Metric Temporal Graph Logic over Typed Attributed Graphs

Holger Giese, Maria Maximova, Lucas Sakizloglou, Sven Schneider

Technische Berichte Nr. 123

des Hasso-Plattner-Instituts für Digital Engineering an der Universität Potsdam

Technische Berichte des Hasso-Plattner-Instituts für
Digital Engineering an der Universität Potsdam

Technische Berichte des Hasso-Plattner-Instituts für
Digital Engineering an der Universität Potsdam | 123

Holger Giese | Maria Maximova | Lucas Sakizloglou | Sven Schneider

Metric Temporal Graph Logic over Typed Attributed Graphs

Universitätsverlag Potsdam

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.dnb.de/ abrufbar.

Universitätsverlag Potsdam 2018
http://verlag.ub.uni-potsdam.de/

Am Neuen Palais 10, 14469 Potsdam
Tel.: +49 (0)331 977 2533 / Fax: 2292
E-Mail: verlag@uni-potsdam.de

Die Schriftenreihe **Technische Berichte des Hasso-Plattner-Instituts für Digital Engineering an der Universität Potsdam** wird herausgegeben von den Professoren des Hasso-Plattner-Instituts für Digital Engineering an der Universität Potsdam.

ISSN (print) 1613-5652
ISSN (online) 2191-1665

Druck: docupoint GmbH Magdeburg

ISBN 978-3-86956-433-3

Zugleich online veröffentlicht auf dem Publikationsserver der Universität Potsdam:
URN urn:nbn:de:kobv:517-opus4-411351
http://nbn-resolving.de/urn:nbn:de:kobv:517-opus4-411351

Various kinds of typed attributed graphs are used to represent states of systems from a broad range of domains. For dynamic systems, established formalisms such as graph transformations provide a formal model for defining state sequences. We consider the extended case where time elapses between states and introduce a logic to reason about these sequences. With this logic we express properties on the structure and attributes of states as well as on the temporal occurrence of states that are related by their inner structure, which no formal logic over graphs accomplishes concisely so far. Firstly, we introduce *graphs with history* by equipping every graph element with the timestamp of its creation and, if applicable, its deletion. Secondly, we define a logic on graphs by integrating the temporal operator *until* into the well-established logic of nested graph conditions. Thirdly, we prove that our logic is equally expressive to nested graph conditions by providing a suitable reduction. Finally, the implementation of this reduction allows for the tool-based analysis of metric temporal properties for state sequences.

Contents

1 Introduction

Various kinds of typed attributed graphs are used to represent states of systems from a broad range of domains. Also, dynamic systems can be described using graph transformation formalisms in which the possible behavior is defined by a set of rules and their application.

For certain domains, verification of this behavior with respect to a specification is of paramount importance and can ideally be done prior to execution using techniques such as static analysis, theorem proving, or model checking. However, the usage of highly expressive rule-patterns typically limits these techniques. Also, such techniques are ineffective for systems where the rules are (partially) unknown or change over the lifespan of the system. *Runtime monitoring* is an alternative analysis approach that verifies at runtime a single state sequence against a specification. It is independent from the expressiveness, the number, and the availability of the systems' rules and can be applied using highly expressive specification formalisms.

In our running example, we discuss an instance of a dynamic system where state changes are governed by unknown rules. States are represented by typed attributed graphs and state changes over time by a sequence comprising such graphs. In particular, we consider a hospital information system where smart devices are used to perform a treatment at a given moment or periodically measure and report the value of a vital sign. The data produced by these devices and the actions of the medical staff are to be logged and monitored at runtime for violations of medical guidelines. For this purpose we require a logic for (a) expressing state properties, which check the data and structure of a single system state, and for (b) expressing sequence properties, which relate state properties at different timepoints to check the timely reachability of a certain state.

As a *first contribution* we define with *Metric Temporal Graph Logic* (MTGL) such a logic for the specification of admissible graph sequences, which allows the usage of MTGL for runtime monitoring. In MTGL we express state properties on the attribute data and graph structure of a single system state, using the well-established logic of nested graph conditions [5] (called graph conditions subsequently). Graph conditions are as expressive as first-order logic on graphs [5, 11] and are commonly used for expressing state properties such as in application conditions of rules in graph transformation systems. Moreover, we build upon *Metric Temporal Logic* (MTL) [7] to express sequence properties such as invariants and the non-violation of deadlines. As shown in chapter 3, the direct combination of MTL and graph conditions is insufficient. As a *second contribution* we show that conditions of MTGL can be translated into graph conditions using attribute constraints to encode metric temporal requirements.

There are several related approaches for specification and verification of sequence properties involving time, either using graphs or relations. A visual notation for the specification of such properties over typed attributed graphs has been introduced in [6] but their approach lacks a definition of logical operators and an implementation. The *Metric First-Order Temporal Logic* (MFOTL) [2] expresses sequence properties by relying on the description of a system state by a set of relations, which are then adapted during the execution. This logic is supported by the state-of-the-art tool *MonPoly*. MFOTL is sufficiently expressive to state relevant properties in a broad range of applications. Nonetheless, we believe that, for graph-based systems, using MTGL to determine a metric temporal specification is less error-prone and potentially more efficient than using MFOTL. Other runtime monitoring tools match the expressiveness of MFOTL. However, either they are not based on a logic, or they do not support *real-time*, i.e., they do not consider the occurrence time of changes, which is required for the dynamic systems discussed in this paper (cf. [1] for a feature overview of other runtime monitoring tools).

The paper is structured as follows. Chapter 2 iterates on required technical preliminaries. Chapter 3 introduces our running example and formulates our problem statement. Chapter 4 shows how our approach consolidates a sequence of typed attributed graphs into a compact representation called *graph with history*. Chapter 5 defines MTGL and introduces the reduction of its conditions to graph conditions. Chapter 6 discusses the tool support for our approach. We conclude in chapter 7 with a summary and remarks on future work.

2 Preliminaries

In this section we recall the basic concepts of typed attributed graphs and nested graph conditions as well as shortly discuss MTL allowing for the formulation of metric temporal properties on timed state sequences.

2.1 Typed Attributed Graphs and Nested Graph Conditions

We rely in the subsequent sections and in our running example on the notion of finite *symbolic graphs* [9], which encode typed attributed graphs, to represent systems' states and to express properties on these states. Symbolic graphs are an adaptation of E-Graphs [4] where a graph does not contain data nodes (i.e., elements that represent actual values) but instead node and edge attributes are connected to variables, which thereby replace the data nodes. Then, such a (sorted) variable x that is part of a symbolic graph can appear in additional attribute constraints that are also contained in the graph such as $x = 5$, $x \leq 5$, and $x =$ "aabb".

As customary, we are only using graphs that are typed over a type graph TG. This typing is given by means of a typing morphism $type : G \to TG$. Thereby, a type graph provides a basic restriction of all typed attributed graphs to an admitted subset of graphs.

On the one hand, instance graphs that are used in the nested graph conditions introduced below (and called in the following graph conditions or GC) are allowed to make use of non-trivial attribute constraints such as $x \leq 5$ to enable their matching to various instance graphs G where the variable x_G, to which x is matched, has some unique value c satisfying the instantiated constraint $c \leq 5$. On the other hand, instance graphs that represent a concrete systems' state are assumed to have a set of attribute constraints that limit the number of possible valuations to one for the variables connected to these attributes, i.e., if G is such an instance graph, α is a node or edge of G, att is an attribute of α, and x is the variable of att, then we require constraints implying that $x = c$ for some concrete value c.

