Amalgamated Graph Transformations and Their Use for Specifying AGG – an Algebraic Graph Grammar System *

Gabriele Taentzer and Martin Beyer

Computer Science Department, Technical University of Berlin,
Franklinstr. 28/29, Sekr. FR 6-1, D-10587 Berlin,
e-mail: {gabi,beyer}@cs.tu-berlin.de

Abstract. The AGG-system is a prototype implementation of the algebraic approach to graph transformation. It consists of a flexible graph editor and a transformation component. The editor allows the graphical representation of production rules, occurrences and transformation results. The transformation component performs direct transformation steps for user–selected production rules and occurrences.
First steps towards a graph specification of an abstract version of the AGG-system are possible by using amalgamated graph transformations. AGG-states are modelled by graphs whereas AGG-operations are described by amalgamated graph transformations combining parallel and sequential rewriting of graphs.

Keywords: graph grammar system, algebraic graph grammars, parallel graph transformation

1 Introduction

Although graph grammars have been investigated for more than 20 years only few graph grammar systems exist. (For example, IPSEN [Nag87, Sch91], PAGGED [Göt87a] or GraphEd [Him91].) The algebraic graph grammar system AGG ([Bey91, LB93]) is a prototype implementation of the single pushout approach to graph rewriting. AGG is a purely graphical system, consisting of a graph editor and a transformation component. It combines the intuitiveness of graphical presentations made accessible by current workstation technology with the mathematical rigour of the algebraic approach to graph rewriting. AGG's editing and visualization features are intended to make it suitable as a tool for case studies that would soon become unwieldy if done on a pen and paper basis. This includes the ability to easily edit and navigate in graphs that are too big to be grasped at one view. E.g. several windows can be opened displaying different views of graphs.

AGG has a mouse/menu - oriented interface running on SUN workstations under X-Window. It is implemented in the object oriented language EIFFEL.

* This work has been partly supported by the ESPRIT Working Group 7183 "Computing by Graph Transformation (COMPUGRAPH II)"

Some AGG-operations may be thought of as consisting of a basic action and a context dependent number of extended actions: e.g. moving a node means changing the representation of a graph (basic action) as well as changing the contents of all windows displaying the node (extended actions). This distinction will be reflected in our graph specification of AGG-operations.

In section 3 we present very first ideas towards a graph specification of an abstract version of the AGG-system. The implementation of the AGG-system ([Bey91]) contains the data structures representing graphs, their graphical attributes and an interface to X-Window. We restrict the specification of the AGG-system to the treatment of the graph structures and the logical part of the graphical representation. AGG-states are represented as graphs where the nodes almost model AGG-objects and the edges in between show their use relation. The specification of AGG-operations is performed by graph transformations, i.e. generally the application of a number of graph productions. Those graph productions which perform the actual user interactions function as a basis for further extended actions that have also to be executed to model the whole AGG-operation. All extended actions can be done in parallel and synchronize in the basic actions. Thus, the resulting specification indicates which actions can be executed simultaneously and how they have to synchronize.

Formalizing the specification concept used to specify the AGG-operations, we present amalgamated graph transformations. This type of graph transformation combines parallel and sequential concepts of graph rewriting and is based on the amalgamation concepts introduced in [BFH87]. Parallel replacement of graphs has been investigated in several approaches to graph grammars. (See [EK76], [JRV82], [NA83], [Nag87], [BCF91], [Kre92], [DDK93], etc.) In [EK76] parallel graph grammars are presented in the algebraic framework. They can be seen as an extension of sequential grammars (see [Ehr79]) on one hand, on the other hand parallel graph grammars generalize L-systems which describe a kind of context-free parallel replacement on strings ([RS86], [RS92]). Amalgamated graph transformations are an extension of the parallel graph grammar approach in [EK76] offering additional features.

