Technische Universität Berlin

Forschungsberichte
der Fakultät IV – Elektrotechnik und Informatik

Process Denition of Adhesive HLR Systems (Long Version)

Frank Hermann

Bericht-Nr. 2008-09
ISSN 1436-9915

Process Definition of Adhesive HLR Systems (Long Version)

Frank Hermann

frank(at)cs.tu-berlin.de
Institut für Softwaretechnik und Theoretische Informatik
Technische Universität Berlin, Germany

Bericht-Nr. 2008/09
ISSN 1436-9915

Process Definition of Adhesive HLR Systems (Long Version)

Frank Hermann

frank(at)cs.tu-berlin.de
Institut für Softwaretechnik und Theoretische Informatik
Technische Universität Berlin, Germany

Abstract

Process models of graph transformation systems are based on the concept of occurrence grammars, which are a generalization of Petri net processes given by occurrence nets. Recently, subobject transformation systems were proposed as an abstract framework for occurrence grammars in adhesive categories, but they are restricted to monomorphic matches for transformation steps. In this paper we review the construction of STSs as processes for plain graph grammars and present an extension to weak adhesive HLR categories with non-monomorphic matching, such that e.g. attributed graph grammars are included.

Keywords: process definition, graph transformation, adhesive HLR systems

1 Introduction

Dynamic systems often show several possibilities for changing the order of their execution steps while their observable behaviour remains equivalent, i.e. the same output is produced for the same input and each execution step is performed in the same way. For this reason, a process model of a specific system run does not only describe the given execution order but also its equivalent ones. Formal models of processes build the basis for a formal analysis and in this way for the verification of relevant properties. Furthermore, equivalent sequences can be compared to find a suitable canonical one.

Graph transformation [12] is a universal modelling technique, which is applied e.g. for the definition of visual languages [8] and for the specification of model transformations [4]. Furthermore, it is especially appropriate to model concurrent and distributed systems, where process analysis is highly relevant. Classical formal models in this context are Petri nets [11], which can be characterized as restricted graph transformation systems (GTSs),

while general GTSs are more expressive. They provide an intuitive and object oriented view on models and based on their formal categorical foundation they enable automated and reliable checks.

Different graph transformation approaches were developed. In order to show results for a variety of transformation approaches the concept of high level replacement systems was introduced in [7, 6]. Adhesive transformation systems based on adhesive categories [10] were presented to form a more compact framework. It covers a variety of important transformation systems, e.g. typed graph transformation systems. Adhesive rewriting systems were extended to the more general concept of adhesive high level replacement systems [5], which covers approaches that also support attribution and inheritance. The most important advantage of the abstract setting is that results are available for all concrete instantiations.

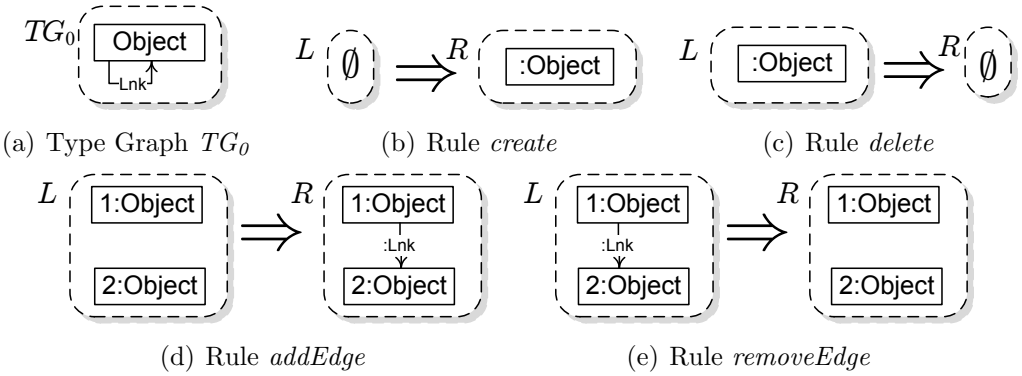
As a main result the static analysis of conflicts between rules of a transformation system can be based on the computation of critical pairs. Possible conflicts between rule applications are detected. If there is no critical pair for a combination of two rules, it is ensured that these rules will never be parallel dependent on each other in any derivation of the transformation system. Furthermore, each dependency, which may arise, corresponds to a critical pair, which can be embedded into the conflicting situation. The analysis of critical pairs is complex in general, because all combination and all overlappings of the left hand sides of the rules are checked.

Occurrence grammars [1] were proposed as a process definitions of GTSs in order to provide a compact model for the analysis of dependencies between the derivation steps of a given derivation in order to analyze the switch equivalence of derivations. They correspond to the concept of occurrence nets for Petri nets. Occurrence grammars were lifted to adhesive rewriting systems [2] and their properties were studied in more detail leading to the notion of subobject transformation systems (STSs)[3].

In this paper we review the concept of STSs in Section 2 and show how to define an STS corresponding to a derivation in a plain graph grammar, such that the STS can be considered as the process corresponding to this derivation. The analysis techniques for STSs (see [3]) - not reviewed in this paper - allow to analyze this process. In Sec. 3 we show how the construction of an STS can be extended to weak adhesive HLR systems with arbitrary matching, if suitable conditions are fulfilled by the concrete category. Exemplarily we show that the conditions hold for attributed graph grammars with non-injective matches. The main construction is based on $\mathcal{E} - \mathcal{M}$-factorization of the matches, where $\mathcal{E}$ and $\mathcal{M}$ are subclasses of epi- and monomorphisms, respectively. We show that the construction of the derived STS with its relations is well defined for adhesive HLR systems with effective unions and $\mathcal{E}$-$\mathcal{M}$-factorizations of matches. The process construction creates an STS, which allows us to analyze sequential and parallel independencies and dependencies in a efficient way. This paper extends our previous work in [9] by the formal proves and constructions.

2 STSs for Adhesive Transformation Systems

The definition of processes of graph grammars is given by occurrence grammars and was extended to the more abstract setting of adhesive rewriting systems [2] based on subobjects. In [3] we elaborated this technique and introduced the generalized concept of subobject transformation systems (STSs). This included new basic relations on rule occurrences, which are shown to compose to the previously defined compound relations for occurrence grammars. The new basic relations separately define all occurring dependencies and conflicts between two derivation steps. The more detailed description of dependencies by the basic relations shall improve the efficiency of an analysis when checking them in concrete instantiations. In this section, we recall STSs and the process construction that builds an STS to a given derivation in an adhesive rewriting system, i.e. within a transformation system over an adhesive category. At first, consider the following example of a graph grammar GG_0 for which we will define the process $\mathcal{S}_0$ later on.



