Optimistic and Pessimistic On-the-fly Analysis for Metric Temporal Graph Logic

Sven Schneider, Lucas Sakizloglou, Maria Maximova, and Holger Giese
University of Potsdam, Hasso Plattner Institute, Potsdam, Germany
{sven.schneider, lucas.sakizloglou, maria.maximova, holger.giese}@hpi.de

Abstract. The nonpropositional Metric Temporal Graph Logic (MTGL) specifies the behavior of timed dynamic systems given by timed graph sequences (TGSs), which contain typed attributed graphs representing system states and the elapsed time between states. MTGL satisfaction can be analyzed for finite TGSs by translating its satisfaction problem to the satisfaction problem of nested graph conditions using a folding operation (aggregating a TGS into a graph with history) and a reduction operation (translating an MTGL condition into a nested graph condition).

In this paper, we introduce an analysis procedure for MTGL to allow for an on-the-fly analysis of finite/infinite TGSs. To this end, we introduce a further (optimistic) reduction of MTGL conditions, which leads to violations during the on-the-fly analysis only when non-satisfaction is guaranteed in the future whereas the former (pessimistic) reduction leads to violations when satisfaction is not guaranteed in the future. We motivate the relevance of our analysis procedure, which uses both reduction operations, by means of a running example. Finally, we discuss prototypical support in the tool AutoGraph.

Keywords: Graph logic with binding · Nonpropositional metric temporal logic · Runtime monitoring · Three-valued logic

1 Introduction

The challenges for developing embedded real-time systems with a high degree of parallelism, data dependencies, and timing constraints that must adhere to a given specification are manifold. The formal verification of such systems given by formal models is often intractable and, moreover, such formal models cannot be obtained for systems with unpredictable behaviors such as human-in-the-loop systems. Model-based testing and runtime monitoring are two standard model-driven approaches supporting the engineering of such systems.

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 241885098, 148420506, 158230677.
© Springer Nature Switzerland AG 2020
F. Gadducci and T. Kehrer (Eds.): ICGT 2020, LNCS 12150, pp. 276–294, 2020.
https://doi.org/10.1007/978-3-030-51372-6_16
In this paper, we check the conformance of a \textit{timed graph sequence} (TGS), given by a sequence of states where time elapses between the states, against a formal specification. TGSs can be (a) generated using timed graph transformation \cite{3,7,17} or can be (b) returned incrementally by a monitor. While offline analysis procedures suffice for finite TGSs, infinite TGSs require an on-the-fly conformance analysis to determine violations as early and as precise as possible.

As a running example, we consider an operating system as an advanced embedded real-time system in which tasks are executed by handlers computing results. For such an operating system, we require the following property $P$.

$P$: Whenever a task $T$ is created in a system $S$, $S$ must have a handler $H$ applicable to $T$ (based on a common id). Moreover, within 10 time units, $H$ must produce a result $R$ with value $ok$ and, during the computation of $R$, no other handler $H'$ applicable to $T$ (based on a common id) may exist in $S$.

For specifying desired system behavior such as the property $P$ from above, we employ an extension of the \textit{Metric Temporal Graph Logic} (MTGL) \cite{8}. This logic permits to concisely express (a) state properties to specify single graphs in a TGS and (b) sequence properties relating graphs at different timepoints in a TGS by their attributes and inner structure. For state properties, MTGL subsumes the graph logic GL of nested graph conditions \cite{9}, which is as expressive as first-order logic on graphs. For sequence properties, MTGL has metric temporal operators that refer to matches of graph patterns in graphs in a TGS as first-class citizens. Due to these operators, MTGL is more expressive compared to metric temporal logics such as MTL \cite{12} only relying on atomic propositions since MTGL allows to keep track of an unbounded number of elements. For example, for the property $P$, we must separately track tasks $T$ and $T'$ for which corresponding results $R$ and $R'$ must be created before the corresponding deadline expires.

The main contributions of this paper are as follows. Firstly, we integrate the \textit{metric-exists-new} operator as a first-class citizen into MTGL, which was not explicitly done in \cite{8}. It matches graph patterns as early as possible to fix a timepoint from which a deadline can be started as in our running example when a new task is matched. Secondly, we formally integrate attribute quantification into MTGL and GL. Thirdly, we develop an on-the-fly checking procedure, which takes a formal MTGL specification and considers a TGS incrementally returning a lower (optimistic) and an upper (pessimistic) bound of the set of \textit{true violations}, which determine a ground truth of violations that would be obtained ideally. The optimistic lower bound does not contain all true violations to handle TGSs that are continued in a way leading to satisfaction in the future whereas the pessimistic upper bound contains additional false violations to handle TGSs that are continued in a way not leading to satisfaction in the future. Returning the optimistic bound in addition to the pessimistic bound as computed in \cite{8} results essentially in a three valued logic\footnote{At each timepoint during the on-the-fly analysis, we return either no violation, only a pessimistic violation, or a pessimistic and an optimistic violation.} where an intervention (e.g. by a user) may depend on whether a pessimistic violation is also an optimistic violation.
In Sect. 2, we recall symbolic graphs and the logic GL of nested graph conditions. In Sect. 3, we extend MTGL by integrating the operator \textit{metric-exists-new}. We present our on-the-fly analysis procedure and discuss its prototypical tool support by AutoGraph in Sect. 4. Finally, we discuss related work in Sect. 5 and conclude the paper with a summary and remarks on future work in Sect. 6.

2 Symbolic Graphs and Graph Logic

We now recall typed attributed graphs and nested graph conditions used for representing system states and properties on these states, respectively.

We use \textit{symbolic graphs} (see e.g. [18–20,24]), called graphs subsequently, to encode typed attributed graphs. Symbolic graphs are an adaptation of E-Graphs [5] where nodes and edges of a graph $G$ are connected to (sorted) variables $X_G$ instead of data nodes representing actual values. To specify the possible values of variables, graphs are equipped with an attribute constraint (AC) $\Theta_G$ over the variables $X_G$ (e.g. $x = 5$, $\exists y. x \leq y$, and $z = \text{“aabb”}$).

An AC $\theta$ is constructed using variables from a set $X$ and the usual operators for the sorts \texttt{bool}, \texttt{int}, \texttt{real}, and \texttt{string}, which range over the set $V$ of all values. Satisfaction of an AC $\theta$ by a valuation $\alpha : X \rightarrow V$ is denoted by $\alpha \models AC \theta$.

If an AC $\theta$ is satisfiable or tautological, we write sat$_\exists(\theta)$ or sat$_\forall(\theta)$, respectively.

In the following, we consider graphs that are typed over a type graph $TG$ using a typing morphism $\texttt{type} : G \rightarrow TG$. Type graphs restrict attributed graphs to an admitted subclass Graphs$(TG)$. The empty graph is denoted by $\emptyset$. For our running example, we employ the type graph $TG$ from Fig. 1a. Examples of graphs that are typed over $TG$ are given in Fig. 3b.