2 The AGG-system

The AGG-system[2] is part of an ongoing implementation effort in the Berlin graph grammar group, initiated by M. Löwe. The development of data structures for an efficient implementation of algebraic graph rewriting has been one initial motivation for the investigation of the single pushout approach (see [Löw93]).

Graphs, morphisms, production rules and occurrences are represented and manipulated in a uniform way. Therefore, AGG supports a graph model with abstraction and higher order edges. It includes the usual definition of graphs as a special case.

Higher order edges, i.e. edges between edges, are used to represent morphisms as graphs. Edges between objects of any order, e.g. between an edge and a node, are also possible. Graphs can be abstracted to nodes, bundles of edges can be abstracted to single edges. The abstraction function induces a partitioning of the graph in

[2] Note that AGG stands for algebraic graph grammars and should not be mixed up with attributed graph grammars.

abstraction levels. The formal definition of this concept which can be found in e.g. [Bey92] regards graphs as a set of objects with three partial functions: the *source, target*, and *abstraction* function. They are totalized by including a special object $\perp$. A node is an object for which both the *source* and *target* function are undefined. There are a few rather obvious consistency conditions that are enforced by the AGG-system, e.g. the *abstraction* function has to commute with the *source* and *target* function and has to be acyclic.

The graph editor provides operations for the input and modification of graphs, different views on graphs and supports the editing and simulating of graph rewriting.

Several graphs can be edited simultaneously. Any number of windows can be opened. Each window displays (a part of) an abstraction level. An abstraction level can be shown in several windows. All windows can be used in the same way for editing or transforming graphs. Any change in the representation of a graph is immediately reflected in all appropiate windows.

Several parameters influencing the display of graphs can be adjusted in each window, like the size of node symbols, the zoom factor or the hiding of objects. It is intended to experiment with graph transformation doing the layout manually, before adding layout possibilities to the system. (See for example [Bra91] or [NP91].)

Some edit operations are especially suited for graph transformations, e.g. a gluing operation which merges two objects to a new object inheriting the incidence relations of the original objects and a copy operation drawing edges between all original nodes and edges and their copies. These operations facilitate the input of transformation rules. A simple example (cf. [LB93]) of a transformation rule is shown in figure 1.

The abstraction concept is supported conveniently by the editor. It is possible to create objects in different abstraction levels before defining the abstraction function. Another possibility is the operation "create abstraction": The user selects a number of objects belonging to one abstraction level and creates an abstraction object for them, i.e. an object is inserted into the next higher abstraction level and is defined as the abstraction of the selected objects. The created abstraction object will be a node if the selected objects include nodes.

The abstraction concept is supported by all edit operations. E.g. when objects are moved, all refinement objects as well as all incident objects are moved accordingly.

Other operations of the editor are the changing of node or edge symbols, an extensive and orthogonal grid functionality for each window, view operations, a bunch of interactive and graph based operations for the selection of objects, copy-, move- and resize-operations and more.

The transformation component implements the single-pushout approach to graph rewriting (see [Löw93]) and is illustrated by a simple example, a graph model of a queue structure. Figure 1 shows a rule which model the insertion of an element at the end ($\square$) of the queue. The head of the queue is marked by $\triangle$.

The abstraction concept is used to present an easy to use interface to the transformation component: graphs and the left and right side of rules are abstracted to single nodes, rule morphisms and occurrences are abstracted to single edges. In this way, the higher abstraction level is used as a diagram level.

Figure 2 shows an occurrence of the insertion rule presented above on the graph level and the diagram level.

A transformation step is performed by triggering the "apply rule"-operation (via

Fig. 1. Transformation rule: graph level and diagram level

Fig. 2. Occurrence: graph level and diagram level

menu or keyboard) and clicking on the occurrence and the place where the derived graph is to be inserted in the window displaying the diagram level. After the execution of the transformation step, the diagram level contains the pushout diagram, the graph level includes the derived graph as well as the morphisms from the old graph and the right hand side of the rule to the derived graph (see figure 3).

