satisfies ϕ. In Section 5.4 we formulate the model checking problem $\mathcal{I} \vDash \phi$ in terms of the constraint satisfaction problem $C^{\mathcal{I}} \wedge C^{\mathcal{I},\neg\phi}$ and prove its correctness. Section 5.5 discusses the complexity of the encoding. Finally, Section 5.6 provides an optimized encoding for specific classes of formulas.

5.1 Constraint language

The model checking problem can be encoded in a system of constraints. The constraint language C follows the following grammar:

$$\begin{aligned} C &= \zeta \sim \zeta \mid C \wedge C \mid \neg C \\ \zeta &= \tau \mid \tau + \tau \mid \tau - \tau \mid c, \end{aligned} \tag{2}$$

where τ denotes a real valued variable, $c \in \mathbb{Q}_0^+$ denotes a rational number constant, and $\sim\ \in \{<, \leq, =, \neq, \geq, >\}$ a relation. We use the commonly used shorthand notations (e.g. $C \vee C \equiv \neg(\neg C \wedge \neg C)$). Furthermore, we use the shorthand $\bigwedge_{1\leq i\leq n} c_i$ to denote $c_1 \wedge c_2 \wedge \cdots \wedge c_n$ and $\bigvee_{1\leq i\leq n} c_i$ to denote $c_1 \vee c_2 \vee \cdots \vee c_n$. Finally, we use $\tau = max_{x\in X}(\zeta_x)$ as a shorthand for $\bigwedge_{x\in X}(\bigwedge_{x'\in X}(\zeta_x \geq \zeta_{x'}) \Rightarrow (\tau = \zeta_x))$ for some finite set X.

Let υ be a function that assigns values to the variables used in a constraint. If υ assigns values such that the constraint C evaluates to true, then υ is a solution to the constraint satisfaction problem, which we denote by $\upsilon \vDash C$.

The constraint language C defined in Equation 2, a quantifier free logic over uninterpreted functions and real arithmetic, is a first-order logic of real-closed fields, which is proven to be decidable by (Tarski and Jónsson, 1949). Hence, since our model-checking problem $\mathcal{I} \vDash \phi$ can be encoded in this formalism, it is decidable as well.

5.2 Encoding the IMSC

An IMSC $\mathcal{I}$ is encoded as a constraint $C^{\mathcal{I}}$. For every event $e \in E$, we introduce a real valued variable τ^e representing the time value of the event and a real valued variable i^e to encode the ordering relation. For every edge $e' \to e$ we introduce a real valued variable $\tau^{e',e}$ representing the time value taken from the interval $I_{e',e}$. For the lower-bound (upper-bound) of the interval we introduce a rational constant $c_l^{e,e'}$ ($c_u^{e,e'}$). We let $\prec$ denote $<$ or $\leq$ and $\succ$ denote $>$ or $\geq$ depending on whether an interval I is open/closed at the left/right side, respectively. The encoding follows the semantics of $P(\mathcal{I})$ as defined in Definition 4 and is given as follows.

Definition 6. Let $\mathcal{I}$ be an IMSC. The constraint $C^{\mathcal{I}}$ that encodes $\mathcal{I}$ is defined as follows:

$$\begin{aligned} C^{\mathcal{I}} = &\bigwedge_{(e',e)\in\to} (c_l^{e',e} \prec \tau^{e',e} \wedge \tau^{e',e} \prec c_u^{e',e} \wedge i^{e'} < i^e) \wedge \\ &\bigwedge_{e\in E}\ \bigwedge_{e'\in E\setminus\{e\}} (\tau^e = \tau^{e'} \Rightarrow i^e \neq i^{e'}) \wedge \\ &\bigwedge_{e\in E^{init}} (\tau^e = 0) \wedge \\ &\bigwedge_{e\notin E^{init}} (\tau^e = max_{e'\in pred(e)}(\tau^{e'} + \tau^{e',e}) \end{aligned}$$

The first two lines of the constraint encode that, given a solution υ, there is a total ordering of events, i.e. the lexicographic ordering of events by their time value variables τ^e and their ordering variables i^e is total.