(a) Type Graph TG_0 (b) Rule *create* (c) Rule *delete*

(d) Rule *addEdge* (e) Rule *removeEdge*

Figure 1: Graph Grammar GG_0

Example 1 (Graph grammar GG_0)**.** *Fig. 1 shows the components of the graph grammar GG_0, where all elements are typed over TG_0, which contains just one node "Object" and one edge "Lnk". Rules create and delete nodes and edges separately, e.g. the rule "addEdge()" takes two nodes and creates a new edge in between. Consider the derivation $d = \left\langle G_0 \overset{create}{\Longrightarrow}{}^3\ G_1 \overset{addEdge}{\Longrightarrow}{}^3\ G_2 \overset{removeEdge}{\Longrightarrow}{}^3\ G_3 \overset{delete}{\Longrightarrow}{}^3\ G_4 \right\rangle$, where each of the productions is applied three times. The intermediate graph G_2 is equal to T_0 in Fig. 2 and $G_0 = G_4 = \emptyset$. Now, an equivalent derivation d' for d can be calculated with the help of a corresponding STS $\mathcal{S}_0$ as shown in Example 2. Many switchings of the derivation steps are possible, which lead to equivalent derivations. For instance an edge may be added already when the needed nodes are present and not as late as all nodes are created. This analysis can be performed without checking the standard independence condition - existence of mediating morphisms in the derivation diagram - but by checking the basic relations of the STS $\mathcal{S}_0$.*

In order to describe the process construction we review the notions needed for STSs. Subobjects of an object T are equivalence classes of monomorphisms $a : A \to T$ in $\mathbf{C}$, and

we write A for short leaving the monomorphism and the equivalence class $[a : A \to T]$ implicit. The category of subobjects of an object T is called $\mathbf{Sub}(T)$. Its objects are the subobjects of T and its morphisms $f : A \to B$ in $\mathbf{Sub}(T)$ are compatible with the implicit monomorphisms, i.e. $b \circ f = a$, implying that they are monomorphisms in $\mathbf{C}$ as well and we write $A \subseteq B$ for short. Furthermore, all morphisms in $\mathbf{Sub}(T)$ are unique making all diagrams commutative. For a graph T in the category **Grahps** the morphisms in $\mathbf{Sub}(T)$ are injective graph morphisms $f = (f_V, f_E) : G_1 = (V_1, E_1, s_1, t_1) \to G_2 = (V_2, E_2, s_2, t_2)$.

Definition 1 (Subobject transformation systems)**.** A *Subobject Transformation System* $\mathcal{S} = (T, P, \pi)$ over an adhesive category $\mathbf{C}$ consists of a super object $T \in \mathbf{C}$, a set of *production names* P, and a function $\pi \colon P \to \mathbf{Sub}(T)^{\cdot\leftarrow\cdot\rightarrow\cdot}$, which maps a production name p to a *production* $L_p \supseteq K_p \subseteq R_p$, i.e., a span in $\mathbf{Sub}(T)$.

Direct derivations of an STS are defined by properties of union and intersection in $\mathbf{Sub}(T)$. The intersection of two objects A and B with morphisms $a : A \to T$ and $b : B \to T$ is defined as the pullback of a and b in $\mathbf{C}$. The union of A and B is given by the pushout of A and B via its intersection $A \cap B$ in $\mathbf{C}$. These constructions coincide with the product of A and B in $\mathbf{Sub}(T)$ for the intersection and the coproduct of A and B in $\mathbf{Sub}(T)$ for the union of both.

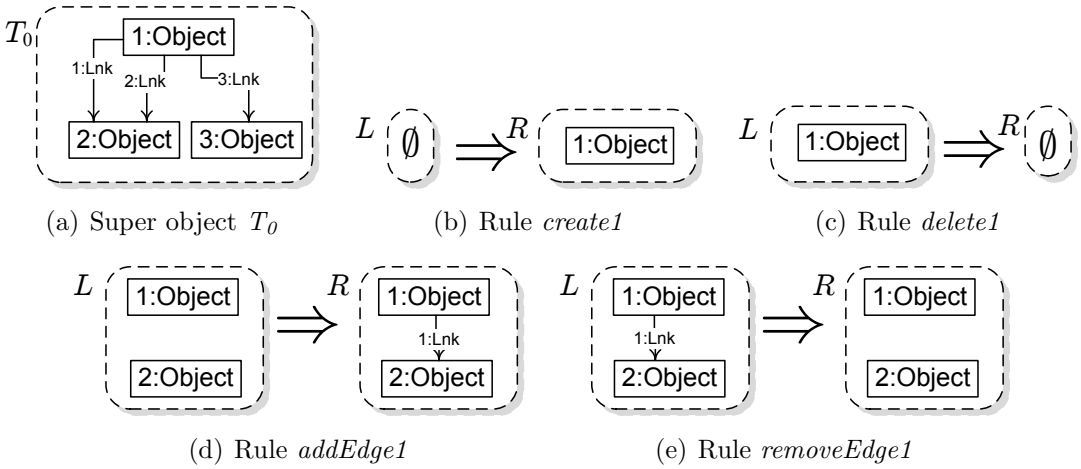
Definition 2 (Direct Derivations)**.** Let $\mathcal{S} = \langle T, P, \pi \rangle$ be a Subobject Transformation System, $\pi(q) = \langle L, K, R \rangle$ be a production, and let G be an object of $\mathbf{Sub}(T)$. Then there is a *direct derivation from G to G' using q*, written $G \overset{q}{\Rightarrow} G'$, if $G' \in \mathbf{Sub}(T)$ and if there exists an object $D \in \mathbf{Sub}(T)$ such that:

$$\begin{array}{llll} (i) & L \cup D \cong G; & (iii) & D \cup R \cong G'; \\ (ii) & L \cap D \cong K; & (iv) & D \cap R \cong K. \end{array}$$

If a production is applicable it is shown in [3] that STS transformations coincide with DPO derivations in $\mathbf{C}$. Note that the condition $G' \in \mathbf{Sub}(T)$ can be characterized by the condition $R \cap G \subseteq L$. Thus there is a special application condition in an STS, which ensures that existing parts in G are not added by a rule, otherwise the DPO step in $\mathbf{C}$ would cause an additional creation an thus, the step would lead to a non-monomorphic embedding of G' into T meaning that G' would not be a subobject of T.