Morphisms $f : G_1 \rightarrow G_2$ between graphs $G_1$ and $G_2$ are defined as usual (see e.g. [24] for a formal definition) and consist of mappings between the components of $G_1$ and $G_2$. In the remainder of this paper, we only use morphisms $f : G_1 \hookrightarrow G_2$ for which all mappings are injective.

Moreover, we distinguish between two kinds of morphisms $f : G_1 \hookrightarrow G_2$. Firstly, \textit{restrictive (mono)morphisms} $f : G_1 \hookrightarrow G_2$ must ensure that the AC of $G_2$ is more restrictive compared to the AC of $G_1$. This means that each valuation that satisfies the AC of $G_2$ also satisfies the $f$-translated AC of $G_1$ (i.e., sat$_\forall(\Theta_{G_2} \rightarrow f_X(\Theta_{G_1}))$ where $f_X$ is the mapping contained in $f$ between the variables of $G_1$ and $G_2$). Secondly, \textit{consistent morphisms} $f : G_1 \hookrightarrow c G_2$ must ensure that the AC of $G_2$ is compatible with the AC of $G_1$. This means that there is at least one valuation that satisfies the AC of $G_2$ as well as the $f$-translated AC of $G_1$ (i.e., sat$_\exists(\Theta_{G_2} \land f_X(\Theta_{G_1})))$. See Fig. 1b for examples of restrictive and consistent morphisms. The initial (mono)morphism $i_G : \emptyset \rightarrow G$ for graph $G$ is restrictive and, when the AC of $G$ is satisfiable, also consistent.

2 The solver Z3 [16] has support for checking satisfiability of ACs but is known to return indefinite results because satisfiability is undecidable for ACs of unrestricted form. While Z3 always succeeds for our running example, we handle this special case in our prototypical implementation by providing warnings to the user.
The graph logic GL over nested graph conditions (GCs) can be used to specify graphs. GL features propositional connectives and the operator $\exists$ (called exists) to extend given matches of graph patterns (given by graphs) in a graph (called host graph) with a satisfiable AC. Technically, the exists operator describes the extension of a graph pattern $H$ using a restrictive monomorphism $f : H \hookrightarrow r H'$.

Definition 1 (Nested Graph Conditions (GCs)). If $H$ is a graph and $f : H \hookrightarrow r H'$ is a restrictive monomorphism, then $\phi_H$ is a nested graph condition (GC) over $H$, written $\phi_H \in \Phi^\text{GC}_H$, as follows.

$$\phi_H ::= \top | \neg \phi_H | \phi_H \land \phi_H | \exists(f, \phi_{H'})$$

We also make use of the operators false ($\bot$), disjunction ($\lor$), and universal quantification ($\forall$), which can be derived from the operators above.

The semantics of GL is given by the satisfaction relation below, which deviates from [24] by using (a) variable valuations $\alpha : X_H \rightarrow V$ and (b) consistent instead of restrictive matches $m : H \hookrightarrow_c G$. Most notably, the GC $\phi = \exists(f : H \hookrightarrow_r H', \phi')$ is satisfied by a match $m$ and a valuation $\alpha$ when they can be extended to a match $m' : H' \hookrightarrow_c G$ and a valuation $\alpha' : X_{H'} \rightarrow V$ that are consistent with $f$. To ensure that the valuation $\alpha'$ is used consistently when evaluating ACs occurring in graphs $\phi$, we require that the consistency condition is satisfied.
Fig. 2. Satisfaction of GCs by graphs

by \( m' \) (i.e., \( \text{sat}_\exists(m'_X(\Theta_{H'}) \land \Theta_G) \)) using a valuation compatible with \( \alpha' \). Finally, as in [9, 24], a graph \( G \) satisfies a GC defined over \( \emptyset \) when the initial morphism \( i_G: \emptyset \to cG \) and the empty valuation \( \alpha: \emptyset \to V \) satisfy the GC.

\[ (i_G, \alpha) \models \phi \]

This novel adaptation of the satisfaction relation (informally handled in [8]) allows to express quantification over attribute values as required for our on-the-fly analysis procedure in Sect. 4. For example, we can state that “for each \( x \in \mathbb{Z} \) satisfying \( 0 \leq x \leq 2 \) there is a variable \( y \in \mathbb{Z} \) with the value of \( x \)” using the GC \( \phi \) in Fig. 2a, which is satisfied by the graph \( G \) from Fig. 2b containing a matchable copy of \( x \).

\section{3 Metric Temporal Graph Logic}

The Metric Temporal Graph Logic (MTGL) over metric temporal graph conditions (MTGCs) [8] extends GL with two metric temporal operators. In particular, (a) the \textit{until} operator \( U \) is well-known from MTL [12] and (b) the formal integration of the \textit{metric-exists-new} operator \( \exists^N \) allows to match graph patterns at a future timepoint (restricted by an interval) when the match is first available.

\[ m'_X : X_G \to X_{H'} \]

The partial function \( m'_X^{-1} : X_G \to X_{H'} \) obtained as the reversal of the injective function \( m'_X : X_{H'} \to X_G \) does not replace variables in \( \Theta_G \) that are not mapped to by \( m'_X \). For simplicity, we assume that \( G \) and \( H' \) have disjoint sets of variables.
Definition 3 (Metric Temporal Graph Conditions (MTGCs)). If $H$ is a graph, $f : H \rightarrow H'$ is a restrictive monomorphism, and $I$ is an interval over $\mathbb{R}_0^+$, then $\phi_H$ is a metric temporal graph condition (MTGC) over $H$, written $\phi_H \in \Phi_{MTGC}^H$, as follows.

$$\phi_H ::= \top \mid \neg \phi_H \mid \phi_H \land \phi_H \mid \exists(f, \phi_H') \mid \exists^N(f, \phi_H') \mid \phi_H \cup I \phi_H$$

The derived operator metric-forall-new ($\forall^N$) is the dual operator to $\exists^N$. Also, operators such as eventually and globally can be derived using the operator $U$.

The integration of graph pattern matching of GCs (for state properties) and the two metric temporal operators (for sequence properties) allows for the formalization of properties where a match into a graph (established using the operators $\exists$ or $\exists^N$) is preserved/extended over multiple timed steps of the system (using the operator $U$) and where the duration of these timed steps can be specified using intervals. For our running example, see Fig. 3a for a formalization of the property $P$ introduced in Sect. 1 in the form of an MTGC $\psi$.

Fig. 3. The MTGC $\psi$ and the TGS $\pi$ from our running example
For well-definedness of the slicing operation (and the satisfaction relation of MTGL later on), we require that \( \text{length}(\pi) = \infty \) implies \( \text{dur}(\pi) = \infty \) ruling out Zeno TGSs.

For the semantics of MTGL, we define that a match \( m \) (also called binding) is propagated over a single timed span \((\delta, \ell, r)\) by adapting it according to the renaming given by \( \ell \) and \( r \). However, the propagation operation is partial when graph elements matched by \( m \) are not preserved across the timed span.