5.3 Encoding the formula with respect to the IMSC

In this section, we obtain the condition $C^{\mathcal{I},\phi}$ that encodes whether or not there exists a timed trace of $\mathcal{I}$ that satisfies ϕ. This constraint is attained according to the structure of the formula interpreted on events. For every until operator with interval I, there are rational valued constants c_l^I and c_u^I encoding the lower- and upper-bound of the interval, respectively. The encoding of the formula ϕ with respect to $\mathcal{I}$ interpreted at event $e \in E$, denoted $C_e^{\mathcal{I},\phi}$, is as follows.

Definition 7. Given an IMSC $\mathcal{I}$, an event $e \in E$, and a formula $\phi \in$ MTL, the constraint $C_e^{\mathcal{I},\phi}$ is inductively computed following the structure of ϕ:

$$C_e^{\mathcal{I},p} = \begin{cases} \mathit{true} & \mathit{if}\ p \in e \\ \mathit{false} & \mathit{if}\ p \notin e \end{cases},$$

$C_e^{\mathcal{I},\psi_1 \wedge \psi_2} = C_e^{\mathcal{I},\psi_1} \wedge C_e^{\mathcal{I},\psi_2}$,

$C_e^{\mathcal{I},\neg\psi} = \neg C_e^{\mathcal{I},\psi}$, and

$$C_e^{\mathcal{I},\psi_1 \mathbf{U}_I \psi_2} = \bigvee_{e' \in E} \big(C_{e'}^{\mathcal{I},\psi_2} \wedge \tau^{e'} - \tau^e \succ c_l^I \wedge \tau^{e'} - \tau^e \prec c_u^I \wedge (\tau^{e'} = \tau^e \Rightarrow i^{e'} \geq i^e) \wedge \bigwedge_{e'' \in E} (\tau^e \leq \tau^{e''} \wedge \tau^{e''} \leq \tau^{e'} \wedge (\tau^e = \tau^{e''} \Rightarrow i^e \leq i^{e''}) \wedge (\tau^{e'} = \tau^{e''} \Rightarrow i^{e''} < i^{e'})) \Rightarrow C_{e''}^{\mathcal{I},\psi_1}\big).$$

The encoding of the formulas p, $\psi_1 \wedge \psi_2$, $\neg\psi$ is straightforward. The intuition behind the encoding of $\psi_1 \mathbf{U}_I \psi_2$ is as follows. The disjunction $\bigvee_{e' \in E}$ ranges over all events. The constraint $C_{e'}^{\mathcal{I},\psi_2}$ encodes that event ϕ_2 holds at e'. An event can only make the right hand side of the until operator true, if it occurs within the interval. This is encoded as $\tau^{e'} - \tau^e \succ c_l^I \wedge \tau^{e'} - \tau^e \prec c_u^I$. Now, if 0 is in the interval, we need to assert that e' occurs after e or is e itself (corresponding to $j \geq i$ in Definition 1). This assertion is encoded as $(\tau^{e'} = \tau^e \Rightarrow i^{e'} \geq i^e)$. Then, for every event e'', if it occurs between e (inclusive) and e' (exclusive), then $(\rho, i_{e''}) \vDash \psi_1$. This is encoded as the conjunction over all events $\bigwedge_{e'' \in E}$. To encode that the time value of e'' lies between the time values of e and e', we have constraints $\tau^e \leq \tau^{e''}$ and $\tau^{e''} \leq \tau^{e'}$. To encode that if e'' occurs at the same time as e implies the index of e'' to be at least as large as the index of e, there is the constraint $(\tau^e = \tau^{e''} \Rightarrow i^e \leq i^{e''})$. To encode that if e'' occurs at the same time as e' implies the index of e'' to be smaller than the index of e', we have constraint $(\tau^{e'} = \tau^{e''} \Rightarrow i^{e''} < i^{e'})$. Finally, $C_{e''}^{\mathcal{I},\psi_1}$ encodes that ψ_1 holds for event e''.

We encoded constraint $C_e^{\mathcal{I},\phi}$ for every event e. A timed trace $\rho \in P(\mathcal{I})$ satisfies ϕ iff the first event in ρ satisfies ϕ. This event must be from the set of initial events $E^{init} = \{e \in E \mid pred(e) = \emptyset\}$. Furthermore, if the trace ρ with first event e satisfies ϕ, then the constraint $C_e^{\mathcal{I},\phi}$ is true, and $C^{\mathcal{I},\phi}$ must also be true. Therefore, since any trace starts with one of the initial events and $C^{\mathcal{I},\phi}$ encodes whether there exists a timed trace in $\mathcal{I}$ that makes ϕ true, it is sufficient to satisfy at least one of the constraints $C_e^{\mathcal{I},\phi}$. Hence, the constraint $C^{\mathcal{I},\phi}$ is the disjunction of the constraints for all initial events: $C^{\mathcal{I},\phi} = \bigvee_{e \in E^{init}} C_e^{\mathcal{I},\phi}$.

