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

Citation for published version (APA):

DOI: 10.1016/j.ifacol.2021.04.045

Document status and date:
Published: 01/01/2020

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.
SMT-based verification of temporal properties for component-based software systems


*** 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 (Kurtnev 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...
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 a transition system 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 (Woźniak-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 space state 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 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 \( \tau = \tau_1 \tau_2 \ldots \) 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 \( a = \sigma_1 \sigma_2 \cdots \sigma_n \) of states where \( \sigma_i \in 2^{AP} \) for all \( 1 \leq i \leq n \).

A timed trace \( \rho = (\sigma, \tau) \) is a pair consisting of a state sequence \( \sigma \) and a time sequence \( \tau \), 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 \( i \leq |\tau| \), \( \rho_i \) refers to the state-time pair \((\sigma_i, \tau_i)\) at position \( i \), and \( (\rho, i) \) refers to the tail of \( \rho \) starting at position \( i \).

The set of MTL formulas, ranged over by \( \phi \), is inductively defined as follows:

\[
\phi = p \mid \phi \land \phi \mid \neg \phi \mid \phi U_1 \phi
\]

where \( p \in AP \) and \( I \subseteq [0, \infty) \) is an interval on \( \mathbb{R}_{0}^{+} \). We use the notation \((a, b], [a, b), (a, b], [a, b), (a, \infty) \) or \([a, \infty) \) \((a, b) \subseteq \mathbb{R}_{0}^{+}\) to represent an open, closed-left, 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 \( \rho = (\sigma_1, \tau_1), (\sigma_2, \tau_2), \ldots, (\sigma_n, \tau_n) \) be a finite timed trace of length \( n \geq 1 \). The satisfaction relation on \( \rho \) starting at position \( 1 \leq i \leq n \) is defined inductively as follows:

- \( (\rho, i) \models p \) iff \( p \in \sigma_i \);
- \( (\rho, i) \models \phi \land \psi \) iff \( (\rho, i) \models \phi \) and \( (\rho, i) \models \psi \);
- \( (\rho, i) \models \neg \phi \) iff \( (\rho, i) \not\models \phi \), and
- \( (\rho, i) \models \phi U_1 \psi \) iff there exists \( j \geq i \) s.t. \( (\rho, j) \models \psi \), \( \tau_j - \tau_i \in I \) and for all \( i \leq k < j \), \( (\rho, k) \not\models \phi \)

The satisfaction of \( \rho \models \phi \) is equivalent to \( (\rho, 1) \models \phi \). We use the following shorthand notations: \( false \equiv p \land \neg p, true \equiv \neg false, \phi \lor \psi \equiv \neg (\neg \phi \land \neg \psi), \phi \lor \psi \equiv \neg (\neg \phi \lor \neg \psi), \phi \land \psi \equiv \phi \lor \psi \land \neg (\neg \phi \land \neg \psi) \). In the shorthand for \( false \) (\( p \land \neg p \)), \( p \) denotes an arbitrary element of \( AP \), which is assumed to be a non-empty set. The operators \( F \) and \( G \) denote that \( \phi \) 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.
Each lifeline contains a component label from the set $C$ of component labels. Function executions are depicted by rectangles drawn on the component’s lifelines, annotated with a function label from the set $F$ 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 $M$ 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 $E = \{(c, f, i, s) \mid c \in C, f \in F, i \in \mathbb{N}^+, s \in S\}$ of events, where $S = \{\uparrow, \downarrow\}$. If $e = (c, f, i, s) \in E$ 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 $\downarrow)$. For events $e$, 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 $U$ is a three-tuple $(E, \rightarrow, m)$, where

- $E \subseteq E$ is a finite set of events,
- $\rightarrow \subseteq E \times E$ defines an ordering relation between events, and
- $m : \rightarrow \rightarrow M$ 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 $\text{pred}(e) = \{e' \in E \mid e' \rightarrow e\}$ and $\text{succ}(e) = \{e' \in E \mid e \rightarrow e'\}$ 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 $I$ is a tuple $(E, \rightarrow, m, I)$, where

- $(E, \rightarrow, m)$ is a UMSC, and
- $I$ is a function that maps each edge $e \rightarrow$ 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 $P(I)$ 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 $P(I)$ 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 $AP = \{C_1, C_2, f_0, f_1, f_2, g_0, g_1, 1, \uparrow, \downarrow\}$. 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 $(C_1, f_1, 1, \downarrow)$. These chosen values determine a time sequence $\tau$. The time value $\tau_e$ for a state $s_e$ 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 $\sigma$ of the poset $(E, \rightarrow)$ such that the time sequences are monotonically ascending. We use the notation $t_e$ to refer to the index at which event $e$ occurs in $\sigma$. Note that $\sigma_{t_e} = e$.

