# Fundamental Theory for Typed Attributed Graph Transformation

Hartmut Ehrig, Ulrike Prange, and Gabriele Taentzer

Technical University of Berlin, Germany
{ehrig,ullip,gabi}@cs.tu-berlin.de

**Abstract.** The concept of typed attributed graph transformation is most significant for modeling and meta modeling in software engineering and visual languages, but up to now there is no adequate theory for this important branch of graph transformation. In this paper we give a new formalization of typed attributed graphs, which allows node and edge attribution. The first main result shows that the corresponding category is isomorphic to the category of algebras over a specific kind of attributed graph structure signature. This allows to prove the second main result showing that the category of typed attributed graphs is an instance of "adhesive HLR categories". This new concept combines adhesive categories introduced by Lack and Sobociński with the well-known approach of high-level replacement (HLR) systems using a new simplified version of HLR conditions. As a consequence we obtain a rigorous approach to typed attributed graph transformation providing as fundamental results the Local Church-Rosser, Parallelism, Concurrency, Embedding and Extension Theorem and a Local Confluence Theorem known as Critical Pair Lemma in the literature.

## 1 Introduction

The algebraic theory of graph transformation based on labeled graphs and the double-pushout approach has already a long tradition (see [1]) with various applications (see [2, 3]). Within the last decade graph transformation has been used as a modeling technique in software engineering and as a meta-language to specify and implement visual modeling techniques like the UML. Especially for these applications it is important to use not only labeled graphs as considered in the classical approach [1], but also typed and attributed graphs. In fact, there are already several different concepts for typed and attributed graph transformation in the literature (see e.g. [4–6]). However, there is no adequate theory for this important branch of graph transformation up to now. The key idea in [5] is to model an attributed graph with node attribution as a pair $AG = (G, A)$ of a graph $G$ and a data type algebra $A$. In this paper we use this idea to model attributed graphs with node and edge attribution, where $G$ is now a new kind of graph, called E-graph, which allows also edges from edges to attribute nodes. This new kind of attributed graphs combined with the concept of typing leads to a category $\mathbf{AGraphs_{ATG}}$ of attributed graphs typed over an attributed type

H. Ehrig et al. (Eds.): ICGT 2004, LNCS 3256, pp. 161–177, 2004.

graph *ATG*. This category seems to be an adequate formal model not only for various applications in software engineering and visual languages, but also for the internal representation of attributed graphs in our graph transformation tool AGG [7].

The main purpose of this paper is to provide the basic concepts and results of graph transformation known in the classical case [1] for this new kind of typed attributed graphs. The straightforward way would be to extend the classical theory in [1] step by step first to attributed graphs and then to typed attributed graphs. In this paper we propose the more elegant solution to obtain the theory of typed attributed graph transformation as an instantiation of the corresponding categorical theory developed in [8]. In [8] we have proposed the new concept of "adhesive HLR categories and systems", which combines the concept of "adhesive categories" presented by Lack and Sobociński in [9] with the concept of high-level replacement systems, short HLR systems, introduced in [10]. In [8] we have shown that not only the Local Church-Rosser, Parallelism and Concurrency Theorem - presented already in [10] for HLR systems -, but also several other results known from the classical theory [1, 9] are valid for adhesive HLR systems satisfying some additional HLR properties.

For this purpose we have to show that the category $\mathbf{AGraphs_{ATG}}$ of typed attributed graphs is an adhesive HLR category in this sense. In Thm. 1 we show that the category $\mathbf{AGraphs_{ATG}}$ is isomorphic to a category of algebras over a suitable signature $AGSIG(ATG)$, which is uniquely defined by the attributed type graph *ATG*. In fact, it is much easier to verify the categorical properties required for adhesive HLR categories for the category of algebras **AGSIG(ATG)-Alg** and to show the isomorphism between **AGSIG(ATG)-Alg** and $\mathbf{AGraphs_{ATG}}$, than to show the categorical properties directly for the category $\mathbf{AGraphs_{ATG}}$. In Thm. 2 we show that **AGSIG(ATG)-Alg** and hence also $\mathbf{AGraphs_{ATG}}$ is an adhesive HLR category. In fact, we show this result for the category **AGSIG-Alg**, where *AGSIG* is a more general kind of attributed graph structure signature in the sense of [4, 11, 12]. Combining the main results of this paper with those of [8] we are able to show that the following basic results shown in Thm. 3 - 5 are valid for typed attributed graph transformation:

1. Local Church-Rosser, Parallelism and Concurrency Theorem,
2. Embedding and Extension Theorem,
3. Local Confluence Theorem (Critical Pair Lemma).

Throughout the paper we use a running example from the area of model transformation to illustrate the main concepts and results. We selected a small set of model elements, basic for all kinds of object-oriented models. It describes the abstract syntax, i.e. the structure of method signatures. These structures are naturally represented by node and edge attributed graphs where node attributes store e.g. names, while edge attributes are useful to keep e.g. the order of parameters belonging to one method. Attributed graph transformation is used to specify simple refactorings on this model part such as adding a parameter,

exchanging two parameters, etc. Usually such refactorings are not always independent of each other. Within this paper we analyse the given refactoring rules concerning potential conflicts and report them as critical pairs.

Node and edge attributed graphs build the basic structures in the graph transformation environment AGG [7]. The attribution is done by Java objects and expressions. We use AGG to implement our running example and to compute all its critical pairs. In GenGED [13], a visual environment for the definition of visual languages, the internal structures are $AGSIG$-algebras for attributed graph structure signatures $AGSIG$ discussed above.

This paper is organized as follows. In section 2 we introduce node and edge attributed graphs and typing and present our first main result. Typed attributed graphs in the framework of adhesive HLR categories are discussed in section 3 together with our second main result. This allows to present the theory of typed attributed graph transformation in section 4 as an instance of the general theory in [8]. Finally we discuss related work and future perspectives in section 5.

For lack of space we can only present short proof ideas in this paper and refer to our technical report [14] for more detail.

## 2 Node and Edge Attributed Graphs and Typing

In this section we present our new notion of node and edge attributed graphs, which generalizes the concept of node attributed graphs in [5], where node attributes are modelled by edges from graph nodes to data nodes. The new concept is based on graphs, called E-graphs, which allows also edges from graph edges to data nodes in order to model edge attributes. An attributed graph $AG = (G, D)$ consists of an E-graph $G$ and a data type $D$, where parts of the data of $D$ are also vertices in $G$. This leads to the category **AGraphs** of attributed graphs and $\mathbf{AGraphs_{ATG}}$ of typed attributed graphs over an attributed type graph $ATG$. The main result in this section shows that $\mathbf{AGraphs_{ATG}}$ is isomorphic to a category **AGSIG(ATG)-Alg** of algebras over a suitable signature $AGSIG(ATG)$, which is in one-to-one correspondence with $ATG$.