Using monomorphisms (called monos subsequently) between these symbolic graphs, we state the existence and nonexistence of graph patterns in a given symbolic graph, which is called host graph. These patterns are stated using graph conditions [5], which are expressively equivalent to first-order logic on graphs [3] as shown in [5, 11].

Definition 1 (Graph Conditions Φ^{GC})**.** *For each graph G we inductively define the set of all* graph conditions Φ^{GC}_G *as follows:*

- *If $S = \{\phi_1, \ldots, \phi_n\}$ is a finite subset of Φ^{GC}_G, then $\wedge S \in \Phi^{GC}_G$.*

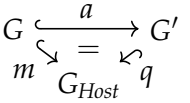
- *If* $\phi \in \Phi_G^{GC}$, *then* $\neg\phi \in \Phi_G^{GC}$.
- *If* $a : G \hookrightarrow H$ *and* $\phi \in \Phi_H^{GC}$, *then* $\exists(a, \phi) \in \Phi_G^{GC}$.

Note that for stating graph conditions we also use infix notation as well as derive the following further operators: true is $\wedge\emptyset$, *false is* $\neg true$, $\vee S$ *is* $\neg \wedge \{\neg\phi \mid \phi \in S\}$, $\phi_1 \Rightarrow \phi_2$ *is* $\neg\phi_1 \vee \phi_2$, *and* $\forall(a, \phi)$ *is* $\neg\exists(a, \neg\phi)$.

Intuitively, a graph condition is satisfied if the positive but not the negative patterns given by the graph condition can be found in the given host graph.

Definition 2 (Satisfaction of Graph Conditions)**.** *A mono* $m : G \hookrightarrow G_{Host}$ satisfies *a graph condition* $\psi \in \Phi_G^{GC}$, *written* $m \models \psi$, *if one of three cases applies:*

- $\psi = \wedge S$ *and* $m \models \phi$ *for each* $\phi \in S$.
- $\psi = \neg\phi$ *and not* $m \models \phi$.
- $\psi = \exists(a : G \hookrightarrow G', \phi)$ *and there exists* $q : G' \hookrightarrow G_{Host}$ *such that* $q \circ a = m$ *and* $q \models \phi$ *(as depicted on the right).*



A host graph G_{Host} *satisfies a graph condition* ψ *over the empty graph if the unique initial morphism* $i_{G_{Host}} : \emptyset \hookrightarrow G_{Host}$ *satisfies* ψ.

2.2 Metric Temporal Logic

The standard temporal logic for expressing properties that need to be satisfied within a certain time interval, is Metric Temporal Logic (MTL) [7, 10]. In the following we recall shortly the syntax and semantics of MTL providing the foundation for the definition of Metric Temporal Graph Conditions in chapter 5.

The set of all MTL formulas is defined using a set AP of atomic propositions, Boolean operators, and the metric *until* operator U as follows:

$$\phi ::= ap \mid \neg\phi \mid \phi \wedge \phi \mid \phi \, \mathrm{U}_I \, \phi$$

where I is an interval over $\mathbf{R}_0^+$.

MTL formulas are satisfied by *timed state sequences* π, which are of the form $(s_1, \delta_1), (s_2, \delta_2), \ldots$ with states $s_i \in S$ and durations $\delta_i \in \mathbf{R}^+$, at timepoints t. The state s_i (for $i \in \mathbf{N}^+$) in a timed state sequence at timepoint t is denoted by $\pi(t)$ and is obtained by requiring $\sum_{j=1}^{i-1} \delta_j \leq t < \sum_{j=1}^{i} \delta_j$. Moreover, a labeling function $L : S \to 2^{AP}$ determines the atomic propositions that are satisfied for any state in S. Then, an MTL formula ϕ is satisfied for a timepoint $t \in \mathbf{R}_0^+$, written $(\pi, t) \models \phi$, if one of the following cases applies:

- $\phi = ap$, $ap \in AP$, and $ap \in L(\pi(t))$.
- $\phi = \neg\phi'$ and not $(\pi, t) \models \phi'$.
- $\phi = \phi_1 \wedge \phi_2$, $(\pi, t) \models \phi_1$, and $(\pi, t) \models \phi_2$.

- $\phi = \phi_1 \, \mathrm{U}_I \, \phi_2$ and there is some $t' \in I$ such that $(\pi, t + t') \models \phi_2$ and for every $t'' \in [0, t')$ it holds that $(\pi, t + t'') \models \phi_1$.

Further operators such as *true*, *eventually*, and *globally* can be derived as in [10].

Intuitively, the formula $\phi_1 \, \mathrm{U}_I \, \phi_2$ is satisfied by a timed state sequence π at an observation timepoint t if ϕ_2 is eventually satisfied at some timepoint from the interval I relative to t and when, additionally, the condition ϕ_1 is satisfied at all points in time between t and strictly before ϕ_1 is satisfied.

3 Running Example and Problem Statement

Throughout the remainder of this paper, we refer to the following property **P** from the medical domain.

P: If a patient P has undergone a dental procedure Pr, then a pump Pu must be attached to that patient. Also, this pump Pu must, in the next two hours, administer a dosage of antibiotics (action A). Until this action is taken, no second pump Pu' should be attached to the patient.

This property is based on a medical guideline [13] and has been adjusted to reflect a setting where smart devices such as a pump are used to support the treatment. We now analyze property **P** to demonstrate the limitations of existing techniques in the formulation of such sequence properties.

As a first step we define the type graph TG from Fig. 3.1 and formalize two state properties occurring in **P** as ϕ_1 and ϕ_2 also shown in Fig. 3.1 based on this type graph. The graph condition ϕ_1 expresses that "a dental procedure has been performed on a patient" and the graph condition ϕ_2 expresses that "a procedure has been performed on a patient to which a pump is attached with an action for administering antibiotics".

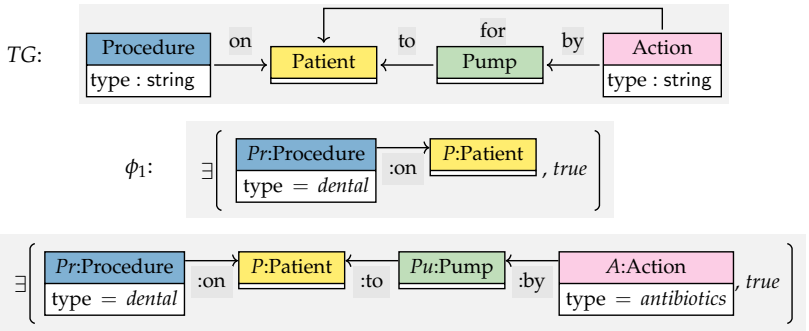


Figure 3.1: The type graph TG (top) for our running example as well as the graph conditions ϕ_1 (middle) and ϕ_2 (bottom) that represent state properties occurring in the sequence property **P**

As a second step we instantiate MTL by providing an interpretation for states, atomic propositions, and the labeling function L as follows. The states are given by all typed attributed graphs, the atomic propositions are given by all graph conditions from $\Phi^{GC}_{\emptyset}$, and the labeling function L uses the satisfaction relation of graph conditions: $L(G) = \{\phi \in \Phi^{GC}_{\emptyset} \mid G \models \phi\}$. Using this interpretation we can

now use MTL over graph conditions to state sequence properties using metric temporal operators. Considering the property **P** from above we can use the until operator to state the MTL formula $\phi_1 \rightarrow \mathit{true}\ \mathrm{U}_I\ \phi_2$. However, a fundamental problem of this interpretation of MTL is that the graph conditions do not expose the matched elements (the patient P in our example), i.e., different patients may be used for the satisfaction of ϕ_1 and ϕ_2. A similar combination of graph conditions and PTCTL with the same limitations was introduced in [8].

