Fundamenta Informaticae ?? (2003) 1–20
IOS Press

Graph Transformation with Time: Causality and Logical Clocks

Szilvia Gyapay, Dániel Varró
Dept. of Measurement and Information Systems,
Budapest University of Technology and Economics
H-1521 Budapest, Hungary
{gyapay, varro}@mit.bme.hu

Reiko Heckel
Institute of Computer Science
University of Paderborn
D-33095 Paderborn, Germany
reiko@upb.de

Abstract. Following TER nets, an approach to the modelling of time in high-level Petri nets, we propose a model of time within (attributed) graph transformation systems where logical clocks are represented as distinguished node attributes. Corresponding axioms for the time model in TER nets are generalised to graph transformation systems and semantic variations are discussed. They are summarised by a general theorem ensuring the consistency of temporal order and casual dependencies.

The resulting notions of *typed graph transformation with time* specialise the algebraic double-pushout (DPO) approach to typed graph transformation. In particular, the concurrency theory of the DPO approach can be used in the transfer of the basic theory of TER nets.

1. Introduction

Recently, a number of authors have advocated the use of graph transformation as a semantic framework for visual modelling techniques both in computer science and engineering (see, e.g., the contributions in [4, 3]). In many such techniques, the modelling of time plays a relevant role. In particular, techniques for embedded and safety critical systems make heavy use of concepts like timeouts, timing constraints, delays, etc., and correctness with respect to these issues is critical to the successful operation of these systems. At the same time, those are exactly the systems where, due to the high penalty of failures, formally based modelling and verification techniques are most successful. Therefore, neglecting the time aspect in the semantics of visual modelling techniques, we disregard one of the crucial aspects of modelling.

So far, the theory of graph transformation provides no support for the modelling of time in a way which would allow for *quantified* statements like “this action takes 200ms of time” or “this message will only be accepted within the next three seconds”, etc. However, from a more abstract, *qualitative* point of view we can speak of temporal and causal ordering of actions thus abstracting from actual clock and timeout values. Particularly relevant in this context is the theory of concurrency of graph transformation, see [13, 6, 1] or [2] for a recent survey.

It is the objective of this paper to propose a quantitative model of time within graph transformation which builds on this more abstract qualitative model. Therefore, we will not add time concepts on top of an existing graph transformation approach, but we show how, in particular, typed graph transformation systems in the double-pushout (DPO) approach [6] can be extended *from within* with a notion of time. This allows both the straightforward transfer of theoretical results and the reuse of existing tools.

The idea is to use dedicated attributes of vertices as time stamps representing the ”‘age”’ of these vertices, and to update these time stamps whenever a rule is applied. To verify the consistency of this encoding with the causal dependencies between transformation steps, we prove the existence of a globally time-ordered sequence of transformations in every shift-equivalence class of sequences satisfying some local axioms. In [11, **?**] we have outlined our approach, proposing several alternative definitions and discussing their consequences with respect to the existence of a globally time-ordered sequences.

The following section outlines our general approach of the problem, which is motivated by a corresponding development in Petri nets, briefly to be reviewed in Section 3. Section 4 develops the basic formalism of typed attributed graph transformation while graph transformation with time is introduced and investigated in Section 5 while Section 7 concludes the paper.

2. From Nets to Graph Transformation, with Time

When trying to incorporate time concepts into graph transformation, it is inspiring to study the representation of time in Petri nets. Nets are formally and conceptually close to graph transformation systems which allows for the transfer of concepts and solutions. This has already happened for relevant parts of the concurrency theory of nets which, as mentioned above, provides a qualitative model of time based on the causal ordering of actions.

In particular, we will follow the approach of time ER nets [10]. These are simple high-level nets which introduce time as a distinguished data type. Then, time values can be associated with individual tokens, read and manipulated like other token attributes when firing transitions. In order to ensure meaningful behaviour (like preventing time from going backwards) constraints are imposed which can be checked for a given net. The advantage of this approach with respect to our aims is the fact that time is modelled within the formalism rather than adding it on top as a new formal concept.

Based on the correspondence of Petri nets and (typed) graph transformation, which regards Petri nets as rewriting systems on multi-sets of vertices [5], we can derive a model of time within typed graph transformation systems with attributes. The correspondence is visualised in Table 1. Besides (low-level) place-transition nets and typed graph transformation systems, it relates (high-level) environment-relationship nets to typed graph transformation with attributes. This relationship, which has first been observed in the case of algebraic high-level nets [7] and attributed graph transformation [16] in [17],

Table 1. Corresponding Petri net and graph transformation variants

	Petri nets	graph transformation systems
low-level	PT nets	typed graph transformation (TGT)
high-level	ER nets	typed graph transformation with attributes (TGTA)
with time	TER nets	typed graph transformation with time (TGTT)

shall enable us to transfer the modelling of time in time ER nets to typed graph transformation with attributes.

Next, we review time environment-relationship (TER) nets [10] in order to prepare for the transfer to typed graph transformation systems in Section 4.

3. Modelling Time in Petri Nets

There are many proposals for adding time to Petri nets. In this paper we concentrate on one of them, time ER nets [10], which is chosen for its general approach of considering time as a token attribute with particular behaviour, rather than as an entirely new concept. As a consequence, time ER nets are a special case of ER nets.

3.1. ER nets

ER (environment-relationship) nets are high-level Petri nets (with the usual net topology) where tokens are environments, i.e., partial functions $e : ID \to V$ associating attribute values from a given set V to attribute identifiers from a given set ID. A marking m is a multi-set of environments (tokens).

To each transition t of the net with pre-domain $p_1 \dots p_n$ and post-domain $p'_1 \dots p'_m$, an action $\alpha(t) \in Env^n \times Env^m$ is associated. The projection of $\alpha(t)$ to the pre-domain represents the firing condition, i.e., a predicate on the tokens in the given marking which controls the enabledness of the transition. If the transition is enabled, i.e., in the given marking m there exist tokens satisfying the predicate, the action relation determines possible successor markings.

Formally, a transition t is enabled in a marking m if there exists a tuple $\langle pre, post\rangle \in \alpha(t)$ such that $pre \leq m$ (in the sense of multiset inclusion). Fixing this tuple, the successor marking m' is computed, as usual, by $m' = (m - pre) + post$, and this firing step is denoted by $m[t(pre, post)\rangle m'$. A firing sequence of $s = m_0[t_1(pre_1, post_1)\rangle \dots [t_{k-1}(pre_{k-1}, post_{k-1})\rangle m_k$ is just a sequence of firing steps adjacent to each other.

3.2. Time ER nets

Time is integrated into ER nets by means of a special attribute, called chronos, representing the time of creation of the token as a time stamp. Constraints on the time stamps of both (i) given tokens and (ii) tokens that are produced can be specified by the action relation associated to transitions. To provide a meaningful model of time, action relations have to satisfy the following axioms with respect to chronos values [10].

Axiom 1: Local monotonicity For any firing, the time stamps of tokens produced by the firing can not be smaller than time stamps of tokens removed by the firing.

Axiom 2: Uniform time stamps For any firing $m[t(pre, post)\rangle m'$ all time stamps of tokens in $post$ have the same value, called the *time of the firing*.

Axiom 3: Firing sequence monotonicity For any firing sequence s, firing times should be monotonically nondecreasing with respect to their occurrence in s.

The first two axioms can be checked locally based on the action relationships of transitions. For the third axiom, it is shown in [10] that every sequence s where all steps satisfy Axioms 1 and 2 is *permutation equivalent* to a sequence s' where also Axiom 3 is valid. Here, permutation equivalence is the equivalence on firing sequences induced by swapping independent steps. Thus, any firing sequence can be viewed as denoting a representative, which satisfies Axiom 3.

It shall be observed that TER nets are a proper subset of ER nets, i.e., the formalism is not extended but specialised. Next, we use the correspondence between graph transformation and Petri nets to transfer this approach of adding time to typed graph transformation systems.

4. Typed Attributed Graph Transformation

Typed graph transformation systems provide a rich theory of concurrency generalising that of Petri nets [2]. In order to represent time as an attribute value, a notion of typed graph transformation with attributes is required. In this section, we propose an integration of the two concepts (types and attributes) which presents attribute values as vertices and attributes as edges.