In our notion of E-graphs we distinguish between two kinds of vertices, called graph and data vertices, and three different kinds of edges, according to the different roles they play for the representation and implementation of attributed graphs.

**Definition 1 (E-graph).** *An E-graph* $G = (V_1, V_2, E_1, E_2, E_3, (source_i, target_i)_{i=1,2,3})$ *consists of sets*

- $V_1$ *and* $V_2$ *called graph resp. data nodes,*
- $E_1$, $E_2$, $E_3$ *called graph, node attribute and edge attribute edges respectively,*

*and source and target functions*

- $source_1 : E_1 \to V_1$, $source_2 : E_2 \to V_1$, $source_3 : E_3 \to E_1$,
- $target_1 : E_1 \to V_1$, $target_2 : E_2 \to V_2$, $target_3 : E_3 \to V_2$.

*An E-graph morphism* $f : G_1 \to G_2$ *is a tuple* $(f_{V_1}, f_{V_2}, f_{E_1}, f_{E_2}, f_{E_3})$ *with* $f_{V_i} : G_{1,V_i} \to G_{2,V_i}$ *and* $f_{E_j} : G_{1,E_j} \to G_{2,E_j}$ *for* $i = 1, 2$, $j = 1, 2, 3$ *such that* $f$ *commutes with all source and target functions.*
*E-graphs combined with E-graph morphisms form the category* **EGraphs**.

The following notions of attributed and typed attributed graphs are in the spirit of (node) attributed graphs of [5], where graphs are replaced by E-graphs in order to allow node and edge attribution. A data signature $DSIG$ is an ordinary algebraic signature (see [15]).

**Definition 2 (attributed graph).** *Consider a data signature* $DSIG = (S_D, OP_D)$ *with attribute value sorts* $S'_D \subseteq S_D$. *An attributed graph* $AG = (G, D)$ *consists of an E-graph* $G$ *together with a* $DSIG$*-algebra* $D$ *such that* $\dot{\cup}_{s \in S'_D} D_s = G_{V_2}$.
*An attributed graph morphism is a pair* $f = (f_G, f_A)$ *with an E-graph morphism* $f_G$ *and an algebra homomorphism* $f_A$ *such that (1) is a pullback for all* $s \in S'_D$.

$$\begin{array}{ccc} D_{1,s} & \xrightarrow{f_{A,s}} & D_{2,s} \\ \downarrow & (1) & \downarrow \\ G_{1,V_2} & \xrightarrow{f_{G,V_2}} & G_{2,V_2} \end{array}$$

*Attributed graphs and attributed graph morphisms form the category* **AGraphs**.

*Remark 1.* The pullback property for the graph morphism is required for Thm. 1, otherwise the categories in this theorem would not be isomorphic.

**Definition 3 (typed attributed graph).** *An attributed type graph is an attributed graph* $ATG = (TG, Z)$ *where* $Z$ *is the final* $DSIG$*-algebra.*
*A typed attributed graph* $(AG, t)$ *over* $ATG$ *consists of an attributed graph* $AG$ *together with an attributed graph morphism* $t : AG \to ATG$. *A typed attributed graph morphism* $f : (AG_1, t_1) \to (AG_2, t_2)$ *is an attributed graph morphism* $f : AG_1 \to AG_2$ *such that* $t_2 \circ f = t_1$.
*Typed attributed graphs over* $ATG$ *and typed attributed graph morphisms form the category* $\mathbf{AGraphs_{ATG}}$. *The class of all attributed type graphs* $ATG$ *is denoted by* **ATG-Graphs**.

*Example 1 (typed attributed graphs).* Given suitable signatures $CHAR$, $STRING$ and $NAT$, the data signature $DSIG$ is defined by
$DSIG = CHAR + STRING + NAT +$
sorts: ParameterDirectionKind
opns: in, out, inout, return: → ParameterDirectionKind
and the set of all data sorts used for attribution is $S'_D = \{$String, Nat, ParameterDirectionKind$\}$. Fig. 1 shows an attributed type graph $ATG = (TG, Z)$ for method signatures. It is an attributed graph where each data element is named after its corresponding sort, because the final $DSIG$-algebra $Z$ has sorts $Z_s = \{s\}$ for all $s \in S_D$. Note that $TG$ is an E-graph with edge attribute edge "order" from "parameter" to "Nat". An attributed graph $AG$ typed over $ATG$ is given in Fig. 2, where only those algebra elements are shown explicitly which are used

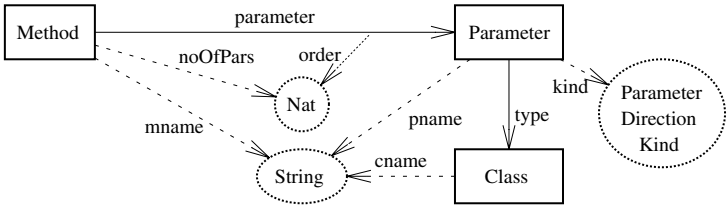



**Fig. 1.**

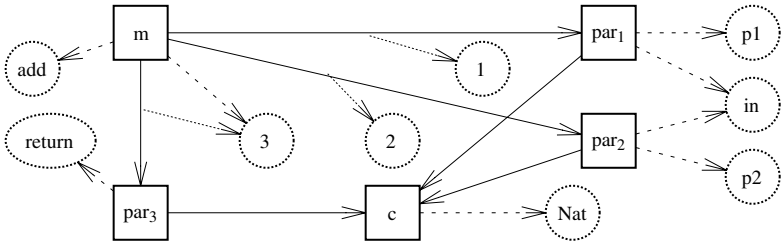



**Fig. 2.**

for attribution. The graph $AG$ is typed over $ATG$ by the attributed graph morphism $t : AG \to ATG$ defined on vertices by $t(\mathrm{m}) = \mathrm{Method}$, $t(\mathrm{par}_1) = t(\mathrm{par}_2) = t(\mathrm{par}_3) = \mathrm{Parameter}$, $t(\mathrm{c}) = \mathrm{Class}$, $t(1) = t(2) = t(3) = \mathrm{Nat}$, $t(\mathrm{return}) = t(\mathrm{in}) = \mathrm{ParameterDirectionKind}$ and $t(\mathrm{p1}) = t(\mathrm{p2}) = t(\mathrm{add}) = t(\mathrm{Nat}) = \mathrm{String}$. In AGG, a typed attributed graph like the one in Fig. 2 is depicted in a more compact notation like the graph in Fig. 3. Each node and edge inscription has two compartments. The upper compartment contains the type of a graph element, while the lower one holds its attributes. The attributes are ordered in a list, just for convenience. Nodes and edges are not explicitly named. While the formal concept of an attributed graph allows partial attribution in the sense that there is no edge from a graph node or edge to a data node, this is not possible in AGG. Thus, parameter $\mathrm{par}_3$ has to be named by an empty string. Furthermore, the formal concept allows several outgoing attribute edges from one graph node or edge which is also not possible in AGG. □

The category $\mathbf{AGraphs_{ATG}}$ introduced above is the basis for our theory of typed attributed graph transformation in this paper. In order to prove properties for $\mathbf{AGraphs_{ATG}}$, however, it is easier to represent $\mathbf{AGraphs_{ATG}}$ as a category **AGSIG(ATG)-Alg** of classical algebras (see [15]) over a suitable signature $AGSIG(ATG)$. For this purpose we introduce the notion of general respectively well-structured attributed graph structure signatures $AGSIG$ where the well-structured case corresponds to attributed graph signatures in the LKW-approach [4]. The signature $AGSIG(ATG)$ becomes a special case of a well-structured $AGSIG$.