5.4 The model-checking problem as an SMT problem

We are now ready to express the IMSC model-checking problem in terms of an SMT problem. As we have established in Section 5.2, the constraint $C^{\mathcal{I}}$ encodes the timing semantics of the IMSC. In Section 5.3, we have encoded as $C^{\mathcal{I},\phi}$ the timing constraints between events that encodes the property of interest (ϕ). We prove the following lemma:

Lemma 1. Let $\mathcal{I}$ be an IMSC and let ϕ be an MTL formula. There exists a timed trace $\rho \in P(\mathcal{I})$ such that $\rho \vDash \phi$ iff there exists a valuation v such that $v \vDash C^{\mathcal{I}} \wedge C^{\mathcal{I},\phi}$.

Proof of Lemma 1. We sketch a proof in both directions. The proof is available in (Jonk et al., 2020).

$\Rightarrow$ Let $\rho \in P(\mathcal{I})$ be such that $\rho \vDash \phi$. Construct a valuation v such that for every event $e \in E$, $v(\tau^e) = \tau_{i_e}$ and $v(i^e) = i_e$. It can be shown that $v \vDash C^{\mathcal{I}}$. Observe that if $v \vDash C_{\sigma_1}^{\mathcal{I},\phi}$, then, since σ_1 must be an initial event, $v \vDash C^{\mathcal{I},\phi}$ follows from the disjunction over all initial events. We can show that $\rho \vDash \phi$ implies $v \vDash C_{\sigma_1}^{\mathcal{I},\phi}$ by induction on the length of the formula.

$\Leftarrow$ Let v be such that for some $e \in E^{init}$, $v \vDash C^{\mathcal{I}} \wedge C_e^{\mathcal{I},\phi}$ and for all $e' \in E$, $v(i^e) \leq v(i^{e'})$. Order the events E according to the lexicographic ordering of ascending time valuations and ascending indices. Construct a timed trace ρ such that for every event $e \in E$, $\tau_{i_e} = v(\tau^e)$ and i_e according to the ordering. It can be shown that $\rho \in P(\mathcal{I})$. Since $v \vDash C_e^{\mathcal{I},\phi}$ and e occurs first in the lexicographic order, it must be the first event in ρ. Then, $\rho \vDash \phi$, which can be shown by induction on the length of the constraint. □

We can now formulate the main theorem of the paper:

Theorem 1. Let $\mathcal{I}$ be an IMSC and let ϕ be a MTL formula. Then, $\mathcal{I} \vDash \phi$ iff $C^{\mathcal{I}} \wedge C^{\mathcal{I},\neg\phi}$ is not satisfiable.

Proof of Theorem 1.

$\mathcal{I} \vDash \phi$ iff $\forall_{\rho \in P(\mathcal{I})}\ (\rho, 1) \vDash \phi$	Definition 5
iff $\neg\exists_{\rho \in P(\mathcal{I})}\ (\rho, 1) \vDash \neg\phi$	DeMorgan
iff $\neg\exists_v\ v \vDash C^I \wedge C^{\mathcal{I},\neg\phi}$	Lemma 1
iff $C^I \wedge C^{\mathcal{I},\neg\phi}$ is unsatisfiable	□

5.5 Complexity analysis

We analyse the time and space complexity of the SMT problem. The variables that occur in $C^{\mathcal{I}} \wedge C^{\mathcal{I},\neg\phi}$ correspond to the events and edges in the IMSC $\mathcal{I}$. We create two variables for each event and one variable for each edge, so the number of variables in the CSP is $2|E| + |{\rightarrow}|$.