The two basic ingredients are graphs, representing dynamic object structures, and algebras representing pre-defined abstract data types. Attributed graphs occur at two levels: the type level (modelling a schema or class diagram) and the instance level (modelling an individual system snapshot).

Attributed graphs. By a *graph* we mean a directed unlabelled graph $G = \langle G_V, G_E, src^G, tar^G\rangle$ with a set of vertices G_V, a set of edges G_E, and functions $src^G : G_E \to G_V$ and $tar^G : G_E \to G_V$ associating to each edge its source and target vertex. A graph homomorphism $f : G \to H$ is a pair of functions $\langle f_V : G_V \to H_V, f_E : G_E \to H_E\rangle$ preserving source and target.

To speak about algebras, throughout the paper we assume a many-sorted signature $\Sigma = \langle S, OP\rangle$ consisting of a set of sort symbols $s \in S$ and a family of sets of operation symbols $op : s_1 \dots s_n \to s \in OP$ indexed by their arities. An many sorted *algebra* $A = ((A_s)_{s\in S}, (op^A)_{op\in OP})$ consists of a family of carrier sets, indexed by sort symbols, and an operation $op^A : A_{s_1} \times \dots \times A_{s_n} \to s$ for each operation symbol $op : s_1 \dots s_n \to s \in OP$. A Σ-*homomorphism* $f_A : A_1 \to A_2$ is given as a family of mappings $f_A = (f_s)_{s\in S}$ compatible with the operation of A_1 and A_2.

Graphs and graph morphisms can be seen as algebras and homomorphisms for the signature with sorts E, V and operation symbols $src, tar : E \to V$.

Definition 4.1. (attributed graphs and morphisms)
An attributed graph (over Σ) is a pair $\langle G, A\rangle$ of a graph G and a Σ-algebra A such that $|A| \subseteq G_V$, where $|A| = \bigcup_{s\in S} A_s$ is the disjoint union of the carrier sets of A, and such that $\forall e \in G_E.\ src(e) \notin |A|$.

An *attributed graph morphism* $f : \langle G_1, A_1 \rangle \to \langle G_2, A_2 \rangle$ is a pair of a Σ-homomorphism $f_A = (f_s)_{s \in S} : A_1 \to A_2$ and a graph homomorphism $f_G = \langle f_V, f_E \rangle : G_1 \to G_2$ such that

- $|f_A| \subseteq f_V$, where $|f_A| = \bigcup_{s \in S} f_s$, and
- $f_A(A_1)$ and $f_V(G_{2V})$ are disjoint.

Attributed graphs and graph morphisms form the *category of* Σ*-attributed graphs*. Often, we will fix the data algebra A in advance—in this case we also speak of a graphs and graph morphisms attributed over A.

Summarizing, data values are represented as vertices of graphs, henceforth called *data vertices* $d \in |A|$ to distinguish them from *object vertices* $v \in G_V \setminus |A|$. Object vertices are linked to data vertices by *attributes*, i.e., edges $a \in G_E$ with $src(a) = v$ and $tar(a) = d$. Edges between object vertices are called *links*. We assume that data vertices have no outgoing edges, and that morphisms of attributed graphs preserve this separation.

Compared with other notions of attributed graphs, like [16], where special attribute carriers are used to relate graph elements and external data values, in our presentation this connection is established by edges within the graph. This simplifies the presentation because attributed graphs can be regarded as a special case of ordinary graphs, subject to the above mentioned constraints. Notice, however, that this limits us to attributed vertices while in [16] both vertices and edges may carry attributes.

Typed graphs. The concept of typed graphs [6] captures the well-known dichotomy between classes and objects, or between database schema and instance, in the case of graphs. Below, it is extended to attributed graphs.

Definition 4.2. (typed attributed graphs)
An *attributed type graph* over Σ is an attributed graph $\langle TG, Z \rangle$ over Σ where Z is the final Σ-algebra having $Z_s = \{s\}$ for all $s \in S$.

An *attributed instance graph* $\langle AG, ag \rangle$ over ATG is an attributed graph AG over the same signature equipped with an attributed graph morphism $ag : AG \to ATG$. A *morphism of typed attributed graphs* $h : \langle AG_1, ag_1 \rangle \to \langle AG_2, ag_2 \rangle$ is a morphism of attributed graphs which preserves the typing, that is, $ag_2 \circ h = ag_1$.

Thus, elements of Z represent the sorts of the signature which are included in TG as types for data vertices. In general, vertices and edges of TG represent vertex and edge types, while attributes in TG are, in fact, attribute declarations.

Instance graphs will be usually infinite, e.g., if the data type $\mathbb{N}$ of natural numbers is present, each $n \in \mathbb{N}$ will be a separate vertex. However, since the data type part will be kept constant during transformation, there is no need to represent this infinite set of vertices as part of the current state. The examples shown contain only those data vertices that are connected to some object vertex.

Example 4.1. (attributed type and instance graphs)
The concepts introduced in this paper shall be illustrated by a small example of a communication system, which models *processes* sending *messages* to each other via *channels*. A message is sent via an *output* channel of a process, which *stores* the message until received via the *input* channel of the other process.

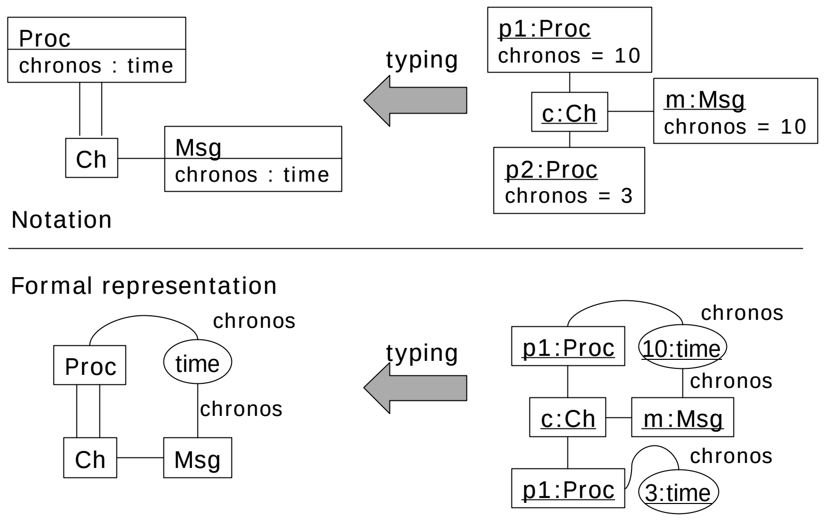


Figure 1. Attributed type and instance graphs: formal presentation (top) and UML-like notation (bottom)

The structure of our communication system is captured by the type graph in the top left of Fig. 1, while a sample system containing only two processes p_1 and p_2 with a single channel c between them is depicted on the right. We use UML notation for class and object diagrams.

The formal representation based on Definition 4.2 is shown in the bottom of Figure 1. Throughout the paper we fix a signature $Time = \langle S, OP \rangle$ with sorts $S = \{time, bool\}$ and operation symbols OP given by $0 :\to time$; $+ : time\ time \to time$; $\geq$: $time\ time \to bool$; $max : time\ time \to time$. This signature is interpreted by the algebras $\mathbb{N}$ of natural numbers and $\mathbb{B}$ of booleans, with the obvious interpretation of $\geq$ and max.

All standard notions, like rule, occurrence, transformation, transformation sequence, etc. can be transfered to the case with attributes. Also, relevant results like the Local Church-Rosser Theorem, the Parallelism theorem, and the corresponding equivalence on transformation sequences based on shifting or swapping independent transformations are easily transferred.

It is worth noticing that, in contrast to ER nets, attributes in our model are typed, that is, different types of nodes may have different selections of attributes. However, like in ER nets, our data types have no syntax: We only consider sets of values without explicit algebraic structure given by operations. As a consequence, we do not explicitly represent variables within rules and variable assignments as part of occurrences: A rule containing variables for attribute calculation and constraints is considered as a syntactic abbreviation for the (possibly infinite) set of its instances where the variables and expressions are replaced by concrete values.

Graph transformation. In the original formulation of the DPO approach [8] the notion of transformation is formalized by two gluing diagrams, called pushouts. Here we have chosen a set-theoretic presentation.

