Metric Temporal Graph Logic over Typed Attributed Graphs: Extended Version

Holger Giese, Maria Maximova, Lucas Sakizloglou, Sven Schneider

Technische Berichte Nr. 127

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 | 127

Holger Giese | Maria Maximova | Lucas Sakizloglou | Sven Schneider

Metric Temporal Graph Logic over Typed Attributed Graphs

Extended Version

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 2019
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

Online veröffentlicht auf dem Publikationsserver der Universität Potsdam
https://doi.org/10.25932/publishup-42752
https://nbn-resolving.org/urn:nbn:de:kobv:517-opus4-427522

Zugleich gedruckt erschienen im Universitätsverlag Potsdam:
ISBN 978-3-86956-463-0

Various kinds of typed attributed graphs can be used to represent states of systems from a broad range of domains. For dynamic systems, established formalisms such as graph transformation can provide a formal model for defining state sequences. We consider the case where time may elapse between state changes and introduce a logic, called *Metric Temporal Graph Logic* (MTGL), to reason about such timed graph sequences. With this logic, we express properties on the structure and attributes of states as well as on the occurrence of states over time that are related by their inner structure, which no formal logic over graphs concisely accomplishes so far.

Firstly, based on timed graph sequences as models for system evolution, we define MTGL by integrating the temporal operator *until* with time bounds into the well-established logic of (nested) graph conditions. Secondly, we outline how a finite timed graph sequence can be represented as a single graph containing all changes over time (called graph with history), how the satisfaction of MTGL conditions can be defined for such a graph and show that both representations satisfy the same MTGL conditions. Thirdly, we present how MTGL conditions can be reduced to (nested) graph conditions and, using this reduction, show that both underlying logics are equally expressive. Finally, we present an extension of the tool AutoGraph allowing to check the satisfaction of MTGL conditions for timed graph sequences by checking the satisfaction of the (nested) graph conditions, obtained using the proposed reduction, for the graph with history corresponding to the timed graph sequence.

1. Introduction

Various kinds of typed attributed graphs are used to represent states of systems from a broad range of domains. Also, the evolution of such systems can be described using a multitude of graph transformation formalisms in which the possible behavior in form of graph sequences is defined by a set of rules and their application. In many cases, the analysis of this induced behavior with respect to a specification in form of a temporal logic that defines the admissible graph sequences is of paramount importance.

In our running example, from which we derive the lack of suitable specification formalisms, we consider a dynamic system describing an operating system which generates timed sequences of (typed attributed) graphs to model the change of the operating system states over time. In this example, users may create tasks with identifiers *id*, the operating system may create handlers specific to task identifiers to allow for the task execution, and the handlers may produce a result when a task has been executed (marking the successful handling of the task). To model the states of the operating system, we employ graphs that store the tasks, the handlers, and the computed results. In the remainder, we refer in the context of this example to the *sequence property* **P** to be checked w.r.t. the *timed graph sequence* at hand describing systems' state changes over time.

P: Whenever a task T with identifier *id* is created on a system S, a handler H for this task (i.e., with a task identifier *t_id* equal to *id* of T) must exist. Moreover, within 120 timeunits, the handler must produce a result R with value *success* and, during the computation of the result, no other handler H' for the same task (i.e., with the same task identifier *t_id*) may exist.

We consider the problem that existing specification formalisms for graph-based systems cannot cover properties such as **P**. The available (metric) temporal logics, such as Metric Temporal Logic (MTL) [16], are defined over Kripke structures abstracting from the system states by labeling each state with a subset of the finite set of atomic propositions. The commonly used operator *until* allows then to formalize the part of property **P** stating that every graph that contains a task T is followed by some graph containing some result R before t time units. However, the existing metric temporal logics do not support the use of *bindings* of elements contained in the graphs to express how a certain matched pattern evolves in a sequence of graphs. Therefore, they are insufficient when e.g. creating different tasks T and T' must be followed by creating the *corresponding* results R and R' while also treating the deadlines for their existence separately.

As a first contribution, we define *Metric Temporal Graph Logic* (MTGL) for the concise specification of systems that generate timed graph sequences. In MTGL, we express properties on *states* using the well-known formalism of nested graph conditions [12, 25] (called GCs for short). The satisfaction of a GC that states the existence of a graph pattern H in the given graph G results in a *match* m from H to G. We extend the logic of GCs to MTGL by extending GCs with the metric temporal operator *until* that may appear in the scope of a previously determined match m. Using this extension, we can express properties, such as property **P**, on the structure and attributes of states as well as on the occurrence of states over time where the preservation/extension of matches during a systems' evolution increases the expressiveness beyond the existing temporal logics.

As a second contribution, we outline how a finite timed graph sequence can be represented as a single graph containing all changes over time (called *graph with history*), how the satisfaction of MTGL conditions can be defined for such a graph, and show that both representations satisfy the same MTGL conditions.

As a third contribution, we show that MTGL conditions can be reduced to GCs using attribute constraints to encode the metric temporal requirements, while preserving the satisfaction for finite timed graph sequences. This encoding enables the direct application of techniques for GCs such as [26].

As a fourth contribution, we present an extension of the tool AutoGraph [26] allowing to check the satisfaction of MTGL conditions for timed graph sequences by checking the satisfaction of the GCs obtained using the proposed reduction for the graph with history corresponding to the timed graph sequence at hand.

The paper is structured as follows. Section 2 discusses related work. Section 3 iterates on technical preliminaries. Section 4 defines timed graph sequences, MTGL, and the satisfaction of MTGL conditions for timed graph sequences. In Section 5, we show how to represent a finite timed graph sequence as a single graph with history, define satisfaction of MTGL conditions for a graph with history, and prove that both representations satisfy the same MTGL conditions. In Section 6, we introduce a reduction of MTGL conditions to GCs and show the equivalence of these two logics. Finally, Section 7 discusses the tool support and Section 8 concludes the paper with a summary and remarks on future work.

2. Related Work

There are several related formal and informal approaches for the specification and verification of different kinds of sequence properties.

In [13] the satisfaction of CTL (state/sequence) properties is checked where the tool GROOVE [10, 11] is used to generate the finite state space of the graph transformation system (GTS) at hand. In [7] invariants are checked for a GTS with a possibly infinite state space. The validity of given pre/post conditions for a program over a GTS has been presented in [24]. In [2, 15] temporal properties for GTS with infinite state space are checked using the tool AUGUR2.

In [20] the satisfaction of graph-based probabilistic timed CTL properties is checked where the tool HENSHIN [1, 9] is used to generate the finite state space of a GTS and where the tool PRISM [17] is used to model check translations of the given properties. In [6] a sequence of timed events are checked against sequence properties given by regular languages based on deterministic finite automata.

The use of bindings, as in this paper, is supported in [3] where bindings are part of the Metric First-Order Temporal Logic in which system states are represented by a set of relations that are adapted during the execution of the system.

A visual but informal notation for the specification of sequence properties involving time and graph bindings was introduced in [14].

In conclusion, existing approaches with a formal semantics do not support either time, bindings, or graphs in a concise manner. Thereby, our graph-based logic MTGL for graph-based systems complements existing approaches since (a) it eases usability in graph-based contexts similarly to the usage of GCs that are favored over first-order logic in these contexts, (b) it enables further developments and combinations with other graph-based techniques such as those in [26], and, (c) as to be shown by future tool-based evaluations, it can be expected that domain-specific tools for checking MTGL conditions are more efficient compared to general-purpose tools such as shown analogously for GCs in [24].

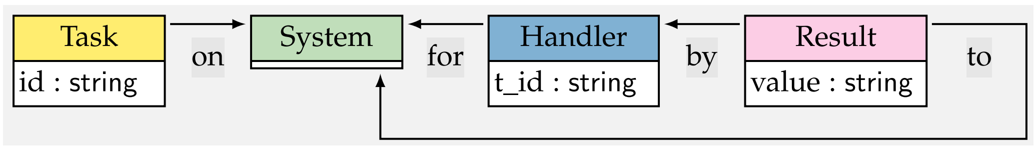


Figure 2.1.: The type graph *TG* for our running example where the attributes cts and dts of sort real used in later sections are omitted in every node and edge to improve readability

3. Typed Attributed Graphs and Graph Conditions

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

We use *symbolic graphs* [22] to encode (finite) typed attributed graphs. Symbolic graphs are an adaptation of E-Graphs [8] 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 replace the data nodes. Symbolic graphs are also equipped with attribute constraints over these (sorted) variables (e.g. $x = 5$, $x \leq 5$, and $y =$ "aabb").

We consider symbolic graphs that are typed over a type graph TG using a typing morphism $type : G \rightarrow TG$. Type graphs restrict attributed graphs to an admitted subset. For our running example, we employ the type graph TG from Fig. 2.1. An example of a symbolic graph that is typed over TG is given in Fig. 5.1.