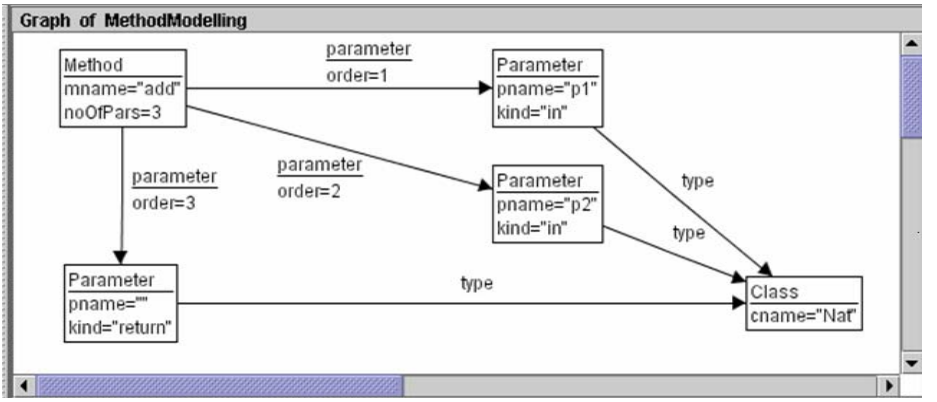



**Fig. 3.**

**Definition 4 (attributed graph structure signature).** *A graph structure signature* $GSIG = (S_G, OP_G)$ *is an algebraic signature with unary operations* $op : s \to s'$ *in* $OP_G$ *only. An attributed graph structure signature* $AGSIG = (GSIG, DSIG)$ *consists of a graph structure signature* $GSIG$ *and a data signature* $DSIG = (S_D, OP_D)$ *with attribute value sorts* $S'_D \subseteq S_D$ *such that* $S'_D = S_D \cap S_G$ *and* $OP_D \cap OP_G = \varnothing$.
*AGSIG is called well-structured if for each* $op : s \to s'$ *in* $OP_G$ *we have* $s \notin S_D$. *The category of all AGSIG-algebras and AGSIG-homomorphisms (see [15]) is denoted by* **AGSIG-Alg**.

**Theorem 1 (Characterization of $\mathbf{AGraphs_{ATG}}$).** *For each attributed type graph ATG there is a well-structured attributed graph structure signature* $AGSIG(ATG)$ *such that* $\mathbf{AGraphs_{ATG}}$ *is isomorphic to the category* **AGSIG(ATG)-Alg** *of* $AGSIG(ATG)$*-algebras:*
$\mathbf{AGraphs_{ATG}} \cong$ **AGSIG(ATG)-Alg**.

*Construction.* Given $ATG = (TG, Z)$ with final $DSIG$-algebra $Z$ we have $TG_{V_2} = \dot{\cup}_{s \in S'_D} Z_s = S'_D$ and define $AGSIG(ATG) = (GSIG = (S_G, OP_G), DSIG)$ with $S_G = S_V \dot{\cup} S_E$ and $S_V = TG_{V_1} \dot{\cup} TG_{V_2}$, $S_E = TG_{E_1} \dot{\cup} TG_{E_2} \dot{\cup} TG_{E_3}$ and $OP_G = \dot{\cup}_{e \in S_E} OP_e$ with $OP_e = \{src_e, tar_e\}$ defined by

- $src_e : e \to v(e)$ for $e \in TG_{E_1}$ with $v(e) = source_1^{TG}(e) \in TG_{V_1}$,
- $tar_e : e \to v'(e)$ for $e \in TG_{E_1}$ with $v'(e) = target_1^{TG}(e) \in TG_{V_1}$,
- $src_e$, $tar_e$ for $e \in TG_{E_2}$ and $e \in TG_{E_3}$ defined analogously.

*Proof idea.* Based on the construction above we are able to construct a functor $F : \boldsymbol{AGraphs_{ATG}} \to \boldsymbol{AGSIG(ATG)\text{-}Alg}$ and a corresponding inverse functor $F^{-1}$. □

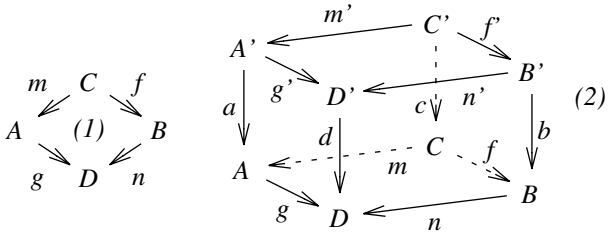
## 3 Typed Attributed Graphs in the Framework of Adhesive HLR Categories

As pointed out in the introduction we are not going to develop the theory of typed attributed graph transformation directly. But we will show that it can be obtained as an instantiation of the theory of adhesive HLR systems, where this new concept (see [8]) is a combination of adhesive categories and grammars (see [9]) and HLR systems introduced in [10]. For this purpose we present in this section the general concept of adhesive HLR categories and we show that **AGSIG-Alg**, **AGSIG(ATG)-Alg** and especially the category $\mathbf{AGraphs_{ATG}}$ of typed attributed graphs are adhesive HLR categories for a suitable class $M$ of morphisms. Moreover our categories satisfy some additional HLR conditions, which are required in the general theory of adhesive HLR systems (see [8]). This allows to apply the corresponding results to typed attributed graph transformation systems, which will be done in the next section.

We start with the new concept of adhesive HLR categories introduced in [8] in more detail.