Definition 4.3. (graph transformation and graph transformation system)
Given a Σ-algebra T, a *graph transformation rule* $p = L \to R$ *over* T consists of a pair of graphs L, R

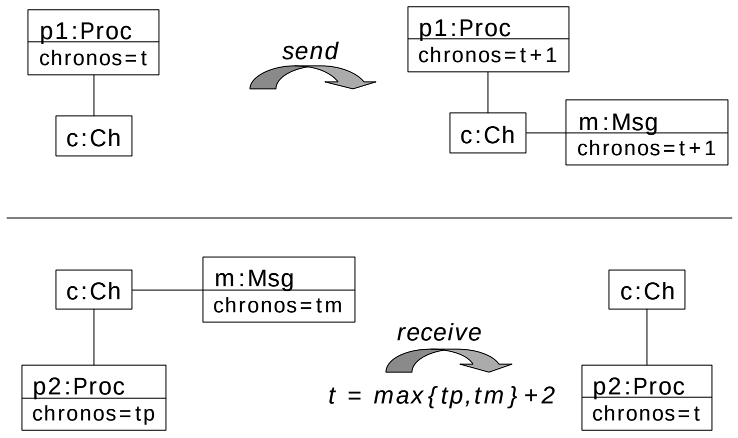


Figure 2. Attributed typed graph transformation rules

attributed over T such that their union $L \cup R$ is a well-defined T-attributed graph.

Given graphs G and H, attributed over a Σ algebra A such that $G \cup H$ is a well-defined A-attributed graph, a *graph transformation* $G \overset{p(o)}{\Longrightarrow} H$ is given by an attributed typed graph morphism $o : L \cup R \to G \cup H$, called *occurrence*, such that

- $o(L) \subseteq G$ and $o(R) \subseteq H$ (the left-hand side of the rule is embedded into the pre-state and the right-hand side into the post-state) and
- $o(L \setminus R) = G \setminus H$ and $o(R \setminus L) = H \setminus G$ (precisely that part of G is deleted which is matched by elements of L not belonging to R and, symmetrically, that part of H is added which is matched by elements new in R).

A *graph transformation system* $GTS = \langle \Sigma, ATG, R \rangle$ consists of a data type signature Σ, an attributed type graph ATG over Σ, and a set R of graph transformation rules over ATG.

A *transformation sequence* $G_0 \overset{*}{\Longrightarrow} G_n = G_0 \overset{p_1(o_1)}{\Longrightarrow} \cdots \overset{p_n(o_n)}{\Longrightarrow} G_n$ in GTS is a sequences of consecutive transformation steps using the rules of GTS.

The union of two graphs L and R is well-defined if, e.g., edges which appear in both L and R are connected to the same vertices in both graphs, edges or vertices with the same name have the same type and attribute values, etc.

The algebras T used within rules will typically be *syntactic*, like the *term algebra* $T_\Sigma(X)$ over a set X of variables[1], consisting of all Σ-terms with variables in X. To express equational application conditions on attributes, the term algebra is replaced by its quotient $T_\Sigma(X)/_E$ with respect to the congruence generated by a set of equations E. This is demonstrated in the example below.

Example 4.2. (attributed graph transformation rule)
Figure 2 provides an examples of attributed typed graph transformation rules over the signature and type graph introduced in Example 1. The two rules model, respectively, the sending and receiving of messages

[1]An S-indexed family of sets of variables $X = (X_s)_{s \in S}$, to be precise.

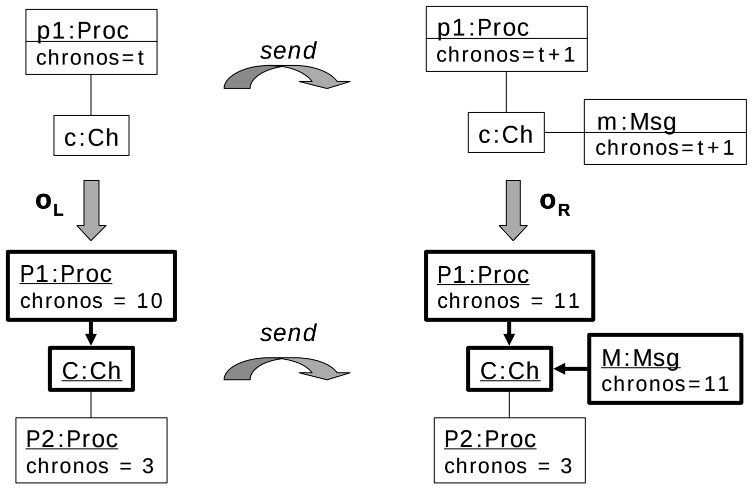


Figure 3. Application of rule send

by processes along channels. Both processes and messages have an attribute chronos to record the time of their last activity.

- **Sending messages:** When process p_1 aims at sending a message, a message object m is generated and placed into the output channel c. The application of the *send* rule takes 2 time units.
- **Receiving messages:** When a message m arrives at the input port of a process p, then the process receives the message by removing the message object from the channel and destroying it afterwards. The application of *receive* rule takes 2 time units as well.

Example 4.3. (attributed graph transformation)
Figure 3 shows an application of the rule send in Figure 2.

Operationally, an attributed graph transformation is performed in three steps. First, find an occurrence o_L of the left-hand side L in the given graph G. This includes an assignment of values from the semantic algebra $\mathbb{N}$ to the variables occurring in L. In our case, the variable t associated with the attribute chronos of process p1 is assigned the value 10.

Second, remove all the vertices, edges, and attribute links from G which are matched by $L \setminus R$. Make sure that the remaining structure $D := G \setminus o(L \setminus R)$ is still a legal graph, i.e., that no edges are left dangling because of the deletion of their source or target vertices. (In this case, the *dangling condition* [8] prohibits the application of the rule.)

Third, glue D with $R \setminus L$ to obtain the derived graph H. This includes the generation of new attribute links to data vertices determined by the evaluation of attribute terms in the algebra A, based on the assignment determined as part of the matching.

Thus, in our example, the attribute links from object vertex P1 to the data vertex 10 would be removed, and replaced by a link from P1 to 12, the evaluation of $x + 2$ where x is bound to 10.

Shift equivalence On transformation sequences, a notion of equivalence is defined which generalises the permutation equivalence on firing sequences: two sequences are equivalent if they can be obtained from each other by repeatedly swapping independent transformation steps. This equivalence has been formalised by the notion of *shift-equivalence* [13] which is based on the following notion of independence of graph transformations. Two transformations $G \overset{p_1(o_1)}{\Longrightarrow} H_1 \overset{p_2(o_2)}{\Longrightarrow} X$ are *independent* if the occurrences $o_1(R_1)$ of the right-hand side of p_1 and $o_2(L_2)$ of the left-hand side of p_2 do only overlap in objects that are preserved by both steps, formally $o_1(R_1) \cap o_2(L_2) \subseteq o_1(L_1 \cap R_1) \cap o_2(L_2 \cap R_2)$. This is more sophisticated than the notion of independent firings of transitions which are required to use entirely disjoint resources.

5. Modelling Time in Graph Transformation Systems

To incorporate time into typed graph transformation with attributes, we follow the approach of TER nets as discussed in Section 3.

Definition 5.1. (type and instance graphs with time)
Let $Time$ be the signature having sort symbol $time$ and operation symbols $+, 0, \geq$ of the obvious arities. A *time data type* $\mathbf{T}$ is an algebra over the signature $Time$ where $\geq^{\mathbf{T}}$ is a partial order with $0^{\mathbf{T}}$ as its least element. Moreover, $\langle +^{\mathbf{T}}, 0^{\mathbf{T}} \rangle$ form a monoid (that is, $+^{\mathbf{T}}$ is associative with neutral element $0^{\mathbf{T}}$) and $+^{\mathbf{T}}$ is monotone wrt. $\geq^{\mathbf{T}}$.

A *type graph with time* $\langle \Sigma, TG \rangle$ is an attributed type graph such that Σ contains $Time$. An instance graph with time over $\langle \Sigma, TG \rangle$ for a given time data type $\mathbf{T}$ is an instance graph $\langle \langle A, G \rangle, ag \rangle$ such that $A|_{Time} = \mathbf{T}$.

Obvious examples of time data types include natural or real numbers with the usual interpretation of the operations, but not dates in the YY:MM:DD format (since, due to the Y2K problem, 0 is not minimal wrt. $\leq$).