Fig. 3. Transformation step: graph level and diagram level

Graphs are not restricted to two abstraction levels. The abstraction concept can also be used to structure large graphs by hiding details in lower abstraction levels.

3 Towards a graph specification of the AGG-system

In this section we present first ideas and examples for modelling the AGG-system by graphs and graph transformations. Within this paper we restrict the model to the level of internal graph structures and logical graph representation which means that we do not care about the layout and position of graphs and windows. Principles of the modelling concept are shown by specifying the AGG-operations "insertion of a node" and "create abstraction".

The internal states of the AGG-system are modelled by graphs where important AGG-objects like windows, graphs, abstraction levels, nodes, edges, etc. are represented by nodes and the use relations between them by edges.

In figure 4 the modelling of an arbitrary state of the AGG-system is shown where the model of the initial state is bordered with dashed lines. Nodes and edges are represented as objects belonging to an abstraction level. Moreover, there is a symbol for each object and each window showing the corresponding level. Such a symbol contains the graphical representation of the object. The cursor points to the current window and abstraction level. All abstraction levels belonging to one graph form a list which is described by edges between different levels. The bottom and the top level do not have a refinement and an abstraction, resp. This is indicated by $\perp$ and $\top$.

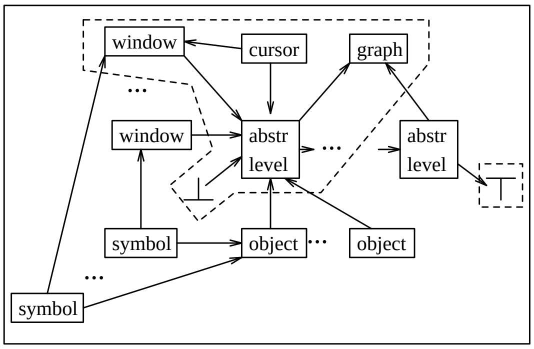

Fig. 4. Section of a state graph

As shown in figure 4, most of the relations between AGG-objects are 1:n-relations. For example a number of windows show one level, a number of objects belong to one level, etc.

The AGG-objects node and edge are represented as object-labelled nodes. They are distinguishable via the pointers to their source (s) and target (t)[3]. For nodes, source and target are not defined, indicated by pointers to $\perp$. Modelling nodes and edges in this uniform way higher-order edges (see section 2) can be easily described (figure 5).

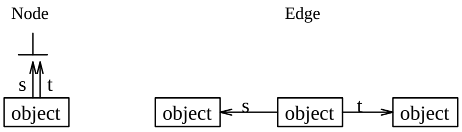

Fig. 5. Node and egde representation

The specification of AGG-operations is performed by graph transformations. The first example we want to discuss is the insertion of a node. What basically should be done is the addition of an object to the current abstraction level. This feature can be described by a simple graph production.

The basic action has to be extended by the insertion of a symbol for the new object for each window showing the current abstraction level. This means that there

[3] For reasons of simplicity, the representation of the abstraction function is first introduced when the abstraction concept is treated.

are n extended actions if the current level is displayed in n windows. In figure 6 the basic part of the graph production is bordered with dashed lines.

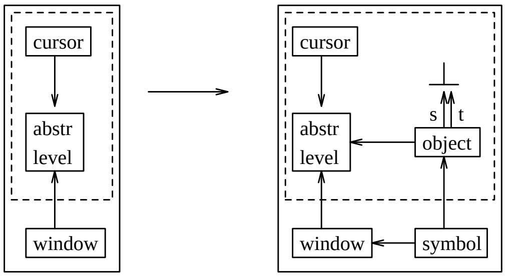

Fig. 6. Insertion of a node

The creation of new symbols can be done simultaneously meaning that all extended actions can be executed in parallel. But the basic action which is the insertion of a node should be done only once. Since all extended actions overlap in this basic action it can also be regarded as interface action where all extended actions have to synchronize in. For example, a node is inserted into an empty abstraction level viewed in two different windows. Another graph containing one node is not changed. The corresponding graph transformation is shown in figure 7.