In the following we explain how an STS $\mathcal{S}$ is obtained from a given derivation d of an adhesive grammar GG. This construction is called process of d and we denote the STS $\mathcal{S}$ by $Prc(d)$. Based on the derived STS we can compute the dependency relations on the rules of $\mathcal{S}$. This leads to a partial order, which completely specifies the dependencies of the steps of the derivation d. The super object T is computed as colimit of the derivation diagram. By composition of the existing monomorphisms all objects are automatically embedded monomorphically into T.

Definition 3 (STS of a derivation)**.** *Let $GG = \langle Q, \pi \rangle$ be a graph transformation system, and let $d = (G_0 \overset{q_1,m_1}{\Longrightarrow} \ldots \overset{q_n,m_n}{\Longrightarrow} G_n)$ be a derivation of GG. The* STS generated from d *is defined as $Prc(d) = \langle T, P, \hat{\pi} \rangle$, where T is the colimit object of the diagram underlying the derivation d, $P = \{k \mid d_k = (G_{k-1} \overset{q_k,m_k}{\Longrightarrow} G_k)$ is a step of $d\}$, and $\hat{\pi}(k) = \langle L_k, K_k, R_k \rangle$, where $q_k = (L_k \leftarrow K_k \rightarrow R_k)$.*



(a) Super object T_0 (b) Rule *create1* (c) Rule *delete1*

(d) Rule *addEdge1* (e) Rule *removeEdge1*

Figure 2: STS $\mathcal{S}_0$

Example 2 (STS $\mathcal{S}_0$ for GG_0)**.** *The components of the STS $\mathcal{S}_0 = (T_0, P_0, \pi_0)$ are shown in Fig. 2. The super object T_0 shows three occurrences of nodes of the type "Object" and three occurrences of edges of the type "Lnk". It is the colimit of the derivation d in Example 1. In this case it is equal to the intermediate graph G_2 of d but it can be bigger than any intermediate graph for other derivations. All rule occurrences in d are instantiated as rules in $\mathcal{S}_0$. The set of production names P contains $create[i]$, $delete[i]$, $addEdge[i]$ and $removeEdge[i]$ for $i = (1, 2, 3)$, where the shown rules are for $i = 1$ and the rule $removeEdge[i]$ is inverse to $addEdge[i]$. Several dependencies can be calculated by relations for STSs. For instance the rule addEdge1 is in "read causality"-relation to the rules create1 and create2. This means that addEdge1 needs elements, which are created by create1 and create2. Therefore, a derivation sequence $G \Rightarrow \ldots \overset{addEdge1}{\Longrightarrow} H$ will always contain both rules, create1 and create2 in the beginning or the nodes "1:Object" and "2:Object" are present in the subobject G already.*

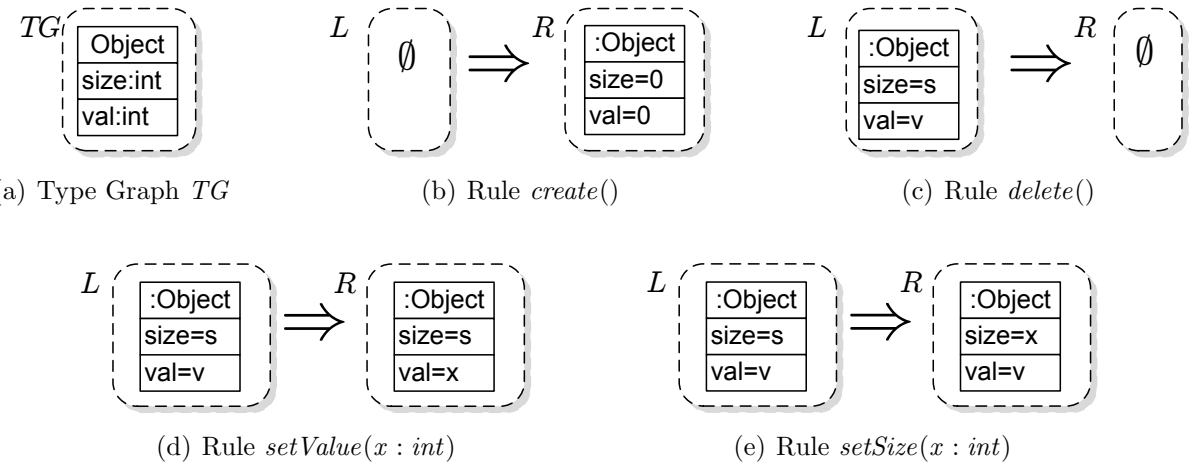
A sequence of rule names with each name occurring exactly once and where all dependencies are respected according to the relations is called a linearization of the STS. Each linearization corresponds to an equivalent derivation d' starting at $G_0 = \emptyset$ and ending at $G_{12} = \emptyset$ as it is the case for d. For example the sequence $s = c1; c2; a1; r1; c3; a2; a3; r2; d2; r3; d3$ with $c = create, a = addEdge, r = removeEdge, d = delete$ defines a corresponding derivation d' in GG_0, which is switching equivalent to d. Each production of $\mathcal{S}_0$ corresponds to a step of the derivation d and hence, it occurs exactly once in a linearization of the process.

Based on the construction of an STS to a given derivation by applying *Prc* it is possible to automatically calculate all switch-equivalent derivations to a given one. The next section shows how this construction can be extended to derivations with arbitrary matching in weak adhesive HLR categories.

3 STSs for weak Adhesive HLR Systems

In this section we show how the process construction for derivations with monomorphic matching in adhesvie rewriting systems as described in Section 2 can be extended to derivations with arbitrary matching in weak adhesive HLR systems [5]. Adhesive high level replacement (AHLR) systems [5] were introduced as a generalization of adhesive rewriting systems in order to additionally cover some sophisticated transformation systems with concepts like attribution and inheritance. In order to analyze the process of such systems the construction of a corresponding STS has to be extended. First of all the construction has to be performed along morphisms of the special class $\mathcal{M}$ of an AHLR category instead of general monomorphisms as before. Furthermore, since matches in AHLR systems cannot be restricted to $\mathcal{M}$-morphisms for practical reasons these morphisms have to be decomposed, such that the process will be build on the components with $\mathcal{M}$-morphisms only. For this purpose we require two conditions for the underlying categories, being the existence of effective unions and $\mathcal{E}$-$\mathcal{M}$-factorization for the morphism class of the matches. We show that these properties are fulfilled by the weak adhesive HLR category $(\mathbf{AGraphs_{ATG}}, \mathcal{M})$ for attributed graphs.