**Definition 4 (Operation \texttt{propagateMatch}).** If \( m : H \to c G_1, m' : H \to c G_2 \) are consistent morphisms, \( \rho = (\delta, \ell : D \to r, G_1, r : D \to r, G_2) \) is a timed span, and there is a consistent morphism \( \bar{m} : H \to c D \) s.t. \( \ell \circ \bar{m} = m \) and \( r \circ \bar{m} = m' \), then \( m' \) is obtained by propagation of \( m \) over \( \rho \), written \( m' \in \text{PM}(m, \rho) \). Also, we extend the operation \texttt{propagateMatch} to finite TGSs as expected.

The semantics of MTGL is given by the satisfaction relation below, which is defined as for GL for the operators \texttt{conjunction}, \texttt{negation}, and \texttt{exists} and uses a current observation timepoint \( t \) (which does not exceed the duration of the TGS) in addition to a consistent match \( m : H \to c G \) and a valuation \( \alpha : X_H \to \mathcal{V} \). Note that the target of the match \( m \) is adapted (via propagation over timed spans from TGS \( \pi \)) and the current timepoint \( t \) is modified (according to the provided intervals) only in the cases of the \texttt{metric-exists-new} and \texttt{until} operators. For these two metric temporal operators, we provide further informal explanations below. Finally, a TGS satisfies an MTGC defined over \( \emptyset \) when the TGS, the initial observation timepoint 0, the initial morphism \( \iota_0 : \emptyset \to c \pi_0 \) representing an empty binding, and the empty valuation \( \alpha : \emptyset \to \mathcal{V} \) satisfy the MTGC.

**Definition 5 (Satisfaction of MTGCs by TGSs).** If \( \phi \in \Phi_H^{\text{MTGC}} \) is an MTGC, \( \pi \in \Pi_G \) is a TGS, \( 0 \leq t \leq \text{dur}(\pi) \) is a timepoint before the end of \( \pi \), \( m : H \to c \pi_t \) is a consistent morphism into the graph at timepoint \( t \), and \( \alpha : X_H \to \mathcal{V} \) is a valuation, then \( (\pi, t, m, \alpha) \models_{\text{TGS}} \phi \), if an item applies.

- \( \phi = \top \).
- \( \phi = \neg \phi' \) and \( (\pi, t, m, \alpha) \not\models_{\text{TGS}} \phi' \).
- \( \phi = \phi_1 \land \phi_2 \), \( (\pi, t, m, \alpha) \models_{\text{TGS}} \phi_1 \), and \( (\pi, t, m, \alpha) \models_{\text{TGS}} \phi_2 \).
- \( \phi = \exists (f : H \to c H', \phi') \) and there are \( m' : H' \to c \pi_t \) and \( \alpha' : H' \to \mathcal{V} \) s.t. \( m' \circ f = m, \alpha' \circ f_X = \alpha, (\pi, t, m', \alpha') \models_{\text{TGS}} \phi' \), and \( \text{sat}_{\exists}(\alpha'((\Theta_H \land \mathcal{M}^{-1}(\Theta_{\pi_t}))) \).

- \( \phi = \exists^v (f : H \to c H', \phi') \) and there are \( t' \in t + I, m' \in \text{PM}(m, \pi_{[t',v']}) \), and \( m'' : H' \to c \pi_{[t',v']} \) s.t.

  - there is \( \alpha' : X_{H'} \to \mathcal{V} \) s.t. \( m'' \circ f = m', \alpha' \circ f_X = \alpha, (\pi, t', m'', \alpha') \models_{\text{TGS}} \phi' \), \( \text{sat}_{\exists}(\alpha'((\Theta_{H'} \land \mathcal{M}^{-1}(\Theta_{\pi_{[t',v']}}))) \).

- for each \( t'' \in [0, t'] \) there is no \( m''' \in \text{PM}(m'', \text{rev}(\pi_{[t',v']})) \).

- \( \phi = \phi'_1 \cup I \phi'_2 \) and there is \( t' \in t + I \) s.t.

  - there is \( m' \in \text{PM}(m, \pi_{[t',v']}) \) s.t. \( (\pi, t', m', \alpha) \models_{\text{TGS}} \phi'_2 \) and

  - for each \( t'' \in [t, t'] \) there is \( m'' \in \text{PM}(m, \pi_{[t',v']} \) s.t. \( (\pi, t'', m'', \alpha) \models_{\text{TGS}} \phi'_1 \).

If \( \phi \in \Phi_\emptyset^{\text{MTGC}} \), \( i_{\pi_0} : \emptyset \to c \pi_0 \), \( \alpha : \emptyset \to \mathcal{V} \), and \( (\pi, 0, i_{\pi_0}, \alpha) \models_{\text{TGS}} \phi \), then \( \pi \models_{\text{TGS}} \phi \).
For the \textit{metric-exists-new} operator, we state that there is some timepoint $t'$ in the future that is compatible with the given interval $I$ where the propagated match $m'$ can be extended to a match $m''$ such that (first item) this extension is compatible with $f$ as required for the case of \textit{exists} and (second item) the extended match $m''$ cannot be propagated to any timepoint in the strict past. For the \textit{until} operator, we state (first item) that $\phi_2'$ is eventually satisfied for some timepoint $t'$ in the future that is compatible with the given interval $I$ and (second item) that $\phi_1'$ is permanently satisfied for all timepoints between the current observation timepoint $t$ and the timepoint $t'$ except for $t'$.

For our running example, see Fig. 3b for a TGS $\pi$, which does not satisfy the MTGC $\psi$ from Fig. 3a. In this TGS, we find a fresh match of a task on a system at timepoint $t = 6$ (after step 2). Moreover, we find immediately a (unique) handler for this task (based on the common id 123) such that there is no second handler for that task (with common id) until we find at timepoint $t = 21$ a result with the successful attribute value \textit{ok} obtained by the handler. However, this result is not obtained within the specified interval of at most 10 timeunits, which corresponds to the global time interval $6 + [0, 10]$ in this case.

4 On-the-fly Analysis for MTGL

We now present an on-the-fly analysis procedure for checking MTGCs against \textit{finite and infinite} TGSs as our main contribution. For this aim, we build upon [8] where an operation $\mathcal{R}$educe for translating an MTGC $\psi$ into a GC $\psi'$ and an operation $\mathcal{F}$old for translating a finite TGS $\pi$ into a so-called graph with history (GH) $G$ were presented. These two operations ensured that $\pi \models_{TGS} \psi$ iff $G \models \psi'$, which allows for an efficient check of MTGL satisfaction for \textit{finite} TGSs by checking GL satisfaction for finite GHs instead. Note that the problem of checking MTGL satisfaction (as for other metric temporal logics) becomes particularly difficult when instances of \textit{until} operators are nested.