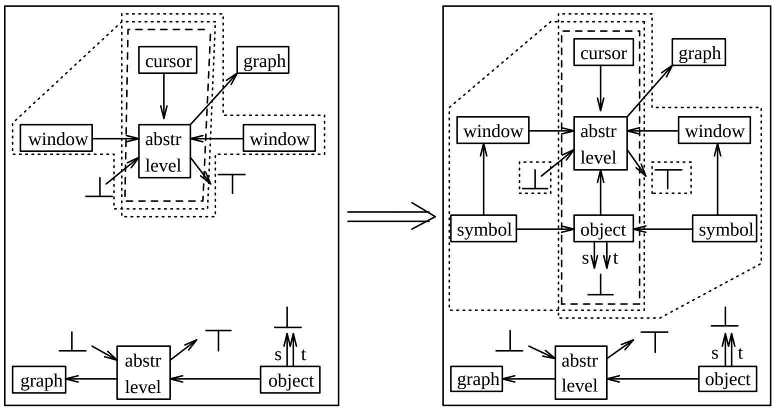

Fig. 7. Example of the insertion of a node

In this example, the basic action is applied once (indicated by dashed bordered lines). Since the current abstraction level is displayed in two different windows the extended action is executed twice (indicated by dotted bordered lines).

Here, performing an AGG-operation corresponds to the application of a basic action and its extended actions.

In the previous section we described the AGG-operation "create abstraction" which allows to abstract graphs to nodes and bundles of edges in the same direction to single edges. Within this paper we show only the abstraction of nodes. Specifying "create abstraction" we have to distinguish two main cases. The first one is that the current abstraction level is the top one. Then a new level has to be created which is the new top level. A new abstraction node abstracting all selected nodes has to be inserted into the new level. All the selected nodes are not yet abstracted. Such an information is described by an a-edge running from a selected node to $\perp$ instead of an abstraction node. Selected objects are indicated by a selected-labelled edge pointing on the object.

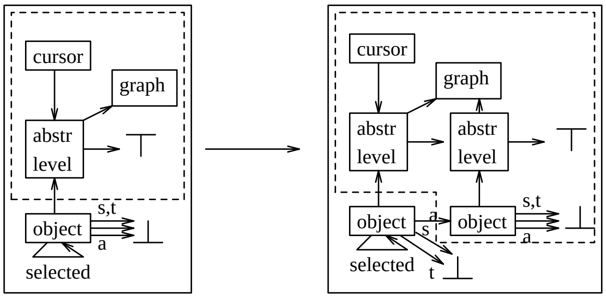

Fig. 8. Create abstraction - case 1

An abstraction node is created in a basic action. For each selected node an abstraction edge is inserted pointing to the abstraction node. This means that there are as many extended actions as objects are selected (figure 8).

The second case is that the abstraction level of the current level already exists. Thus, only a new abstraction node has to be inserted which is the basic action. Similar to the first case all selected nodes are joined to the abstraction node by an a-edge (figure 9).

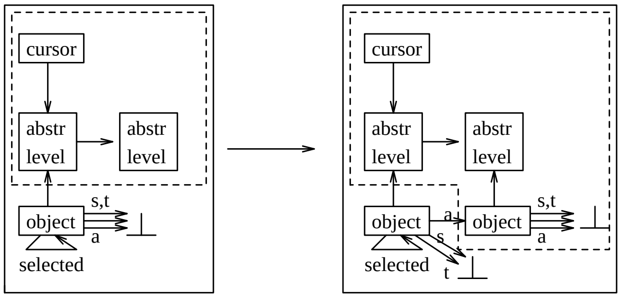

Fig. 9. Create abstraction - case 2.1

This production models the first kind of extended actions. The second one follows from the fact that the abstraction level of the current one can already be shown in any number of windows which is not possible in the case that the abstraction level is newly created. For each window showing the abstraction level a new symbol of the abstraction node has to be created (figure 10). Thus, for this type of AGG-operation two different kinds of extended actions are needed. But, although different kinds of extended actions are used to describe the abstraction of nodes they all can be applied in parallel due to the fact that they overlap in one basic action.