In order to transfer the axioms for modelling time in ER nets to attributed graph transformations, we introduce the following terminology: Given a graph transformation rule $p = L \to R$ over a type graph with time, we say that

- p *reads the chronos value* c *of* v if $v \in L$ has a chronos attribute of value c, that is, there exists an edge $e \in L$ with $src(e) = v$ and $tar(e) = c \in D_{time}$.
- p *writes the chronos value* c *of* v if $v \in R$ has a chronos attribute of value c which is not present in L, i.e., there exists an edge $e \in L$ with $src(e) = v$ and $tar(e) = c \in D_{time}$ and $e \notin L$.

Given a transformation $G \overset{p(o)}{\Longrightarrow} H$ we say that $p(o)$ *reads / writes the chronos value of* w if there exists $v \in L \cup R$ such that $o(v) = w$ and p reads / writes the chronos value of v.

It is important to note that, writing an attribute value of a vertex v which is preserved by the rule (i.e., it belongs both to L and R) means deleting the edge from v to the old value and creating a new link to another value. Therefore, writing implies reading the value.

The definition of graph transformation rules with time has to take into account the particular properties of time as expressed, for example, by the axioms in Section 3. The direct transfer of axioms 1 and 2 leads to the following well-formedness conditions.

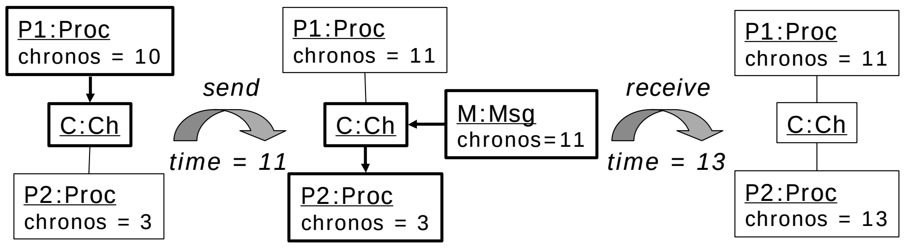


Figure 4. A transformation sequence using the rules in Figure 2

Definition 5.2. (graph transformation system with time)
A *graph transformation rule with time* is a graph transformation rule over a type graph with time satisfying the following conditions.

1. *Local monotonicity:* All chronos values written by p are not smaller than any of the chronos values read by p.
2. *Uniform duration:* All chronos values written by p are equal.

Given a transformation $G \overset{p(o)}{\Longrightarrow} H$ using rule p, the uniform chronos value of axiom 2 is called the *firing time* of the transformation, denoted by $time(p(o))$.

A *graph transformation system with time* is an attributed graph transformation system over a type graph with time whose rules satisfy the conditions above.

Example 5.1. (graph transformation with time)
The attributed graph transformation system introduced in Example 4.2 is in fact a graph transformation system with time. Figure 4 sequence shows a two-step transformation sequence where the firing time is given below the arrow for each step.

One can easily check that both rules satisfy the well-formedness conditions for graph transformation rules with time. The *send* rule computes its time from the chronos value of the sender process p_1, while the *receive* rule takes its time from the maximum of the receiver process p_2 and the message.

The axioms of Definition 5.2 ensure a behaviour of time which can be described informally as follows. According to condition 1, an operation or transaction specified by a rule cannot take negative time, i.e., it cannot decrease the clock values of the nodes it is applied to. Condition 2 states an assumption about atomicity of rule application, that is, all effects specified in the right-hand side are observed at the same time.

Due to the more general nature of typed graph transformation in comparison with ER nets, there exist some additional degrees of freedom.

Existence of time-less vertex types: ER nets are untyped (that is, all tokens have (potentially) the same attributes) while in typed graph transformation we can declare dedicated attributes for every vertex type. Therefore, we do not have to assume an attribute chronos for all vertex types, but could leave the decision about how to distribute chronos attributes to the designer. As we consider time as a distinguished semantic concept, which should not be confused with time-valued data, we do not allow more than one chronos attribute per vertex. This does not forbid us to model additional time-valued data by ordinary attributes.

Update of chronos values for preserved vertices: The second degree of freedom comes from the (well-known) fact that graph transformations generalize Petri nets by allowing contextual rewriting: All tokens in the post-domain of a transition are newly created while in the right-hand side of a graph transformation rule there may be vertices that are preserved. This allows to leave the chronos values of vertices in $L \cap R$ unchanged, creating new timestamps only for the newly generated items.

The type graph in Figure 1 does not declare a chronos attribute for Ch vertices. Thus Ch is a timeless vertex type in the sense of the first item above. The transformation rules in Figure 4.2 based on this type graph do update all chronos values they encounter, for both new and preserved vertices.

If we take in both cases the most restrictive choice, i.e., *chronos values for all types* and *update of chronos values for all vertices in R*, we can show, in analogy with TER nets, that for each transformation sequence s using only rules that satisfy the above two conditions, there exists an equivalent sequence s' such that s' is time-ordered, that is, time is monotonically non-decreasing as the sequence advances.

This is no longer true in general with the more liberal interpretations, as will be shown in Example 5.2.

Theorem 5.1. (global monotonicity)
Given a graph transformation system with time $\mathcal{G}$ such that

- its type graph declares a chronos attribute for every vertex type
- its rules write the chronos values of all vertices in their right-hand sides.

In this case, for every transformation sequence s in $\mathcal{G}$ there exists an equivalent sequence $s' = G_0 \overset{p_1(o_1)}{\Longrightarrow} \dots \overset{p_n(o_n)}{\Longrightarrow} G_n$ in $\mathcal{G}$ such that s' is time-ordered, that is, $time(p_i(o_i)) \leq time(p_{i+1}(o_{i-1}))$ for all $i \in \{0, \dots, n\}$.

Proof:
As a consequence of Theorem 5.2 below. □

Thus, a safe solution to our counter example would be to declare chronos values for both Ch and Msg vertices. However, the example system in Figure 1 and 2 suggests that we can do better than that.

In fact, the problem is to simultaneously ensure the consistency of causality and time in the sense that, whenever two steps are causally dependent, they must communicate their clock values. This idea is crucial to many algorithms for establishing consistent global time in distributed systems, based on logical clocks. The next theorem formalises this statement.

Theorem 5.2. (global monotonicity)
Given a graph transformation system with time $\mathcal{G}$ such that for all transformations $G \overset{p_1(o_1)}{\Longrightarrow} X \overset{p_2(o_2)}{\Longrightarrow} H$ in $\mathcal{G}$ that are *not* sequentially independent, there exists a vertex $v \in o_1(R_1) \cap o_2(L_2)$ whose chronos value is written by p_1 and read by p_2. In this case, for every transformation sequence s in $\mathcal{G}$ there exists an equivalent sequence $s' = G_0 \overset{p_1(o_1)}{\Longrightarrow} \dots \overset{p_n(o_n)}{\Longrightarrow} G_n$ in $\mathcal{G}$ such that s' is time-ordered.

Proof:
The main line of the proof is as follows.

1. Our first observation is that the fact that two transformations $G \overset{p_1(o_1)}{\Longrightarrow} X \overset{p_2(o_2)}{\Longrightarrow} H$ are *not sequentially independent* implies that they are *time ordered*, i.e., $time(p_1(o_1)) \leq time(p_2(o_2))$. This is guaranteed by the existence of a common vertex $v \in o_1(R_1) \cap o_2(l_2)$ with a chronos value written by p_1 and read by p_2, which is
   (a) *exactly* the time of transformation $p_1(o_1)$ (due to the *"uniform duration"* condition),
   (b) *at most* the time of transformation $p_2(o_2)$ (as a consequence of the *"local monotonicity"* condition).
2. Then if two transformations are *not time ordered* and they are *sequentially independent*, we swap them in the rule application sequence[2] . We continue the swap operation until no such transformation pairs can be found.
3. We state that after the *termination* of this swapping algorithm, a time ordered transformation sequence is obtained.
   (a) Let us suppose indirectly that there exist two transformations $G \overset{p_a(o_a)}{\Longrightarrow} X \overset{p_b(o_b)}{\Longrightarrow} H$ that violate the condition of time ordered sequences, i.e. $time(p_a(o_a)) > time(p_b(o_b))$.
   (b) However, if these transformations are *sequentially independent* then the algorithm in Item 2 can still be applied to them, which contradicts the assumption of termination.
   (c) On the other hand, if transformations $p_a(o_a)$ and $p_b(o_b)$ are *not sequentially independent* (but they are not time ordered by the indirect assumption), then we have a contradiction with our first observation, which established that two sequentially dependent transformations with a common vertex are always time ordered.