We state the existence and nonexistence of graph patterns in a given symbolic graph, which is called a *host graph*, by representing graph patterns by symbolic graphs and by using monomorphisms (called *monos* and denoted using $\hookrightarrow$ subsequently) to extend graph patterns. Formally, we rely on the notion of nested graph conditions (GCs) [12], which are expressively equivalent to first-order logic on graphs [5] as shown in [12, 25].

Definition 1 (Graph Conditions (GCs)) *The class of* graph conditions (GCs) Φ^{GC}_{H} *for the graph H contains ψ if one of the following cases applies.*

- $\psi = \wedge S$ *and* $S = \{\phi_1, \ldots, \phi_n\} \subseteq \Phi^{\mathrm{GC}}_{H}$.
- $\psi = \neg\phi$ *and* $\phi \in \Phi^{\mathrm{GC}}_{H}$.
- $\psi = \exists(a, \phi)$, $a : H \hookrightarrow H'$, *and* $\phi \in \Phi^{\mathrm{GC}}_{H'}$.

GCs allow for further abbreviations such as true, false, $\vee S$, *and* $\forall(a, \phi)$.

Intuitively, a GC is satisfied if the positive but not the negative patterns given by the GC can be found in the given host graph. For the case of the *exists* operator, a previously determined match m must be extendable using a monomorphism q according to the monomorphism a from the GC.

Definition 2 (Satisfaction of GCs) *A GC* $\psi \in \Phi^{\mathrm{GC}}_{H}$ *is* satisfied *by a monomorphism* $m : H \hookrightarrow G$, *written* $m \models \psi$, *if one of the following cases applies.*

$$\begin{tikzcd} H \arrow[rr, hook, "a"] \arrow[dr, hook, "m"'] & \arrow[d, phantom, "="] & H' \arrow[dl, hook', "q"] \\ & G & \end{tikzcd}$$

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

A GC ψ over the empty graph is satisfied by a graph G, written $G \models \psi$, if $\mathrm{i}_G \models \psi$ where $\mathrm{i}_G : \emptyset \hookrightarrow G$ is the initial morphism to G.

To ease presentation, we omit here a more formal and technical handling of requiring a consistent variable assignment and meta-variables needed for conjunctions.

4. Metric Temporal Graph Logic

We build upon GCs [12] and the future fragment of MTL [16, 23] to introduce *Metric Temporal Graph Logic* (MTGL) by defining its syntax and semantics.

We assume a graph transformation based formalism for the definition of steps changing a graph while possibly also determining a progress of time. We abstract from the actual timed graph transformation formalism employed but only assume that it is capable to generate so-called *timed graph sequences* (short TGSs), which contain the graphs, their modifications, and the elapsed time between successive graphs. In the following, we are concerned with TGSs in which either only the past states of sequences are given in the form of *finite* TGSs or where, alternatively, an *infinite* TGS describes a nonterminating evolution of a system.

A step from a graph G to a graph G' where G has remained unchanged for a duration of δ, which may be determined by a timed graph transformation formalism, is represented by $G \cdot (\delta, l, r) \cdot G'$ in our notion of TGSs. In this representation, the monos $l : IG \hookrightarrow G$ and $r : IG \hookrightarrow G'$ identify the graph elements 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 to obtain G' and $G' - r(IG)$ are the nodes and edges that do not exist in G but are created to obtain G'.[1]

Definition 3 (Timed Graph Sequences (TGSs)) *We inductively define the class of finite* timed graph sequences *(TGSs) Π_{fin} as follows:*

- *If $\pi = G_{init}$ is the sequence containing only the graph G_{init}, then $\pi \in \Pi_{fin}$.*
- *If $\pi \in \Pi_{fin}$ is a TGS ending with a graph G, $l : IG \hookrightarrow G$, $r : IG \hookrightarrow G'$ are monos (for an* interface graph *IG), and $\delta \in \mathbf{R}^+$ is the timepoint where the graph G is changed relative to the previous change, then $\pi \cdot (\delta, l, r) \cdot G' \in \Pi_{fin}$.*

The class of TGSs Π contains the finite TGSs Π_{fin} from above and all infinite sequences that have only finite TGSs from Π_{fin} as prefixes.

Moreover, $\mathrm{dur}(\pi)$ *denotes the sum of all durations δ contained in π. Additionally, if* $\mathrm{dur}(\pi) = \infty$*, π_t denotes the unique graph at time t, i.e., if $\pi = G$ then $\pi_t = G$ and if $\pi = G \cdot (\delta, l, r) \cdot \pi'$ then ($\pi_t = G$ for $t < \delta$) and ($\pi_t = \pi'_{t-\delta}$ for $t \geq \delta$). Finally, if* $\mathrm{dur}(\pi) = \infty$*, $\pi_{[t_1,t_2]}$ denotes the finite TGS contained in π from π_{t_1} to π_{t_2}.*

We do not require that every step modifies the current graph (i.e., we permit $G = G'$ possibly using $l = r = \mathrm{id}_G$). Also, for well-definedness of the satisfaction relation for TGSs we require that time diverges in every infinite TGS π (i.e., $\mathrm{dur}(\pi) = \infty$).

[1]The span $G \xleftarrow{l} IG \xrightarrow{r} G'$ does not correspond to a rule as used in the DPO approach but rather to a rule application describing changes between the graphs G and G'.

In our running example, we simplify the presentation by using only inclusions l and r. The TGS π given in Fig. 4.1 contains five graphs G_i for $i \in \{0,1,2,3,4\}$ showing the system states in five different points in time, namely 0, 5, 10, 13, and 15. The corresponding durations where the respective graphs G_i remain unchanged are denoted by δ_i for $i \in \{0,1,2,3\}$.

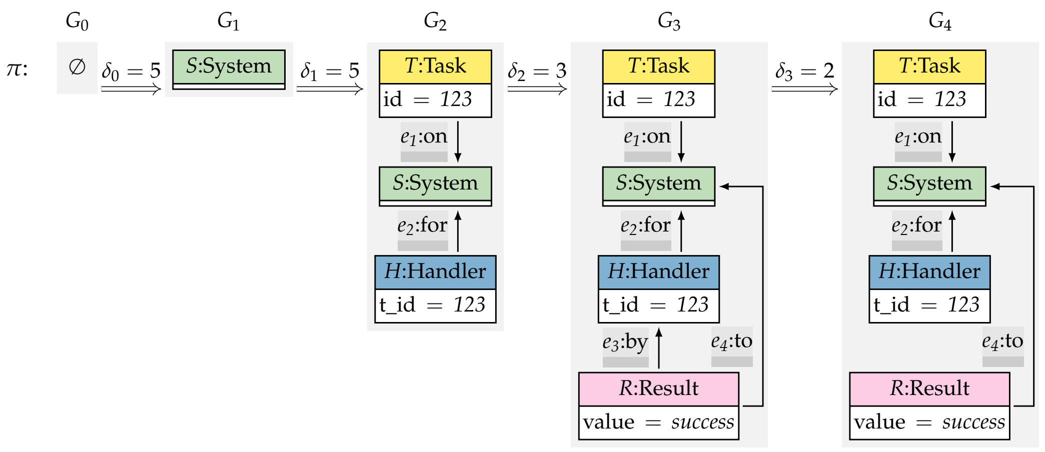


Figure 4.1.: A TGS π for our running example. For $i \in \{0,1,2,3\}$, the arrows $\overset{\delta_i}{\Longrightarrow}$ between graphs of the TGS describe changes $G_i \cdot (\delta_i, l_i, r_i) \cdot G_{i+1}$ where the inclusions l_i and r_i are implicitly given by the usage of the same names in all graphs

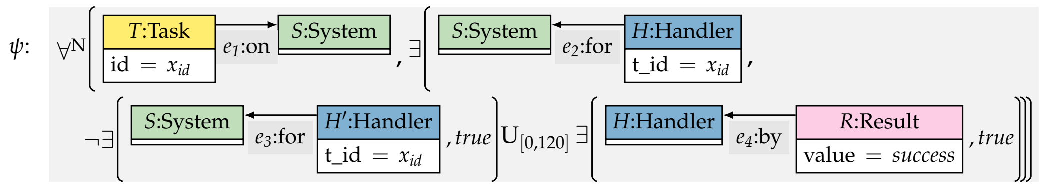


Figure 4.2.: The property **P** from our running example formalized by the MTGC ψ

The syntax of MTGL is given by *Metric Temporal Graph Conditions* (short MTGCs) introduced in the following definition. The distinguishing feature of MTGL is the extension of the binding of graph elements used by the operator *exists* in GCs to the *until* operator of MTL. This allows for the formalization of properties where a match into a graph is preserved/extended over multiple timepoints in the subsequently introduced semantics for TGSs.