**Definition 5 (adhesive HLR category).** *A category $\boldsymbol{C}$ with a morphism class $M$ is called adhesive HLR category, if*

1. *$M$ is a class of monomorphisms closed under isomorphisms and closed under composition ($f : A \to B \in M$, $g : B \to C \in M \Rightarrow g \circ f \in M$) and decomposition ($g \circ f \in M$, $g \in M \Rightarrow f \in M$),*
2. *$\boldsymbol{C}$ has pushouts and pullbacks along $M$-morphisms, i.e. if one of the given morphisms is in $M$, then also the opposite one is in $M$, and $M$-morphisms are closed under pushouts and pullbacks,*
3. *pushouts in $\boldsymbol{C}$ along $M$-morphisms are van Kampen (VK) squares, where a pushout (1) is called VK square, if for any commutative cube (2) with (1) in the bottom and pullbacks in the back faces we have: the top is pushout $\Leftrightarrow$ the front faces are pullbacks.*




Important examples of adhesive HLR categories are the category (**Sets**, $M_{inj}$) of sets with class $M_{inj}$ of all injective functions, the category (**Graph**, $M_{inj}$) of graphs with class $M_{inj}$ of injective graph morphisms and different kinds of labelled and typed graphs (see [8]). Moreover all HLR1 and HLR2 categories presented in [10] are adhesive HLR categories. In the following we will show that also our categories **AGSIG-Alg**, **AGSIG(ATG)-Alg** and $\mathbf{AGraphs_{ATG}}$ presented in the previous section are adhesive HLR categories for the class $M$ of

all injective morphisms with isomorphic data type part, which is used for typed attributed graph transformation systems in the next section.

**Definition 6 (class $M$ for typed attributed graph transformation).** *The distinguished class $M$ is defined by $f \in M$ if*

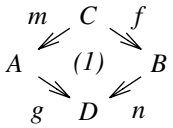
1. *$f_G$ is injective, $f_A$ is isomorphism for $f = (f_G, f_A)$ in* **AGraphs$_{\mathbf{ATG}}$** *and $AG = (G, A)$,*
2. *$f_{GSIG}$ is injective, $f_{DSIG}$ is isomorphism for $f$ in* **AGSIG-Alg** *or* **AGSIG(ATG)-Alg** *and $AGSIG = (GSIG, DSIG)$.*

*Remark 2.* The corresponding categories (**AGraphs$_{\mathbf{ATG}}$**, $M$), (**AGSIG-Alg**, $M$) and (**AGSIG(ATG)-Alg**, $M$) are adhesive HLR categories (see Thm. 2). For simplicity we use the same notation $M$ in all three cases. For practical applications we assume that $f_A$ and $f_{DSIG}$ are identities.

This class $M$ of morphisms is on the one hand the adequate class to define productions of typed attributed graph transformation systems (see Def. 7), on the other hand it allows to construct pushouts along $M$-morphisms componentwise. This is essential to verify the properties of adhesive HLR categories.

**Lemma 1 (properties of pushouts and pullbacks in (AGSIG-Alg, $M$)).**

1. *Given $m : C \to A$ and $f : C \to B$ with $m \in M$ then there is a pushout (1) in* **AGSIG-Alg** *with $n \in M$.*




   *Moreover given (1) commutative with $m \in M$ then (1) is a pushout in* **AGSIG-Alg** *iff (1) is a componentwise pushout in* **Sets**. *If $m \in M$ then also $n \in M$.*
2. *Given $g : A \to D$ and $n : B \to D$ then there is a pullback (1) in* **AGSIG-Alg**. *Moreover given (1) commutative then (1) is a pullback in* **AGSIG-Alg** *iff (1) is a componentwise pullback in* **Sets**. *If $n \in M$ then also $m \in M$.*

*Proof.* (see [14])

*Remark 3.* Since $AGSIG(ATG)$ is a special case of $AGSIG$, the lemma is also true for **AGSIG(ATG)-Alg**. It is well-known that **AGSIG-Alg** - as a category of algebras - has pushouts even if $m \notin M$, but in general such pushouts cannot be constructed componentwise.

**Theorem 2 (adhesive HLR categories).** *The category (***AGraphs$_{\mathbf{ATG}}$***, $M$) of typed attributed graphs and also (***AGSIG-Alg***, $M$) and (***AGSIG(ATG)-Alg***, $M$) are adhesive HLR categories.*

*Proof.* It is sufficient to prove the properties for (**AGSIG-Alg**, $M$), because (**AGSIG(ATG)-Alg**, $M$) is a special case of (**AGSIG-Alg**, $M$) and (**AGraphs$_{\mathbf{ATG}}$**, $M$) $\cong$ **AGSIG(ATG)-Alg** by Thm. 1.

1. The class $M$ given in Def. 6 is a subclass of monomorphisms, because monomorphisms in **AGSIG-Alg** are exactly the injective homomorphisms, and it is closed under composition and decomposition.
2. (**AGSIG-Alg**, $M$) has pushouts and pullbacks along $M$-morphisms and $M$-morphisms are closed under pushouts and pullbacks due to Lem. 1.
3. Pushouts along $M$-morphisms in **AGSIG-Alg** are VK squares because pushouts and pullbacks are constructed componentwise in **Sets** (see Lem. 1) and for (**Sets**, $M_{inj}$) pushouts along monomorphisms are VK squares as shown in [9].

## 4 Theory of Typed Attributed Graph Transformation

After the preparations in the previous sections we are now ready to present the basic notions and results for typed attributed graph transformation systems. In fact, we obtain all the results presented for adhesive HLR systems in [8] in our case, because we are able to show the corresponding HLR conditions for (**AGraphs$_{\mathbf{ATG}}$**, $M$). This category (**AGraphs$_{\mathbf{ATG}}$**, $M$) is fixed now for this section.

**Definition 7 (typed attributed graph transformation system).** *A typed attributed graph transformation system $GTS = (DSIG, ATG, S, P)$ based on (**AGraphs$_{\mathbf{ATG}}$**, $M$) consists of a data type signature $DSIG$, an attributed type graph $ATG$, a typed attributed graph $S$, called start graph, and a set $P$ of productions, where*

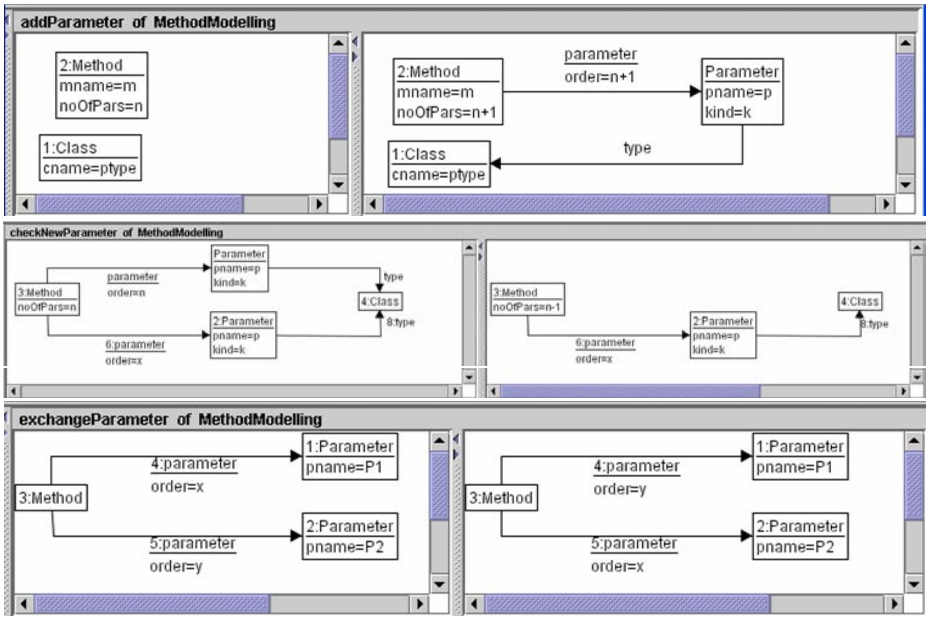
1. *a production $p = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ consists of typed attributed graphs $L$, $K$ and $R$ attributed over the term algebra $T_{DSIG}(X)$ with variables $X$, called left hand side $L$, gluing object $K$ and right hand side $R$ respectively, and morphisms $l, r \in M$, i.e. $l$ and $r$ are injective and isomorphisms on the data type $T_{DSIG}(X)$,*
2. *a direct transformation $G \overset{p,m}{\Longrightarrow} H$ via a production $p$ and a morphism $m : L \rightarrow G$, called match, is given by the following diagram, called double pushout diagram, where (1) and (2) are pushouts in **AGraphs$_{\mathbf{ATG}}$**,*