□

Notice that the condition above can be effectively verified by checking all non-independent two-step sequences in $\mathcal{G}$ where $x = o_1(R_1) \cup o_2(L_2)$.

Example 5.2. (a counter example)
The graph transformation system with time shown in Figure 5 provides us an example where the property of global monotonicity is violated. It coincides with the example introduced in Figure 1 and 2 since the type graph does not define a chronos attribute for messages in this case. However, all chronos values that are encountered are updated by the rules. While in the first system, every sequence is equivalent to one which is time-ordered, this is not the case for the system in Figure 5.

The sequence in Figure 6 gives a counter example. It is not time-ordered because the first step has a higher firing time than the second. Now observe that the two steps are not sequentially independent because the message consumed by the second step has to be generated by the first. Therefore, no equivalent sequence exists where the rules are applied in the reverse order.

[2]This algorithm is, in fact, conceptually similarly to the trick applied in the construction of a shift equivalent transformation sequence.

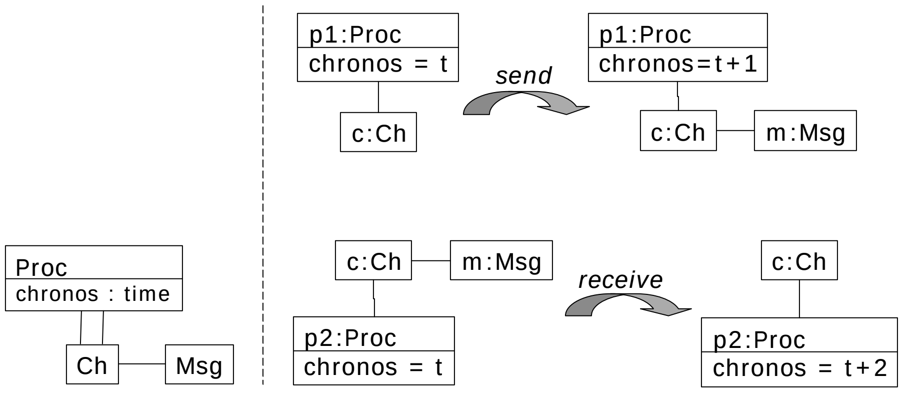


Figure 5. Another graph transformation system with time

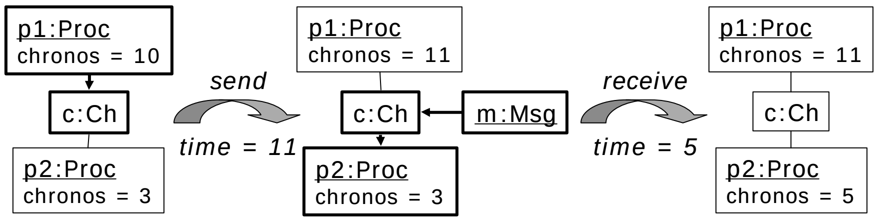


Figure 6. A sequence in the GTS of Figure 5 that is not time-ordered

The conceptual explanation is that, since no timestamps are attached to messages, the receiver cannot synchronize its clock to the sender when the message is processed. In fact, the problem does not occur in the (otherwise similar) sequence in Figure 4 because, in this system, Msg has a chronos attribute as well.

This time, our global monotonicity theorem trivially holds, since the chronos value of each message object is written by the send rule and read by the receive rule. Thus in a transformation sequence where a certain application of send precedes the application of receive, the time of receive cannot be less then the time of send due to the well-formedness conditions 1 and 2.

6. Strong Semantics

In applications it is often desirable to enforce a certain order of actions, e.g., to ensure that messages are delivered in the same order in which they are sent. In many cases, such requirements can be coded into the model by additional vertices and edges serving as control structures. Heavily used, however, this leads to cluttered and unreadable models. Thus, in this section, we will discuss semantic solutions to this problem, again following the line of TER nets.

The basic idea of strong semantics is to give priority to transformations with smaller firing time. That is, before choosing a transformation which is bound to occur at a later point in time, all possible earlier transformations should be performed. In this way, for example, the global preservation of message order can be enforced at a semantic level.

Example 6.1. (motivating example)
For a motivating example, let us consider the communication process depicted in Fig. 7. Note that not the entire state space of the system is depicted to improve the clarity of the figure, i.e., executable transformation sequences are missing.

Our (first) objective for introducing strong semantics of graph transformation is to semantically ensure that the message M1 sent by process P1 at time unit 1 is received earlier by process P3 than message M2 issued by process P2 at time unit 5 supposing that the load of communication channels is equally balanced (i.e., driven by graph transformation rules send and receive of Fig. 2). In this respect, the result graph IG_{5a} shows a desired situation, while graph IG_{5b} depicts an undesired execution.

Note that while the sending of messages is independent of each other (steps send(M1) and send(M2)), there is a potential conflict in receiving messages since the chronos attribute of process P3 is shared.

Now, let us define globally strong transformation sequences in a formal way.

Definition 6.1. (globally strong sequences)
A transformation sequence $s = G_0 \overset{t_1}{\Longrightarrow} G_1 \overset{t_2}{\Longrightarrow} \cdots \overset{t_n}{\Longrightarrow} G_n$ in GTS is called (globally) strong if for all $i = 1 \ldots n$ and transformations $G_i \overset{t'_i}{\Longrightarrow} G'_i$ in GTS: $time(t_i) \leq time(t'_i)$.

Example 6.2. (a globally strong sequence)
We can easily notice that the transformation sequence $s_1 = IG_1 \overset{t_1}{\Longrightarrow} IG_2 \overset{t_2}{\Longrightarrow} IG_{3a} \overset{t_3}{\Longrightarrow} IG_{4a} \overset{t_4}{\Longrightarrow} IG_{5a}$ is globally strong, as at each step, we selected the transformation with the minimal firing time (time(send(M1))=3, time(receive(M1))=5, time(send(M2))=7, time(receive(M2))=9, respectively).

Unsurprisingly, this transformation sequence is time-ordered as well, which turns out to be a general property of globally strong sequences generated by our well-known subclass of graph transformation systems with time.

Theorem 6.1. (globally strong is time ordered)
Let $\mathcal{G}$ be a graph transformation system with time such that for all transformations $G \overset{p_1(o_1)}{\Longrightarrow} X \overset{p_2(o_2)}{\Longrightarrow} H$ in $\mathcal{G}$ that are *not* sequentially independent, there exists a vertex $v \in o_1(R_1) \cap o_2(L_2)$ whose chronos value is written by p_1 and read by p_2 (i.e., identical to the assumption of Theorem 5.2).

Let s be a transformation sequence in such a GTS. If s is globally strong then s is time ordered.

Proof:
This theorem can be proved by induction on the length of the globally strong sequence s.

1. Any globally strong sequence of length 1 is time-ordered by definition.

2. Let us suppose that sequence s is provenly time ordered up to length i. Now we prove that it remains time ordered at length $i + 1$.

   (a) We suppose by contradiction that t_{i+1} violates the condition of time orderedness, i.e., $time(t_{i+1}) < time(t_i)$.

   (b) The selection mechanism of globally strong transformation sequences guarantees that at each step t_j of a globally strong transformation sequence s, t_j can be a member of the sequence only if its time $time(t_j)$ is less than or equal to the time of any transformation step t'_j that is enabled and executable. As a consequence, the occurrence of t_{i+1} was non-existent at step i (i.e., prior to the application of t_i), otherwise t_{i+1} would have been selected instead t_i at this previous step (more precisely, at *some* previous step).

   (c) Equivalently speaking, the execution of step t_i generated some new elements of the graph required for the successful matching of t_{i+1}, therefore t_{i+1} is *not sequentially independent* on t_i. However, according to our first observation in the proof of Theorem 5.2, in such a case $time(t_i) \leq time(t_{i+1})$ (due to the existence of a vertex written by t_i and read by t_{i+1}), which contradicts our indirect assumption and thus finishes the proof.