**Definition 4.** (Timed traces of IMSCs). Let $I$ be an IMSC. The set $P(I)$ contains all timed traces $\rho = (\sigma, \tau)$ where $\sigma$ is a linearisation of the poset $(E, \rightarrow)$ and where $\tau$ is a non-decreasing time sequence constructed inductively according to the following rules for the timing of events:
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 \( \tau_{ie} = 0 \),
- if \( pred(e) = \{e_1, e_2, \ldots, e_n\} \), then there exist \( d_{e_1,e} \in I(e_1 \rightarrow e), d_{e_2,e} \in I(e_2 \rightarrow e), \ldots, d_{e_n,e} \in I(e_n \rightarrow e) \) such that \( \tau_{ie} = \max_{k \in \{1, n\}} \tau_{ik} + d_{ek,e} \).

4.2 Model checking problem

Our goal is to determine \( I \models \phi \) which expresses that \( \phi \) is satisfied for all traces in \( P(I) \).

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

Since \( P(I) \) 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 \( I \) as a constraint \( C^I \) that models the semantics of IMSCs given in Definition 4. Then, in Section 5.3, we phrase a constraint \( C^{I,\phi} \) that encodes whether or not a timed trace of \( I \) exists that satisfies \( \phi \). In Section 5.4 we formulate the model checking problem \( I \models \phi \) in terms of the constraint satisfaction problem \( C^{I} \land C^{I,\phi} \) 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:

\[ C = \zeta \sim \zeta \mid C \land C \mid \neg C \]
\[ \zeta = \tau \mid \tau + \tau \mid \tau - \tau \mid c, \]

where \( \tau \) denotes a real valued variable, \( c \in \mathbb{Q}^+ \) denotes a rational number constant, and \( \sim \in \{<, \leq, =, \neq, \geq, >\} \) a relation. We use the commonly used shorthanded notations (e.g. \( C \lor C \equiv \neg(\neg C \land \neg C) \)). Furthermore, we use the shorthand \( \bigwedge_{i=1}^{n} c_i \) to denote \( c_1 \lor c_2 \lor \cdots \lor c_n \).

Finally, we use \( \tau = \max_{x \in X}(\zeta_x) \) as a shorthand for \( \bigwedge_{x \in X}(\bigwedge_{x' \in X}(\zeta_x \geq \zeta_{x'})) \) for some finite set \( X \).

Let \( v \) be a function that assigns values to the variables used in a constraint. If \( v \) assigns values such that the constraint \( C \) evaluates to true, then \( v \) is a solution to the constraint satisfaction problem, which we denote by \( v \models C \).

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 \( I \models \phi \) can be encoded in this formalism, it is decidable as well.

5.2 Encoding the IMSC

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

Definition 6. Let \( I \) be an IMSC. The constraint \( C^I \) that encodes \( I \) is defined as follows:

\[
C^I = \bigwedge_{(e',e) \in \rightarrow} (c^{e'}_i < \tau^{e',e} \land \tau^{e',e} < c^{e'}_i + i^{e'} < i^e) \land \\
\bigwedge_{e \in E} \bigwedge_{e' \in E \setminus \{e\}} (\tau^e = \tau^{e'} \Rightarrow i^e \neq i^{e'}) \land \\
\bigwedge_{e \in E^{init}} (\tau^e = 0) \land \\
\bigwedge_{e \notin E^{init}} (\tau^e = \max_{e' \in pred(e)} (\tau^{e'} + \tau^{e,e})).
\]

The first two lines of the constraint encode that, given a solution \( v \), there is a total ordering of events, i.e. the lexicographic ordering of events by their time values \( \tau^e \) 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 \( C^{I,\phi} \) that encodes whether or not there exists a timed trace of \( I \) that satisfies \( \phi \). 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 \( c^e_1 \) and \( c^e_2 \) encoding the lower- and upper-bound of the interval, respectively. The encoding of the formula \( \phi \) with respect to \( I \) interpreted at event \( e \in E \), denoted \( C^e_{I,\phi} \), is as follows.

Definition 7. Given an IMSC \( I \), an event \( e \in E \), and a formula \( \phi \in \text{MTL} \), the constraint \( C^e_{I,\phi} \) is inductively computed following the structure of \( \phi \):

\[
C^e_{I,\phi} = \begin{cases} 
\text{true} & \text{if } p \in e \\
\text{false} & \text{if } p \notin e
\end{cases}
\]
The encoding of the formulas \( p, \psi_1 \land \psi_2, \neg \psi \) is straightforward. The intuition behind the encoding of \( \psi_1 \cup \psi_2 \) is as follows. The disjunction \( \bigvee_{e \in E} \) ranges over all events. The constraint \( C_{\psi} \) encodes that event \( \phi_2 \) 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 \( \tau^{e'} - \tau^e \geq c^1_i - \tau^e \land \exists u_i \land (\tau^e = \tau^{e'} \implies \tau^e \geq \tau^{e''}) \land \land_{e'' \in E} (\tau^{e''} \leq \tau^{e''} \leq \tau^{e''} \land (\tau^e = \tau^{e''} \implies \forall (\tau^{e''} < \tau^{e''})) \implies C_{\psi}^{\psi_1}) \).