Definition 4 (Metric Temporal Graph Conditions (MTGCs)) *The class of* metric temporal graph conditions (MTGCs) Φ_H^{MTGC} *for the graph H contains ψ if one of the following cases applies.*

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

Further metric temporal operators can be defined as for MTL and GCs.

For our running example, we formalize the property **P** from chapter 1 by the MTGC ψ depicted in Fig. 4.2. In this MTGC, we additionally use the *forall-new* operator in the form of $\forall^{\mathrm{N}}(a : H \hookrightarrow H', \phi)$ to match the pattern H' into the considered TGS as soon as possible, i.e., precisely at the minimal timepoint, at which all elements of H' exist. A formal handling of the *forall-new* operator is left for future work. Moreover, we use notational conventions to simplify our presentation of MTGCs by omitting elements in subconditions. Firstly, we omit nodes (such as T) if no new edges or attributes are attached to them. Secondly, we omit edges (such as e_1) if no new attributes are attached to them. Finally, we omit attributes (such as *id* of T) in general.

The MTGC ψ properly formalizes the property **P** using the binding capabilities of MTGL as follows: the nodes T, S, and H (together with the edges e_1, e_2 as well as their attributes) are shared among the two subconditions of the *until* operator. This implies that the Handler node that must be matched by the right subcondition of the *until* operator is the previously bound Handler node H. Similarly, the System node that may be matched by the left subcondition of the *until* operator is the previously bound System node S.

Next we present the MTGL *semantics for TGSs* that defines when a given TGS satisfies a given MTGC. For the definition of this semantics, we first introduce the concept of a *match that is preserved over a finite number of steps* given by a finite TGS. In the following, we also call such a preserved match a *binding*. The preservation of the match is guaranteed by adapting it according to the renaming determined by the steps of the TGS for the case where these steps do not remove any element initially matched.

Definition 5 (Preserved Match for a Finite TGS) *A monomorphism $m : H \hookrightarrow G_0$ is* preserved over a finite TGS *π that starts in G_0 and ends in G_n resulting in a monomorphism $m' : H \hookrightarrow G_n$, written $m \overset{\pi}{\rightsquigarrow} m'$, if one of the following cases applies.*

- *$\pi = G_0 = G_n$ and $m = m'$.*
- *$\pi = G_0 \cdot (\delta, l : IG \hookrightarrow G_0, r : IG \hookrightarrow G_1) \cdot \pi'$ and there is $m'' : H \hookrightarrow IG$ such that $m = l \circ m''$ and $r \circ m'' \overset{\pi'}{\rightsquigarrow} m'$.*

$$\begin{tikzcd}
G_0 & IG \arrow[l, hook', "l"'] \arrow[r, hook, "r"] & G_1 \\
& H \arrow[ul, hook, bend left, "m"] \arrow[u, hook, "m''"'] \arrow[ur, hook', bend right, "m'"'] \arrow[ul, phantom, "="] \arrow[ur, phantom, "="] &
\end{tikzcd}$$

The fact that the step does not remove elements that are matched by a monomorphism m is obtained from the existence of a monomorphism m'' making the triangle $m = l \circ m''$ commute. The required renaming is then performed by replacing the match m by $r \circ m''$. The monomorphism m'' is uniquely defined when it exists.

Based on the preservation of matches, we now define the semantics for TGSs.

Definition 6 (Satisfaction of MTGCs by TGSs) *A given MTGC $\psi \in \Phi_H^{\mathrm{MTGC}}$ is* satisfied *by a TGS π, an observation timepoint $t \in \mathbf{R}_0^+$ (where $0 \leq t \leq \mathrm{dur}(\pi)$), and a monomorphism $m : H \hookrightarrow \pi_t$, written $(\pi, t, m) \models_{\mathrm{TGS}} \psi$, if one of the following cases applies.*

- *$\psi = \wedge S$ and $(\pi, t, m) \models_{\mathrm{TGS}} \phi$ for each $\phi \in S$.*
- *$\psi = \neg\phi$ and not $(\pi, t, m) \models_{\mathrm{TGS}} \phi$.*
- *$\psi = \exists(a : H \hookrightarrow H', \phi)$ and there is some $q : H' \hookrightarrow \pi_t$ such that $q \circ a = m$ and $(\pi, t, q) \models_{\mathrm{TGS}} \phi$.*
- *$\psi = \phi_1 \mathrm{U}_I \phi_2$ and there is some $t' \in I$ such that*
  ● there is $m' : H \hookrightarrow \pi_{t+t'}$ s.t. $m \overset{\pi_{[t,t+t']}}{\rightsquigarrow} m'$ and $(\pi, t+t', m') \models_{\mathrm{TGS}} \phi_2$ and
  ● for every $t'' \in [0, t')$ it holds that there is an $m'' : H \hookrightarrow \pi_{t+t''}$ such that $m \overset{\pi_{[t,t+t'']}}{\rightsquigarrow} m''$ and $(\pi, t+t'', m'') \models_{\mathrm{TGS}} \phi_1$.

An MTGC ψ over the empty graph is satisfied by a TGS π, written $\pi \models_{\mathrm{TGS}} \psi$, if $(\pi, 0, \mathrm{i}_{\pi_0}) \models_{\mathrm{TGS}} \psi$ where $\mathrm{i}_{\pi_0} : \emptyset \hookrightarrow \pi_0$ is the initial morphism to the graph at timepoint 0 of π (i.e., the first graph of π).

This semantics is similar to the semantics of GCs for *conjunction*, *negation*, and the *exists* operator since for the triple (π, t, m) it always holds that the codomain of m is the graph π_t and since the checked MTGC is defined for the domain of m. As for GCs, we omit here a more formal and technical handling for variable assignments. The TGS π and the current timepoint t are used in the case for the *until* operator where we rely on the *preserved match* relation from above to change the codomain of a match from π_t to the graphs $\pi_{t+t'}$ and $\pi_{t+t''}$ at later timepoints.

Example 1 (TGS satisfies MTGC) *Considering our running example, we argue that the MTGC given in Fig. 4.2 is satisfied by the TGS given in Fig. 4.1. Firstly, the* forall-new *operator matches the nodes T, S and the edge e_1 in G_2 at timepoint 10, which is the maximal creation timepoint of these three elements. Then, the* exists *operator matches the node H together with the edge e_2 in G_2 at the same timepoint. Finally, the* until *operator matches subsequently the node R and the edge e_3 in G_3 at the timepoint 13 and the remainder true is trivially satisfied for the timepoint 13. In addition, as also required by the* until *operator, for every timepoint in the interval $[10, 13)$ it is not possible to match a second Handler node H' that is connected to S. This holds because the graph in π for the timepoints in this interval is the graph G_2, which indeed does not contain such a second Handler node.*

5. Mapping of TGSs to Graphs with History

Subsequently, we are concerned with finite TGSs π (which have a finite number of steps and therefore also satisfy $\mathrm{dur}(\pi) < \infty$) for which the satisfaction of an MTGC ψ is decidable [4] when replacing in ψ right-open intervals $[r, \infty)$ and (r, ∞) by $[r, \mathrm{dur}(\pi))$ and $(r, \mathrm{dur}(\pi))$, respectively. Such an adaptation of intervals leads to an MTGC ψ' that is *bounded* and for which the satisfaction by the finite TGS π is equivalent (i.e., $\pi \models_{\mathrm{TGS}} \psi \iff \pi \models_{\mathrm{TGS}} \psi'$).

To analyze the satisfaction of an MTGC by a given finite TGS, we now introduce the notion of *graphs with history* (in short, GHs) as an equivalent representation of a given finite TGS. Afterwards, we introduce a semantics operating on this alternative representation (called in the following *semantics for GHs*) that is compatible with the semantics introduced before for TGSs. The translation from finite TGSs to GHs reduces the size of the representation in terms of the stored data. Moreover, it decouples the observation of modifications, resulting in a GH, and the subsequent satisfaction check for possibly several MTGCs.

The notion of GHs for capturing the changes to a current graph over time as given by a TGS π, requires 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.[1]

Definition 7 (Graphs with History (GHs)) *Let TG be a type graph where all nodes and edges have attributes* cts *denoting the timepoint of their creation and* dts *denoting the timepoint of their deletion. Then G_H is a* graph with history (GH) *if it is typed over TG satisfying the following consistency requirements.*[2]

- *There is precisely one* cts *attribute for every graph node and edge.*
- *There is at most one* dts *attribute for every graph node and edge.*
- *For an edge e, the value of the* cts *attributes of the source and the target nodes of e are less or equal to the* cts *attribute of e.*
- *For an edge e, the value of the* dts *attributes of the source and the target nodes of e are greater or equal to the* dts *attribute of e.*

[1]The total timepoints of additions and removals of attributes and their values can be encoded by moving attributes into separate nodes, for which their cts and dts attributes then encode the relevant timepoints.