In this section we explain the formal conditions which have to be shown for this challenge and provide the corresponding proofs. The suggested constructions and solutions are explained by means of the following intuitive and compact example of an attributed graph transformation system.



(a) Type Graph *TG* (b) Rule *create*() (c) Rule *delete*()

(d) Rule $setValue(x : int)$ (e) Rule $setSize(x : int)$

Figure 3: Graph Grammar *GG*

Example 3 (Graph Grammar GG)**.** *Rules in Fig. 3 define simple operations on objects with two attributes, being "size" and "val", both of type "int". For keeping the example compact the type graph TG of the graph grammar $GG = (TG, S, R)$ contains only one node "Object" containing these attributes. The start graph $S = \emptyset$ is empty and the set of rules R consists of the four given rules in Fig. 3. The rule "create()" has an empty left hand side meaning that there is no precondition for applying the rule, and the effect is an insertion of a new node of type "Object" with both attributes set to* 0*. The opposite behaviour is defined by "delete()". Finally, the rule "setValue(x:int)" changes the attribute "val" of a selected object and the analogous rule setSize(x:int) changes the attribute "size" instead.*

Weak adhesive HLR categories fulfill the following three conditions, which are less restrictive than the conditions for adhesive categories - mainly in the way that the conditions have to hold only for a subclass $\mathcal{M}$ of monomorphisms and furthermore, some of the considered morphisms in the conditions are additionally required to be $\mathcal{M}$-morphisms. Weak adhesive categories are a generalization of adhesive categories, for which the VK-property in condition 3 is required with less preconditions.

Definition 4 (weak adhesive HLR category)**.** *A category* $\mathbf{C}$ *with a morphism class* $\mathcal{M}$ *is called a weak adhesive HLR category if:*

1. *$\mathcal{M}$ is a class of monomorphisms closed under isomorphisms, composition ($f : A \to B \in \mathcal{M}, g : B \to C \in \mathcal{M} \Rightarrow g \circ f \in \mathcal{M}$), and decomposition ($g \circ f \in \mathcal{M}, g \in \mathcal{M} \Rightarrow f \in \mathcal{M}$).*

2. *$\mathbf{C}$ has pushouts and pullbacks along $\mathcal{M}$-morphisms, and $\mathcal{M}$-morphisms are closed under pushouts and pullbacks.*

3. *Pushouts in $\mathbf{C}$ along $\mathcal{M}$-morphisms are weak VK squares, i.e. the VK square property holds for all commutative cubes with $m \in \mathcal{M}$ and ($f \in \mathcal{M}$ or $b, c, d \in \mathcal{M}$) (see Fig. 4).*

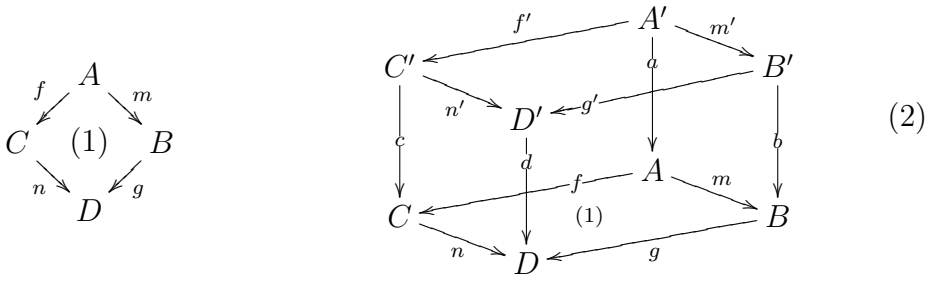


Figure 4: Pushout (1) and commutative cube (2) for VK property.

Based on the class $\mathcal{M}$ of monomorphisms of a weak adhesive HLR category we define the category of $\mathcal{M}$-subobjects of a given object T in order to extend the construction of an STS to derivations in a weak adhesive HLR system.

Definition 5 (Category of $\mathcal{M}$-Subobjects)**.** Let $\mathbf{C}$ be a (weak) AHLR category with $\mathcal{M}$ its subclass of monomorphisms and T be an object in $\mathbf{C}$. The category of $\mathcal{M}$-subobjects of $\mathbf{C}$ is called $\mathbf{Sub}_{\mathcal{M}}(T)$. Its objects are the subobjects $[a : A \to T]$ of T, where a is an $\mathcal{M}$-morphism. A morphism $f : A \to B$ in $\mathbf{Sub}_{\mathcal{M}}(T)$ is compatible with the implicit $\mathcal{M}$-morphisms, i.e. $b \circ f = a$, implying that f is an $\mathcal{M}$-morphism in $\mathbf{C}$ as well and we write $A \subseteq B$ for short.

Using the notion of $\mathcal{M}$-subobjects we can now modify the constructions "intersection" and "union" which were defined for subobjects in Sec. 2.

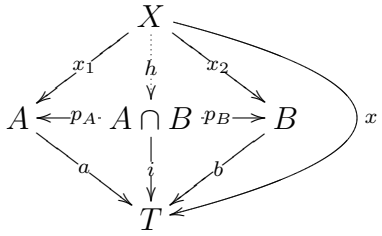
Definition 6 (Intersection and Union in $\mathbf{Sub}_{\mathcal{M}}(T)$)**.** Let $\mathbf{C}$ be a (weak) AHLR category with $\mathcal{M}$ its subclass of monomorphisms and T be an object in $\mathbf{C}$. Let $\mathbf{Sub}_{\mathcal{M}}(T)$ be the subcategory of $\mathcal{M}$-subobjects of T. The intersection of two objects A and B in $\mathbf{Sub}_{\mathcal{M}}(T)$ is the product in $\mathbf{Sub}_{\mathcal{M}}(T)$. The union of A and B in $\mathbf{Sub}_{\mathcal{M}}(T)$ is the coproduct in $\mathbf{Sub}_{\mathcal{M}}(T)$.