Problem statement: A more sophisticated logic such as MTGL introduced in chapter 5 is required to suitably define metric temporal specifications.

4 Timed Graph Sequences and Graphs with History

In the domain of runtime monitoring the sequence of states is automatically generated from a system that usually behaves according to unknown rules. In the following we abstract from the actual mechanism that generates a sequence of graphs to be analyzed.

4.1 Timed Graph Sequences

To include the concept of time, we equip all steps between graphs G and G' in the state sequences with the lifespan where G remains unchanged. Note that MTL considers infinite timed state sequences whereas we are concerned with the runtime monitoring perspective, in which only the past states of sequences are given in the form of finite timed state sequences.

Definition 3 (Timed Graph Sequence (TGS))**.** *We inductively define the set of all* timed graph sequences *(TGS)* Π *as follows:*

- *If* $\pi = \emptyset$ *is the word containing only the empty graph, then* $\pi \in \Pi$.
- *If* $\pi \cdot G \in \Pi$ *is a TGS ending with a graph* G, $l : IG \hookrightarrow G$, $r : IG \hookrightarrow G'$ *are inclusions but at least one of them is no isomorphism (for an interface graph* IG*), and* $\delta \in \mathbf{R}^+$ *is the time duration where the graph* G *remains unchanged, then* $\pi \cdot G \cdot (\delta \cdot l \cdot r) \cdot G' \in \Pi$ *is also a TGS.*

For a TGS $G \cdot (\delta \cdot l \cdot r) \cdot G' \in \Pi$, the graphs G and G' correspond to the states s_1 and s_2, respectively, of a timed state sequence $(s_1, \delta_1), (s_2, \delta_2), \ldots$ (defined for MTL in Sec. 2.2) and the duration δ in the TGS corresponds to the duration δ_1 of the timed state sequence. The inclusions $l : IG \hookrightarrow G$ and $r : IG \hookrightarrow G'$ are used in TGS to identify the nodes and edges that are preserved from G to G', i.e., $G - l(IG)$ are the nodes and edges that are present in G but are deleted from G' and $G' - r(IG)$ are the nodes and edges that do not exist in G but are created in G'. Furthermore, we require that at least one of the morphisms l or r is no isomorphism implying that $G - l(IG)$ or $G' - r(IG)$ is not empty because, during the monitoring process, we want to capture only those states of a system as new graphs in a TGS where some changes over time occurred.

An example for a TGS is given in Fig. 4.1 where the top part shows a TGS π represented by five graphs $G_0 = \emptyset, \ldots, G_4$ showing the monitored states in five different points in time, namely 0, 5, 10, 13, and 15. The durations where the respective graphs G_i remain unchanged are denoted by δ_i for $i \in \{0, 1, 2, 3\}$.

4.2 Graphs with History

To capture changes to a current graph over time, which is given by a TGS π as defined above, we introduce graphs with history. For this purpose we assume that the used type graph TG contains for all nodes and edges the attributes cts and dts of sort real to capture the total timepoint at which an element was created and (if applicable) deleted, respectively.

Definition 4 (Graph with History (GH))**.** *If TG is a type graph where each node and each edge has attributes* cts *denoting the timestamp of creation and* dts *denoting the timestamp of deletion, then TG is a* history type graph*. If G_H is a graph typed over a history type graph, then G_H is a* graph with history (GH)*.*

For consistency, we ensure that all considered GH satisfy (a) that there is precisely one cts attribute for every graph node and edge, (b) that there is at most one dts attribute for every graph node and edge, (c) that for an edge e the value of the cts attributes of the source and target nodes of e are less equal to the cts attribute of e, and (d) that for an edge e the value of the dts attributes of the source and target nodes of e are greater equal to the dts attribute of e. Note that these consistency requirements are not automatically guaranteed by the formalisms of E-Graphs or symbolic graphs. Moreover, we denote the set of all creation timestamps and deletion timestamps occurring in a graph with history G_H by $\text{cts}(G_H)$ and $\text{dts}(G_H)$, respectively.

4.3 Operation Fold

We now define the operation fold, which recursively converts a (finite) TGS π into the corresponding graph with history G_H.

Definition 5 (Map TGS to GH (Operation fold))**.** *We define operation* fold *as follows:*

- *If $\pi = \emptyset$ is a word containing only the empty graph, then* $\text{fold}(\pi) = \emptyset$*.*
- *If $\pi = \pi' \cdot G \cdot (\delta \cdot l \cdot r) \cdot G'$ is a TGS, $G'_H = \text{fold}(\pi' \cdot G)$ is the GH obtained from the translation of the TGS $\pi' \cdot G$ using the operation* fold*, and $t = \max(\{0\} \cup \text{cts}(G'_H) \cup \text{dts}(G'_H))$ is the maximal timestamp where a node or edge from G'_H was created or deleted, then* $\text{fold}(\pi)$ *is constructed from G'_H by adding the attributes $\text{dts}(x) = t + \delta$ to each node or edge $x \in G - l(IG)$ in G'_H, by adding the nodes and edges from $G' - r(IG)$ to G'_H, and by adding the attributes $\text{cts}(x) = t + \delta$ to each node or edge $x \in G' - r(IG)$ in G'_H.*

Note that we obtain the history type graph used for the result of the operation fold from the type graph used for the graphs occurring in the TGS π by adding the attributes cts and dts to all nodes and edges (assuming that such attributes are not contained before). Furthermore, in the recursive case in the definition above we encode the removal of nodes and edges, which are deleted in the TGS since they

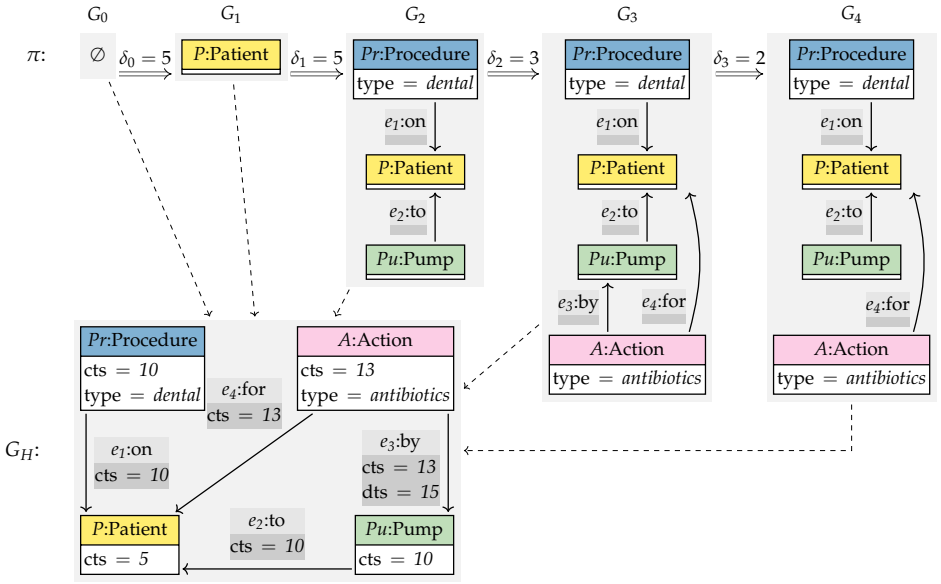


Figure 4.1: Mapping of a TGS π to the corresponding graph with history G_H using the operation fold. The arrows $\overset{\delta_i}{\Rightarrow}$ between states of the TGS describe changes $G_i \cdot (\delta_i, l_i, r_i) \cdot G_{i+1}$ where the inclusions l_i and r_i are implicitly given by the usage of the same names in all graphs. The dashed arrows from the states of the TGS to the GH visualize that the changes of the currently monitored system state are integrated into the GH

are contained in $G - l(IG)$, by adding a deletion timestamp to them in the GH and we encode the creation of nodes and edges, which are created in the TGS since they are contained in $G' - r(IG)$, by first adding these nodes and edges to the GH and then by adding to them a creation timestamp.