$$\begin{array}{ccccc} L & \xleftarrow{\quad} & K & \xrightarrow{\quad} & R \\ {\scriptstyle m}\downarrow & \overset{l}{(1)} & \downarrow & \overset{r}{(2)} & \downarrow \\ G & \xleftarrow{\quad} & D & \xrightarrow{\quad} & H \end{array}$$

3. *a typed attributed graph transformation, short transformation, is a sequence $G_0 \Rightarrow G_1 \Rightarrow ... \Rightarrow G_n$ of direct transformations, written $G_0 \overset{*}{\Rightarrow} G_n$,*
4. *the language $L(GTS)$ is defined by $L(GTS) = \{G \mid S \overset{*}{\Rightarrow} G\}$.*

*Remark 4.* A typed attributed graph transformation system is an adhesive HLR system in the sense of [8] based on the adhesive HLR category ($\mathbf{AGraphs_{ATG}}$, $M$).

*Example 2 (typed attributed graph transformation system).* In the following, we start to define our typed attributed graph transformation system *MethodModelling* by giving the productions. All graphs occuring are attributed by term algebra $T_{DSIG}(X)$ with $DSIG$ being the data signature presented in Ex. 1 and $X = \bigcup_{s \in S'_D} X_s$, i.e. $X = X_{String} \cup X_{int} \cup X_{ParameterDirectionKind}$ with $X_{String} = \{m, p, ptype, P1, P2\}$, $X_{ParameterDirectionKind} = \{k\}$ and $X_{int} = \{n, x, y\}$. We present the productions in the form of AGG productions where we have the possibility to define a subset of variables of $X$ as input parameters. That means a partial match is fixed by the user before the proper matching procedure starts. Each production is given by its name followed by the left and the right-hand side as well as a partial mapping from left to right given by numbers. From this partial mapping the gluing graph $K$ can be deduced being the domain of the mapping. Parameters are m, p, k, ptype, x and y. We use a graph notation similarly to Fig. 3.




AGG productions are restricted concerning the attribution of the left-hand sides. To avoid the computation of most general unifiers of two general terms, nodes and edges of left-hand sides are allowed to be attributed only by constants and variables. This restriction is not a real one, since attribute conditions may be used. A term in the left-hand side is equivalent to a new variable and a new attribute condition stating the equivalence of the term and this new variable.

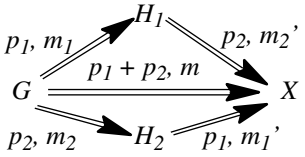
Productions $addMethod$ and $addClass$ with empty left-hand side and a single method respectively class on the right-hand side are not shown. Together with production $addParameter$ they are necessary to build up method signatures. A new parameter is inserted as last one in the parameter list. Production $checkNewParameter$ checks if a newly inserted parameter is already in the list. In this case it is removed. Production $exchangeParameter$ is useful for changing the order of parameters in the list.

The start graph $S$ is empty, i.e. $S = \emptyset$, while data signature $DSIG$ and type graph $T$ have already been given in Ex. 1. Summarizing, the typed attributed graph transformation system is given by $MethodModelling = (DSIG, ATG, S, P)$ with $P = \{addMethod, addClass, addParameter, exchangeParameter, checkNewParameter\}$. □

In the following we show that the basic results known in the classical theory of graph transformation in [1] and in the theory of HLR systems in [10] are also valid for typed attributed graph transformation systems.

The Local Church-Rosser Theorem states that direct transformations $G \overset{p_1,m_1}{\Longrightarrow} H_1$ and $G \overset{p_2,m_2}{\Longrightarrow} H_2$ can be extended by direct transformations $H_1 \overset{p_2,m_2'}{\Longrightarrow} X$ and $H_2 \overset{p_1,m_1'}{\Longrightarrow} X$ leading to the same $X$, provided that they are parallel independent. Parallel independence means that the matches $m_1$ and $m_2$ overlap only in common gluing items, i.e. $m_1(L_1) \cap m_2(L_2) \subseteq m_1(l_1(K_1)) \cap m_2(l_2(K_2))$.

The Parallelism Theorem states that in the case of parallel independence we can apply the parallel production $p_1 + p_2 = (L_1 + L_2 \overset{l_1+l_2}{\Longleftarrow} K_1 + K_2 \overset{r_1+r_2}{\Longrightarrow} R_1 + R_2)$ in one step $G \overset{p_1+p_2,m}{\Longrightarrow} X$ from $G$ to $X$. Vice versa each such direct parallel derivation can be sequentialized in any order leading to two sequential independent sequences $G \overset{p_1,m_1}{\Longrightarrow} H_1 \overset{p_2,m_2'}{\Longrightarrow} X$ and $G \overset{p_2,m_2}{\Longrightarrow} H_2 \overset{p_1,m_1'}{\Longrightarrow} X$.




The case of general sequences, which may be sequentially dependend, is handled by the Concurrency Theorem. Roughly spoken, for each sequence $G \overset{p_1,m_1}{\Longrightarrow} H_1 \overset{p_2,m_2'}{\Longrightarrow} X$ there is a production $p_1 * p_2$, called concurrent production, which allows to construct a direct transformation $G \overset{p_1*p_2}{\Longrightarrow} X$ and vice versa, leading, however, only to one sequentialization.

**Theorem 3 (Local Church-Rosser, Parallelism and Concurrency Theorem).** *The Local Church-Rosser Theorems I and II, the Parallelism Theorem and the Concurrency Theorem as stated in [10] are valid for each graph transformation system based on* ($\mathbf{AGraphs_{ATG}}$, $M$).