When considering a TGS $\pi$ in an on-the-fly scenario where timed spans are added one-by-one, we cannot simply apply the procedure from [8] to all prefixes of $\pi$ because the MTGL satisfaction relation is inherently pessimistic not returning the desirable results. For example, the MTGC $\psi$ from Fig. 3a would be violated by the TGS $\pi$ from Fig. 3b not only at timepoint 21 when the violation of the deadline is detected (as discussed before) but also at timepoint 6 since the prefix of length 2 of $\pi$ does not contain any node of type Result. Note that we would indeed expect a violation at timepoint 6 already when the prefix of length 2 of $\pi$ would be the entire TGS to be considered. As a ground truth of violations that would ideally be returned by our procedure, we define \textit{true violations} later on in Definition 12 where also all subsequent behavior given by the timed spans in the TGS, which is not available to the on-the-fly procedure, is also taken into account.

For our on-the-fly analysis procedure (see Fig. 4 for an overview), we employ extensions/adaptations of the operations $\mathcal{R}$educe and $\mathcal{F}$old from [8]. As inputs, we consider an MTGC $\psi$ and a finite/infinite TGS $\pi$ that is incrementally considered. If the given TGS $\pi$ is finite, it may include a node of type Terminated in
its last graph to denote that it is not going to be continued (see Fig. 3b). Firstly, we employ a modification of the operation \( \text{Reduce} \) (see Definition 9) to obtain for \( \psi \) (via a parameter \( \text{mode} \in \{\text{Pes}, \text{Opt}\} \)), in addition to the pessimistic GC \( \psi_{\text{pes}} \) as in [8], a second optimistic GC \( \psi_{\text{opt}} \). Note that we apply the operation \( \text{Reduce} \) offline (once for each mode) before considering the timed spans of the TGS for increased efficiency. Secondly, we split the operation \( \text{Fold} \) into two operations \( \text{Fold}^{\text{1st}} \) and \( \text{Fold}^{\text{F}} \) to allow for an incremental rather than a batch folding of a TGS. That is, \( \text{Fold}^{\text{1st}} \) produces a first GH from the start graph of the TGS and \( \text{Fold}^{\text{F}} \) modifies a given GH \( G'_i \) into a GH \( G'_{i+1} \) for each timed span from the TGS as soon as that timed span is available. The sequence of GHS constructed in this way results in a \( \text{GH TGS} \pi' \) that corresponds to the prefix of the TGS available so far. Thirdly, we check for the conformance of each \( G'_i \) with \( \psi_{\text{pes}} \) and \( \psi_{\text{opt}} \) as soon as \( G'_i \) is available by separately applying the GL satisfaction relation to \( G'_i \) and \( \psi_{\text{pes}} \) as well as to \( G'_i \) and \( \psi_{\text{opt}} \). Each determined non-satisfaction is a violation for which we add the global time of its occurrence (which is given by the sum of all \( \delta_i \) so far) to the resulting sets of pessimistic and optimistic violations.

The operations \( \text{Fold}^{\text{1st}} \) and \( \text{Fold}^{\text{F}} \) ensure that each GH resulting from folding a TGS contains for each node/edge occurring in the TGS the timepoints of its creation and (if it was deleted) its deletion using additional cts and dts attributes. For our running example, the type graph \( TG \) from Fig. 1a contains these cts and dts attributes already and the GH eventually obtained for the entire finite terminated TGS \( \pi \) from Fig. 3b is given in Fig. 5c.

The operation \( \text{Reduce} \) returns GCS that encode the satisfaction checks for MTGL operators according to Definition 5 using ACs. These ACs make use of (a) the cts and dts attributes as added to the GH to control the matching of elements, (b) additional variables for quantifying over observation timepoints as in Definition 5, and (c) a variable \( x_{t_0} \) storing the current global time (i.e., the duration of the considered TGS prefix \( \pi' \)). As for the variable \( x : A x : \text{int} \) in Fig. 2, the additional quantified variables and the variable \( x_{t_0} \) for the current global time are required to be contained in the GHS. The operation \( \text{Reduce} \) returns

\[ \text{Fig. 4. Overview of the on-the-fly analysis procedure} \]
for this purpose, besides the GC, also a graph $G_{tv}$ containing these additional variables including the distinguished global time variable $x_{tv}$. For our running example, the graph $G_{tv}$ is given in Fig. 5b where all variables are associated with:

- Encoding nodes to decrease the number of matches via the num attributes that need to be considered when we check whether a GH subsuming $G_{tv}$ satisfies the obtained GC. Note that the construction of :Encoding nodes is omitted later in Definition 9 to ease presentation. While $G_{tv}$ is a subgraph of each constructed GH, we add the AC $x_{tv} = \text{dur}(\pi')$ to the current GH assigning the current global time to the variable $x_{tv}$ just before checking GL satisfaction whenever a new GH has been constructed during the on-the-fly analysis.

We now define the operations used in the on-the-fly analysis procedure. The predicate $Gtv$ identifies a unique global time variable $x_{tv}$ in a GH $G$ (subsuming the graph $G_{tv}$ obtained from folding) and the operation $Gta$ adds the AC that assigns the current global time to the $x_{tv}$ into the graph $G$.

**Definition 6 (Predicate $Gtv$ and Operation $Gta$).** If $G$ is a GH with a variable $x$ of sort real, then $G$ is a graph with global time variable $x$, written $Gtv(G, x)$. If, additionally, $t \in \mathbb{R}_0^+$ is a global timepoint and $G'$ is the graph obtained from $G$ by adding the AC $x = t$, then $G'$ is the time-assigned version of $G$ for the timepoint $t$, written $Gta(G, x, t) = G'$.

For incrementally folding a TGS starting in a graph $H$ into a GH, we use in the first step the following operation $\mathcal{F}old^{1st}$, which joins the graphs $G_{tv}$ and $H$ as well as adds cts attributes to all nodes and edges originating from $H$.

**Definition 7 (Operation $\mathcal{F}old^{1st}$).** If $G'$ is the componentwise disjoint union of the graphs $G_{tv}$ and $H$ where the attributes cts($\alpha$) = 0 and dts($\alpha$) = −1 are added to each node and edge $\alpha$ originating from $H$, then $\mathcal{F}old^{1st}(G_{tv}, H) = G'$.

The operation $\mathcal{F}old^{F}$ adapts a GH $G$ reached at global timepoint $t$ to a GH $G'$ by incorporating the changes described by a timed span $\rho = (\delta, \ell, r)$. Firstly, dts attributes of nodes and edges deleted by $\ell$ are updated to the new global timepoint $t + \delta$. Secondly, cts attributes with new global timepoint $t + \delta$ and dts attributes with default value $−1$ are added to all nodes and edges created by $r$.

**Definition 8 (Operation $\mathcal{F}old^{F}$).** If $G$ is a graph, $t \in \mathbb{R}_0^+$ is the current global timepoint, $\rho = (\delta, \ell, r : D \rel_1 H_1, r : D \rel_2 H_2)$ is a timed span, and $G'$ is a graph constructed from $G$ by (a) changing the attribute dts($\alpha$) to $t + \delta$ for each node or edge $\alpha \in H_1 - \ell(D)$, (b) renaming each node and edge $\alpha \in \ell(D)$ according to $\ell$, (c) renaming each node and edge $\alpha \in r(D)$ according to $r$, (d) adding each node and edge $\alpha \in H_2 - r(D)$, and (e) adding the attributes cts($\alpha$) = $t + \delta$ and dts($\alpha$) = $−1$ to each node and edge $\alpha \in H_2 - r(D)$, then $\mathcal{F}old^{F}(G, t, \rho) = G'$.