Example 1 (Map TGS to GH). *We translate the finite TGS π from the top part of Fig. 4.1 into the graph with history G_H shown in the bottom part of Fig. 4.1 using the operation* fold *as follows. Every TGS starts with an empty graph. Thus, as the first step, we translate the empty graph into the empty GH. The second state of the TGS given by G_1, which includes the Patient node P, is monitored and added to the TGS after 5 timeunits have elapsed. We translate this TGS state into the GH by adding P to the empty GH and by, additionally, equipping this node with the creation timestamp* cts $= 5$. *After another 5 timeunits have elapsed, an additional Procedure node Pr, a Pump node Pu, and edges e_1, e_2 between the existing Patient node P and the new Procedure node Pr resp. the new Pump node Pu are monitored. These changes are again translated into the GH by adding the Procedure node Pr, the Pump node Pu, and the edges e_1, e_2 to the current version of G_H as well as by additionally equipping them with the creation timestamps* cts $= 10$. *In the similar manner the Action node A together with the edges e_3 and e_4 (see G_3 in Fig. 4.1) are added to the GH with the creation timestamps* cts $= 13$. *Finally, after 2 timeunits in addition, we have monitored that the edge e_3 is deleted. To reflect this in the GH, we add to the edge e_3 in G_H the additional deletion timestamp* dts $= 15$.

5 Metric Temporal Graph Logic

We build upon graph conditions [5] and MTL [7, 10] to introduce in this section *Metric Temporal Graph Logic* (MTGL) by defining its syntax and semantics.

5.1 Syntax

To define the syntax of MTGL we introduce *Metric Temporal Graph Conditions* (MTGC), which feature the operators known from graph conditions and additionally include the *until* operator from MTL. In the following definition of MTGC we extend the binding of graph elements used by the graph condition operator $\exists$ to the until operator as we motivated in chapter 3.

Definition 6 (Metric Temporal Graph Conditions (MTGC))**.** *We inductively define the set of* metric temporal graph conditions *(MTGC)* Φ_G^{MTGC} *over a graph G as follows:*

- *If* $S = \{\phi_1, \ldots, \phi_n\} \in \Phi_G^{\mathrm{MTGC}}$*, then* $\wedge S \in \Phi_G^{\mathrm{MTGC}}$*.*
- *If* $\phi \in \Phi_G^{\mathrm{MTGC}}$*, then* $\neg\phi \in \Phi_G^{\mathrm{MTGC}}$*.*
- *If* $a : G \hookrightarrow H$ *and* $\phi \in \Phi_H^{\mathrm{MTGC}}$*, then* $\exists(a, \phi) \in \Phi_G^{\mathrm{MTGC}}$*.*
- *If* $\phi_1, \phi_2 \in \Phi_G^{\mathrm{MTGC}}$ *and I is an interval over* $\mathbf{R}_0^+$*, then* $\phi_1 \, \mathrm{U}_I \, \phi_2 \in \Phi_G^{\mathrm{MTGC}}$*.*

Various additional operators can be defined along the usual ways as for MTL and graph conditions. In our running example (see Fig. 5.1) we use the *forall-new* operator in the form of $\forall^{\mathrm{N}}(a : G \hookrightarrow H, \phi)$ to match the pattern H into the considered GH as soon as possible, i.e., precisely at the minimal timepoint, at which all its elements are created in the GH. Note that this operator can be encoded by the MTGC $\neg((\neg\,\exists(a : G \hookrightarrow H, \neg\phi))\ \mathrm{U}_{[0,\infty)}\ \exists(a : G \hookrightarrow H, \neg\phi))$.

Consider the formalization of the property **P** from chapter 3 using the MTGC in Fig. 5.1. In this figure and also in Fig. 5.2 we omit nodes in subconditions (such as *Pr*) if no new edges or attributes are attached to them. Moreover, in subconditions we omit edges (such as e_1) if no new attributes are attached to them and attributes (such as *type* of *Pr*) in general. Also note that we established the desired binding of graph elements in this example as follows: the nodes *Pr*, *P*, and *Pu* (together with the edges e_1, e_2 and the *type* attribute of *Pr*) are shared among the two subconditions of the until operator. This implies that the Pump node that must be matched by the right subcondition of the until operator is the previously bound Pump node *Pu*. Similarly, the Patient node that may be matched by the left subcondition of the until operator is the previously bound Patient node *P*. Hence, with the introduced concept of binding we have solved the problem stated in chapter 3.

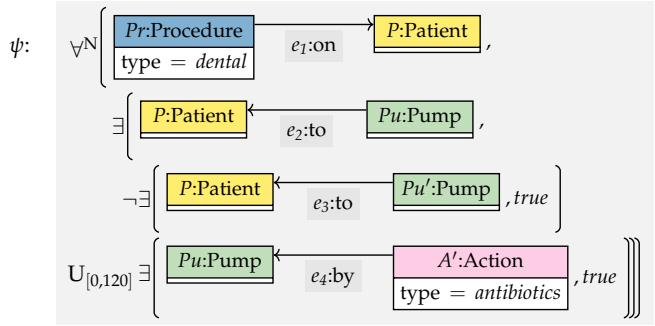


Figure 5.1: The property **P** from our running example expressed by the MTGC ψ

5.2 Semantics

We now define the semantics of MTGL by introducing the satisfaction relation, which explains when an MTGC is satisfied by a GH. This satisfaction relation handles the MTGL operators similarly to their handling for graph conditions and MTL. For the satisfaction of an MTGC of the form $\exists(a : G \hookrightarrow G', \phi)$, where the $\exists$ operator is inherited from graph conditions, it is still required that the pattern that is found so far (given by some mono $m : G \hookrightarrow G_H$) in the graph with history G_H can be extended to a larger pattern (given by some mono $m' : G' \hookrightarrow G_H$). Additionally, we have to check that all matched elements are already created (because the GH also contains the elements created with higher cts values) but not yet deleted (because the GH also contains the elements deleted at earlier dts values). For the satisfaction of an MTGC of the form $\phi_1 \, \mathrm{U}_I \, \phi_2$, where the until operator is inherited from MTL, it is still required that ϕ_2 must be satisfied at some timepoint t' in the interval I relative to the current observation timepoint t and that ϕ_1 is continuously satisfied (by a possibly varying match for each timepoint) for all timepoints preceding t'.