[2]Note that the consistency requirements used in this definition are not guaranteed by the formalisms of E-Graphs or symbolic graphs.

We now define the operation $\mathcal{F}\mathrm{old}$, which converts a finite TGS π (i.e., a TGS with a finite number of steps) into the corresponding GH G_H. This recursive operation handles the renaming given by the monos l and r in the steps of π and, moreover, encodes the insertion of additional nodes/edges α by adding attributes $\mathrm{cts} = t$ for these nodes/edges in the constructed G_H and by equipping removed nodes/edges α with an additional attribute $\mathrm{dts} = t$ where t is the current total time of the considered TGS π in both cases.

Definition 8 (Map TGS to GH (Operation $\mathcal{F}\mathrm{old}$)) *We define the operation $\mathcal{F}\mathrm{old}$ recursively as follows.*

- *If $\pi = G_{init}$, then $G_H = \mathcal{F}\mathrm{old}(\pi)$ is obtained from G_{init} by adding the attributes $\mathrm{cts}(\alpha) = 0$ to each node or edge α in G_{init}.*
- *If $\pi = \pi' \cdot (\delta, l : IG \hookrightarrow G, r : IG \hookrightarrow G') \cdot G'$ is a TGS, $G'_H = \mathcal{F}\mathrm{old}(\pi')$ is the GH obtained from the mapping of the TGS π' using the operation $\mathcal{F}\mathrm{old}$, and $t = \mathrm{dur}(\pi')$ is the total time of G'_H, then $G_H = \mathcal{F}\mathrm{old}(\pi)$ is constructed from G'_H by adding the attributes $\mathrm{dts}(\alpha) = t + \delta$ to each node or edge $\alpha \in G - l(IG)$, by renaming each node and edge $\alpha \in l(IG)$ according to l, by adding each node and edge $\alpha \in G' - r(IG)$, by renaming each node and edge $\alpha \in r(IG)$ according to r, and by adding the attributes $\mathrm{cts}(\alpha) = t + \delta$ to each node or edge $\alpha \in G' - r(IG)$.*

The following example covers an application of $\mathcal{F}\mathrm{old}$ to a finite TGS.

Example 2 (Map TGS to GH) *We map the finite TGS π from Fig. 4.1 to the GH G_H shown in Fig. 5.1 using the operation $\mathcal{F}\mathrm{old}$ as follows. Since π starts with an empty graph G_0, we first map it into the empty GH. The second state of π given by G_1 including the System node S is added to the TGS after 5 timeunits. We map this TGS state to the GH by adding S to the empty GH and by, additionally, equipping this node with the creation timepoint $\mathrm{cts} = 5$. After another 5 timeunits, an additional Task node T, a Handler node H, and edges e_1, e_2 between the existing System node S and the new Task node T resp. the new Handler node H are added to the TGS resulting in the TGS state G_2. These changes are again mapped to the GH by adding the Task node T, the Handler node H, and the edges e_1, e_2 to the current version of G_H as well as by additionally equipping them with the creation timepoints $\mathrm{cts} = 10$. In a similar manner the Result node R together with the edges e_3 and e_4 (see the TGS state G_3) are added to the GH with the creation timepoints $\mathrm{cts} = 13$. Finally, after 2 timeunits, the edge e_3 is deleted to obtain the TGS state G_4. To reflect this in the GH, we add to the edge e_3 in G_H the additional deletion timepoint $\mathrm{dts} = 15$.*

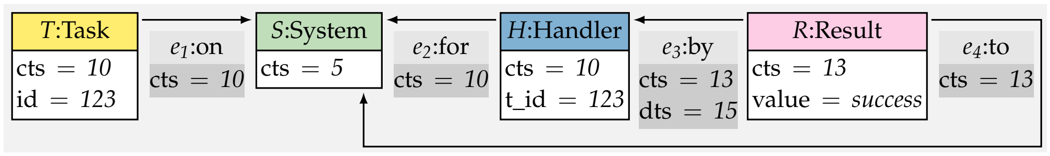


Figure 5.1.: Mapping of the TGS π from Fig. 4.1 to the GH $G_H = \mathcal{F}\mathrm{old}(\pi)$

For the satisfaction of an MTGC of the form $\exists(a : H \hookrightarrow H', \phi)$, where the *exists* operator is inherited from GCs, it is still required that the pattern that is found so far (given by some monomorphism $m : G \hookrightarrow G_H$) in the host graph G_H can be extended to a larger pattern (given by some monomorphism $m' : G' \hookrightarrow G_H$). 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 timepoints). 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 9 (Satisfaction of MTGCs by GHs) *An MTGC $\psi \in \Phi_H^{\mathrm{MTGC}}$ is satisfied by a monomorphism $m : H \hookrightarrow G_H$ and an observation timepoint $t \in \mathbf{R}_0^+$, written $(m,t) \models_{\mathrm{GH}} \psi$, if $\max(\{0\} \cup \mathrm{cts}(m(H))) \leq t < \min(\{\infty\} \cup \mathrm{dts}(m(H)))$ and one of the following cases applies.*

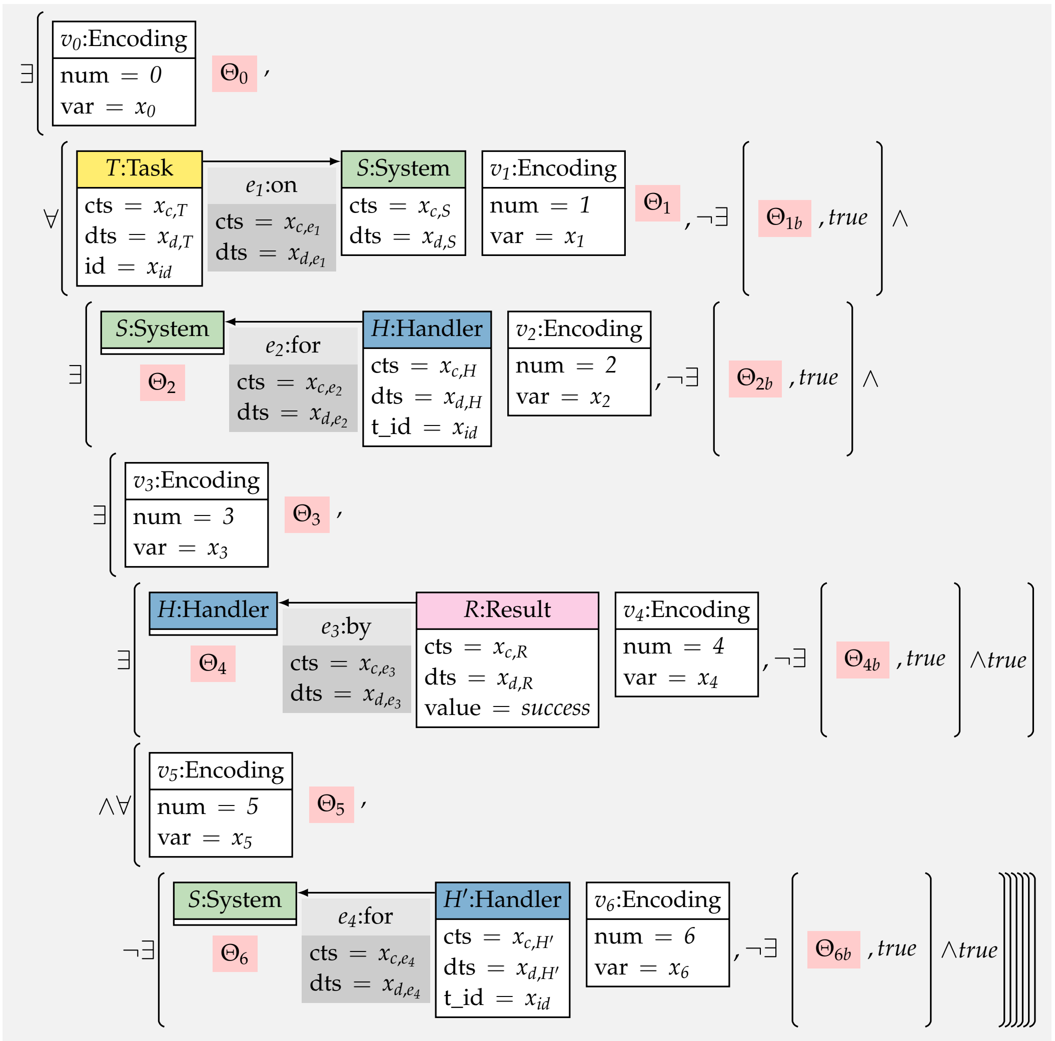
- *$\psi = \wedge\{\phi_1, \dots, \phi_n\}$ and $(m,t) \models_{\mathrm{GH}} \phi_i$ (for all $1 \leq i \leq n$).*
- *$\psi = \neg\phi$ and not $(m,t) \models_{\mathrm{GH}} \phi$.*
- *$\psi = \exists(a : H \hookrightarrow H', \phi)$ and there is some $q : H' \hookrightarrow G_H$ such that $q \circ a = m$ and $(q,t) \models_{\mathrm{GH}} \phi$.*
- *$\psi = \phi_1 \mathrm{U}_I \phi_2$ and there is some $t' \in I$ such that $(m, t+t') \models_{\mathrm{GH}} \phi_2$ and for every $t'' \in [0, t')$ it holds that $(m, t+t'') \models_{\mathrm{GH}} \phi_1$.*