The operations $\mathcal{F}old^{1st}$ and $\mathcal{F}old^{F}$ preserve the predicate $Gtv$ implying that the variable $x_{tv}$ for the current global time can still be identified and used after the folding steps during the on-the-fly analysis. Moreover, the operation $\mathcal{F}old^{F}$ induces a timed span between the GHs $G$ and $G'$ leading to a GH TGS as discussed above. For our running example, the result of applying $\mathcal{F}old^{1st}$ and
then incrementally $F_{\text{old}}$ to the TGS $\pi$ from Fig. 3b leads to the graph $G'_3$ in Fig. 5c where the elements of the graph $G_{tv}$ from Fig. 5b are omitted for brevity. Since no elements are deleted in $\pi$, all elements in $G'_3$ have dts attributes of value $-1$.

We apply the operation $\text{Reduce}$, which is equipped with a $\text{mode}$ parameter, to separately construct the GCs $\psi_{\text{pes}}$ and $\psi_{\text{opt}}$ for a given MTGC $\psi$. The following definition of $\text{Reduce}$ extends the definition in [8] by additionally covering the optimistic reduction for $\text{mode} = \text{Opt}$ and explicitly integrating the formal reduction for the MTGL operator $\text{metric-exists-new}$ introduced in Definition 3.

**Definition 9 (Operation $\text{Reduce}$).** If $\psi \in \Phi_0^\text{MTGC}$ is an MTGC and $\text{mode} \in \{\text{Pes}, \text{Opt}\}$ is the chosen mode for reduction, then $\text{Reduce}(\text{mode}, \psi) = (G_{tv}, x_{tv}, \exists(i_{G_0}, \text{Reduce}_{\text{rec}}(\psi, x_0)))$ where the graph $G_0$ contains the variable $x_{tv}$ and an additional variable $x$ for the initial observation timepoint $0$. The employed recursive operation $\text{Reduce}_{\text{rec}}$ is homomorphic for true, conjunction, and negation and adds cts and dts attributes to all nodes and edges in the resulting GC. For the remaining operators, $\text{Reduce}_{\text{rec}}$ is defined as follows where the formal parameter $x_t$ corresponds to the variable $t$ in Definition 5 capturing the timepoint at which the MTGC provided to $\text{Reduce}_{\text{rec}}$ is checked for satisfaction.

- $\text{Reduce}_{\text{rec}}(\exists(f : H \leftarrow_r H', \phi'), x_t) = \exists(f', \text{Reduce}_{\text{rec}}(\phi', x_t))$ where $f'$ additionally requires alive($x_t, H'$)\(^4\) ensuring that $H'$ is matchable at timepoint $x_t$.
- $\text{Reduce}_{\text{rec}}(\exists f (f : H \leftarrow_r H', \phi'), x_t) = \exists(f', \text{Reduce}_{\text{rec}}(\phi', x_t))$ where $f'$ additionally requires a variable $x_t'$ satisfying $x_t' \in (x_t + I) \land \text{alive}(x_t, H') \land \text{earliest}(x_t', H')\(^5\) ensuring that $H'$ is matchable at timepoint $x_t'$ but not earlier.
- $\text{Reduce}_{\text{rec}}(\phi'_1 \cup I \phi'_{2}, x_t) = \psi_{\text{pes}} \lor \psi_{\text{ext}}$ with
  - $\psi_{\text{pes}} = \exists(f_0, \text{Reduce}_{\text{rec}}(\phi'_2, x_t') \land \forall(f_1, \text{Reduce}_{\text{rec}}(\phi'_1, x_t''))) \) where $f_0$ additionally requires a variable $x_t'$ satisfying $x_t' \in [0, x_{tv}] \cap (x_t + I)$ ensuring that $x_t'$ is a future timepoint where $\phi'_2$ is satisfied and $f_1$ additionally requires a variable $x_t''$ satisfying $x_t'' \in [x_t, x_t')$ for checking that $\phi'_1$ is satisfied until timepoint $x_t'$.
  - $\psi_{\text{ext}} = \bot$ for $\text{mode} = \text{Pes}$ disabling the optimistic check.
  - $\psi_{\text{ext}} = \neg \exists(f_0, \top) \land \exists(f_1, \top) \land \forall(f_2, \text{Reduce}_{\text{rec}}(\phi'_1, x_t''))$ for $\text{mode} = \text{Opt}$ where $f_0$ additionally requires a :Terminated node disabling the optimistic check when the TGS corresponding to the GH against which the resulting GC is checked is known to have ended, $f_1$ additionally requires a variable $x_t'$ satisfying $x_t' \in (x_{tv}, \infty) \cap (x_t + I)$ ensuring that there is still a timepoint in the strict future at which $\phi'_2$ could be satisfied, and $f_2$ additionally requires a variable $x_t''$ satisfying $x_t'' \in [x_t, x_{tv}]$ for checking that $\phi'_1$ was satisfied at least until the current global time $x_{tv}$.

The returned graph $G_{tv}$ contains all additional variables used in the reduction.\(^6\)

---

4 alive($x, H$) is an AC based on cts and dts attributes stating that all nodes and edges in $H$ are created and not yet deleted at timepoint $x$.

5 earliest($x, H$) is an AC stating that the highest cts attribute value in $H$ is $x$.

6 Note that the predicate $\text{Gtv}(G_{tv}, x_{tv})$ is satisfied by construction.
Fig. 5. The results of reducing the MTGC $\psi$ from Fig. 3a and the result of incrementally folding the entire TGS $\pi$ from Fig. 3b

For our running example, the result of the reduction of the MTGC $\psi$ from Fig. 3a is given in Fig. 5a for both reduction modes. The first line requires that the initial observation timepoint $x_0$ is 0 and that the variable $x_{tv}$ is matched into the GH thereby binding it to the current global time. The second line (a) attempts to match $T$, $e_1$, and $S$ for some observation timepoint $x_2$ in the future of $x_0$ such that the matched elements are alive at timepoint $x_2$ and not all of them are alive at any earlier timepoint (encoding the metric-forall-new operator) and (b) then checks whether a suitable handler with the same id
exists at timepoint $x_2$. The third line (for mode = $\mathcal{P}_{es}$) tries to determine some timepoint $x_3$ in the future of $x_2$ restricted by the given interval $[0, 10]$ of the $\mathbf{untill}$ operator where (a) a result with value $ok$ can be matched and where (b) at all timepoints $x_4 \in [x_2, x_3)$ no second handler with the same id can be matched. The fourth line (for mode = $\mathcal{O}_{pt}$) checks whether the TGS has not terminated already (which disables the optimistic checking), whether there is still some time left for the right-hand side condition of the $\mathbf{untill}$ operator to become satisfied, and whether the left-hand side condition of the $\mathbf{untill}$ operator was satisfied at all timepoints $x_6$ since $x_2$ and up-to the current global time.