Definition 7 (Satisfaction of MTGC for a Graph with History)**.** *An* MTGC $\psi \in \Phi_G^{\mathrm{MTGC}}$ *is satisfied by a mono* $m : G \hookrightarrow G_H$*, where* G_H *is a graph with history, and an observation timepoint* $t \in \mathbf{R}_0^+$*, written* $(m, t) \models \psi$*, if one of the following cases applies:*

- $\psi = \wedge\{\phi_1, \ldots, \phi_n\}$ *and* $(m, t) \models \phi_i$ *(for all* $1 \leq i \leq n$*).*
- $\psi = \neg\phi$ *and not* $(m, t) \models \phi$*.*
- $\psi = \exists(a : G \hookrightarrow G', \phi)$ *and there is some* $q : G' \hookrightarrow G_H$ *such that* $q \circ a = m$*,* $(q, t) \models \phi$*, and* $\max(\{0\} \cup \mathrm{cts}(q(G'))) \leq t < \min(\{\infty\} \cup \mathrm{dts}(q(G')))$*.*
- $\psi = \phi_1 \, \mathrm{U}_I \, \phi_2$ *and there is some* $t' \in I$ *such that* $(m, t + t') \models \phi_2$ *and for every* $t'' \in [0, t')$ *it holds that* $(m, t + t'') \models \phi_1$*.*

A graph with history G_H *satisfies an* MTGC ψ *over the empty graph if the unique initial morphism* $i_{G_H} : \emptyset \hookrightarrow G_H$ *satisfies* ψ *for the observation timepoint* 0*.*

Considering our running example we can easily see that the MTGC given in Fig. 5.1 is satisfied by the GH given in Fig. 4.1. Firstly, the $\forall^{\mathrm{N}}$ operator matches

the nodes Pr, P and the edge e_1 changing thereby the observation timepoint to $t = 10$, which is the maximal creation timestamp of these three elements. Then, the $\exists$ operator matches the node Pu together with the edge e_2 and keeps the current observation timepoint at $t = 10$. Finally, the until operator matches subsequently the node A' and the edge e_3 at the observation timepoint $t = 15$ and the remainder *true* is trivially satisfied for the observation timepoint $t = 15$. In addition, as also required by the until operator, for every timepoint in $[10, 15)$ it is not possible to match a second Pump node Pu' that is connected to P.

As already discussed in chapter 4, MTL considers an infinite timed state sequence whereas MTGC focuses on runtime monitoring, in which only past states are considered, hence its satisfaction is determined for finite timed state sequences. Consequently, an MTGC that reasons on states at future timepoints may have different results compared to infinite semantics such as defined for MTL. Such a semantics definition could increase the efficiency of runtime monitoring but is left as future work. In the following, we assume the considered sequences to be not arbitrarily truncated, such that all expected reactions to specific conditions (as formulated, for example, in **P**) can be found in the monitored sequence.

5.3 Operation Reduce

We now present the operation reduce for reducing an MTGC to the corresponding graph condition. It encodes in the resulting graph condition all parts of the satisfaction relation of MTGL that are not covered by the satisfaction relation for graph conditions. In particular, the operation reduce removes all occurrences of the until operator and encodes the check that the elements that matched by the $\exists$ operator have all been created as well as that none of them has yet been deleted.

Definition 8 (Reduce MTGC into GC (Operation reduce))**.** *The operation* reduce *takes 3 arguments: a graph with history G_H that has been obtained by application of the* fold *operation to a TGS π, an observation timepoint $t \in \mathbf{R}_0^+$, and an* MTGC *ψ over the empty graph $\emptyset$. G_H and all graphs contained in ψ are typed over the history type graph TG.*

The operation reduce *returns a pair (G'_H, ψ') of a typed attributed graph G'_H (which is a slight modification of G_H) and a graph condition $\psi' \in \Phi^{\mathrm{GC}}_{\emptyset}$ over the empty graph $\emptyset$. The graph G'_H and all graphs contained in ψ' are typed over an adapted type graph TG' also introduced below.*

In the subsequently given procedure, additional elements added to graphs are fresh (w.r.t. the existing ones) and pairwise distinct.

1. *(Construction of adapted type graph TG')*
   We adapt the original type graph TG to TG' by adding a node Encoding with attributes num : int *and* var : real.

2. *(Construction of reduced graph condition ψ')*
   We obtain ψ' from ψ in multiple steps.

a) *We add the attributes* $\mathsf{cts} = x_{c,\alpha}$ *and* $\mathsf{dts} = x_{d,\alpha}$ *to all nodes and edges* α *contained in graphs in* ψ.

b) *We wrap* ψ *into an existential quantification* $\exists(i_\emptyset : \emptyset \hookrightarrow \emptyset, \psi)$.

c) *For all subconditions of* ψ *of the form* $\exists(m : G \hookrightarrow G', \phi)$ *we add an Encoding node with attributes* $\mathrm{num} = n$ *and* $\mathrm{var} = x_n$ *to* G' *where* n *is fresh in each case. We also add for every other node and edge* α *in* G' *the attribute constraints* $\mathsf{le}(x_{c,ff}, x_n)$ *and* $\mathsf{or}(\mathsf{eq}(x_{d,ff}, -1), \mathsf{lt}(x_n, x_{d,ff}))$. *Also, the added nodes and attribute constraints are additionally inserted into all graphs occurring in* ϕ. *Moreover, the node that is added to the target graph of the outermost morphism must use the variable* x_{outer} *for the* var *attribute. Finally, we add the constraint* $\mathsf{eq}(x_{outer}, t)$ *to this graph.*

d) *We replace all subconditions of* ψ *of the form* $\phi_1 \ \mathrm{U}_I \ \phi_2$ *over a graph* G *by* $\exists(m_1 : G \hookrightarrow G', \phi_2' \ \wedge \ \forall(m_2 : G' \hookrightarrow G'', \phi_1'))$. *The graphs* G' *and* G'' *are obtained from* G *by adding nodes and attribute constraints as described subsequently (i.e.,* $G \subset G' \subset G''$ *and* m_1, m_2 *are inclusions). The conditions* ϕ_1 *and* ϕ_2 *inherit the additions to* G' *and to* G'' *as before, respectively, resulting in the conditions* ϕ_1' *and* ϕ_2'.

- *Due to the previous step, there is an Encoding node with the attribute* $\mathrm{var} = x_n$ *in* G *that was not added to a graph of an enclosing subcondition of* ψ.
- *We add an Encoding node with attributes* $\mathrm{num} = n$ *and* $\mathrm{var} = y_n$ *to* G *to obtain* G' *where* n *is fresh in each case. We also add to* G *the attribute constraints* $\mathsf{ge}(y_n, \mathsf{add}(x_n, i_1))$ *and* $\mathsf{le}(y_n, \mathsf{add}(x_n, i_2))$ *for the interval* $I = [i_1, i_2]$ *(and similarly for left or right open intervals also omitting the second constraint when* $i_2 = \infty$*).*
- *We add an Encoding node with attributes* $\mathrm{num} = m$ *and* $\mathrm{var} = y_m$ *to* G' *to obtain* G'' *where* m *is fresh in each case. We also add to* G' *the attribute constraints* $\mathsf{le}(x_n, y_m)$ *and* $\mathsf{lt}(y_m, y_n)$.

e) *Again, we consider all subconditions of* ψ *of the form* $\exists(m : G \hookrightarrow G', \phi)$ *where there are Encoding nodes* v_1 *and* v_2 *with attributes* $\mathrm{var} = z_n$ *and* $\mathrm{var} = x_m$ *in* G *and* G', *respectively, that were not added to a graph of an enclosing subcondition of* ψ *and where* x_m *is no* y_n *from 2d. We then add the attribute constraint* $\mathsf{eq}(z_n, x_m)$ *to the graph* G' *and all graphs contained in* ϕ.