The encoding of $C^{\mathcal{I}}$ generates four clauses per edge (three from the first constraint in Definition 6 and one from the last) and one clause for every combination of events, so in total $\mathcal{O}(\frac{1}{2}|E|^2 + 4|{\rightarrow}|)$ clauses. The encoding of $C^{\mathcal{I},\phi}$ depends on the length of the formula. The encoding of an until operator is a disjunction over all events, and for each event there is a conjunction over all events. The complexity for an until operator is thus $\mathcal{O}(|E|^2)$. In the worst case, until formulas are nested. Therefore, the total worst case number of clauses in the CSP is $\mathcal{O}(|E|^{2|\phi|})$. In our experiments, we use the z3 theorem prover. z3 uses CDCL (Marques-Silva and Sakallah (1999)), which is based on DPLL algorithms (Davis et al. (1961)), which have a worst-case running time of $\mathcal{O}(2^n)$ and a worst case space complexity of $\mathcal{O}(n)$, where n is the size of the input (number of variables plus number of clauses). The time complexity of the SMT problem is therefore $\mathcal{O}(2^{(|E|^{|\phi|})})$ and the space complexity is $\mathcal{O}(|E|^{2|\phi|})$.

5.6 Optimization of the encoding

We discuss two practical optimizations for the encoding of the IMSC. The first optimization concerns formulas for which the until operator only appears as a finally or globally operator. The second optimization concerns removing and simplifying clauses from the constraints that do not influence the result.

In our encoding of the formula $\psi_1 \mathbf{U}_I \psi_2$, we require constraints on the indices to guarantee the ordering of events. For formulas for which ψ_1 only occurs as *true* for until operators, i.e., only for finally and globally operators, then the constraint $C_{e''}^{\mathcal{I},true}$ is equivalent to *true* for all e''. This results in a more compact constraint $C_e^{\mathcal{I},true\mathbf{U}_I\psi_2}$: $\bigvee_{e' \in E} C_{e'}^{\mathcal{I},\psi_2} \wedge \tau^{e'} - \tau^e \succ c_l^I \wedge \tau^{e'} - \tau^e \prec c_u^I \wedge (\tau^{e'} = \tau^e \Rightarrow i^{e'} \geq i^e)$. We let $C^{\mathcal{I},*}$ be the encoding of the IMSC $\mathcal{I}$ without the constraints $\bigwedge_{e \in E} \bigwedge_{e' \in E \setminus \{e\}} (\tau^e = \tau^{e'} \Rightarrow i^e \neq i^{e'})$.

Lemma 2. Let $\mathcal{I}$ be an IMSC and ϕ be an MTL formula for which until operators only occur as finally or globally operators. Then, there exists a timed trace $\rho \in P(\mathcal{I})$ such that $\rho \vDash \phi$, if and only if there exists a valuation υ such that $\upsilon \vDash C^{\mathcal{I},*} \wedge C^{\mathcal{I},\phi}$.

Proof of Lemma 2. The proof for "⇒" is identical to the proof of Lemma 1. The proof for "⇐" follows from assuming a total order for events that are unordered in the poset $(E, \rightarrow)$ to construct a timed trace ρ, and otherwise follows the proof of Lemma 1. □

As a corollary to Lemma 2, we conclude a similar result as Theorem 1 for $C^{\mathcal{I},*}$.

Theorem 2. Let $\mathcal{I}$ be an IMSC and let ϕ be an MTL formula where until operators only occur as finally/globally operators. Then, $\mathcal{I} \vDash \phi$ iff $C^{\mathcal{I},*} \wedge C^{\mathcal{I},\neg\phi}$ is not satisfiable.

The second optimization follows from annihilation of conjunctions containing *false* (disjunctions containing *true*). We call $C_r^{\mathcal{I},\phi}$ the result of annihilating such clauses from $C^{\mathcal{I},\phi}$. Not every variable τ^e and i^e occur in $C_r^{\mathcal{I},\phi}$. Therefore, we can eliminate the events corresponding to these variables from the IMSC such that the encoding of the IMSC does not affect the model checking problem. An event e can be eliminated from $\mathcal{I}$ if τ^e and i^e do not occur in $C_r^{\mathcal{I},\neg\phi}$, according to the following rules:

R1 if $succ(e) = \emptyset$: remove e and all edges $e' \rightarrow e$.
R2 if $pred(e) = \{e_p\}$ and $succ(e) = \{e_s\}$: remove e and the edges $e_p \rightarrow e$ and $e \rightarrow e_s$. Then, if there does not exist an edge $e_p \rightarrow e_s$, add an edge $e_p \rightarrow e_s$ with interval the sum of the lower- and upper-bounds of the removed intervals. Otherwise, adjust the lower- and upper-bounds of the existing edge such that the interval on the edge is the maximum of the existing lower- and upper-bounds and the sum of the lower- and upper-bounds of the removed intervals.