The theorem on soundness of $\mathcal{R}_{educe}$ from [8] is now adapted for the included operator metric-exists-new and extended to cover also the optimistic mode.

**Theorem 1 (Soundness of $\mathcal{R}_{educe}$).** If $\psi \in \Phi_{\mathcal{MTGC}}^{MTGC}$ is an MTGC, $\pi \in \Pi_{\mathcal{H}}^{\mathcal{fin}}$ is a finite TGS starting in $H$ and ending in $H'$, $(G_{tv}, x_{tv}, \psi_{pes}) = \mathcal{R}_{educe}(\mathcal{P}_{es}, \psi)$ is the result of pessimistic reduction, $(G_{tv}, x_{tv}, \psi_{opt}) = \mathcal{R}_{educe}(\mathcal{O}_{pt}, \psi)$ is the result of optimistic reduction, and $G$ is obtained from $\pi$ and $G_{tv}$ using $\mathcal{F}_{old}^{1st}$ and the iterated application of $\mathcal{F}_{old}^{\mathcal{F}}$, then (a) $\pi \models \mathcal{T}_{GS} \psi$ iff $G \models \psi_{pes}$ and (b) $\exists \pi' \in \Pi_{H'}. \pi; \pi' \models \mathcal{T}_{GS} \psi$ only if $G \models \psi_{opt}$.

**Proof Idea.** Straightforward inductions on $\psi$ and $\pi$ along the lines of [8].

The reverse direction of the item (b) in the theorem above does not hold for the MTGC $\phi = \top \cup U_{[2, 2]} \perp$ even though non-satisfaction in the future is guaranteed. Moreover, no other reduction can achieve the equivalence for item (b) since satisfiability is undecidable for GCs and hence also for MTGCs implying that guaranteed non-satisfiability as for $\phi$ cannot always be determined. However, we may simplify generated GCs using the sound and refutationally complete procedure from [24] for checking GL satisfiability. For example, the GC obtained by reducing the MTGC $\phi$ from above can be simplified to $\perp$ using this approach.

The following two operations Analyze$^{1st}$ and Analyze$^{\mathcal{F}}$ rely on the operations $\mathcal{F}_{old}^{1st}$ and $\mathcal{F}_{old}^{\mathcal{F}}$, respectively, and are used in our on-the-fly analysis procedure, which is given in pseudo code in Procedure 1. To simplify presentation, Procedure 1 describes the on-the-fly analysis using only one of both modes requiring that two instances of Procedure 1 are executed concurrently to cover the optimistic and the pessimistic mode. Procedure 1 maintains during its execution a triple $(G, t, V)$ consisting of the current GH $G$, the current global time $t$, and the set of computed violations $V$. The operation Analyze$^{1st}$ constructs the first triple $(G, t, V)$ by applying $\mathcal{F}_{old}^{1st}$ and by inserting the initial timepoint 0 into the set of violations when the resulting GH $G$ does not satisfy the GC $\psi'$.

**Definition 10 (Operation Analyze$^{1st}$).** If $(G_{tv}, x_{tv}, \psi')$ was returned by an application of the operation $\mathcal{R}_{educe}$, $\pi \in \Pi_{\mathcal{H}}^{\mathcal{fin}}$ is a finite TGS starting in graph $H$, $G$ is the GH obtained using $\mathcal{F}_{old}^{1st}(G_{tv}, H)$, and if $Gta(G, x_{tv}, 0) \not\models \psi'$ then $V = \{0\}$ else $V = \emptyset$, then Analyze$^{1st}$($(G_{tv}, x_{tv}, \psi'), \pi) = (0, G, V)$.

The operation Analyze$^{\mathcal{F}}$ modifies a triple $(G, t, V)$ to a triple $(G', t', V')$ according to a timed span $\rho$ by modifying the GH $G$ to a GH $G'$, by increasing the
global time \( t \) to \( t' \), and by inserting the global time \( t' \) into the set of violations \( V \) when \( G' \) does not satisfy the GC \( \psi' \).

**Definition 11 (Operation Analyze\(^F\)).** If \( (G_{tv}, x_{tv}, \psi') \) was returned by an application of the operation Reduce, \( \rho = (\delta, \ell, r) \) is a timed span, \( t \in R_0^\uparrow \) is the global time before \( \rho \), \( G \) is a GH with time-storing variable \( x_{tv} \) satisfying \( \text{Gtv}(G, x_{tv}) \), \( V \subseteq R_0^\uparrow \) is a set of violations computed until the timepoint \( t \), \( t' = t + \delta \) is the global time after \( \rho \), \( G' = \text{Fold}^F(G, t, \rho) \) is the modification of the GH \( G \) according to \( \rho \), and if \( \text{Gta}(G', x_{tv}, t') \not\in \psi' \) then \( V' = V \cup \{ t' \} \) else \( V' = V \) implements the conditional addition of a violation \( t' \) to the set of violations \( V \), then the triple \( (t, G, V) \) is modified to the triple \( (t', G', V') \) according to the timed span \( \rho \), written \( (t, G, V) \text{ Analyze}^F_{x_{tv}, \psi', \rho} (t', G', V') \).

Moreover, we extend the operation Analyze\(^F\) to finite TGSs using its iterated application starting with the triple obtained using Analyze\(^1\text{st}\)(\( (G_{tv}, x_{tv}, \psi') \), \( \pi \)).

For our running example, Procedure 1 returns the sets \{6, 21\} and \{21\} of violations for the pessimistic and the optimistic mode, respectively. Note that the given MTGC is violated at each timepoint in \((16, \infty)\), i.e., the violation is detected with a delay of about 5 timeunits. However, we believe that assuming a sufficiently high timed span rate (checking for violations permanently in a hot loop) mitigates this problem of delayed detection. Also, we assume a user-provided policy for deciding whether timepoints are recorded as violations when employed solvers such as Z3 are unable to decide the satisfaction problem in time.

Subsequently, we formally evaluate Procedure 1 by comparing the violations printed by it with a ground truth of violations that would be returned ideally. This ground truth is defined by the operation trueViolations delivering the set of true violations of an MTGC \( \psi \) in a TGS \( \pi \) for a maximum checking index \( n \). This set of true violations contains the durations of all prefixes \( \pi' \) of the TGS \( \pi \) where the MTGC \( \psi \) is not satisfied subsequently by any continuation \( \tilde{\pi} \) that is
a prefix of \( \pi \). Observe that a precise detection of violations thereby requires the knowledge of future steps, which is unavailable in the context of our proposed on-the-fly analysis approach.