□

Note, however, that globally strong transformation sequences and time-ordered transformation sequences are not equivalent. In other words, there may exist time-ordered sequences that do not conform to the globally strong semantics.

Example 6.3. (strong sequences vs. time-orderedness)
For instance, both transformation sequences $s_1 = IG_1 \overset{t_1}{\Longrightarrow} IG_2 \overset{t_2}{\Longrightarrow} IG_{3a} \overset{t_3}{\Longrightarrow} IG_{4a} \overset{t_4}{\Longrightarrow} IG_{5a}$ and $s_2 = IG_1 \overset{t_1}{\Longrightarrow} IG_2 \overset{t'_2}{\Longrightarrow} IG_{3b} \overset{t'_3}{\Longrightarrow} IG_{4b} \overset{t'_4}{\Longrightarrow} IG_{5b}$ in Fig. 7 are obviously time ordered, however, s_2 is not globally strong since when transformation step t_2 is applied on graph IG_2, $time(t'_2 > time(t_2)$ which contradicts the previous definition.

In fact, there are no other globally strong transformation sequences in Fig. 7.

Despite globally strong semantics of rule applications looks rather intuitive at first sight, unfortunately, the condition potentially violates the concurrent character of graph transformation. This is based on the fact that when considering concurrency, the applicability of a rule depends only on local information, i.e., the firing times of independent matchings are incomparable.

Example 6.4. (globally strong vs. shift-equivalent sequences)
To demonstrate the problem, we show two shift-equivalent sequences in Fig. 7, one of which is ruled out by the global priority while the other one is not.

Consider, for instance, transformation sequences $s_1 = IG_1 \xrightarrow{t_1} IG_2 \xrightarrow{t_2} IG_{3a} \xrightarrow{t_3} IG_{4a} \xrightarrow{t_4} IG_{5a}$ and $s_2 = IG_1 \xrightarrow{t_1} IG_2 \xrightarrow{t_3} IG_{3b} \xrightarrow{t_2} IG_{4a} \xrightarrow{t_4} IG_{5a}$. Since t_2 and t_3 are independent of each other (as demonstrated by the white and the grey shaded matchings in IG_2 which are not overlapping) sequences s_1 and s_2 are shift equivalent.

However, s_2 is ruled out by the globally strong semantics as the sending of message M2 at time unit 7 cannot happen before the receiving of message M2 at time unit 6 regarding from a global point of view.

As a consequence, we propose a weakening of the condition which does only apply the priority to such transformations which are in conflict. This condition, called locally strong, is shown to be compatible with shift-equivalence.

Definition 6.2. (locally strong sequences)
A transformation sequence $s = G_0 \xrightarrow{t_1} G_1 \xrightarrow{t_2} \cdots \xrightarrow{t_n} G_n$ in GTS is called locally strong if for all $i = 1 \ldots n$ and transformations $G_{i-1} \xrightarrow{t'_i} G'_i$ in GTS where t_i is in conflict with t'_i $time(t_i) \leq time(t'_i)$.

Unsurprisingly, the set of locally strong and time-ordered sequences for a given GTS with time are incomparable, i.e., locally strong transformation sequences are not required to be time-ordered, and on the other hand, there may be time-ordered sequences which are not locally strong.

Example 6.5. (locally strong vs. time-ordered sequences)
To demonstrate the difference, we show two sequences in Fig. 7, one of which is not locally strong while the other one is not time-ordered.

Consider, for instance, transformation sequences $s_1 = IG_1 \xrightarrow{t_1} IG_2 \xrightarrow{t_2} IG_{3b} \xrightarrow{t_3} IG_{4a} \xrightarrow{t_4} IG_{5a}$ and $s_2 = IG_1 \xrightarrow{t_1} IG_2 \xrightarrow{t_2} IG_{3b} \xrightarrow{t'_3} IG_{4b} \xrightarrow{t'_4} IG_{5b}$. Since the t_3 and t'_3 are in conflict (because of the conflicting firing times in IG_{3b}) transformation t_3 takes priority over t'_3 according to the locally strong semantics, therefore, sequence s_2 is not locally strong.

However, when regarding the firing times of s_1 we easily notice that s_1 is not time ordered since $time(t_2) = 7$ while $time(t_3) = 6$.

This notion of strong semantics provides a satisfactory compromise between our original goal of enforcing priority of earlier steps and the local nature of matching and rule application. This is true as long as we consider rules with a fixed firing time. However, if the firing time is *flexible* (e.g., the precise time to deliver a message is unknown, except for an upper and lower bound), the present condition would lead to a behavior where, not only earlier steps have priority over later ones, but where everything would happen as soon as possible.

For example, the rule in Figure 8 models a receive operation with a delivery time between 2 and 6 seconds. The inequation $max(tp, tm)+2 \leq t \leq max(tp, tm)+6$ can be expressed by the two equations

$t = max(tp, tm) + 2 + x$ and $x + y = 4$. Thus, formally we stay in the framework of rules attributed with a quotient term algebra $T_\Sigma(X)/_E$. If we require locally strong semantics, the message would always take exactly 2 seconds to deliver because the two applications of the rule, which differ only for the assignment of values to x and y, are in conflict. Hence we call this version of strong semantics the *eager* one.

To recover the desired flexibility, we introduce a notion of non-eager strong semantics, based on the concept of the maximal firing time of a step. This is the latest point in time at which a transformation using a given rule and match may happen according to the constraints expressed on the chronos attributes.

Definition 6.3. (maximal possible firing time)
Given a transformation (step) $G \overset{p(o)}{\Longrightarrow} H$ in GTS, its maximal possible firing time is defined as $maxtime(t) = max\{time(t') \mid t' = G \overset{p(o')}{\Longrightarrow} H' \text{ where } o'_{L,G} = o_{L,G}\}$.

Recall that $o'_{L,G}, o_{L,G}$ denote the graph components of the attributed graph morphisms o'_L, o_L, respectively. That is, the maximal firing time of a step is the maximum of all firing times of steps with the same matching of the left-hand sides graph L, thereby implicitly enforcing the assignment of all variables occurring in L. The point is that variables like x and y in the rule of Figure 8 remain unconstrained—therefore the firing times of the steps may still vary.

In practice, possible firing times of a transformation step are frequently represented as firing *intervals* although the definition does not require to use intervals. For informal discussions and illustrations, we this intuitive interpretation shall be helpful.

Now the non-eager version of strong semantics requires that if (the graphical components of) a matching of a transformation step is enabled, and remains enabled for all possible time values at which it can be executed then it *must* be executed.

Definition 6.4. (globally strong sequences, lazy)
A transformation sequence $s = G_0 \overset{t_1}{\Longrightarrow} G_1 \overset{t_2}{\Longrightarrow} \cdots \overset{t_n}{\Longrightarrow} G_n$ in GTS is called globally strong and lazy if for all $i = 1 \ldots n$ and all applicable transformations $G_{i-1} \overset{t'_i}{\Longrightarrow} G'_i$ in GTS: $time(t_i) \leq maxtime(t'_i)$.

Definition 6.5. (locally strong sequences, lazy)
A transformation sequence $s = G_0 \overset{t_1}{\Longrightarrow} G_1 \overset{t_2}{\Longrightarrow} \cdots \overset{t_n}{\Longrightarrow} G_n$ in GTS is called locally strong and lazy if for all $i = 1 \ldots n$ and transformations $G_i \overset{t'_i}{\Longrightarrow} G'_i$ in GTS where t_i is in conflict with t'_i $time(t_i) \leq maxtime(t'_i)$.

In other terms, let $s = G_0 \overset{t_1}{\Longrightarrow} G_1 \overset{t_2}{\Longrightarrow} \cdots \overset{t_i}{\Longrightarrow} G_i, i \geq 0$ be a strong lazy firing sequence in GTS and let us examine which transformation step may be executed at this point. Any enabled step $G_i \overset{t_{i+1}}{\Longrightarrow} G_{i+1}$ can be chosen to be applied if for all *enabled* and *conflicting* steps $G_i \overset{t'_i}{\Longrightarrow} G'_i$, the actual firing time t_i of step $G_i \overset{t_{i+1}}{\Longrightarrow} G_{i+1}$ is less than the maximal firing time of t'_i. Or equivalently, if there exists no other enabled steps $G_i \overset{t'_i}{\Longrightarrow} G'_i$ such that the maximal firing time of t'_i is less than t_i.