$R1$ states that events at the end of the IMSC can be removed if their variables do not occur in $C_r^{\mathcal{I},\phi}$. $R2$ states that events with one incoming and one outgoing event can be removed and the intervals on the incoming edge and outgoing edge can be added. If this results in two edges between the predecessor and the successor, the edges are merged with by taking the maximum values for the lower- and upper-bounds, respectively.

$R1$ and $R2$ can be repeatedly applied on the events in $\mathcal{I}$ until no more events can be removed. This results in a reduced IMSC $\mathcal{I}_r$. The following theorem states that the reduced model checking problem is equivalent to the model checking problem in Theorem 1.

Theorem 3. Let $\mathcal{I}$ be an IMSC and ϕ be a formula. $C^{\mathcal{I}_r} \wedge C_r^{\mathcal{I},\neg\phi}$ is not satisfiable iff $C^{\mathcal{I}} \wedge C^{\mathcal{I},\neg\phi}$ is not satisfiable.

Proof of Theorem 3. The proof follows by showing that the time valuations of the remaining variables remain unchanged. The full proof is available in (Jonk et al., 2020).

6. EXPERIMENTAL EVALUATION

In this section we evaluate our model checking method and compare the SMT based solution of the model checking problem to the automaton based strategy, choosing Uppaal as a tool. We do so with a synthetic case study to demonstrate the scalability in terms of computation time and memory usage for our encoding and via manually translating the IMSC to an equivalent timed automaton. We also validate the method using an industrial case study. We implemented the encoding according to Definitions 6 and 7. The encoding is done such that it can be used as input for (z3, 2011). The experiments are done on an Intel Core i7-6560 2.20GHz quad core CPU with 16GB of RAM on a Windows 8.1 Enterprise 64-bit operating system.

6.1 A synthetic case-study: TGPP

We use the Timed Generic Pipeline Paradigm (TGPP) as a synthetic case study. The case study is inspired by the case study used in (Woźna-Szcześniak and Zbrzezny, 2014) and (Woźna-Szcześniak et al., 2017), based on a Generic Pipeline Paradigm introduced in (Peled, 1993). The proposed case study is originally presented in the context of timed automata. We adapt the case study to fit our formalism of IMSCs. We also create a timed automaton following the semantics of the IMSC to be used as input to Uppaal.

Figure 3 shows the UMSC where two pieces of data are being produced, processed by one intermediate node and finally consumed. The *Producer* component consists of a function *prod_loop* which subsequently calls the functions *prod* and *send* repeatedly. After a piece of data is produced, it is sent to the first node. When the *Node1* component is notified of the message from the producer, it starts a handler which calls the functions *rec*, *process* and *send* in sequence. At the end of *rec*, it notifies the producer it has received the piece of data. After processing the data, it sends it to the next node, or, as in Figure 3, to the consumer, if it is the last node. The node may now receive the next piece of data. Finally, when a piece of data reaches the consumer, it starts a handler which receives the data and consumes it by calling function *cons*.

The synthetic case can be scaled by increasing the number n of intermediate nodes and/or the number r of data pieces that are produced by the producer. For our experiments, we pick $r = 10$ for all experiments, and vary n between 0 and 30. There is a time-out at 600 seconds for each

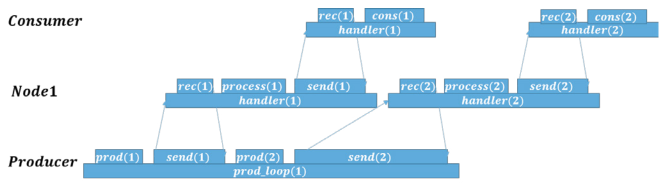
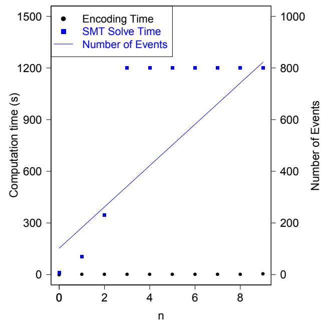
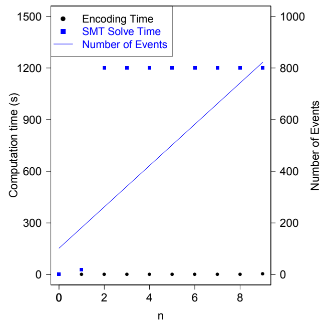


Fig. 3. The UMSC of a Timed Generic Pipeline Paradigm execution with one intermediate processing node.