*Proof idea.* The Local Church-Rosser, Parallelism and Concurrency Theorem are verified for HLR2 categories in [10] and they are shown for adhesive HLR

systems in [8], where only the Parallelism Theorem requires in addition the existence of binary coproducts compatible with $M$. Compatibility with $M$ means $f$, $g \in M$ implies $f + g \in M$. In Thm. 2 we have shown that ($\mathbf{AGraphs_{ATG}}$, $M$) is an adhesive HLR category. Binary coproducts compatible with $M$ can be constructed already in **AGSIG-Alg** with well-structured $AGSIG$ and transfered to $\mathbf{AGraphs_{ATG}}$ by Thm. 1. In (**AGSIG-Alg**, $M$) binary coproducts can be constructed separately for the $DSIG$-part and componentwise in **Sets** for all sorts $s \in S_G \backslash S_D$, which implies compatibility with $M$. If $AGSIG$ is not well-structured we still have binary coproducts in **AGSIG-Alg** - as in any category of algebras - but they may not be compatible with $M$. □

The next basic result in the classical theory of graph transformation systems is the Embedding Theorem (see [1]) in the framework of adhesive HLR systems. The main idea of the Embedding Theorem according to [1] is to show under which conditions a transformation $t : G_0 \overset{*}{\Rightarrow} G_n$ can be extended to a transformation $t' : G'_0 \overset{*}{\Rightarrow} G'_n$ for a given "embedding" morphism $k_0 : G_0 \to G'_0$. In the case of typed attributed graph transformation we consider the following class $M'$ of "graph part embeddings": $M'$ consists of all morphisms $k_0$ where the E-graph part of $k_0$ is injective except of data nodes (see Def. 1 - 3). This means, that the algebra part of $k_0$ is not restricted to be injective.

Similar to the graph case it is also possible in the case of typed attributed graphs to construct a boundary $B$ and context $C$ leading to a pushout (1) over $k_0$ in Fig. 4 with $b_0 \in M$, i.e. $G'_0$ is the gluing of $g_0$ and context $C$ along the boundary $B$. This boundary-context pushout (1) over $k_0$ turns out to be an initial pushout over $k_0$ in the sense of [8].

Now the morphism $k_0 \in M'$ is called consistent with respect to the transformation $t : G_0 \overset{*}{\Rightarrow} G_n$, if the boundary $B$ is "preserved" by $t$ leading to a morphism $b_n : B \to G_n \in M$ in Fig. 4. (For a formal notion of consistency see [8].)

The following Embedding and Extension Theorem shows that consistency is necessary and sufficient in order to extend $t : G_0 \overset{*}{\Rightarrow} G_n$ for $k_0 : G_0 \to G'_0$ to $t' : G'_0 \overset{*}{\Rightarrow} G'_n$.

**Theorem 4 (Embedding and Extension Theorem).** *Let $GTS$ be a typed attributed graph transformation system based on ($\mathbf{AGraphs_{ATG}}$, $M$) and $M'$ the class of all graph part embeddings defined above. Given a transformation $t : G_0 \overset{*}{\Rightarrow} G_n$ and a morphism $k_0 : G_0 \to G'_0 \in M'$ with boundary-context pushout (1) over $k_0$ we have: The transformation $t$ can be extended to a transformation $t' : G'_0 \overset{*}{\Rightarrow} G'_n$ with morphism $k_n : G_n \to G'_n \in M'$ leading to diagram (2), called extension diagram, and a boundary-context pushout (3) over $k_n$ if and only if the morphism $k_0$ is consistent with respect to $t$.*

*Proof idea.* This theorem follows from the Embedding and Extension Theorems in [8] shown for adhesive HLR systems over an adhesive HLR category ($\mathbf{C}$, $M$). It requires initial pushouts over $M'$-morphisms for some class $M'$, which is closed under pushouts and pullbacks along $M$-morphisms. By Thm. 2 we know that ($\boldsymbol{AGraphs_{ATG}}$, $M$) is an adhesive HLR category. In addition it can be shown

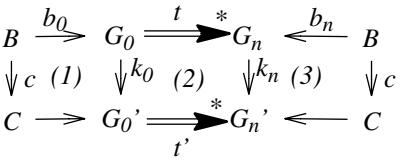
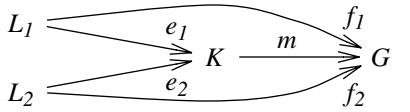



**Fig. 4.**

that for each $k_0 \in M'$, where $M'$ is the class of all graph part embeddings, there is a boundary-context pushout (1) over $k_0$, which is already an initial pushout over $k_0$ in the sense of [8]. Moreover it can be shown that $M'$ is closed under pushouts and pullbacks along $M$-morphisms. □

The Embedding and Extension Theorems are used in [8] to show the Local Confluence Theorem, also known as critical pair lemma, in the framework of adhesive HLR systems, where in addition to initial pushouts also the existence of an $E'$-$M'$ pair factorization is used.

**Definition 8 ($E'$-$M'$ pair factorization).** *Given a class $E'$ of morphism pairs $(e_1, e_2)$ with the same codomain and $M'$ the class of all graph part embeddings defined above. We say that a typed attributed graph transformation system based on ($\mathbf{AGraphs}_{\mathbf{ATG}}$, $M$) has $E'$-$M'$ pair factorization, if for each pair of matches $f_1 : L_1 \to G$, $f_2 : L_2 \to G$ there is a pair $e_1 : L_1 \to K$, $e_2 : L_2 \to K$ with $(e_1, e_2) \in E'$ and a morphism $m : K \to G$ with $m \in M'$ such that $m \circ e_1 = f_1$ and $m \circ e_2 = f_2$.*




*Remark 5.* For simplicity we have fixed $M'$ to be the class of all graph part embeddings, which implies that $M'$ is closed under pushouts and pullbacks along $M$-morphisms as required for $E'$-$M'$ pair factorization in [8] with general class $M'$.

*Example 3.* 1. Let $E'$ be the class of jointly surjective morphisms in $\boldsymbol{AGraphs}_{\mathbf{ATG}}$ with same codomain. Given $f_1$ and $f_2$ we obtain an induced morphism $f_{12} : L_1 + L_2 \to G$ with coproduct injections $i_1 : L_1 \to L_1 + L_2$ and $i_2 : L_2 \to L_1 + L_2$. Now let $f_{12} = m \circ e$ an epi-mono factorization of $f_{12}$ leading to $e_1 = e \circ i_1$ and $e_2 = e \circ i_2$ with $(e_1, e_2) \in E'$. In this case $m : K \to G$ is injective and the data type part of $K$ is a quotient term algebra $T_\Sigma(X_1 + X_2)|_\equiv$, where $T_\Sigma(X_1)$ and $T_\Sigma(X_2)$ are the algebras of $L_1$ and $L_2$ respectively. This corresponds to one possible choice of congruence $\equiv$ considered in [5].