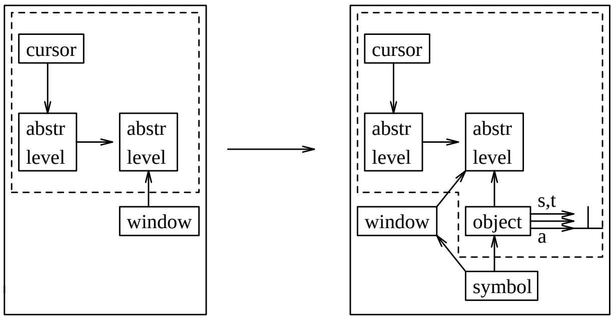

Fig. 10. Create abstraction - case 2.2

The input of the abstraction function is completed automatically as far as this is possible unambigously. This concept is more complex than the described ones and would lead to a set of actions where not all can be done in parallel. However, other AGG-operations as described in section 2 can be treated in the same way as this has been done with the sample AGG-operations in this section. AGG-operations like node and edge insertion and deletion, selection of objects, creation of windows and graphs and changing to another graph are already specified in [Tae92] in a similar way.

The initial state displayed in figure 4 together with sets of actions for each AGG-operation form a graph specification of the AGG-system which describes all possible state transitions of it. Moreover, the specification gives hints to what actions can be done in parallel and how they have to be synchronized.

The type of graph transformation used in this section is formalized in the next section dealing with amalgamated graph transformations.

4 Amalgamated Graph Transformations

Amalgamated graph transformations extend the algebraic kind of parallel graph rewriting which is described in [EK76]. That approach to parallel rewriting of graphs is a basic version where the whole mother graph has to be covered by occurrences of left hand sides of productions. Interfaces between the different occurrences have to remain constant during the parallel replacement.

For amalgamated graph transformations these restrictions are removed. There, dynamic interfaces are allowed which means that interfaces can change during a rewriting step. Moreover, the covering of the mother graph by occurrences can be partial. Only the covered part is replaced, the context is joined unchanged to the derived graph (cf. figure 7). When describing complex operations by graph transformations all graph productions involved are put together into an operational production set. These sets are similar to tables in other approaches to parallel rewriting grouping several productions together.

Note that the AGG-system implements the single-pushout approach to graph rewriting, but a first graph specification as introduced in section 3 is now formalized using amalgamated graph transformations in the double-pushout approach. Although one could see a conflict in using two different kinds of graph rewriting the main emphasis is put in the usage of graph transformations for the specification of the AGG-system, anyway. Moreover, it seems to be easy to generalize the concepts of amalgamated transformations also to partial graph morphisms ([Löw93]), at least using partial morphisms for the description of productions. This means performing amalgamated transformations in the single pushout approach to rewriting, too.

The amalgamated graph transformations build on labelled graphs and total label preserving graph morphisms which form a category $\underline{\mathrm{GRAPH}}$. (For this category and its properties see [Ehr79], [EHKP92], etc.) Using category $\underline{\mathrm{GRAPH}}$ amalgamated graph transformations are one instantiation of parallel high-level replacement which is formulated in an algebraic framework based on categories in [Tae92]. Parallel high-level replacement systems generalize the transformation process known from graph grammars to other kinds of replacement systems. Such systems allow, for example, relational structures, algebraic specifications, Petri nets, etc. to be replaced. The general framework of high-level replacement systems is introduced in [EHKP92] and extended to the parallel case where different kinds of parallel transformations are introduced.