(a) Computation time for $x = 12$

(b) Computation time for $x = 20 + 6n$

Fig. 4. The computation time and number of events for ϕ_1 as a function of n.

experiment. All intervals on the edges between events are set to $[1, 2]$. We evaluate the following formulas:

- $\phi_1 = \mathbf{F}_{[0,x]}(prod \wedge \downarrow)\mathbf{U}_{[0,\infty)}(prod \wedge \downarrow \wedge\, r)$ *"Always until the producer has produced r pieces of data, new data is produced within x time units" (we evaluate using $x = 12$ and $x = 20 + 6n$),*
- $\phi_2 = \phi_{2,1} \wedge \phi_{2,2} \wedge \cdots \wedge \phi_{2,r}$, where

$$\begin{aligned}\phi_{2,i} =\,& \mathbf{G}_{[0,\infty)}((prod \wedge i \wedge \downarrow) \Rightarrow \\ & \mathbf{F}_{[6+6n,12+12n]}(cons \wedge i \wedge \downarrow))\end{aligned}$$

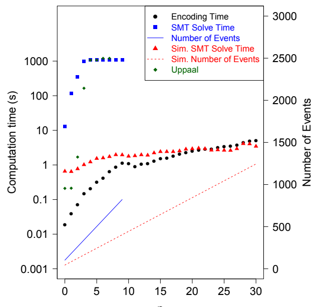
  "For each piece of data, when it is produced, it is consumed within $6 + 6n$ to $12 + 12n$ time units," and
- $\phi_3 = \mathbf{G}_{[0,\infty)}((cons \wedge \downarrow \wedge \neg r \Rightarrow \mathbf{F}_{[1,16+6n]}(cons \wedge \uparrow))$ *"Whenever a cons execution finishes (except the last), the next cons execution will start within 1 to $16 + 6n$ time units."*

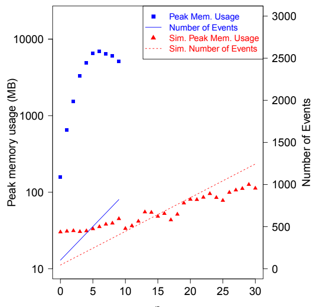
The following formulas are expected to be true: ϕ_1 for $x = 20 + 6n$, $\phi_{2,1}$ and ϕ_3. The other formulas are expected to be false. We evaluate the formulas on the Timed Generic Pipeline Paradigm IMSC and equivalent formulas for ϕ_2 and ϕ_3 in Uppaal on the timed automaton.

Figure 4 shows the results for ϕ_1. Since the formula contains an until operator, the optimizations discussed in Section 5.6 do not help. The computation time to encode the constraint (black circles), the time to solve the constraint problem (blue line) and the number of events (blue squares) are shown for $x = 12$ (a) and $x = 20 + 6n$ (b). The figure shows that the encoding in SMT takes significantly less time than solving the SMT problem. Even for small n, the solving time for the SMT reaches the time-out of 1200 seconds.

Figure 5 and 6 show the results for ϕ_2 and ϕ_3 for the unoptimized encoding. We evaluated each sub-formula $\phi_{2,i}$ separately. In (a), the (mean) computation time to encode the constraint (black circles), the mean computation time to solve the constraint problem (blue line) and the number of events (blue squares) are shown, and in (b) the peak memory usage and the number of events are shown. The figures show that Uppaal performs better for smaller models, but does not scale as well as our method.