An MTGC ψ over the empty graph is satisfied by a GH G_H, written $G_H \models_{\mathrm{GH}} \psi$, if $(\mathrm{i}_{G_H}, 0) \models_{\mathrm{GH}} \psi$ where $\mathrm{i}_{G_H} : \emptyset \hookrightarrow G_H$ is the initial morphism to G_H.

Note that the reasoning for the satisfaction of the MTGC ψ from Fig. 4.2 by $G_H = \mathcal{F}\mathrm{old}(\pi)$ from Fig. 5.1 proceeds analogously to Ex. 1.

In the following theorem (see the appendix Appendix B for its proof), we state the compatibility of the two satisfaction relations for the case of finite TGSs showing that they can be used interchangeably to determine the satisfaction of an MTGC in this case.

Theorem 1 (Soundness of Operation $\mathcal{F}$old) *If $\pi \in \Pi_{fin}$ and $\psi \in \Phi_{\emptyset}^{\mathrm{MTGC}}$, then $\pi \models_{\mathrm{TGS}} \psi$ iff $\mathcal{F}\mathrm{old}(\pi) \models_{\mathrm{GH}} \psi$.*



$\Theta_0 = \{x_0 = 10\}$

$\Theta_1 = \Theta_0 \cup \{x_1 = x_0\}$

$\Theta_{1b} = \Theta_1 \cup \{\neg \mathrm{alive}(x_1, \{T, S, e_1\})\}$

$\Theta_2 = \Theta_1 \cup \{x_2 = x_1\}$

$\Theta_{2b} = \Theta_2 \cup \{\neg \mathrm{alive}(x_2, \{T, S, e_1, H, e_2\})\}$

$\Theta_3 = \Theta_2 \cup \{x_2 + 0 \le x_3, x_3 \le x_2 + 120\}$

$\Theta_4 = \Theta_3 \cup \{x_4 = x_3\}$

$\Theta_{4b} = \Theta_4 \cup \{\neg \mathrm{alive}(x_4, \{T, S, e_1, H, e_2, R, e_3\})\}$

$\Theta_5 = \Theta_3 \cup \{x_2 \le x_5, x_5 < x_3\}$

$\Theta_6 = \Theta_5 \cup \{x_6 = x_5\}$

$\Theta_{6b} = \Theta_6 \cup \{\neg \mathrm{alive}(x_6, \{T, S, e_1, H, e_2, H', e_4\})\}$

Figure 5.2.: The GC ψ' resulting from applying the operation $\mathcal{R}$educe to the timepoint $t = 10$, and the MTGC ψ from Fig. 4.2 (where the outermost *forall-new* operator has been simplified to the *forall* operator)

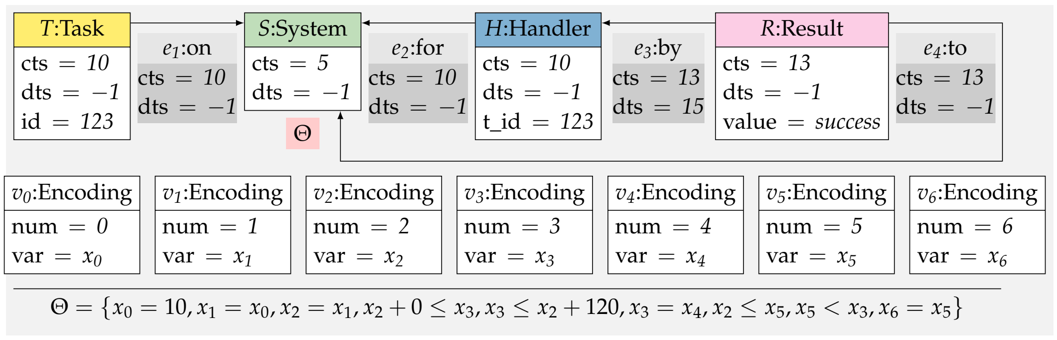


Figure 5.3.: The adapted graph G'_H resulting from applying the operation $\mathcal{R}$educe to the GH from Fig. 5.1, the timepoint $t = 10$, and the MTGC ψ from Fig. 4.2 (where the outermost *forall-new* operator has been simplified to the *forall* operator)

6. Reduction of MTGL to GCs

We now introduce a procedure for checking the satisfaction of an MTGC by a GH using a reduction of an MTGC to a corresponding GC. Based on the $\mathcal{F}\mathrm{old}$ operation from the previous section, we thereby obtain a checking procedure for finite TGSs as well. Moreover, this reduction shows that MTGL is as expressive as the logic of GCs on finite TGSs (since every GC is trivially also an MTGC).

We first present the operation $\mathcal{R}\mathrm{educe}$ for translating an MTGC into the corresponding GC and then show that this translation (also called *reduction* in the following) is compatible with our semantics for GHs and the operation $\mathcal{F}\mathrm{old}$ from before. The operation $\mathcal{R}\mathrm{educe}$ encodes in the resulting GC all parts of the satisfaction relation $\models_{\mathrm{GH}}$ that are not covered by the satisfaction relation $\models$ for GCs. In particular, the operation $\mathcal{R}\mathrm{educe}$ removes all occurrences of the *until* operator and encodes the check that the elements that are matched by the *exists* operator have all been created as well as that none of them has yet been deleted.

Technically, we translate a GH $G_H = \mathcal{F}\mathrm{old}(\pi)$ for a finite TGS π, $\psi \in \Phi_{\emptyset}^{\mathrm{MTGC}}$, and an observation timepoint $t \in \mathbf{R}_0^+$ (where G_H and ψ are typed over a type graph TG) into a graph G'_H and $\psi' \in \Phi_{\emptyset}^{\mathrm{GC}}$ (where both are typed over a changed type graph TG') using the procedure presented in Def. 10. We obtain ψ' from ψ by encoding the *until* operator suitably and by implementing the checks of cts and dts attributes according to Def. 9 for the *exists* and *until* operators using attribute constraints, for which we add variables to ψ. We also add the same variables to G_H to obtain G'_H.

Definition 10 (Reduce MTGC to GC (Operation $\mathcal{R}\mathrm{educe}$)) *The recursive operation* $\mathcal{R}\mathrm{educe}$ *takes 3 arguments: a GH G_H that has been obtained by application of the operation* $\mathcal{F}\mathrm{old}$ *to a TGS π, an observation timepoint $t \in \mathbf{R}_0^+$, and an MTGC $\psi \in \Phi_{\emptyset}^{\mathrm{MTGC}}$. G_H and all graphs contained in ψ are typed over the type graph TG.*

The operation $\mathcal{R}\mathrm{educe}$ *returns a pair (G'_H, ψ') consisting of a graph G'_H (which is a slight modification of G_H) and a GC $\psi' \in \Phi_{\emptyset}^{\mathrm{GC}}$. The graph G'_H and all graphs contained in ψ' are typed over an adapted type graph TG' (called a* reduction type graph*) introduced below.*

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

2. *(Construction of the MTGC ψ_{att} with* cts *and* dts *attributes):*
   We obtain ψ_{att} from ψ by adding the attributes $\mathrm{cts} = x_{c,\alpha}$ *and* $\mathrm{dts} = x_{d,\alpha}$ *to all nodes and edges α contained in graphs in ψ.*