Example 6.6. The intuitive meaning of locally strong and lazy semantics is demonstrated in Fig. 9, where a timing diagram is depicted to guide the execution of conflicting transformation steps receive(M1) and receive(M2) applied to the instance graph IG_{3b} of Fig. 7.

Timing constraints are those expressed in the rule receive depicted in Fig. 8. Therefore, we can conclude that the maximal possible firing time of receive(M1) is 10, while the respective parameter of receive(M2) is 13.

Let τ denote the time when, according to locally strong and lazy semantics, the next transformation step is scheduled for execution.

- If $\tau < 3$, then none of the matchings are existent therefore nothing happens.
- If $3 \leq \tau < 6$, then the matching of receive(M1)) becomes existent but this step is not allowed to be executed as $\tau < max(3,4) + 2 = 6$ (which is the earliest time point in the possible firing interval of receive(M1)).
- If $6 \leq \tau < 7$, then receive(M1) can fire at any time, since there are other no conflicting matchings (note that message M2 is not available yet in the channel).
- If $7 \leq \tau < 9$, then the matching of receive(M2)) becomes existent, but receive(M1) can still fire at any time, since while the graphical parts of steps receive(M1) and receive(M2) are in already in conflict, the transformation step receive(M2) is not executable yet (not within its due time).
- If $9 \leq \tau < 10$, then both transformation steps receive(M1) and receive(M2) can fire since none of the maximal possible firing time points have arrived (both of them are within their firing interval).
- If $10 \leq \tau$ then the maximal possible firing time of receive(M1) has arrived, therefore, we *must* execute this transformation step. Otherwise the axiom of locally strong lazy semantics would be violated since receive(M1) is a transformation step that is in conflict with receive(M2), but the execution time $\tau > 10$ of receive(M2) is greater than the maximal possible firing time of receive(M1).

Next we show that locally strong sequences (eager and lazy) are compatible with shift-equivalence.

Theorem 6.2. (locally strong is closed under shift)
Let s and s' be transformation sequences in GTS. If s is locally strong and s' is equivalent to s, then s' is locally strong.

Proof:
By way of contradiction, assume transformation sequences s and s' in GTS that are shift-equivalent, and where s is locally strong while s' is not. We will show that s' not locally strong implies that s is not locally strong.

By definition of shift-equivalence, s can be obtained from s' by a finite number of swaps of sequentially independent steps. If $s = s'$, this number is zero and the statement follows trivially. Otherwise, there exists s'' equivalent to s' by means of n swaps and such that s can be obtained from s'' by another swap. In particular, let $s'' = (G_0 \overset{*}{\Longrightarrow} G_n \overset{p(o)}{\Longrightarrow} G_{n+1} \overset{q(m)}{\Longrightarrow} G_{n+2} \overset{*}{\Longrightarrow} G_{n+k})$ and $s = (G_0 \overset{*}{\Longrightarrow} G_n \overset{q(m')}{\Longrightarrow} G'_{n+1} \overset{p(o')}{\Longrightarrow} G'_{n+2} \overset{*}{\Longrightarrow} G_{n+k})$. The relevant steps are depicted in Figure 10.

By induction hypothesis, s'' is not locally strong. Therefore, for some step $G_{i-1} \overset{t_i}{\Longrightarrow} G_i$ there exists a conflicting step $G_{i-1} \overset{t'_i}{\Longrightarrow} G'_i$ in GTS such that $time(t_i) > maxtime(t'_i)$.

If $i < n$ or $i \geq n + 2$, it follows directly that s is not locally strong. It remains to analyze $i = n$ and $i = n + 1$.

□

7. Conclusion

The ability to specify time-dependent behavior is an important feature for any modeling technique aiming at concurrent and safety-critical systems. In this paper, we have developed a model of time in attributed graph transformation systems inspired by the concepts of TER nets, a notion of high-level Petri nets with time.

We have discussed several semantic choices and their consequences, leading to a *global monotonicity theorem* which provides conditions for the existence of time ordered transformation sequences. This theorem generalizes the idea behind familiar algorithms for establishing consistent logical clocks via time stamps in distributed systems [15].

Further, we have investigated a stronger semantic model where, by assumption of a local or global scheduling mechanism, steps with an earlier firing time are granted priority. The local version is shown to be compatible with the concurrency theory of graph transformation.

Future work will include a deeper analysis of applications, in particular, the semantics of time in diagrammatic techniques like statecharts or sequence diagrams and their formalization using graph transformation (cf. [9, 12, 14]).

Acknowledgement. We wish to thank Luciano Baresi for his introduction to Petri nets with time.

References

[1] Baldan, P.: *Modelling Concurrent Computations: from Contextual Petri Nets to Graph Grammars*, Ph.D. Thesis, Dipartimento di Informatica, Università di Pisa, 2000.

[2] Baldan, P., Corradini, A., Ehrig, H., Löwe, M., Montanari, U., Rossi, F.: Concurrent Semantics of Algebraic Graph Transformation, in: *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 3: Concurrency and Distribution* (H. Ehrig, H.-J. Kreowski, U. Montanari, G. Rozenberg, Eds.), World Scientific, 1999, 107 – 188.

[3] Baresi, L., Pezzé, M., Taentzer, G., Eds.: *Proc. ICALP 2001 Workshop on Graph Transformation and Visual Modeling Techniques, Heraklion, Greece*, Electronic Notes in TCS, Elsevier Science, July 2001.

[4] Corradini, A., Heckel, R., Eds.: *Proc. ICALP 2000 Workshop on Graph Transformation and Visual Modeling Techniques*, Carleton Scientific, Geneva, Switzerland, July 2000.

[5] Corradini, A., Montanari, U.: Specification of Concurrent Systems: from Petri Nets to Graph Grammars, *Quality of Communication-Based Systems*, Kluwer Academic Publishers, 1995.

[6] Corradini, A., Montanari, U., Rossi, F.: Graph processes, *Fundamenta Informaticae*, **26**(3,4), 1996, 241–266.

[7] Ehrig, H., Padberg, J., Ribeiro, L.: Algebraic high-level nets: Petri nets revisited, *Recent Trends in Data Type Specification*, Springer-Verlag, Caldes de Malavella, Spain, 1994, Lecture Notes in Computer Science 785.

[8] Ehrig, H., Pfender, M., Schneider, H.: Graph grammars: an algebraic approach, *14th Annual IEEE Symposium on Switching and Automata Theory*, IEEE, 1973.

[9] Engels, G., Hausmann, J., Heckel, R., Sauer, S.: Dynamic Meta Modeling: A Graphical Approach to the Operational Semantics of Behavioral Diagrams in UML, *Proc. UML 2000, York, UK* (A. Evans, S. Kent, B. Selic, Eds.), 1939, Springer-Verlag, 2000.

[10] Ghezzi, C., Mandrioli, D., Morasca, S., Pezzè: A Unified High-Level Petri Net Formalism for Time-Critical Systems, *IEEE Transactions on Software Engineering*, **17**(2), 1991, 160–172.

[11] Gyapay, S., Heckel, R.: Towards Graph Transformation with Time, *Proc. ETAPS 2002 Workshop on Application of Graph Transformation, Grenoble, France* (H.-J. Kreowski, Ed.), April 2002.

[12] Hausmann, J., Heckel, R., Sauer, S.: Towards Dynamic Meta Modeling of UML Extensions: An Extensible Semantics for UML Sequence Diagrams, *Symposium on Visual Languages and Formal Methods, IEEE Symposia on Human Computer Interaction (HCI 2001), Stresa, Italy* (M. Minas, A. Schürr, Eds.), IEEE Computer Society Press, Los Alamitos, CA, September 2001.