We implemented the optimizations discussed in Section 5.6 and verified formulas ϕ_2 and ϕ_3 again. Figures 5 and 6 show the results for ϕ_2 and ϕ_3, respectively. At the left-hand side of the figures, the (mean) time to solve the constraint problem (red dashed line) and the number of

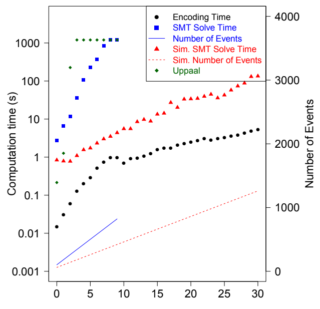


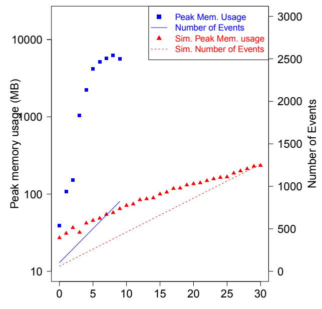


(a) Computation time

(b) Peak memory usage

Fig. 5. The mean computation time, peak memory usage and number of events for $\phi_{2,i}$ as a function of n.





(a) Computation time

(b) Peak memory usage

Fig. 6. The mean computation time, peak memory usage and number of events for ϕ_3 as a function of n.

events (red triangles) for the simplified problem are shown, and at the right-hand side of the figures, the peak memory usage (red dashed line) and the number of events (red triangles) are shown. Since ϕ_2 only uses finally and globally operators, the optimization is effective at reducing the number of events, and shows a clear improvement over the regular IMSC. The figures show that ϕ_3 takes significantly longer to compute than each individual $\phi_{2,i}$, while the memory usage is similar to the memory usage for ϕ_2. The results for the simplified constraints are an improvement over the regular constraints. Uppaal is able to verify up to depth five, whereas our simplified approach allow us to verify properties upto a pipeline depth of thirty.

6.2 Industrial case study

We apply the verification technique to a case study on lithography scanners. Lithography scanners use an optical system to project an image of a pattern on a quartz plate, called the reticle, onto a photosensitive layer on a substrate, called the wafer. Since one wafer can contain many ICs, typically 100 or more, the wafer needs to be repositioned from exposure to exposure. Exposures take place during scanning motions of the wafer, as illustrated in Figure 7 (redrawn from (Butler, 2011)). A wafer is divided in image fields, which are scanned one by one according to a step and scan pattern. Each physical step and scan action involves a motion path of the wafer defined in terms of a setpoint profile which has to be tracked with nanometer positional accuracy. Due to physical disturbances, e.g. temperature perturbations, feed-back setpoint adjustments are computed based on system-wide sensor and data input. To obtain optimal system throughput, all

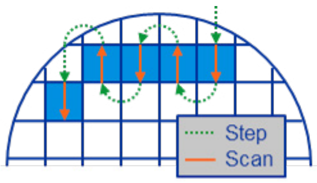


Fig. 7. A schematic overview of a wafer being exposed. Redrawn from Figure 15 in (Butler, 2011).

step and scan actions must be executed in a concatenated fashion. Therefore, each setpoint adjustment computation is subject to latency requirements.

We created a UMSC $\mathcal{U}$ and measured execution times for each edge from a run of a TWINSCAN machine. $\mathcal{U}$ contains 48.7 million events and 60.4 million edges. We fold the function executions and the execution times occurring in each setpoint adjustment computation to an IMSC $\mathcal{I}_{e\_l}$ covering a single setpoint computation, where the lower- and upper-bound of each interval is determined by the minimum and maximum execution times observed for each edge. The resulting IMSC $\mathcal{I}_{e\_l}$ contains 952 events and 987 edges. Requirements are formulas of the form $\mathbf{G}_{[0,\infty)}\big(a \Rightarrow \mathbf{F}_{[0,t)}(b)\big)$. That is, from the moment event 'a' occurs, event 'b' occurs within t time units. We verified four latency requirements on $\mathcal{I}_{e\_l}$. All of the latency requirements gave a result within fifteen seconds.

7. CONCLUSIONS

We have studied the verification of temporal properties on Interval Message Sequence Charts with temporal concepts modelled by interval-labelled edges between events. We approached the problem by encoding the model and properties as an SMT problem. Our encoding to a constraint satisfaction problem supports the full MTL. We have validated the encoding of the model checking problem in an SMT problem for a scalable synthetic case and an industrial case. The results of the synthetic case show that verifying properties on a system containing in the order of thousand events is feasible for formulas that do not contain the until operator. These properties are verified a lot more time-efficiently than equivalent properties that verified using Uppaal. The industrial case study shows the method to be applicable to verify a relevant property of the software of a lithography scanner.