3. *(Construction of the GC ψ'):*
   $\psi' = \exists(\mathrm{i}_{G_0}, \mathcal{R}\mathrm{educe}_{\mathrm{rec}}(\psi_{att}, x_0, G_0, \emptyset))$ *where G_0 is the graph containing the Encoding node v_0 with the attributes* num $=$ *0,* var $= x_0$ *as well as the attribute constraint $x_0 = t$ and $\mathrm{i}_{G_0} : \emptyset \hookrightarrow G_0$ is the initial morphism to G_0. Then, $\mathcal{R}\mathrm{educe}_{\mathrm{rec}}(\psi_{att}, x_o, G_a, G) = \psi'_{att}$ if one of the following cases applies (where ψ_{att} is the condition to be reduced, x_o is the timepoint at which the subcondition must be satisfied, G_a is the graph containing additional nodes, edges, and attribute constraints to be added to the graphs in conditions constructed, and G is the graph over which the condition ψ_{att} is defined).*
   a) $\psi_{att} = \wedge S$ *and* $\psi'_{att} = \wedge\{\mathcal{R}\mathrm{educe}_{\mathrm{rec}}(\phi, x_o, G_a, G) \mid \phi \in S\}$.
   b) $\psi_{att} = \neg\phi$ *and* $\psi'_{att} = \neg\,\mathcal{R}\mathrm{educe}_{\mathrm{rec}}(\phi, x_o, G_a, G)$.
   c) $\psi_{att} = \exists(a : H_1 \hookrightarrow H_2, \phi)$ *and* $\psi'_{att} = \exists(a' : H'_1 \hookrightarrow H'_2, \neg\exists(m : H'_2 \hookrightarrow H'_3, true) \wedge \mathcal{R}\mathrm{educe}_{\mathrm{rec}}(\phi, x_n, G'_a, H'_2))$ *where G'_a equals the graph G_a, to which an Encoding node v_n with the attributes* num $= n$, var $= x_n$ *(where no Encoding node has been created in the reduction for n so far) and the attribute constraint $x_n = x_o$ have been added, $H'_1 = G_a \cup H_1$, $H'_2 = G'_a \cup H_2$, H'_3 equals the graph H'_2, to which the attribute constraints $\neg\,\mathrm{alive}(x_n, H_2)$ have been added,*[1] *a' is obtained as the union of a and the identity morphism id_{G_a}, and m is an inclusion.*
   d) $\psi_{att} = \phi_1 \ \mathrm{U}_I \ \phi_2$ *and* $\psi'_{att} = \exists(m_0 : G_0 \hookrightarrow G_1, \mathcal{R}\mathrm{educe}_{\mathrm{rec}}(\phi_2, x_{n_1}, G'_a, G_1) \wedge \forall(m_1 : G_1 \hookrightarrow G_2, \mathcal{R}\mathrm{educe}_{\mathrm{rec}}(\phi_1, x_{n_2}, G''_a, G_2)))$ *where G'_a equals the graph G_a, to which an Encoding node v_{n_1} with the attributes* num $= n_1$, var $= x_{n_1}$ *(where no Encoding node has been created in the reduction for n_1 so far) and the attribute constraints equivalent to $x_{n_1} \in I$ have been added, $G_0 = G \cup G_a$, $G_1 = G \cup G'_a$, m_0 is an inclusion, G''_a equals the graph G'_a, to which an Encoding node v_{n_2} with the attributes* num $= n_2$, var $= x_{n_2}$ *(where no Encoding node has been created in the reduction for n_2 so far) and the attribute constraints equivalent to $x_{n_2} \in [x_o, x_o + x_{n_1})$ have been added, $G_2 = G_1 \cup G''_a$, and m_1 is an inclusion.*
4. *(Construction of the graph G'_H):*
   We obtain G'_H by adding elements to G_H as follows:
   a) *We add the attribute* dts $= -1$ *to all nodes/edges without that attribute.*
   b) *We insert all Encoding nodes contained in graphs in ψ' together with their* num $= n$ *and* var $= x_n$ *attributes.*
   c) *We add the attribute constraints added during the reduction except for the* alive *constraints.*

We now demonstrate how the operation $\mathcal{R}$educe can be applied to the MTGC from our running example.

[1] For a graph H, $\mathrm{alive}(x, H)$ equals $\mathrm{alive}(x, S)$ for the disjoint union S of the nodes and edges of H. For a set S of nodes and edges, $\mathrm{alive}(x, S)$ equals $\cup\{\mathrm{alive}(x, \alpha) \mid \alpha \in S\}$. For a node or an edge α, $\mathrm{alive}(x, \alpha)$ equals $\{x_{c,\alpha} \leq x,\ x_{d,\alpha} = -1 \vee x < x_{d,\alpha}\}$.

Example 3 (Reduce MTGC to GC) *We now apply the $\mathcal{R}$educe operation to GH from Fig. 5.1, the timepoint $t = 10$, and the MTGC ψ from Fig. 4.2 resulting in G'_H and ψ' given in Fig. 5.3 and Fig. 5.2, respectively. However, to simplify the presentation, we replaced the enclosing* forall-new *operator by the* forall *operator. A formal handling of the* forall-new *operator is left for future work.*

1. *We add the attribute* dts $= x_{d,ff}$ *to all nodes/edges α of G_H without* dts *attribute and add the attribute constraint $x_{d,\alpha} = -1$ to the set of constraints. With these additional attributes and the* cts $= x_{c,ff}$ *attributes introduced by the operation $\mathcal{F}$old, we are able to state the existence of nodes/edges at a given timepoint x_n using attribute constraints in the resulting GC ψ'.*

2. *We add a unique Encoding node to each graph in ψ' as a container for additional variables x_n that are used in attribute constraints to encode the current observation timepoint (the num attributes are included to decrease the number of matches to be considered). Initially, we add an enclosing* exists *operator with the attribute constraint $x_0 = t$ (see Θ_0) where t is the input observation timepoint that is 10 for this application of $\mathcal{R}$educe. Further attribute constraints then relate the additional variables x_n for existential/universal quantifications (see Θ_1, Θ_2, Θ_4, and Θ_6). For the encoding of the* until *operator, these observation timepoints (x_3 in Θ_3 and x_5 in Θ_5) are restricted to some interval as described below.*

3. *We encode the* exists *operator $\exists(a : H_1 \hookrightarrow H_2, \phi)$ for the MTGC ϕ according to Def. 9 using an additional negative graph condition stating that the matched nodes/edges α are not violating the attribute constraints in* $\text{alive}(x_n, \alpha)$. *The set* $\text{alive}(x_n, \alpha)$ *contains the constraint $x_n \leq x_{c,\alpha}$ (to state that α was created before x_n) and the constraint $x_{d,\alpha} = -1 \vee x_n < x_{d,\alpha}$ (to state that α was not deleted or that it is deleted later than x_n).*

4. *We encode the* until *operator $\phi_1 \; \mathrm{U}_I \; \phi_2$ for the MTGCs ϕ_1 and ϕ_2 according to Def. 9 using the* exists *operator (the* forall *operator used in the GC below is only an abbreviation for a usage of the* exists *operator according to Def. 1). Informally, $\phi_1 \; \mathrm{U}_{[t_1,t_2]} \; \phi_2$ (the construction is similar for other kinds of intervals) is equivalent to $\exists(t' \in [x_n + t_1, x_n + t_2], \phi'_2 \wedge \forall(t'' \in [x_n + t_1, t'), \phi'_1))$ where ϕ'_1 and ϕ'_2 are the reductions of ϕ_1 and ϕ_2, respectively. The variable x_n refers to the current observation timepoint that depends on the timepoint where an enclosing condition has been matched. In the example, the variables x_n, t', and t'' are represented in ψ' by the variables x_2, x_3, and x_5, respectively. The reduction is recursively applied to ϕ_1 and ϕ_2 resulting in ϕ'_1 and ϕ'_2, respectively. The replacement GC for the* until *subcondition spans the last four lines of ψ' in Fig. 5.2.*

5. *We add all Encoding nodes occurring in ψ' to G_H as depicted in Fig. 5.3. The Encoding nodes are used in ψ' as containers for the additional variables employed in the attribute constraints and are required in G'_H to allow for matchings from the adapted graphs of ψ' to G'_H.*

In the following theorem (see the appendix Appendix B for its proof), we state that the operation $\mathcal{R}$educe is sound w.r.t. the satisfaction relations for MTGCs and GCs.

Theorem 2 (Soundness of Operation $\mathcal{R}$educe) *If $\pi \in \Pi_{fin}$, $G_H = \mathcal{F}\text{old}(\pi)$, $\psi \in \Phi_{\emptyset}^{\text{MTGC}}$, $t \in \mathbf{R}_0^+$ is a timepoint, $\text{i}_{G_H} : \emptyset \hookrightarrow G_H$ is the initial morphism to G_H, and $(G'_H, \psi') = \mathcal{R}\text{educe}(G_H, t, \psi)$, then $(\text{i}_{G_H}, t) \models_{\text{GH}} \psi$ iff $G'_H \models \psi'$.*

By application of Thm. 2, we can deduce for our running example that the MTGC ψ from Fig. 4.2 translated by the operation $\mathcal{R}$educe is satisfied by the graph G'_H (given in Fig. 5.2 and Fig. 5.3). For this purpose observe that ψ from Fig. 4.2 (simplified as stated in Fig. 5.3 and Fig. 5.2) is satisfied by the GH from Fig. 5.1 for the timepoint $t = 10$ since the unique match of the Task node T, the on edge e_1, and the System node S satisfies the remaining condition starting at timepoint $t = 10$.