2. In order to obtain a minimal number of critical pairs it is essential to consider also the case, where $\equiv$ is the trivial congruence with $T_\Sigma(X) \cong T_\Sigma(X)|_\equiv$. In fact, a most general unifier construction $\sigma_n : X \to T_\Sigma(X)$ considered in [5] leads to a different $E'$-$M'$ pair factorization of $f_1$, $f_2$ with $(e_1, e_2) \in E'$,

$m \in M'$, $e_1 : L_1 \to K$, $e_2 : L_2 \to K$ and $m : K \to G$, where $(e_1, e_2)$ is jointly surjective for non-data nodes, the data type part of $K$ is $T_\Sigma(X)$ and $m$ is injective for non-data nodes, where non-data nodes are all nodes and edges, which are not data nodes (see Def. 1).

Dependent on the choice of an $E'$-$M'$ pair factorization we are now able to define critical pairs and strict confluence.

**Definition 9 (critical pair and strict confluence).** *Given an $E'$-$M'$ pair factorization, a critical pair is a pair of non-parallel independent direct transformations $P_1 \overset{p_1,o_1}{\Longleftarrow} K \overset{p_2,o_2}{\Longrightarrow} P_2$ such that $(o_1, o_2) \in E'$ for the corresponding matches $o_1$ and $o_2$. The critical pair is called strictly confluent, if we have*

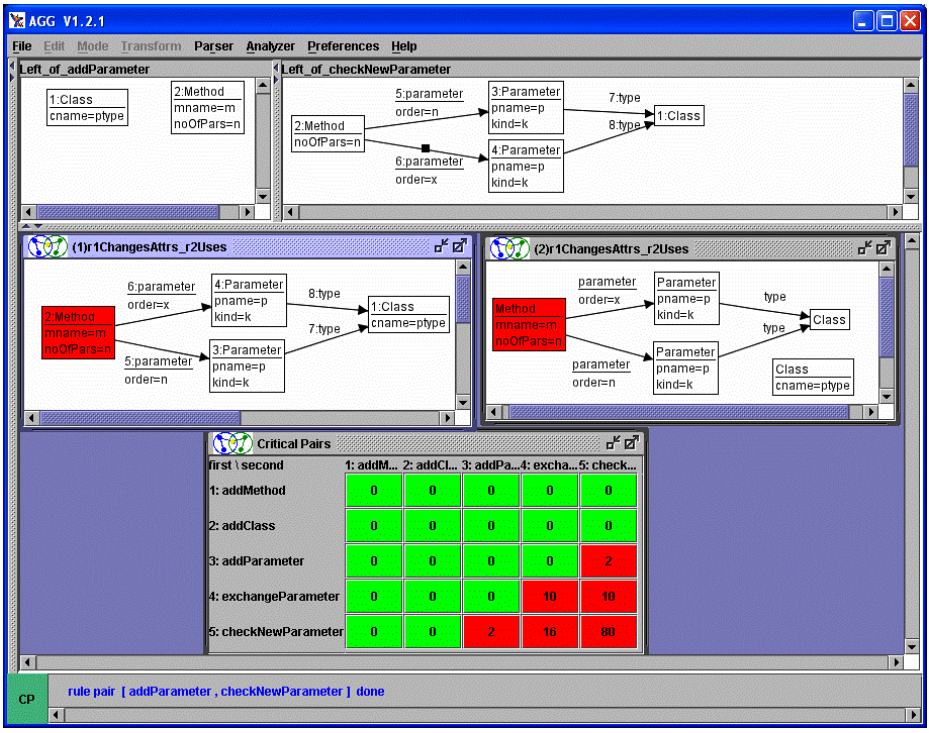
1. *Confluence: There are transformations $P_1 \overset{*}{\Rightarrow} K'$ and $P_2 \overset{*}{\Rightarrow} K'$.*
2. *Strictness: Let $N$ be the common subobject of $K$, which is preserved by the direct transformations $K \Longrightarrow P_1$ and $K \Longrightarrow P_2$ of the critical pair, then $N$ is also preserved by $P_1 \overset{*}{\Rightarrow} K'$ and $P_2 \overset{*}{\Rightarrow} K'$, such that the restrictions of the transformatios $K \Rightarrow P_1 \overset{*}{\Rightarrow} K'$ and $K \Rightarrow P_2 \overset{*}{\Rightarrow} K'$ yield the same morphism $N \to K'$. (See [8] for a more formal version of strict confluence.)*

**Theorem 5 (Local Confluence Theorem - Critical Pair Lemma).** *Given a typed attributed graph transformation system $GTS$ based on ($\mathbf{AGraphs}_{\mathbf{ATG}}$, $M$), and an $E'$-$M'$ pair factorization, where $M'$ is the class of all graph part embeddings, then $GTS$ is locally confluent, if all its critical pairs are strictly confluent.*
*Local confluence of $GTS$ means that for each pair of direct transformations $H_1 \Leftarrow G \Rightarrow H_2$ there are transformations $H_1 \overset{*}{\Rightarrow} X$ and $H_2 \overset{*}{\Rightarrow} X$.*

*Proof idea.* This theorem follows from the Local Confluence Theorem in [8] shown for adhesive HLR systems over ($\mathbf{C}$,$M$). It requires initial pushouts over $M'$-morphisms for a class $M'$ "compatible" with $M$. The proof in [8] is based on completeness of critical pairs shown by using an $M$-$M'$ pushout pullback decomposition property. In our case $M'$ is the class of all graph part embeddings, which can be shown to satisfy this property, which implies that $M'$ is "compatible" with $M$. In the proof idea of Thm. 4 we have discussed already how to verify the remaining properties which are required in the general framework of [8]. □

*Example 4 (critical pairs).* Considering our typed attributed graph transformation system $MethodModelling$ (see Ex. 2) we now analyse its critical pairs. This analysis is supported by AGG. Due to the restriction of attributes in the left-hand side of a production to constants and variables we restrict ourselves to very simple congruence relations on terms of overlapping graphs. Variables may be identified with constants or with other variables. All other congruences are the identities. In the following, the AGG user interface for critical pair analysis is shown. It presents an overview on all critical pairs in form of a table containing all possible pairs of $MethodModelling$ at the bottom of the figure. Applying for example first $addParameter$, $exchangeParameter$ or $checkNewParameter$

and second *checkNewParameter* leads to critical pairs. We consider the two critical pairs of productions *addParameter* and *checkNewParameter* closer. On the top of the figure the left-hand sides of both productions are displayed. The center shows the two overlapping graphs which lead to critical pairs. Both overlapping graphs show the conflict on attribute 'noOfPars' which is increased by production *addParameter* but decreased by *checkNewParameter*. In the left graph both classes are identified, i.e. the new parameter would be of the same type as the the two already existing ones which are equal, while the right graph shows two classes.