REFERENCES

Alur, R. and Yannakakis, M. (1999). Model checking of message sequence charts. In *International Conference on Concurrency Theory*, 114–129. Springer.

Blom, S., Ioustinova, N., and Sidorova, N. (2003). Timed verification with μCRL. In *International Andrei Ershov Memorial Conference on Perspectives of System Informatics*, 178–191. Springer.

Butler, H. (2011). Position control in lithographic equipment [applications of control]. *IEEE control systems*, 31(5), 28–47.

Davis, M., Logemann, G., and Loveland, D.W. (1961). *A machine program for theorem-proving.* New York University, Institute of Mathematical Sciences.

Groote, J.F., Mathijssen, A., Reniers, M., Usenko, Y., and Van Weerdenburg, M. (2007). The formal specification language mCRL2. In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

Jonk, R., Voeten, J., Geilen, M., Basten, T., and Schiffelers, R. (2020). SMT-based verification of temporal properties for component-based software systems. *ES Report*, 2020(01), http://www.es.ele.tue.nl/esreports.

Koymans, R. (1990). Specifying real-time properties with metric temporal logic. *Real-time systems*, 2(4), 255–299.

Kurtev, I., Schuts, M., Hooman, J., and Swagerman, D.J. (2017). Integrating Interface Modeling and Analysis in an Industrial Setting. In *MODELSWARD*, 345–352.

Loose, R., van der Sanden, B., Reniers, M., and Schiffelers, R. (2018). Component-wise supervisory controller synthesis in a client/server architecture. *IFAC-PapersOnLine*, 51(7), 381–387.

Malinowski, J. and Niebert, P. (2010). SAT based bounded model checking with partial order semantics for timed automata. In *Tools and Algorithms for the Construction and Analysis of Systems*, 405–419. Springer.

Marques-Silva, J.P. and Sakallah, K.A. (1999). GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5), 506–521.

Matena, V., Stearns, B., and DeMichiel, L. (2003). *Applying enterprise JavaBeans: component-based development for the J2EE platform.* Pearson Education.

Peled, D. (1993). All from one, one for all: on model checking using representatives. In *International Conference on Computer Aided Verification*, 409–423. Springer.

Sharp, D.C. (1998). Reducing avionics software cost through component based product line development. In *Digital Avionics Systems Conference, 1998. Proceedings., 17th DASC.*, volume 2, G32–1. IEEE.

Tarski, A. and Jónsson, B. (1949). Cardinal algebras.

UML (2005). Unified Modeling Language 2.0. `http://omg.org/spec/UML/2.0/About-UML/`. Accessed: 29-11-2019.

Uppaal (1995). Uppaal. `http://www.uppaal.org/`. Accessed: 3-12-2019.

Van Ommering, R., Van Der Linden, F., Kramer, J., and Magee, J. (2000). The Koala component model for consumer electronics software. *Computer*, 33(3), 78–85.

Wilhelm, R. et al. (2008). The Worst-case Execution-time Problem - Overview of Methods and Survey of Tools. *ACM Transactions on Embedded Computing Systems*, 7(3), 36:1–36:53.

Woźna-Szcześniak, B., Zbrzezny, A.M., and Zbrzezny, A. (2017). SMT-based Searching for k-quasi-optimal Runs in Weighted Timed Automata. *Fundamenta Informaticae*, 152(4), 411–433.

Woźna-Szcześniak, B. and Zbrzezny, A. (2014). Checking MTL properties of discrete timed automata via bounded model checking. *Fundam. Inform.*, 135(4), 553–568.

z3 (2011). Z3. `http://github.com/Z3Prover/z3`. Accessed: 29-11-2019.

Zbrzezny, A.M., Szymoniak, S., and Kurkowski, M. (2019). Efficient Verification of Security Protocols Time Properties Using SMT Solvers. In *International Joint Conference: CISIS 2019 and ICEUTE 2019*, 25–35. Springer.

Zbrzezny, A. (2005). Sat-based reachability checking for timed automata with diagonal constraints. *Fundamenta Informaticae*, 67(1-3), 303–322.