We now characterize the constructions "intersection" and "union" for $\mathcal{M}$-subobjects in order to show the further properties - in particular the distributivity - which are needed for the construction and analysis of an STS of a given derivation.

Lemma 1 (Intersection in $\mathbf{Sub}_{\mathcal{M}}(T)$)**.** *Let $\mathbf{C}$ be a (weak) AHLR category with $\mathcal{M}$ its subclass of monomorphisms and T be an object in $\mathbf{C}$. Let $\mathbf{Sub}_{\mathcal{M}}(T)$ be the subcategory of $\mathcal{M}$-subobjects of T. The intersection $A \cap B$ of two objects A and B in $\mathbf{Sub}_{\mathcal{M}}(T)$ exists and is given by the pullback (1) in $\mathbf{C}$ with the $\mathcal{M}$-morphism $i : A \cap B \xrightarrow{a \circ p_A} T$.*

$$\begin{array}{ccc} A \cap B & \xrightarrow{p_A} & A \\ {\scriptstyle p_B}\downarrow & (1) & \downarrow{\scriptstyle a} \\ B & \xrightarrow[b]{} & T \end{array}$$

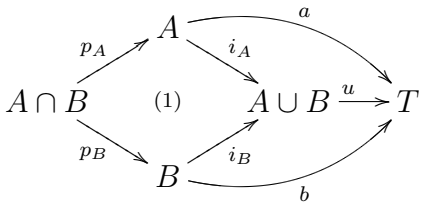
Proof. Let A, B be two objects in $\mathbf{Sub}_{\mathcal{M}}(T)$ and (1) be a pullback in $\mathbf{C}$. The pullback exists, because $a \in \mathcal{M}$. The projections p_A, p_B are in $\mathcal{M}$, because $a, b \in \mathcal{M}$ and $\mathcal{M}$ is closed under pullbacks. Furthermore, p_A, p_B are morphisms in $\mathbf{Sub}_{\mathcal{M}}(T)$ by the commutativity of the pullback construction and the definition of ab. Now, a comparison object X for the product $A \cap B$ in $\mathbf{Sub}_{\mathcal{M}}(T)$ is also a comparison object for the pullback $A \cap B$ in $\mathbf{C}$, because every diagram in $\mathbf{Sub}_{\mathcal{M}}(T)$ commutes. Thus, there is a unique morphism h satisfying the universal property of both, the pullback in $\mathbf{C}$ and the product in $\mathbf{Sub}_{\mathcal{M}}(T)$. Furthermore, $h \in \mathcal{M}$ by decomposition of x_1 and h is a morphism in $\mathbf{Sub}_{\mathcal{M}}(T)$ by the commutativity of the diagram beneath. □



We now show that the union for the weak adhesive HLR category $\mathbf{AGraphs_{ATG}}$ exists and can be constructed within $\mathbf{AGraphs_{ATG}}$ as the pushout in $\mathbf{AGraphs_{ATG}}$ over the

intersection in $\mathbf{Sub}_{\mathcal{M}}(T)$. The main part here is to show that unions are effective, i.e. the constructed object is again a subobject in $\mathbf{Sub}_{\mathcal{M}}(T)$.

Theorem 2 (Union in $\mathbf{Sub}_{\mathcal{M}}(T)$ for $(\mathbf{AGraphs}_{\mathbf{ATG}}, \mathcal{M})$)**.** *Let T be an object in $\mathbf{AGraphs}_{\mathbf{ATG}}$ and $\mathcal{M}$ be the class of monomorphisms, which are isomorphisms on the data part. The union $A \cup B$ of two objects A and B in $\mathbf{Sub}_{\mathcal{M}}(T)$ exists and is given by the pushout* (1) *over the intersection $A \cap B$ with the induced $\mathcal{M}$-morphism $u : A \cup B \to T$ of the pushout* (1)*.*



Proof. The intersection exists by Lemma 1 and the pushout exists, because $p_A \in \mathcal{M}$. The induced morphism u is monomorphic using Thm. 4.7 in [10] and the existence of pullbacks along $\mathcal{M}$-morphisms. It remains to show that u is also an $\mathcal{M}$-morphisms. This means that u is additionally an isomorphism on the data part. Since a and i_A are $\mathcal{M}$-morphisms they are isomorphisms on the data part and using the inverse isomorphism of i_A on its data part we have that u is an isomorphism on the data part by the composition of isomorphisms. □

Theorem 3 (Distributivity)**.** *The union and intersection constructions in $\mathbf{Sub}_{\mathcal{M}}(T)$ are distributive, i.e. $(i) : A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ and $(ii) : A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$.*

Proof. We can perform the same steps as for the proof in Corollary 4.8 of [10], because all Morphisms in the constructed cube are in $\mathcal{M}$ and thus, the van Kampen property holds as required using the general result for pullback decomposition. We derive $(i) : A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$. The direction $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$, which is not explicitly shown in [10], follows directly:

$(A \cup B) \cap (A \cup C)$
$= ((A \cup B) \cap A) \cup ((A \cup B) \cap C)$ using (i)
$= A \cup ((A \cup B) \cap C)$ using $(A \cup B) \cap A = A$ given by $A \subseteq A \cup B$
$= A \cup (A \cap C) \cup (B \cap C)$ using (i)
$= A \cup (B \cap C)$ using $A \cup (A \cap C) = A$ given by $A \cap C \subseteq A$.

□

The further constructions in [3] are based on the intersection and union constructions together with the distributivity result and hence, they can be performed as well for weak adhesive HLR categories with effective unions, like $\mathbf{AGraphs}_{\mathbf{ATG}}$.

We now want to consider derivations with arbitrary matching and therefore, we show how they can be instantiated to derivations with $\mathcal{M}$-matching, such that the previous constructions can be performed again. Even though the rule applications of GG may never identify structural nodes by their matches, because the left hand sides do not contain more than one graph node, it is obvious that different attribute variables of a rule

may be matched to the same value of an instance graph. According to the definition of attribution in the category $\mathbf{AGraphs_{ATG}}$ in [5] attribute values are nodes of E-graphs, which implies that rule matches will identify E-graph nodes, meaning that the matches are not monomorphic. For this reason a restriction to monomorphic matches will prohibit the analysis of practical examples with attribution. Consider e.g. the presented example. The rule setValue(x:int) shall be applicable for all possible values that can be assigned to an object. This includes those cases in which the attribute size is equal to the same attribute value as the one of the attribute val. Thus, matches cannot be restricted to monomorphic ones in this situation. Therefore, the process construction using STSs is not applicable directly.