3. *(Construction of adapted graph with history* G_H'*)*
We obtain G_H' *by adding elements to* G_H *as follows:*

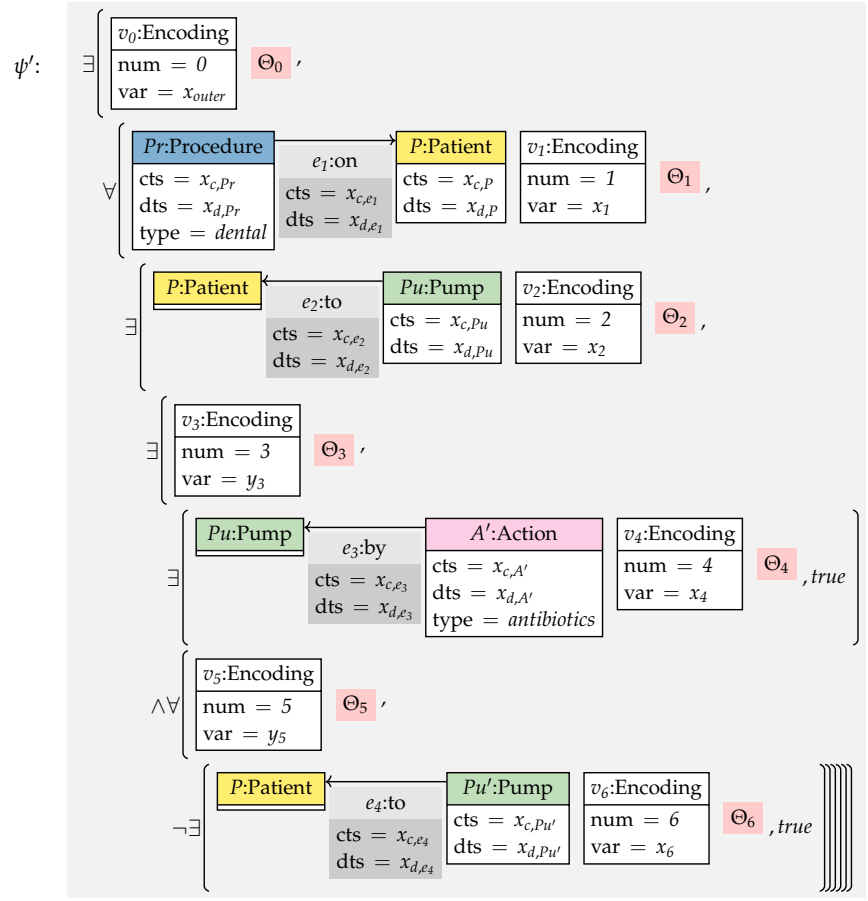
a) *We add the attribute* $\mathsf{dts} = -1$ *to all nodes/edges without that attribute.*

b) *We insert all Encoding nodes contained in graphs in* ψ'.

For our running example, Fig. 5.2 visualizes the outcome of the application of the reduce operation to the GH given in Fig. 4.1, the observation timepoint $t = 10$, and the MTGC depicted in Fig. 5.1. However, to simplify the presentation we have replaced the enclosing $\forall^{\mathrm{N}}$ operator by the $\forall$ operator to avoid the substitution of

the $\forall^{\mathrm{N}}$ operator by its encoding mentioned above since this would result in an MTGC of more than twice the size.

We now state that the operation reduce is sound w.r.t. the satisfaction relations for MTGC and graph conditions where $i_{G_H} : \emptyset \hookrightarrow G_H$ and $i_{G'_H} : \emptyset \hookrightarrow G'_H$ are again the unique initial morphisms.



$$
\begin{aligned}
\Theta_0 &= \{\ \mathsf{eq}(x_{outer}, 10)\ \}\\
\Theta_1 &= \Theta_0 \cup \{\ \mathsf{eq}(x_{outer}, x_1),\ \mathsf{le}(x_{c,Pr}, x_1),\ \mathsf{or}(\mathsf{eq}(x_{d,Pr}, -1), \mathsf{lt}(x_1, x_{d,Pr})),\\
&\qquad \mathsf{le}(x_{c,P}, x_1),\ \mathsf{or}(\mathsf{eq}(x_{d,P}, -1), \mathsf{lt}(x_1, x_{d,P})),\\
&\qquad \mathsf{le}(x_{c,e_1}, x_1),\ \mathsf{or}(\mathsf{eq}(x_{d,e_1}, -1), \mathsf{lt}(x_1, x_{d,e_1}))\ \}\\
\Theta_2 &= \Theta_1 \cup \{\ \mathsf{eq}(x_1, x_2),\ \mathsf{le}(x_{c,Pu}, x_2),\ \mathsf{or}(\mathsf{eq}(x_{d,Pu}, -1), \mathsf{lt}(x_2, x_{d,Pu})),\\
&\qquad \mathsf{le}(x_{c,e_2}, x_2),\ \mathsf{or}(\mathsf{eq}(x_{d,e_2}, -1), \mathsf{lt}(x_2, x_{d,e_2}))\ \}\\
\Theta_3 &= \Theta_2 \cup \{\ \mathsf{ge}(y_3, \mathsf{add}(x_2, 0)),\ \mathsf{le}(y_3, \mathsf{add}(x_2, 120))\}\\
\Theta_4 &= \Theta_3 \cup \{\ \mathsf{eq}(y_3, x_4),\ \mathsf{le}(x_{c,A'}, x_4),\ \mathsf{or}(\mathsf{eq}(x_{d,A'}, -1), \mathsf{lt}(x_4, x_{d,A'})),\\
&\qquad \mathsf{le}(x_{c,e_3}, x_4),\ \mathsf{or}(\mathsf{eq}(x_{d,e_3}, -1), \mathsf{lt}(x_4, x_{d,e_3}))\ \}\\
\Theta_5 &= \Theta_3 \cup \{\ \mathsf{le}(x_2, y_5),\ \mathsf{lt}(y_5, y_3)\}\\
\Theta_6 &= \Theta_5 \cup \{\ \mathsf{eq}(y_5, x_6),\ \mathsf{le}(x_{c,Pu'}, x_6),\ \mathsf{or}(\mathsf{eq}(x_{d,Pu'}, -1), \mathsf{lt}(x_6, x_{d,Pu'})),\\
&\qquad \mathsf{le}(x_{c,e_4}, x_6),\ \mathsf{or}(\mathsf{eq}(x_{d,e_4}, -1), \mathsf{lt}(x_6, x_{d,e_4}))\ \}
\end{aligned}
$$