**Definition 12 (Operation trueViolations).** If \( \psi \in \Phi^\text{MTGC} \) is an MTGC, \( \pi \in \Pi_H \) is a TGS, \( n \in \mathbb{N} \) is a maximum checking index, and \( \text{length}(\pi) \geq n \), then

\[
\text{trueViolations}(\psi,\pi,n) = \{ t \mid \exists \pi'. (\pi' \sqsubseteq \pi) \land (\text{length}(\pi') \leq n) \land (t = \text{dur}(\pi')) \land (\forall \tilde{\pi}. (\pi' \sqsubseteq \tilde{\pi} \sqsubseteq \pi) \rightarrow (\tilde{\pi} \not\models \text{TGS} \; \psi)\}.
\]

To enable a comparison of Procedure 1 with the set of true violations, we now introduce the operation check, which gathers the optimistic and pessimistic violations printed by Procedure 1 up to a given index \( n \).

**Definition 13 (Operation check).** If \((G_{tv}, x_{tv}, \psi') = \text{Reduce}(\text{mode}, \psi)\) is the reduction obtained using the operation \( \text{Reduce} \), \( \pi \in \Pi_H \) is a TGS of length at least \( n \in \mathbb{N} \) starting in graph \( H \), \( \bar{\pi} \) is the prefix of length \( n \) of \( \pi \), \((t, G, V) = \text{Analyze}^{1st}(G_{tv}, x_{tv}, \psi', \pi)\) is the first triple for the on-the-fly analysis, and \((t, G, V) \, \text{Analyze}^{2nd}_{x_{tv}, \psi}\, \bar{\pi} (t', G', V')\) computes the violations \( V' \) using the on-the-fly analysis on the prefix \( \bar{\pi} \), then \( \text{check}(\text{mode}, \psi, \pi, n) = V' \).

Based on the presented definitions, we introduce our main result stating that the optimistic and the pessimistic on-the-fly analysis carried out using Procedure 1 (and formalized using the operation check) determines under- and over-approximations of the set of true violations, respectively. That is, optimistic violations must be true violations and pessimistic violations may be true violations.

**Theorem 2 (Approximate Detection of True Violations using Optimistic and Pessimistic On-the-fly Analysis).** If \( \psi \in \Phi^\text{MTGC} \) is an MTGC, \( \pi \in \Pi_H \) is a TGS, \( n \in \mathbb{N} \) is a maximum checking index, and \( \text{length}(\pi) \geq n \), then

- \( \text{check}(\text{Opt}, \psi, \pi, n) \subseteq \text{trueViolations}(\psi, \pi, n) \) and
- \( \text{trueViolations}(\psi, \pi, n) \subseteq \text{check}(\text{Pess}, \psi, \pi, n) \).

**Proof Idea.** The first item holds using Theorem 1 since every violation of the GC \( \psi' \) obtained using reduction for \( \text{mode} = \text{Opt} \) ensures that the future evolution used in the operation \( \text{trueViolations} \) is also guaranteed to violate the MTGC \( \psi \) as the violation cannot be undone in any continuation. The second item holds using Theorem 1 because (via contraposition) when the GC \( \psi' \) obtained using reduction for \( \text{mode} = \text{Pess} \) is not violated, this means that the considered TGS prefix \( \pi' \) satisfies the MTGC \( \psi \) and therefore at least the empty continuation \( \pi' = \bar{\pi} \) satisfies \( \psi \) as well implying that the current global timepoint is no true violation. \( \square \)

For our running example from Fig. 3 where the TGS \( \pi \) has length 3, we observe that \( \text{check}(\text{Opt}, \psi, \pi, 3) = \{21\} = \text{trueViolations}(\psi, \pi, 3) \subseteq \{6, 21\} = \text{check}(\text{Pess}, \psi, \pi, 3) \). This means that the optimistic on-the-fly analysis detects only one true violation at global timepoint 21 but the pessimistic on-the-fly analysis returns additionally a second violation at global timepoint 6 indicating that the MTGC \( \psi \) may not be satisfied by all continuations, which indeed is the
case for the TGS $\pi$. When changing $\delta_2$ to 1 in Fig. 3b, the pessimistic on-the-fly analysis still generates the violation at global timepoint 6 but 6 is no true violation since the result $R$ is then generated in time.

Based on Theorem 2, we obtain, as mentioned in Sect. 1, a three-valued interpretation of MTGL when using Procedure 1 (formalized using the operation check) to generate optimistic and pessimistic violations. In this interpretation, the absence of a pessimistic violation $t$ guarantees satisfaction, each optimistic violation $t$ guarantees nonsatisfaction, and a pessimistic violation $t$ that is no optimistic violation is an indifferent judgement on satisfaction.

**Definition 14 (Three-valued Interpretation of MTGL).** If $\psi \in \Phi_{\text{MTGC}}$ is an MTGC, $\pi \in \Pi_H$ is a TGS of length at least $n \in \mathbb{N}$, and $\bar{\pi}$ is the prefix of length $n$ of $\pi$, then $\llbracket \pi, \psi \rrbracket_n = \text{true}$ if $\text{dur}(\bar{\pi}) \notin \text{check}(\text{Pes}, \psi, \pi, n)$, $\llbracket \pi, \psi \rrbracket_n = \text{false}$ if $\text{dur}(\bar{\pi}) \in \text{check}(\text{Opt}, \psi, \pi, n)$, and $\llbracket \pi, \psi \rrbracket_n = \text{indifferent}$ otherwise.

From a practical point of view, Procedure 1 solves the satisfaction problem for three kinds of TGSs. Firstly, non-terminating systems can be analyzed throughout their entire runtime in an on-the-fly manner. Secondly, finite slices of TGSs generated by long-running systems can be analyzed in an offline manner producing pessimistic but no optimistic violations for cases where the ongoing evolution of the system may be admissible. Lastly, terminating systems where a Terminated node is added by the last timed span can be analyzed where pessimistic and optimistic violations coincide after the last step.

However, for formal specifications given by complex MTGCs, we intend to equip violations given by timepoints with human-readable explanations. For this purpose, we may use partial MTGC satisfaction trees following GC satisfaction trees from [25]. We expect that this would also permit an analysis of the causal dependencies among violations and their origins. For our running example, the violation at timepoint 21 has the pessimistic violation at timepoint 6 as an origin because the problematic task $T$ connected to system $S$ was freshly matched at timepoint 6 leading to the violated deadline at timepoint 21.

We implemented Procedure 1 in the tool AUTOGRAPH [24], which supports GL and MTGL and relies on the constraint solver Z3 [16] for checking the satisfiability of ACs. For a high level of confidence, we applied extensive testing of the implemented functionality for diverse and deeply nested MTGCs also covering our running example. Overall, the implementation is promising and demonstrates the feasibility of our approach.

## 5 Related Work

Verification approaches for graph transformation systems are incomplete due to their expressiveness. For example, logics such as CTL and PTCTL can be applied to entire state spaces [6,15] but have limited support for sequence properties relying on atomic propositions. Similarly, invariant verification [4,23] as an example of static analysis considers graph sequences but only state invariants.
On-the-fly analysis of dynamic systems is used (before deployment) in testing and (after deployment) in specification-based monitoring using specifications given by temporal logics, automata with quantification, and rule-based systems [1]. These approaches are difficult to compare due to highly domain-specific requirements regarding expressiveness, efficiency, and usability. Logic-based approaches e.g. [11,14] often lack support for key features of MTGL such as data elements, bindings, or metric bounds in temporal operators. A notable exception is the Metric First-Order Temporal Logic (MFOTL) [2], supported by the tool MonPoly, that represents a system state as a set of relations, supports the binding of elements and uses a point based rather than an interval based semantics as MTGL. Note that the encoding of MTGCs in MFOTL is highly technical and error-prone (similarly, the logic of nested graph conditions [9] is advantageous compared to FOL on graphs in graph centered scenarios). Finally, MonPoly imposes syntactic limitations on MFOTL conditions to ensure that provided conditions are satisfied/violated by a finite future.