Example 4 (Derivation)**.** *Derivation d in Fig. 5 consists of four derivation steps of the graph grammar GG:* $d = \langle d_1; d_2; d_3; d_4 \rangle = \Big\langle G_0 \overset{create(),m_1}{\Longrightarrow} G_1 \overset{create(),m_2}{\Longrightarrow} G_2 \overset{setAttribute(5),m_3}{\Longrightarrow} G_3 \overset{delete(),m_4}{\Longrightarrow} G_4 \Big\rangle$*. Some steps are sequentially independent, while others are not. For instance d_3 changes an attribute of node 1, such that it depends on d_1, which creates this node. Therefore, d_3 can only occur after d_1 in an equivalent derivation. The following table shows all equivalent permutations for the derivation d.*

$\pi_i \backslash$ *step*	d_1	d_2	d_3	d_4
π_1	*1*	*2*	*3*	*4*
π_2	*1*	*3*	*2*	*4*
π_3	*2*	*1*	*3*	*4*
π_4	*2*	*1*	*4*	*3*
π_5	*2*	*4*	*1*	*3*

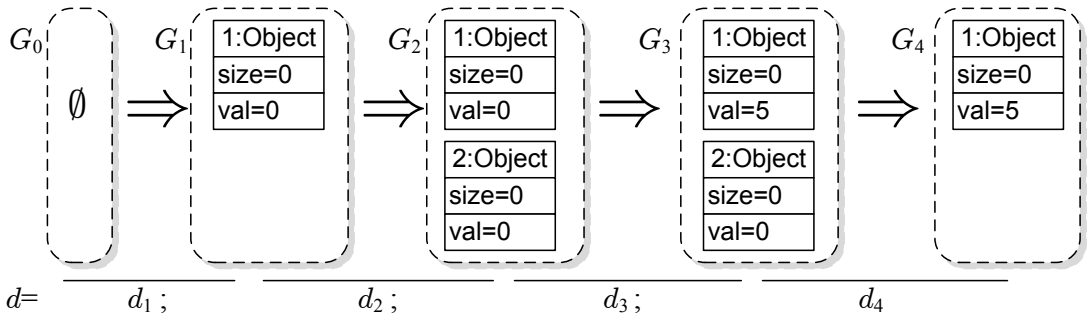
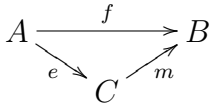


Figure 5: Derivation

In order to automatically derive a process of the derivation d the process construction *Prc* has to be extended to graph grammars with attribution and general matching. Since rules may use terms to specify attribute values it is possible that two terms are matched to the same value as in step d_3, where rule variables s and v are both mapped to 0. This implies that the match in this step is not injective. Therefore the process construction has to be extended to non-injective matches in general. For this purpose we first instantiate

the derivation in the way that all identifications are performed to the rules itself and thereafter we apply the existing process construction. This instantiation is based on $\mathcal{E}-\mathcal{M}$-factorizations of the matches.

Definition 7 ($\mathcal{E}$-$\mathcal{M}$ factorization)**.** **C** *has an* $\mathcal{E}$-$\mathcal{M}$ *factorization for given morphism classes* $\mathcal{E}$ *and* $\mathcal{M}$ *if for each* f *there is a decomposition, unique up to isomorphism,* $f = m \circ e$ *with* $e \in \mathcal{E}$ *and* $m \in M$.



Usually $\mathcal{E}$ is a subclass of epimorphisms and $\mathcal{M}$ is a subclass of monomorphisms. In the context of attributed graphs as in the category $\mathbf{AGraphs_{ATG}}$ the class $\mathcal{E}$ contains all epimorphisms and $\mathcal{M}$ consists of the monomorphisms f, which are injective on its graph components f_V, f_E and isomorphic on the algebra component. We now show that the category $\mathbf{AGraphs_{ATG}}$ has $\mathcal{E}$-$\mathcal{M}$-factorizations for a restricted class of morphisms, which is enough in our case, because matches of derivations are included in this class.

Theorem 4 ($\mathcal{E}$-$\mathcal{M}$-factorization for $(\mathbf{AGraphs_{ATG}}, \mathcal{M})$)**.** *Let* $\mathcal{E}$ *be the class of all epimorphisms in the weak adhesive HLR category* $\mathbf{AGraphs_{ATG}}$ *with* $\mathcal{M}$ *the class of monomorphism that are isomorphisms on the data part. Let* $f = (f_G, f_D) : AG \to AH$ *be a morphism in* $\mathbf{AGraphs_{ATG}}$*, where* $AG = (G, A_G)$ *and* $AH = (H, A_H)$ *are attributed graphs and the algebra* $A_G = T_{Sigma}(X)$ *and* A_H *is term generated. Then, there is an* $\mathcal{E}$-$\mathcal{M}$*-factorization* $(e, m) = ((e_G, e_D), (m_G, m_D))$ *for* f*. The morphisms* (e_G, m_G) *are given by the epi-mono factorization for typed graphs and the morphisms* (e_D, m_D) *are given by the the epi-mono factorization for* $\mathbf{Alg}(\mathbf{\Sigma})$.

Proof. By A.15 in [5] we know that $\mathbf{Graphs_{TG}}$ and $\mathbf{Alg}(\mathbf{\Sigma})$ have epi-mono-factorizations. Thus, e and m uniquely exist. It remains to show that $m \in \mathcal{M}$. Since A_H is term generated we know that $f_D = xeval(ass)$ is surjective and by decomposition also m_D is surjective. By m_D being mono according to the factorization we have that m_D is an isomorphism. Thus, $m \in \mathcal{M}$. □

Remark 1. *In the case that we consider derivations with algebras that are not term generated we can still perform a factorization* $AG \xrightarrow{e} AG' \xrightarrow{m} AH$*, but this time with* e *being a general morphism and* $m \in \mathcal{M}$*. In this case* AG' *is obtained as before on the graph part, but for the data part we use the A-quotient term algebra extended by a new variable for each element not reached by* $xeval(ass) : T_{Sigma}(X) \to A_H$*. Based on this construction we can also construct the STS, because we only use the lower part of the factorized derivation diagrams, where only* $\mathcal{M}$*-morphisms occur.*

Based on the factorization of the matches we instantiate the derivation steps of a derivation leading to a new derivation, where all matches are $\mathcal{M}$-morphisms.