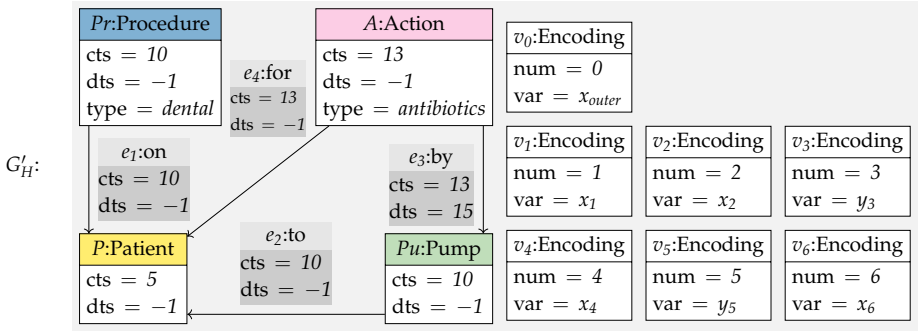


Figure 5.2: The graph condition ψ' and the adapted graph G'_H resulting from applying the operation reduce to the GH from Fig. 4.1, the timepoint $t = 10$, and the property ψ from Fig. 5.1 (where the outermost $\forall^{\mathrm{N}}$ has been simplified to $\forall$)

Theorem 1 (Soundness of Operation reduce)**.** *If π is a* TGS*, $G_H = \text{fold}(\pi)$ is a graph with history, ψ is an* MTGC *over the empty graph, $t \in \mathbf{R}_0^+$ is a timepoint, and $(G'_H, \psi') = \text{reduce}(G_H, t, \psi)$, then $(i_{G_H}, t) \models \psi$ iff $i_{G'_H} \models \psi'$.*

Idea. Throughout the proof we refer to items given in Def. 8.
(⇒): In the first step we establish a connection between the two satisfaction statements that is then verified by structural induction on the condition ψ in the second step.

The outermost existential quantification added in item 2b can be matched to G'_H because the required Encoding node is present in G'_H due to item 3b and the used match is unique due to the *num* attribute. The variable x_{outer} is also restricted to the value of t according to item 2c.

Hence, a satisfaction proof state (m, t, ϕ) of MTGC is connected to a satisfaction proof state (m', ϕ') of GC as follows: the match m' is an extension of m where additionally some Encoding nodes with their attributes and variables are matched (initially this is $i_{G'_H}$, which is matching the Encoding node added in item 2c) and the timestamp t is represented by the variable used in the most recently matched Encoding node (initially this is x_{outer}).
We now proceed by induction on the conditions ϕ omitting the trivial cases on conjunction and negation.

- **($\exists$ operator):** We assume that $(m, t) \models \exists(a : G_1 \hookrightarrow G_2, \phi)$ and show that $m' \models \exists(a' : G'_1 \hookrightarrow G'_2, \phi')$ for the result obtained by application of the operation reduce according to item 2c.

  Due to the assumption and by Def. 7 there is some $q : G_2 \hookrightarrow G_H$ such that $q \circ a = m$, $(q, t) \models \phi$, and $\max(\{0\} \cup \text{cts}(q(G_2))) \leq t < \min(\{\infty\} \cup \text{dts}(q(G_2)))$. This mono q can be used to extend m' to a mono $q' : G'_2 \hookrightarrow G'_H$. Here q' matches a further Encoding node v that could not have been matched before and that is unique due to the *num* attribute. The node v has a variable x_n for the *var* attribute. By the attribute constraint on x_n we have that x_n is equal to the outer variable that encodes the current timepoint t due to item 2e. Hence, x_n encodes the timepoint that is also used for the MTGL satisfaction statement $(q, t) \models \phi$ from above. Moreover, the attribute constraints added for all nodes and edges, according to item 2c, encodes the statement $\max(\{0\} \cup \text{cts}(q(G_2))) \leq t < \min(\{\infty\} \cup \text{dts}(q(G_2)))$ from above. The graph condition is able to make a statement on all dts attributes because the dts attribute was added to all nodes and edges in item 2a and, additionally, the graph condition can still be matched using q' to G'_H because G_H has been adapted in item 3a to also contain all dts attributes. The subconstraint checking for -1 is then required to only consider the actually deleted nodes and edges. Finally, the translated condition ϕ' is then also satisfied by the induction hypothesis.

- **(until operator):** We assume that $(m, t) \models \phi_1 \, \mathrm{U}_I \, \phi_2$ and show that $m' \models \exists(m_1 : G \hookrightarrow G', \phi'_2 \wedge \forall(m_2 : G' \hookrightarrow G'', \phi'_1))$ for the result obtained by application of the operation reduce according to item 2d.

Due to the assumption and by Def. 7 there is some $t' \in I$ such that $(m, t + t') \models \phi_2$ and for every $t'' \in [0, t')$ it holds that $(m, t + t'') \models \phi_1$. By item 2e we can assume that the outer variable x_o encodes the current timepoint t as in the previous item. The existentially quantified $t' \in I$ is now covered by the graph G', in which we use the variable y_n to encode the value of t' and the attribute constraints to restrict y_n to the interval I. Then, $(m, t + t') \models \phi_2$ implies that the match m' can be extended to a match m'', as required by the existential quantification, because the variable y_n along with its Encoding node can be matched to G'_H due to item 3b. The translated condition ϕ'_2 is then also satisfied by the induction hypothesis. Moreover, the universally quantified t'' is then represented by the variable y_m in G'', which is also universally quantified. We assume that t'' is fixed in that interval satisfying $(m, t + t'') \models \phi_1$. Hence, the match m'' can be extended to a match m''' as required by matching y_m to the corresponding variable in G'_H due to item 3b. Also, the attribute constraints on y_m given in item 2d are satisfied because t'' is taken from the interval $[0, t')$. The translated condition ϕ'_1 is then also satisfied by the induction hypothesis.

($\Leftarrow$): The reverse reasoning applies for the if direction in all these steps. It is important however to realize that the patterns obtained due to the encoding of the operators $\exists$ and U must be matched entirely for the reverse direction to preserve the correspondence with the MTGC satisfaction proof state. □

By application of our theorem above we can deduce for our running example that the property translated by the operation reduce is satisfied by the adapted GH (both given in Fig. 5.2). For this purpose observe that the property from Fig. 5.1 (simplified as stated in Fig. 5.2) is satisfied by the GH from Fig. 4.1 for the timepoint $t = 10$ since the unique match of the Procedure node *Pr*, the on edge e_1, and the Patient node *P* satisfies the remaining condition at timepoint $t = 10$ as discussed before.

6 Tool Support

We have extended the tool AutoGraph [12] to support MTGL for the analysis of timed graph sequences and for the runtime monitoring of dynamic systems generating such sequences. Firstly, following chapter 4 we have implemented the operation fold consolidating a TGS to a GH. For this purpose we process a timed sequence of modification events issued by the system (which characterize the inclusion morphisms l and r from Def. 3) and construct from them the resulting GH. Secondly, we have added the required syntactical support for MTGC (see Def. 6) by adding the metric until operator to the syntax of graph conditions. Thirdly, we have implemented the operation reduce translating a satisfaction statement on an MTGC and a GH into the corresponding satisfaction statement on a graph condition and a graph. The application to our running example from the medical domain demonstrates the validity of our general approach but detailed analysis regarding the performance using benchmarks from runtime monitoring is left as future work.

7 Conclusion and Future Work

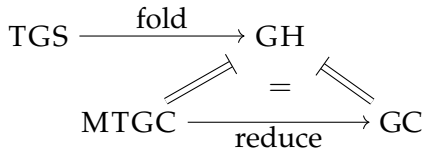


Figure 7.1: Overview of our approach

We have defined MTGL to express state properties using graph conditions and sequence properties using the until operator, which allows to maintain an established binding of graph elements throughout the analysis of a timed sequence of typed attributed graphs. We present how TGS and MTGC can be reduced to GH and GC by the operations fold and reduce, respectively, as shown in Fig. 7.1. Tool-support for MTGL is achieved by extending the tool AutoGraph.