7. Tool Support

We provide tool support for checking finite TGSs against MTGCs as an extension of AutoGraph [26]. Firstly, we extended the support of AutoGraph to handle TGSs and MTGCs. Secondly, we implemented the operation $\mathcal{F}$old from Def. 8 to consolidate a TGS π to a GH G_H. Thirdly, we implemented the operation $\mathcal{R}$educe from Def. 10 to reduce an MTGC ψ to a GC ψ' and to adapt G_H to a graph G'_H. On the foundation of these three steps and as applications of our theoretical results (see Thm. 1 and Thm. 2), we then use the built-in support of AutoGraph for checking whether the obtained graph G'_H satisfies the reduced GC ψ'. Note that AutoGraph depends in this scenario on the constraint solver Z3 [21] to check satisfiability of expressions involving the values of cts and dts attributes of sort real as well as the additional constraints introduced by $\mathcal{R}$educe that contain further variables of sort real.

Considering our running example, we observed negligible runtime and memory consumption when verifying that the finite TGS π from Fig. 4.1 satisfies the MTGC ψ from Fig. 4.2 using our implementation due to the short length of π. Overall, the application of the AutoGraph extension *to our running example* shows promising results albeit the potential of further improvements regarding efficiency for handling more elaborate problem instances.

8. Conclusion and Future Work

We defined *Metric Temporal Graph Logic* (MTGL) by integrating the metric temporal operator *until* with time bounds into the well-established logic of (nested) graph conditions (GCs). This new logic allows to maintain an established binding of graph elements throughout the analysis of a timed sequence of (typed attributed) graphs (TGSs). Furthermore, to enable a satisfaction check for MTGL conditions by finite TGSs, we introduced a mapping of a finite TGS π into a graph with history $G_H = \mathcal{F}\text{old}(\pi)$ and defined a reduction of an MTGL condition ψ to a GC ψ' given by $(G_H, \psi') = \mathcal{R}\text{educe}(G_H, 0, \psi)$ where the graph with history G_H is extended to a graph G'_H. For this mapping and this reduction, we have proven that the satisfaction checks for the different representations are consistent (i.e., $\pi \models_{\text{TGS}} \psi \iff G_H \models_{\text{GH}} \psi \iff G'_H \models \psi'$). Finally, we presented an extension of the tool AutoGraph allowing to check the satisfaction of MTGL conditions by finite TGSs via the introduced mapping and reduction.

In the future, we want to formally handle the *forall-new* operator from our running example as well as the *since* operator from MTL. Based on the current paper, we want to develop checking procedures for bounded MTGL conditions such that only violations that hold for any possible continuation are reported. Moreover, we intend to use our reduction of MTGL conditions to related GC counterparts for invariant checking for graph transformation systems as considered in [7]. Furthermore, we want to develop extensions of MTGL that include branching such as in timed CTL, that are applicable to the setting of probabilistic timed graph transformation systems as introduced in [20], or that support additional features e.g. permitting variables in the interval bounds of MTGL conditions or in attribute constraints. Finally, we intend to develop a model checking procedure for MTGL and these extensions. Besides these technical advancements we intend to evaluate and compare our approach based on benchmarks from applications domains such as runtime monitoring [19].

References

[1] T. Arendt, E. Biermann, S. Jurack, C. Krause, and G. Taentzer. “Henshin: Advanced Concepts and Tools for In-Place EMF Model Transformations”. In: *Model Driven Engineering Languages and Systems – 13th International Conference, MODELS 2010, Oslo, Norway, October 3–8, 2010, Proceedings, Part I.* Edited by D. C. Petriu, N. Rouquette, and Ø. Haugen. Volume 6394. Lecture Notes in Computer Science. Springer, 2010, pages 121–135. ISBN: 978-3-642-16144-5. DOI: 10.1007/978-3-642-16145-2_9.

[2] P. Baldan, A. Corradini, and B. König. “A Framework for the Verification of Infinite-state Graph Transformation Systems”. In: *Inf. Comput.* 206.7 (2008), pages 869–907.

[3] 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.

[4] P. Bouyer, N. Markey, J. Ouaknine, and J. Worrell. “The Cost of Punctuality”. In: *22nd IEEE Symposium on Logic in Computer Science (LICS 2007), 10–12 July 2007, Wroclaw, Poland, Proceedings*. IEEE Computer Society, 2007, pages 109–120. ISBN: 0-7695-2908-9. DOI: 10.1109/LICS.2007.49.

[5] 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. ISBN: 981022-88-48.

[6] I. Dávid, I. Ráth, and D. Varró. “Foundations for Streaming Model Transformations by Complex Event Processing”. In: *Software and System Modeling* 17.1 (2018), pages 135–162. DOI: 10.1007/s10270-016-0533-1.

[7] J. Dyck and H. Giese. “K-Inductive Invariant Checking for Graph Transformation Systems”. In: *ICGT*. Edited by J. de Lara and D. Plump. Volume 10373. Lecture Notes in Computer Science. Springer, 2017, pages 142–158. ISBN: 978-3-319-61469-4. DOI: 10.1007/978-3-319-61470-0_9.

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

[9] *EMF Henshin*. *The Eclipse Foundation*. 2013.

[10] A. H. Ghamarian, M. de Mol, A. Rensink, E. Zambon, and M. Zimakova. “Modelling and analysis using GROOVE”. In: *STTT* 14.1 (2012), pages 15–40. DOI: 10.1007/s10009-011-0186-x.

[11] *Graphs for Object-Oriented Verification (GROOVE)*. University of Twente. 2011.

[12] 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.

[13] E. Jakumeit, S. Buchwald, D. Wagelaar, L. Dan, Á. Hegedüs, M. Herrmannsdörfer, T. Horn, E. Kalnina, C. Krause, K. Lano, M. Lepper, A. Rensink, L. M. Rose, S. Wätzoldt, and S. Mazanek. "A survey and comparison of transformation tools based on the transformation tool contest". In: *Sci. Comput. Program.* 85 (2014), pages 41–99. DOI: 10.1016/j.scico.2013.10.009.

[14] 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. ISBN: 978-3-540-71288-6. DOI: 10.1007/978-3-540-71289-3_16.

[15] B. König and V. Kozioura. "Augur 2 – A New Version of a Tool for the Analysis of Graph Transformation Systems". In: *ENTCS* 211 (2008), pages 201–210. DOI: 10.1016/j.entcs.2008.04.042.

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

[17] M. Kwiatkowska, G. Norman, and D. Parker. "PRISM 4.0: Verification of Probabilistic Real-time Systems". In: *Proc. CAV'11*. LNCS 6806. DOI: 10.1007/978-3-642-22110-1_47. Springer, 2011, pages 585–591.

[18] J. de Lara and D. Plump, editors. *Graph Transformation – 10th International Conference, ICGT 2017, Held as Part of STAF 2017, Marburg, Germany, July 18–19, 2017, Proceedings*. Volume 10373. Lecture Notes in Computer Science. Springer, 2017. ISBN: 978-3-319-61469-4. DOI: 10.1007/978-3-319-61470-0.

[19] M. Leucker and C. Schallhart. "A brief account of runtime verification". In: *J. Log. Algebr. Program.* 78.5 (2009), pages 293–303. DOI: 10.1016/j.jlap.2008.08.004.

[20] 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*. Edited by J. de Lara and D. Plump. Volume 10373. Lecture Notes in Computer Science. Springer, 2017, pages 159–175. ISBN: 978-3-319-61469-4. DOI: 10.1007/978-3-319-61470-0_10.

[21] Microsoft Corporation. *Z3*. https://github.com/Z3Prover/z3. Accessed: 2017-09-19.

[22] 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.

[23] 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. ISBN: 978-3-540-85777-8. DOI: 10.1007/978-3-540-85778-5_1.

[24] K.-H. Pennemann. *Development of Correct Graph Transformation Systems, PhD Thesis*. Dept. Informatik, Univ. Oldenburg, 2009.

[25] 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.

[26] S. Schneider, L. Lambers, and F. Orejas. "Automated reasoning for attributed graph properties". In: *STTT* 20.6 (2018), pages 705–737. DOI: 10.1007/s10009-018-0496-3.