Definition 8 (Instantiation of a derivation)**.** *Let* $G \overset{p,m}{\Longrightarrow} H$ *be a derivation step in a weak adhesive HLR System with* $\mathcal{E}$-$\mathcal{M}$*-factorization for the morphism class of the matches. The instantiated derivation step* $G \overset{p',m'}{\Longrightarrow} H$ *is given by diagrams* (5) *and* (6) *below, which*

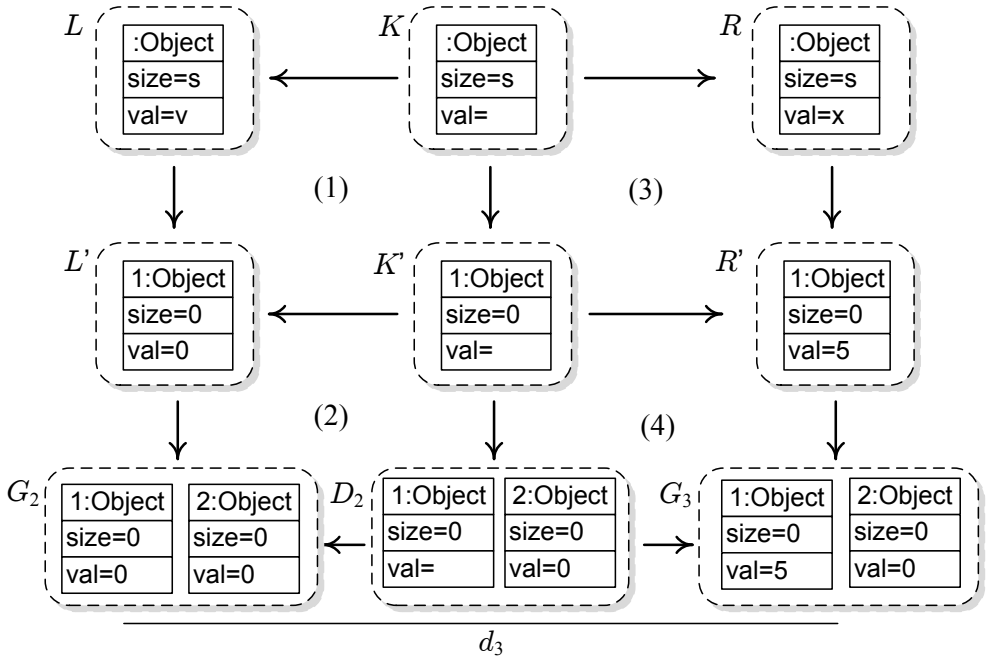
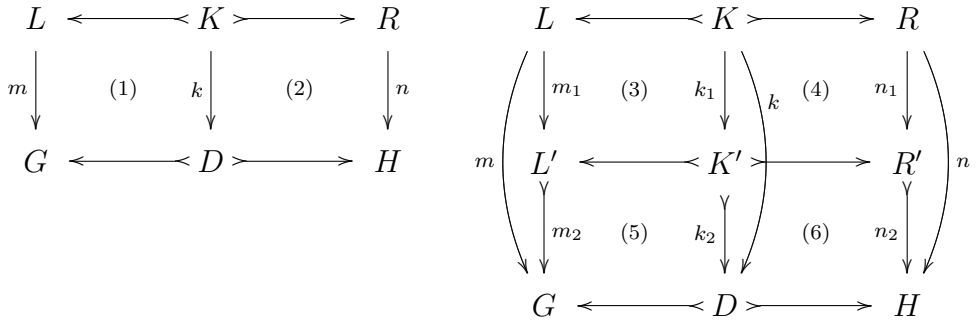


Figure 6: Instantiated derivation step d_3, Rule setAttribute(x:int)

are constructed as follows. The match m is factorized in the $\mathcal{E}$-morphism m_1 and the $\mathcal{M}$-morphism m_2. Diagram (5) *is constructed as pullback, implying that $k_2 \in \mathcal{M}$, because $\mathcal{M}$ is closed under pullbacks. Morphism k_1 is obtained as the induced morphism from the pullback* (1) *into K' and we have that* (3) *is a pullback by pullback decomposition and* (5) *is a pushout by the pushou-pullback decomposition lemma for weak adhesive HLR categories. Diagram* (4) *is obtained as a pushout and the morphism n_2 is derived as the induced morphism from the pushout* (4) *into H. Thus,* (6) *is a pushout by pushout decomposition making* ((4), (6)) *a DPO diagram.*

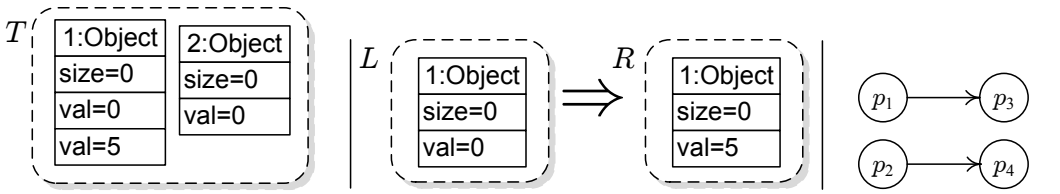


Let d be a derivation in a weak adhesive HLR System with $\mathcal{E}$-$\mathcal{M}$-factorization for the morphism class of the matches. The instantiated derivation d' is derived by instantiating each derivation step as defined above.

Example 5 (Instantiation)**.** *We show the instantiation of the derivation d exemplarily at step d_3. Here, the attribute variables are identified by the match to the same value* 0 *making*

the match non-injective. If identifications of graph nodes occur, which is not the case in the example, they are handled equivalently. Fig. 6 shows the step d_3 in the top most and bottom line. The intermediate line contains the instantiated rule. All together squares (2) and (4) are pushouts and applying this construction to the complete derivation d we have a new instantiated DPO derivation with matches in $\mathcal{M}$ only.

Using the instantiation of a derivation we can again apply the process construction *Prc* on a derivation diagram with $\mathcal{M}$-morphisms only. The resulting STS of the example derivation d is constructed by applying *Prc* to the instantiated derivation d' of d. The results of an analysis of a derived STS concern the lower DPO diagrams of an instantiated derivation d at first. But the computed equivalent derivations for the lower part lead to equivalent derivations of d by composing the new DPO diagrams with the corresponding but unchanged DPO diagrams of the upper line of the instantiation.