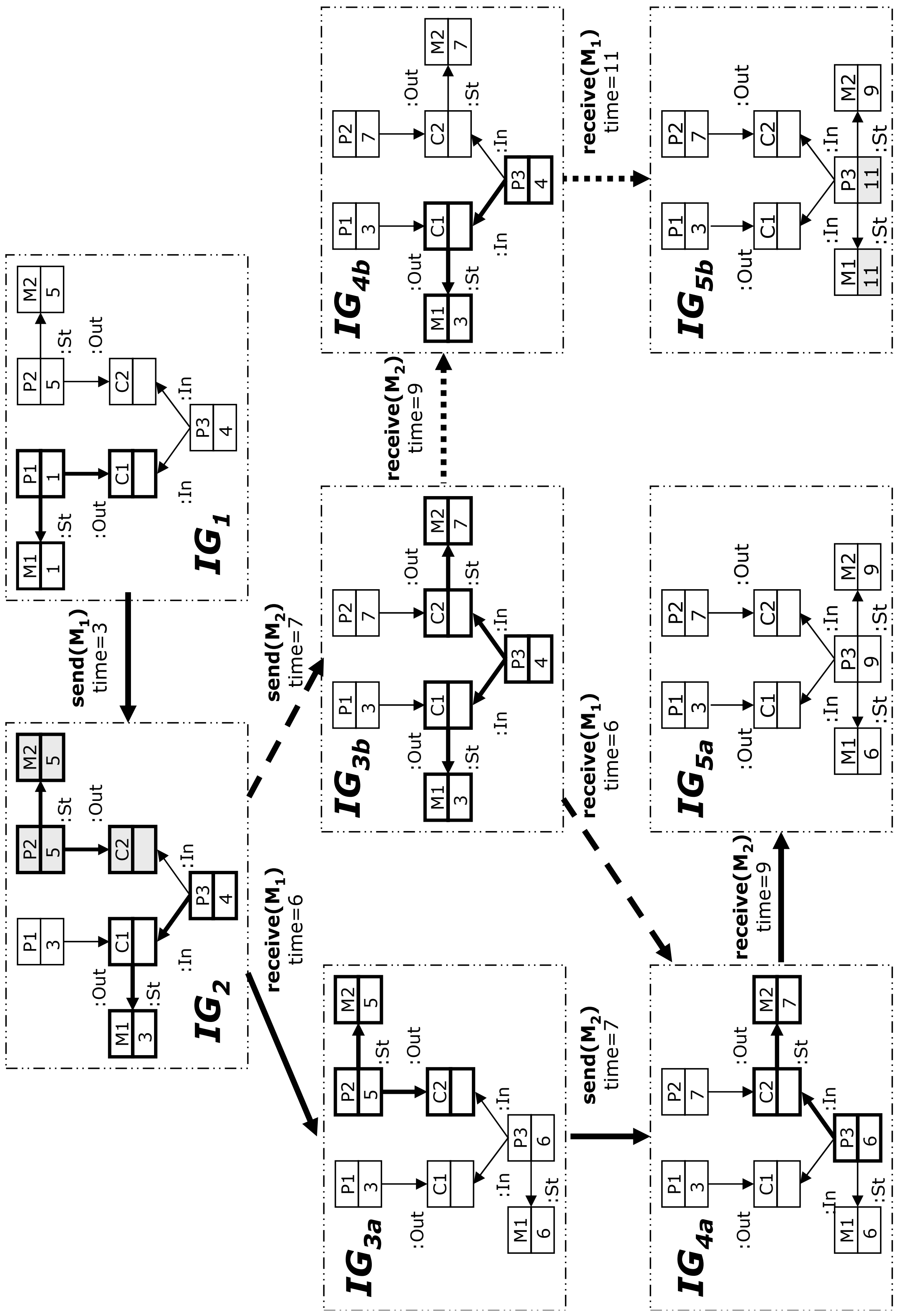
[13] Kreowski, H.-J.: *Manipulation von Graphmanipulationen*, Ph.D. Thesis, Technical University of Berlin, Dep. of Comp. Sci., 1977.

[14] Kuske, S.: A Formal Semantics of UML State Machines Based on Structured Graph Transformation, *Proc. UML 2001, Toronto, Kanada* (M. Gogolla, C. Kobryn, Eds.), 2185, Springer-Verlag, 2001.

[15] Lamport, L.: Time, clocks, and the ordering of events in a distributed system, *Communications of the ACM*, **21**(7), July 1978.

[16] Löwe, M., Korff, M., Wagner, A.: An Algebraic Framework for the Transformation of Attributed Graphs, in: *Term Graph Rewriting: Theory and Practice* (M. R. Sleep, M. J. Plasmeijer, M. van Eekelen, Eds.), chapter 14, John Wiley & Sons Ltd, 1993, 185–199.

[17] Ribeiro, L.: *Parallel Composition and Unfolding Semantics of Graph Grammars*, Ph.D. Thesis, TU Berlin, 1996.


IG1
send(M1) time=3
IG2
send(M2) time=7
receive(M1) time=6
IG3a
IG3b
receive(M2) time=9
receive(M1) time=6
send(M2) time=7
IG4a
IG4b
receive(M1) time=11
receive(M2) time=9
IG5a
IG5b
P1
P2
P3
C1
C2
M1
M2
:In
:Out
:St

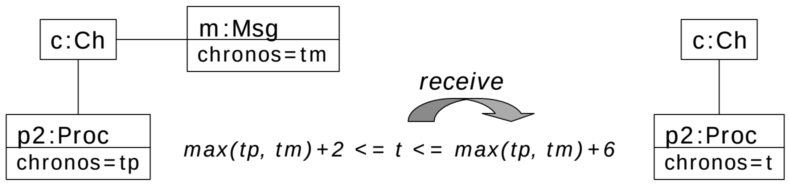


Figure 8. A rule with flexible firing time

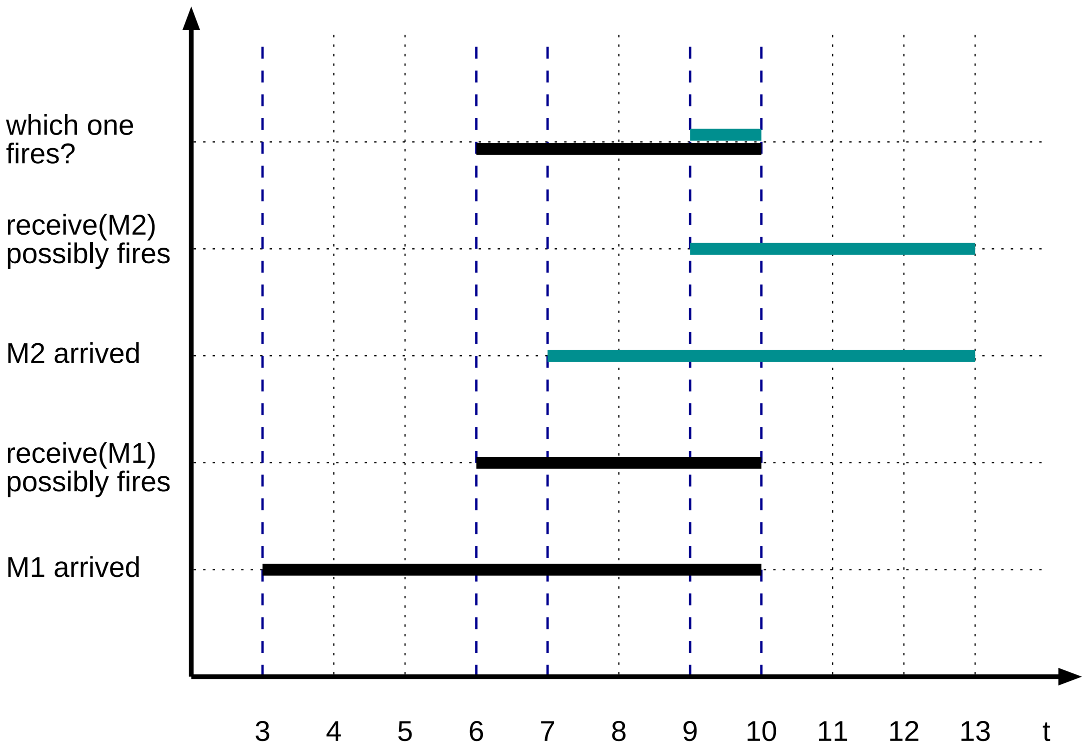


Figure 9. An intuitive interpretation of lazy semantics

$$\begin{tikzcd}
 & G_n \arrow[dl, Rightarrow, "p(o)"'] \arrow[dr, Rightarrow, "q(m')"] & \\
G_{n+1} \arrow[dr, Rightarrow, "q(m)"'] & & G'_{n+1} \arrow[dl, Rightarrow, "p(o')"] \\
 & G_{n+2} &
\end{tikzcd}$$

Figure 10. Swapping independent of transformation steps