We can now formulate the main theorem of the paper:

**Theorem 1.** Let \( I \) be an IMSC and let \( \phi \) be a MTL formula. Then, \( I \models \phi \iff C_1 \land C_{\neg \phi} \) is not satisfiable.

**Proof of Theorem 1.**

\[ I \models \phi \iff \forall e \in E \models C_{\phi} \lor C_{\neg \phi} \land \neg \exists_{\rho \in \operatorname{P}(I)} (\rho, 1) \models \phi \]

Definition 5

\[ \iff \neg \exists_{\rho \in \operatorname{P}(I)} (\rho, 1) \models \phi \]

DeMorgan

\[ \iff \neg \exists_{\rho, v} \models C_1 \land C_{\neg \phi} \]

Lemma 1

\[ \iff C_I \land C_{\neg \phi} \text{ is unsatisfiable} \]

\( \Box \)

5.5 Complexity analysis

We analyse the time and space complexity of the SMT problem. The variables that occur in \( C_1 \land C_{\neg \phi} \) correspond to the events and edges in the IMSC \( I \). 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 \( C_1 \) 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 \( O(|2| E| + |\rightarrow|) \) clauses. The encoding of \( C_{\neg \phi} \) 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 \( O(|E|^2) \). In the worst case, until formulas are nested. Therefore, the total worst case number of clauses in the CSP is \( O(|E|^2|\phi|) \).

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 \( O(2^n) \) and a worst case space complexity of \( O(n) \), where \( n \) is the size of the input (number of variables plus number of clauses). The time complexity of the SMT problem is therefore \( O(2^n|\phi|) \) and the space complexity is \( O(|E|^2|\phi|) \).

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 $\psi_1 \cup I \psi_2$, we require constraints on the indices to guarantee the ordering of events. For formulas for which $\psi_1$ only occurs as true for until operators, i.e., only for finally and globally operators, then the constraint $C_{\psi_1}^{I}$ is equivalent to true for all $e'$. This results in a more compact constraint $C_{\psi_1}^{I}$:

$$\forall e' \in E, C_{\psi_1}^{I} \land \tau' = \tau \land c = c' \land \tau' < c'_u \land (\tau' = \tau \Rightarrow i' \geq i').$$

We let $C_{\psi_1}^{I}$ be the encoding of the IMSC $I$ without the constraints $\land e \in E \land c \in \{c\}(\tau' = \tau \Rightarrow i' \neq i')$.

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

**Proof of Lemma 2.** The proof for “$\Rightarrow$” is identical to the proof of Lemma 1. The proof for “$\Leftarrow$” follows from assuming a total order for events that are unordered in the poset $(E, \rightarrow)$ to construct a timed trace $\rho$, and otherwise follows the proof of Lemma 1. \qed

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

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

The second optimization follows from annihilation of conjunctions containing $\text{false}$ (disjunctions containing $\text{true}$). We call $C_{\phi}^{I}$ the result of annihilating such clauses from $C_{\phi}^{I}$. Not every variable $\tau$ and $i$ occur in $C_{\phi}^{I}$. 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 $I$ if $\tau$ and $i$ do not occur in $C_{\phi}^{I}$, according to the following rules:

R1: if $\text{succ}(e) = \emptyset$: remove $e$ and all edges $e' \rightarrow e$.

R2: if $\text{pred}(e) = \{e_p\}$ and $\text{succ}(e) = \{e_s\}$: 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 the 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 $C_{\phi}^{I}$. 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 $I$ until no more events can be removed. This results in a reduced IMSC $I_r$. The following theorem states that the reduced model checking problem is equivalent to the model checking problem in Theorem 1.

**Theorem 3.** Let $I$ be an IMSC and $\phi$ be a formula. $C_{\phi}^{I} \land C_{\phi}^{\neg}$ is not satisfiable if $C_{\phi}^{I} \land C_{\phi}^{\neg}$ 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 i7-6560u 2.20 GHz 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 $\text{prod}$ which subsequently calls the functions $\text{prod}$ and $\text{send}$ repeatedly. After a piece of data is produced, it is sent to the first node. When the Node 1 component is notified of the message from the producer, it starts a handler which calls the functions $\text{rec}$, $\text{process}$ and $\text{send}$ in sequence. At the end of $\text{rec}$, 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 $\text{cons}$.

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.
The second optimization follows from annihilation of con-...

As a corollary to Lemma 2, we conclude a similar result...
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 $I_{eq}$ 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 $I_{eq}$ contains 952 events and 987 edges. Requirements are formulas of the form $G[0,\infty)(a \Rightarrow F_{[0,t]}(b))$. That is, from the moment event $'a'$ occurs, event $'b'$ occurs within $t$ time units. We verified four latency requirements on $I_{eq}$. 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