(a) Super Object T (b) STS rule p_3 (c) Dependencies

Figure 7: Parts of STS S

Example 6 (STS with Attribution)**.** *Applying Prc to the derivation d in Example 4 we derive an STS $S = (T, P, \pi)$, with the super object T as shown in Fig. 7(a) constructed as the colimit of the lower line in the derivation diagram and in our example it is equal to the pushout of $G_2 \leftarrow D_2 \rightarrow G_3$ in Fig. 6. The set of production names $P = \{p_1, p_2, p_3, p_4\}$ contains four elements which are mapped by π to the rules, which are instantiations of the rules in GG. For instance, the rule name p_3 is mapped to an instantiation of "setAttribute(x:int)" as shown in Fig. 7(b). Now, STS relations presented in [3] can be used to calculate a concrete dependency relation Dep and all conflicts Confl for all rules p_i in P. In this example we have no conflicts and $Dep = \{(p_1, p_3), (p_2, p_4)\}$ as visualized in Fig. 7(c).*

Remark 2 (Multiple Attribute Values)**.** *Attribute values in the category* $\mathbf{AGraphs_{ATG}}$ *are defined as edges to actual value nodes. Therefore, a correctly typed node is allowed to have zero or multiple attribute values by having zero or multiple edges of the same attribution type. This loose typing is needed, e.g. to specify attribute changes by a rule, because an attribute edge has to be deleted and recreated again to connect the node to a new data value. Furthermore, multiple attribute values are of main interest in the context of process construction. When applying Prc to construct an STS each occurrence of a node and an edge is added to the super object T separately. This means that elements, which are deleted and created again, appear as different elements in T. Thus, after changing an attribute*

value as in rule setValue($x : int$), the edge connecting the object node with its attribute is deleted and created again to the new value. This leads to two separate edges in T, which is the case in the example super object T with value 0 and 5 for the node of type "Object". If this multiple structure would not be created, derivation steps, which depend on each other at attributes, would be recognized misleadingly as independent.

The example has shown how processes of derivations with attributes and non-injective matching are constructed. More complex processes can be calculated automatically and the formal results justify the correctness of a process analysis based on the derived STS.

4 Conclusion

Process definitions for graph grammars [2, 1] have been studied in analogy to and as generalization of existing process constructions for Petri nets given by occurrence nets. These constructions are limited to monomorphic matches and plain graph grammars without attribution only.

Extension The extension to arbitrary matches and attribution is the main contribution of this paper. It was presented on behalf of an intuitive and compact example, but the construction is highly scalable. A special condition for the new construction is the existence of $\mathcal{E}$-$\mathcal{M}$-factorizations of matches in the regarded category of the transformation system. Most categories for graphs, such as typed attributed graphs, as well as many other categories have $\mathcal{E}$-$\mathcal{M}$-factorizations for all or a suitable subclass of morphisms, which ensures that the condition is no restriction of the approach for practical cases. The formal foundation and correctness were shown in this paper.

Further Extensions Practical applications of graph grammars often use negative application conditions (NACs) for restricting the application situations of the rules. In order to extend the presented process definition to handle also NACs we will instantiate NACs. Furthermore, there are new conflict situations in a derivation with NACs. They will have to be captured by new relations in the corresponding STS. Moreover, the notion of switch equivalence has to be generalized, because there are derivations with NACs that shall be seen to be equivalent, but they are not switch-equivalent using sequential independence with NACs. The reason is that rule applications may be possible in an equivalent way at several positions, which are not situated next to each other and these permutations cannot be derived by switching independent steps.

The presented extension of process definitions of graph transformations opens new possibilities for the analysis of object oriented systems including the analysis of dependencies caused by attribute changes. Case studies will benchmark the power of this approach and may inspire further extensions.

References

[1] P. Baldan. *Modelling Concurrent Computations: from Contextual Petri Nets to Graph Grammars*. PhD thesis, Computer Science Department - University of Pisa, 2000.

[2] P. Baldan, A. Corradini, T. Heindel, B. König, and P. Sobocinski. Processes for adhesive rewriting systems. In L. Aceto and A. Ingólfsdóttir, editors, *FoSSaCS*, volume 3921 of *Lecture Notes in Computer Science*, pages 202–216. Springer, 2006.

[3] A. Corradini, F. Hermann, and P. Sobociński. Subobject Transformation Systems. *Applied Categorical Structures*, February 2008.

[4] H. Ehrig, K. Ehrig, C. Ermel, F. Hermann, and G. Taentzer. Information preserving bidirectional model transformations. In M. B. Dwyer and A. Lopes, editors, *Fundamental Approaches to Software Engineering*, volume 4422 of *LNCS*, pages 72–86. Springer, 2007.

[5] H. Ehrig, K. Ehrig, U. Prange, and G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. EATCS Monographs in Theoretical Computer Science. Springer, 2006.

[6] H. Ehrig, A. Habel, H.-J. Kreowski, and F. Parisi-Presicce. Parallelism and concurrency in high level replacement systems. Technical Report 90-35, Technical University of Berlin, 1990.

[7] H. Ehrig, A. Habel, H.-J. Kreowski, and F. Parisi-Presicce. From graph grammars to high level replacement systems. In *4th Int. Workshop on Graph Grammars and their Application to Computer Science*, volume 532 of *LNCS*, pages 269–291. Springer, 1991.

[8] C. Ermel, K. Ehrig, G. Taentzer, and E. Weiss. Object Oriented and Rule-based Design of Visual Languages using TIGER. In *Proc. Third International Workshop on Graph-Based Tools (GraBaTs'06)*, volume 1, Natal, Brazil, September 2006. Electronic Communications of the EASST.

[9] F. Hermann and H. Ehrig. Process Definition using Subobject Transformation Systems. *EATCS Bulletin*, 95:153–163, 2008.

[10] S. Lack and P. Sobociński. Adhesive Categories. In *Proc. FOSSACS 2004*, volume 2987 of *LNCS*, pages 273–288. Springer, 2004.

[11] W. Reisig. *Petri Nets: An Introduction*, volume 4 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1985.

[12] G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*. World Scientific, 1997.