We believe MTGL will prove advantageous in many future applications. We plan to develop two more finite semantics, also for arbitrary truncated sequences: the optimistic, where it guarantees that no violations are reported too early, and the pessimistic, where it guarantees that violations are reported, when possible, n timeunits before their definitive occurrence. Furthermore, we plan to compare our approach w.r.t. efficiency to other runtime monitoring tools on standard benchmarks, and identify subsets of MTGL that can be checked more efficiently (e.g. incrementally). Regarding the analysis, we plan to study whether our reduction could reduce other cases of checking MTGL to related GC counterparts such as invariant checking. Finally, initial results indicate that the concept of binding used in MTGL can be extended to allow for a more convenient resp. more compact specification of sequence properties.

References

[1] E. Bartocci, Y. Falcone, B. Bonakdarpour, C. Colombo, N. Decker, K. Havelund, Y. Joshi, F. Klaedtke, R. Milewicz, G. Reger, et al. "First international competition on runtime verification: rules, benchmarks, tools, and final results of CRV 2014". In: *International Journal on Software Tools for Technology Transfer* (2017), pages 1–40.

[2] D. Basin, F. Klaedtke, S. Müller, and E. Zălinescu. "Monitoring metric first-order temporal properties". In: *Journal of the ACM (JACM)* 62.2 (2015), page 15.

[3] B. Courcelle. "The Expression of Graph Properties and Graph Transformations in Monadic Second-Order Logic". In: *Handbook of Graph Grammars*. Edited by G. Rozenberg. World Scientific, 1997, pages 313–400.

[4] H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of algebraic graph transformation*. Springer-Verlag, 2006.

[5] A. Habel and K.-H. Pennemann. "Correctness of high-level transformation systems relative to nested conditions". In: *Mathematical Structures in Computer Science* 19 (2009), pages 1–52.

[6] F. Klein and H. Giese. "Joint Structural and Temporal Property Specification Using Timed Story Scenario Diagrams". In: *Fundamental Approaches to Software Engineering, 10th International Conference, FASE 2007, Held as Part of the Joint European Conferences, on Theory and Practice of Software, ETAPS 2007, Braga, Portugal, March 24 - April 1, 2007, Proceedings*. Edited by M. B. Dwyer and A. Lopes. Volume 4422. Lecture Notes in Computer Science. Springer, 2007, pages 185–199. DOI: 10.1007/978-3-540-71289-3_16.

[7] R. Koymans. "Specifying real-time properties with metric temporal logic". In: *Real-time systems* 2.4 (1990), pages 255–299.

[8] M. Maximova, H. Giese, and C. Krause. "Probabilistic Timed Graph Transformation Systems". In: *Graph Transformation - 10th International Conference, ICGT 2017, Held as Part of STAF 2017, Marburg, Germany, July 18-19, 2017, Proceedings*. 2017, pages 159–175. DOI: 10.1007/978-3-319-61470-0_10.

[9] F. Orejas. "Symbolic graphs for attributed graph constraints". In: *J. Symb. Comput.* 46.3 (2011), pages 294–315. DOI: 10.1016/j.jsc.2010.09.009.

[10] J. Ouaknine and J. Worrell. "Some Recent Results in Metric Temporal Logic". In: *Formal Modeling and Analysis of Timed Systems, 6th International Conference, FORMATS 2008, Saint Malo, France, September 15-17, 2008. Proceedings*. Edited by F. Cassez and C. Jard. Volume 5215. Lecture Notes in Computer Science. Springer, 2008, pages 1–13. DOI: 10.1007/978-3-540-85778-5_1.

[11] A. Rensink. "Representing First-Order Logic Using Graphs". In: *Proc. ICGT 2004*. Edited by H. Ehrig, G. Engels, F. Parisi-Presicce, and G. Rozenberg. Volume 3256. LNCS. Springer, 2004, pages 319–335. ISBN: 3-540-23207-9.

[12] S. Schneider, L. Lambers, and F. Orejas. "Symbolic Model Generation for Graph Properties". In: *Fundamental Approaches to Software Engineering - 20th International Conference, FASE 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*. Edited by M. Huisman and J. Rubin. Volume 10202. Lecture Notes in Computer Science. Springer, 2017, pages 226–243. DOI: 10.1007/978-3-662-54494-5_13.

[13] W. Wilson, K. A. Taubert, M. Gewitz, P. B. Lockhart, L. M. Baddour, M. Levison, A. Bolger, C. H. Cabell, M. Takahashi, R. S. Baltimore, et al. "Prevention of Infective Endocarditis: A Guideline From the American Heart Association". In: *Circulation* 116.15 (2007), pages 1736–1754.

Aktuelle Technische Berichte des Hasso-Plattner-Instituts

Band	ISBN	Titel	Autoren / Redaktion
122	978-3-86956-432-6	**Proceedings of the Fifth HPI Cloud Symposium "Operating the Cloud" 2017**	Estee van der Walt, Isaac Odun-Ayo, Matthias Bastian, Mohamed Esam Eldin Elsaid
121	978-3-86956-430-2	**Towards version control in object-based systems**	Jakob Reschke, Marcel Taeumel, Tobias Pape, Fabio Niephaus, Robert Hirschfeld
120	978-3-86956-422-7	**Squimera : a live, Smalltalk-based IDE for dynamic programming languages**	Fabio Niephaus, Tim Felgentreff, Robert Hirschfeld
119	978-3-86956-406-7	**k-Inductive invariant Checking for Graph Transformation Systems**	Johannes Dyck, Holger Giese
118	978-3-86956-405-0	**Probabilistic timed graph transformation systems**	Maria Maximova, Holger Giese, Christian Krause
117	978-3-86956-401-2	**Proceedings of the Fourth HPI Cloud Symposium "Operating the Cloud" 2016**	Stefan Klauck, Fabian Maschler, Karsten Tausche
116	978-3-86956-397-8	**Die Cloud für Schulen in Deutschland : Konzept und Pilotierung der Schul-Cloud**	Jan Renz, Catrina Grella, Nils Karn, Christiane Hagedorn, Christoph Meinel
115	978-3-86956-396-1	**Symbolic model generation for graph properties**	Sven Schneider, Leen Lambers, Fernando Orejas
114	978-3-86956-395-4	**Management Digitaler Identitäten : aktueller Status und zukünftige Trends**	Christian Tietz, Chris Pelchen, Christoph Meinel, Maxim Schnjakin
113	978-3-86956-394-7	**Blockchain : Technologie, Funktionen, Einsatzbereiche**	Tatiana Gayvoronskaya, Christoph Meinel, Maxim Schnjakin
112	978-3-86956-391-6	**Automatic verification of behavior preservation at the transformation level for relational model transformation**	Johannes Dyck, Holger Giese, Leen Lambers
111	978-3-86956-390-9	**Proceedings of the 10th Ph.D. retreat of the HPI research school on service-oriented systems engineering**	Christoph Meinel, Hasso Plattner, Mathias Weske, Andreas Polze, Robert Hirschfeld, Felix Naumann, Holger Giese, Patrick Baudisch, Tobias Friedrich, Emmanuel Müller
110	978-3-86956-387-9	**Transmorphic : mapping direct manipulation to source code transformations**	Robin Schreiber, Robert Krahn, Daniel H. H. Ingalls, Robert Hirschfeld

ISBN 978-3-86956-433-3
ISSN 1613-5652