Both critical pairs of productions *addParameter* and *checkNewParameter* are strictly confluent. Applying first $addParameter(mname, \text{“}n1\text{”}, cname, \text{“}in\text{”})$ and then $exchangeParameter(n, n+1)$ and $checkNewParameter()$ the resulting graph is isomorphic to applying first $checkNewParameter()$ and then $addParameter(mname, \text{“}n1\text{”}, cname, \text{“}in\text{”})$. The common subgraph $N$ of the conflicting transformations is the result graph applying *checkParameter* to the overlapping graph.

## 5 Related Work and Conclusions

A variety of approaches to attributed graph transformation [4–6, 11, 12, 16] has already been developed where attributed graphs consist of a graph part and a

data part. These approaches are compared with ours in the following. Moreover, general forms of algebraic structures and their transformation have been considered in e.g. [17–19].

A simple form of attributed graphs, where only nodes are attributed, is chosen in [16] and [5]. In [16], nodes are directly mapped to data values which makes relabelling difficult. In [5], nodes are attributed by special edges which are deleted and newly created in order to change attributes. We have defined attributed graphs in the spirit of the later notion of node-attributed graphs, but extended the notion also to edge attribution. In [4, 11], attributed graph structures are formulated by algebras of a special signature where the graph part and the data part are separate from each other and only connected by attribution operations. The graph part comprises so-called attribute carriers which play the same role as attribution edges. They assign attributes to graph elements and allow relabelling. I.e. attributed graph signatures can also be formalized by well-structured $AGSIG$-algebras. In [4, 11], they are transformed using the single-pushout approach. In [12] and [6] this attribution concept is extended by allowing partial algebras. Using e.g. partial attribution operations, no carriers or edges are needed for attribution. This leads to a slightly more compact notion, but however, causes a more difficult formalization. We are convinced that attributed graphs as defined in Sec. 2 are a good compromise between the expressiveness of the attribution concept on one hand and the complexity of the formalism on the other.

The theory we provide in this paper includes fundamental results for graph transformation which are now available for typed attributed graph transformation in the sense of sections 2 and 4. The general strategy to extend the theory for typed attributed graph transformation is to formulate the corresponding results in adhesive HLR categories and to verify additional HLR properties for the category ($\mathbf{AGraphs_{ATG}}$, $M$), if they are required. But the theory presented in Sec. 4 is also valid in the context of well-structured attributed graph structure signatures $AGSIG$, which correspond to attributed graph signatures in [4, 11]. In fact, the HLR conditions required for all the results in [8] are already valid for the category (**AGSIG-Alg**, $M$) with well-structured $AGSIG$. This means, that our theory is also true for attributed graph transformation based on the adhesive HLR category (**AGSIG-Alg**, $M$) with general $AGSIG$ for Thm. 3 and well-structured $AGSIG$ for Thm. 4 and 5.

Future work is needed to obtain corresponding results for extensions of typed attributed graph transformation by further concepts such as application conditions for productions or type graphs with inheritance.

## References

1. Ehrig, H.: Introduction to the Algebraic Theory of Graph Grammars (A Survey). In: Graph Grammars and their Application to Computer Science and Biology. Volume 73 of LNCS. Springer (1979) 1–69
2. Ehrig, H., Engels, G., Kreowski, H.J., Rozenberg, G., eds.: Handbook of Graph Grammars and Computing by Graph Transformation, Volume 2: Applications, Languages and Tools. World Scientific (1999)

3. Ehrig, H., Kreowski, H.J., Montanari, U., Rozenberg, G., eds.: Handbook of Graph Grammars and Computing by Graph Transformation. Vol 3: Concurrency, Parallelism and Distribution. World Scientific (1999)
4. Löwe, M., Korff, M., Wagner, A.: An Algebraic Framework for the Transformation of Attributed Graphs. In: Term Graph Rewriting: Theory and Practice. John Wiley and Sons Ltd. (1993) 185–199
5. Heckel, R., Küster, J., Taentzer, G.: Confluence of Typed Attributed Graph Transformation with Constraints. In: Proc. ICGT 2002. Volume 2505 of LNCS., Springer (2002) 161–176
6. Berthold, M., Fischer, I., Koch, M.: Attributed Graph Transformation with Partial Attribution. Technical Report 2000-2 (2000)
7. Ermel, C., Rudolf, M., Taentzer, G.: The AGG-Approach: Language and Tool Environment. In Ehrig, H., Engels, G., Kreowski, H.J., Rozenberg, G., eds.: Handbook of Graph Grammars and Computing by Graph Transformation, Volume 2, World Scientific (1999) 551–603
8. Ehrig, H., Habel, A., Padberg, J., Prange, U.: Adhesive High-Level Replacement Categories and Systems. In: Proc. ICGT 2004. LNCS, Springer (2004) (this volume).
9. Lack, S., Sobociński, P.: Adhesive Categories. In: Proc. FOSSACS 2004. Volume 2987 of LNCS., Springer (2004) 273–288
10. Ehrig, H., Habel, A., Kreowski, H.J., Parisi-Presicce, F.: Parallelism and Concurrency in High-Level Replacement Systems. Math. Struct. in Comp. Science **1** (1991) 361–404
11. Claßen, I., Löwe, M.: Scheme Evolution in Object Oriented Models: A Graph Transformation Approach. In: Proc. Workshop on Formal Methods at the ISCE'95, Seattle (U.S.A.). (1995)
12. Fischer, I., Koch, M., Taentzer, G., Volle, V.: Distributed Graph Transformation with Application to Visual Design of Distributed Systems. In Ehrig, H., Kreowski, H.J., Montanari, U., Rozenberg, G., eds.: Handbook of Graph Grammars and Computing by Graph Transformation, Volume 3, World Scientific (1999) 269–340
13. Bardohl, R.: A Visual Environment for Visual Languages. Science of Computer Programming (SCP) **44** (2002) 181–203
14. Ehrig, H., Prange, U., Taentzer, G.: Fundamental Theory for Typed Attributed Graph Transformation: Long Version. Technical Report TU Berlin. (2004)
15. Ehrig, H., Mahr, B.: Fundamentals of Algebraic Specification 1: Equations and Initial Semantics. Volume 6 of EATCS Monographs on TCS. Springer, Berlin (1985)
16. Schied, G.: Über Graphgrammatiken, eine Spezifikationsmethode für Programmiersprachen und verteilte Regelsysteme. Arbeitsber. des Inst. für math. Maschinen und Datenverarbeitung, PhD Thesis, University of Erlangen (1992)
17. Wagner, A.: A Formal Object Specification Technique Using Rule-Based Transformation of Partial Algebras. PhD thesis, TU Berlin (1997)
18. Llabres, M., Rossello, F.: Pushout Complements for Arbitrary Partial Algebras. In Ehrig, H., Engels, G., Kreowski, H.J., Rozenberg, G., eds.: Theory and Applications of Graph Transformation. Volume 1764., Springer (2000) 131–144
19. Große-Rhode, M.: Semantic Integration of Heterogeneuos Software Specifications. EATCS Monographs on Theoretical Computer Science. Springer, Berlin (2004)