A. Encoding of Attribute Modifications

Example 4 (Encoding of Attribute Modifications) *The modification of the attributes (see Fig. A.1) of the node v_A is encoded in additional nodes v_A^{A1} and v_A^{A2}. Thereby the attribute* attValueA *has the value* 24 *in the time interval* $[10, 16)$ *and the value* 30 *in the time interval* $[16, 64)$. *The attribute of the edge e_A is encoded in the additional node e_A^A that is identified using* num $= 1$ *and the connections to the nodes v_A and v_B. Note that we omit in this example the additional four edge types used for the encoding.*

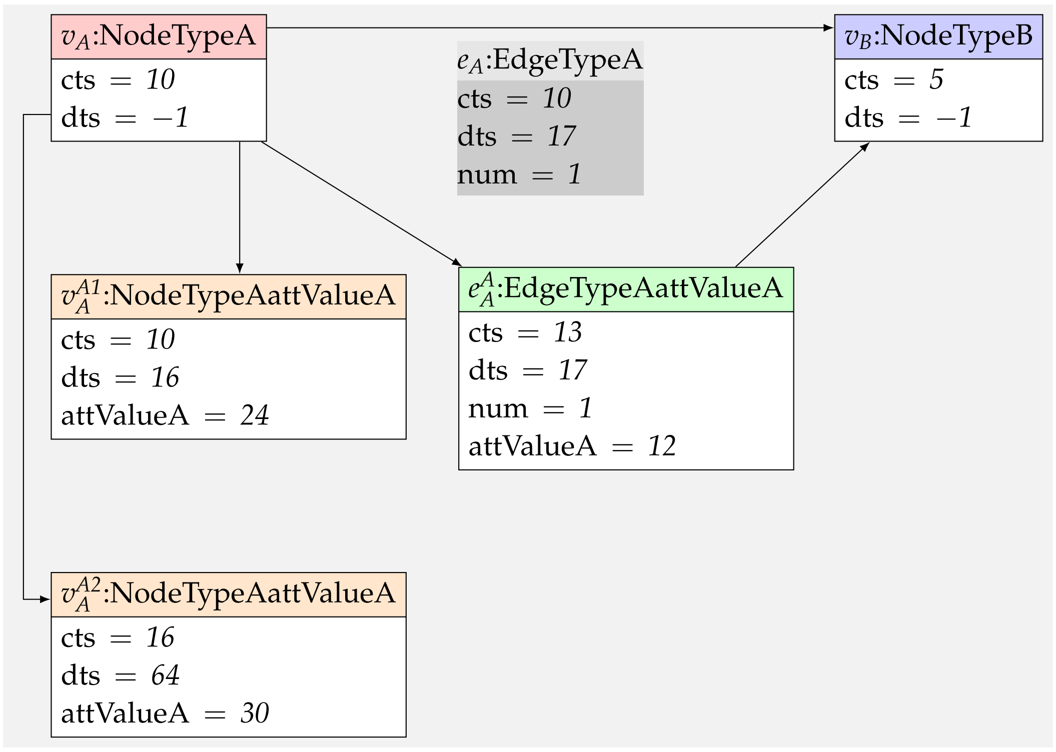


Figure A.1.: Encoding of attribute modifications (see Ex. 4)

B. Proofs

We provide proofs for the theorems contained in the main body of this report.

Lemma 1 (Soundness of Operation $\mathcal{F}\text{old}$) *If $\pi \in \Pi_{fin}$ of length n, $\psi \in \Phi_H^{\text{MTGC}}$, $t \in \{\sum_{i \leq m} \text{delta}(\pi, i) \mid m \leq n\}$ where $\text{delta}(\pi, i)$ is the delay of the ith step in π, and $f_t : \pi_t \hookrightarrow \mathcal{F}\text{old}(\pi_{[0,t]})$ is the obvious inclusion induced by $\mathcal{F}\text{old}$, then $(\pi, t, m : H \hookrightarrow \pi_t) \models_{\text{TGS}} \psi$ iff $(f_t \circ m, t) \models_{\text{GH}} \psi$*

Proof 1 (for Lemma 1) *By induction on ψ.*

The proof is straight-forward for conjunction and negation.

For existential quantification, we obtain the morphism q required for satisfaction of ψ by π (see Def. 6) from Def. 9 and, vice versa, the morphism q required for satisfaction of ψ by $\mathcal{F}\text{old}(\pi)$ (see Def. 9) is induced by Def. 6. This is possible since the definition of $\mathcal{F}\text{old}$ ensures that elements exist in π_t iff their cts *attributes are lower than the timepoint t and their* dts *attributes (if existent) are strictly higher than t.*

For the until *operator, we apply induction on the finite set of graphs corresponding to the possible values of t. Then, the satisfaction of the less deeply nested subconditions ϕ_1 and ϕ_2 of $\phi_1 \text{ U}_I\ \phi_2$ is preserved and reflected between the two semantics due to the induction hypothesis. The central step is then that the* preserved match *relation (over k steps of π) is compatible with applying k times the single-step adaption used in the $\mathcal{F}\text{old}$ operation of the current GH for the same steps. This compatibility ensures that the matches used for the satisfaction of the subconditions coincide in both cases.*

Proof 2 (for Thm. 1) *from Lemma 1*

Proof 3 (for Thm. 2) *Throughout the proof we refer to items given in Def. 10.*
(⇒): *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 can be matched to G'_H because the required Encoding node is present in G'_H due to Item 4b 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 4c.

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 $\text{i}_{G'_H}$, which is matching the Encoding node added in Item 4c) and the timepoint 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 $(m, t) \models_{\text{GH}} \exists(a : G_1 \hookrightarrow G_2, \phi)$ and show that $m' \models \exists(a' : G'_1 \hookrightarrow G'_2, \phi')$.*

Due to the assumption and by Def. 9 there is some $q : G_2 \hookrightarrow G_H$ such that $q \circ a = m$ and $(q, t) \models_{\mathrm{GH}} \phi$, which already implies $\max(\{0\} \cup \mathrm{cts}(q(G_2))) \leq t < \min(\{\infty\} \cup \mathrm{dts}(q(G_2)))$. This monomorphism q can be used to extend m' to a monomorphism $q' : G_2' \hookrightarrow G_H'$. 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. Hence, x_n encodes the timepoint that is also used for the MTGL satisfaction statement $(q, t) \models_{\mathrm{GH}} \phi$ from above. Moreover, the attribute constraints added for all nodes and edges encodes the statement $\max(\{0\} \cup \mathrm{cts}(q(G_2))) \leq t < \min(\{\infty\} \cup \mathrm{dts}(q(G_2)))$ *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 and, additionally, the graph condition can still be matched using q' to G_H' because G_H has been adapted to also contain all* dts *attributes. The subconstraint checking for -1 is then required to only consider the actually deleted nodes and edges. We make use of the negated subcondition containing the negated alive constraints to state that it is not the case that the nodes matched are not alive; this is required due to the implication check for the attribute constraints required for q to be a well-formed morphism on symbolic graphs. Finally, the translated condition ϕ' is then also satisfied by the induction hypothesis.*

- **(until *operator*):** *We assume that $(m, t) \models_{\mathrm{GH}} \phi_1 \ \mathrm{U}_I \ \phi_2$ and show that $m' \models \exists(m_1 : G \hookrightarrow G', \wedge\phi_2', \forall(m_2 : G' \hookrightarrow G'', \phi_1'))$ for the result obtained by application of the operation* $\mathcal{R}$educe.

  Due to the assumption and by Def. 9 there is some $t' \in I$ such that $(m, t + t') \models_{\mathrm{GH}} \phi_2$ and for every $t'' \in [0, t')$ it holds that $(m, t + t'') \models_{\mathrm{GH}} \phi_1$. 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_1, in which we use the variable x_{n1} to encode the value of t' and the attribute constraints to restrict x_{n1} to the interval I. Then, $(m, t + t') \models_{\mathrm{GH}} \phi_2$ implies that the match m' can be extended to a match m'', as required by the existential quantification, because the variable x_{n1} along with its Encoding node can be matched to G_H'. The translated condition ϕ_2' is then also satisfied by the induction hypothesis. Moreover, the universally quantified t'' is then represented by the variable x_{n2} in G_2, which is also universally quantified. We assume that t'' is fixed in that interval satisfying $(m, t + t'') \models_{\mathrm{GH}} \phi_1$. Hence, the match m'' can be extended to a match m''' as required by matching x_{n2} to the corresponding variable in G_H'. Also, the attribute constraints on x_{n2} 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* until *must be matched entirely for the reverse direction to preserve the correspondence with the MTGC satisfaction proof state.*

ISBN 978-3-86956-463-0
ISSN 1613-5652