Logics such as MTGL and MFOTL can be directly applied in the context of runtime monitoring [1,13]. A roadmap towards such an application using an extended subset of MTGL is presented in an informal way in [21,22].

6 Conclusion and Future Work

We introduced an on-the-fly analysis procedure for the satisfaction of MTGCs by infinite TGSs (generated by non-terminating systems) and finite TGSs (generated by terminating systems or representing prefixes of infinite TGSs). The analysis procedure results in a three-valued interpretation of MTGL where unavoidable non-satisfaction is detected via optimistic violations and where potential non-satisfaction in the future is detected via pessimistic violations as soon as possible. The two sets of violations approximate the ground truth given by the set of true violations, which can only be determined by offline analysis for finite terminated TGSs. The on-the-fly analysis procedure including both sets of violations is supported by our extension of the tool AutoGraph.

In the future, we will (a) integrate the since operator into MTGL and the proposed analysis procedure, (b) improve applicability of our approach using more detailed violations, (c) improve the optimistic reduction by simplifying the reduced MTGC using the constraint solver approach from [24], (d) employ incremental GC checking to improve the on-the-fly checking performance, and (e) compare our approach to other tools such as MonPoly w.r.t. efficiency.

References

1. Bartocci, E., Deshmukh, J., Donzé, A., Fainekos, G., Maler, O., Ničković, D., Sankaranarayanan, S.: Specification-based monitoring of cyber-physical systems: a survey on theory, tools and applications. In: Bartocci, E., Falcone, Y. (eds.) Lectures on Runtime Verification. LNCS, vol. 10457, pp. 135–175. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-75632-5_5
2. Basin, D.A., Klaedtke, F., Müller, S., Zălinescu, E.: Monitoring metric first-order temporal properties. J. ACM 62(2), 15:1–15:45 (2015). https://doi.org/10.1145/2699444

3. Becker, B., Giese, H.: On safe service-oriented real-time coordination for autonomous vehicles. In: ISORC 2008, pp. 203–210. IEEE Computer Society (2008). https://doi.org/10.1109/ISORC.2008.13

4. Dyck, J., Giese, H.: k-inductive invariant checking for graph transformation systems. In: de Lara, J., Plump, D. (eds.) ICGT 2017. LNCS, vol. 10373, pp. 142–158. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-61470-0_9

5. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. Springer, Heidelberg (2006). https://doi.org/10.1007/3-540-31188-2

6. Ghamarian, A.H., de Mol, M., Rensink, A., Zambon, E., Zimakova, M.: Modelling and analysis using GROOVE. STTT 14(1), 15–40 (2012). https://doi.org/10.1007/s10009-011-0186-x

7. Giese, H.: Modeling and verification of cooperative self-adaptive mechatronic systems. In: Kordon, F., Sztipanovits, J. (eds.) Monterey Workshop 2005. LNCS, vol. 4322, pp. 258–280. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71156-8_14

8. Giese, H., Maximova, M., Sakizloglou, L., Schneider, S.: Metric temporal graph logic over typed attributed graphs. In: Hähnle and van der Aalst [10], pp. 282–298 (2019). https://doi.org/10.1007/978-3-030-16722-6_16

9. Habel, A., Pennemann, K.: Correctness of high-level transformation systems relative to nested conditions. Math. Struct. Comput. Sci. 19(2), 245–296 (2009). https://doi.org/10.1017/S0960129508007202

10. Hähnle, R., van der Aalst, W. (eds.): FASE 2019. LNCS, vol. 11424. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-16722-6

11. Havelund, K., Peled, D.: Runtime verification: from propositional to first-order temporal logic. In: Colombo, C., Leucker, M. (eds.) RV 2018. LNCS, vol. 11237, pp. 90–112. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03769-7_7

12. Koymans, R.: Specifying real-time properties with metric temporal logic. Real-Time Syst. 2(4), 255–299 (1990). https://doi.org/10.1007/BF01995674

13. Leucker, M., Schallhart, C.: A brief account of runtime verification. J. Log. Algebr. Program. 78(5), 293–303 (2009). https://doi.org/10.1016/j.jlap.2008.08.004

14. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Lakhnech, Y., Yovine, S. (eds.) FORMATS/FTRTFT 2004. LNCS, vol. 3253, pp. 152–166. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30206-3_12

15. Maximova, M., Giese, H., Krause, C.: Probabilistic timed graph transformation systems. J. Log. Algebr. Meth. Program. 101, 110–131 (2018). https://doi.org/10.1016/j.jlamp.2018.09.003

16. Microsoft Corporation: Z3. https://github.com/Z3Prover/z3

17. Neumann, S.: Modellierung und Verifikation zeitbehafteter Graphtransformationssysteme mittels Groove. Master’s thesis, University of Paderborn (2007)

18. Orejas, F.: Symbolic graphs for attributed graph constraints. J. Symb. Comput. 46(3), 294–315 (2011). https://doi.org/10.1016/j.jsc.2010.09.009

19. Orejas, F., Lambers, L.: Symbolic attributed graphs for attributed graph transformation. ECEASST 30 (2010). http://journal.ub.tu-berlin.de/index.php/eceasst/article/view/405

20. Orejas, F., Lambers, L.: Lazy graph transformation. Fundam. Inf. 118(1–2), 65–96 (2012). https://doi.org/10.3233/FI-2012-706
21. Sakizloglou, L., Ghahremani, S., Brand, T., Barkowsky, M., Giese, H.: Towards highly scalable runtime models with history. In: SEAMS 2020. IEEE Computer Society (2020). (to appear)
22. Sakizloglou, L., Ghahremani, S., Brand, T., Barkowsky, M., Giese, H.: Towards highly scalable runtime models with history. Technical Report (2020). arxiv: 2004.03727
23. Schneider, S., Dyck, J., Giese, H.: Formal verification of invariants for attributed graph transformation systems based on nested attributed graph conditions. In: Gadducci, F., Kehrer, T. (eds.) ICGT 2020, LNCS. Springer, Heidelberg (2020). (to appear)
24. Schneider, S., Lambers, L., Orejas, F.: Automated reasoning for attributed graph properties. STTT 20(6), 705–737 (2018). https://doi.org/10.1007/s10009-018-0496-3
25. Schneider, S., Lambers, L., Orejas, F.: A logic-based incremental approach to graph repair. In: Hähnle and van der Aalst [10], pp. 151–167 (2019). https://doi.org/10.1007/978-3-030-16722-6_9