First, we review some basic concepts of the algebraic theory to graph grammars:

- Let C_E and C_N be fixed color alphabets for nodes and edges. A graph $G = (G_E, G_N, s_G, t_G, m_{GE}, m_{GN})$ consists of the sets of edges and nodes, G_E and G_N, mappings $s_G, t_G : G_E \to G_N$, called source resp. target map, and the coloring maps for arcs and nodes, $m_{GE} : G_E \to C_E$ and $m_{GN} : G_N \to C_N$.
- Given two graphs G and G' a graph morphism $f : G \to G'$ is a pair of maps $(f_E : G_E \to G'_E, f_N : G_N \to G'_N)$ such that f_E and f_N are compatible with source and target mappings and label preserving.
- A production $p = (L \leftarrow K \to R)$ consists of three graphs L, K and R, called left hand side, gluing graph and right hand side, resp. and two graph morphisms $l : K \to L$ and $r : K \to R$. If l and r are injective p is called injective.
- An occurrence of the left hand side L in a graph G is a graph morphism. It is also called occurrence of production p. Production p is applicable to G if its occurrence satisfies the gluing condition ([Ehr79]).

Definition 1 (Subproduction). Given a production $p = (L \stackrel{l}{\leftarrow} K \stackrel{r}{\to} R)$ a production $s = (L_s \stackrel{l_s}{\leftarrow} K_s \stackrel{r_s}{\to} R_s)$ is called subproduction of p if there are graph morphisms $l' : L_s \to L$, $k' : K_s \to K$ and $r' : R_s \to R$ called embedding of s into p (short $s \to p$) such that $l' \circ l_s = l \circ k'$ and $r' \circ r_s = r \circ k'$.

Subproduction s is called proper w.r.t. p if productions s and p and morphisms l', k' and r' are injective and $l'^{-1} \circ l(K) \subseteq l_s(K_s)$ and $r'^{-1} \circ r(K) \subseteq r_s(K_s)$.

Example 1. In figure 6 the left and right hand side of a production are shown whereas the gluing graph is equal to the left hand side. Morphisms run between objects named alike and corresponding edges existing in between.

Modelling AGG-operations of section 3 we distinguished basic actions and extended actions. Describing extended actions by productions basic actions have to be subproductions of them.

Looking at figure 6 again, the subproduction is displayed by the part bordered with dashed lines and interpreted in the way it was done above. It can be included in the production solidly framed.

In section 3 we specified one AGG-operation by a basic action and a set of extended actions. All extended actions overlap in the basic action which should be executed only once. There, the basic action can also be regarded as an interface action between all extended actions. This schema is now generalized by allowing more than one interface between different actions and leads to operational production sets. They consist of a number of productions each equipped with a set of their subproductions. They model the possible interfaces each action can have to other actions.

Definition 2 (Operational production set). An operational production set $S = \{(e_k, SP_k) | k \in I\!N\}$ is a set of pairs consisting of a production e_k which is also called elementary production together with a set of subproductions SP_k of e_k.

An operational production set is called simple if each SP_k in S consists of just one subproduction which has only one embedding into e_k.

Example 2. We specified the second case of AGG-operation "create abstraction" in section 3 by one basic action and two extended actions. So, the corresponding simple operational production set has two pairs where each one consists of an elementary production modelling an extended action and a set of one subproduction representing the basic action.

The concept of operational production sets is more expressive than that of tables in other approaches to parallel rewriting of graphs (f.ex. in [NA83], [DDK93]) because the set of possible interfaces between productions has to be specified which cannot be done using tables.

For finding a partial covering of the mother graph G we have to look for all different occurrences of the elementary productions of an operational production set S. This means that extended actions are done *for all* different settings in the state they can be done.

Specifying all possible interfaces in S an elementary production is used only for a covering of G if it provides an interface for every other production that will be applied to G. This concept ensures that all extended actions have to overlap in one of their interface actions pairwise.

In the case of specifying the AGG-operations in section 3 all extended actions have only one interface, the basic action. Looking for corresponding coverings all productions overlap in this single subproduction.

To get a partial covering of the mother graph all occurrences of productions found there are glued at their interfaces.

Definition 3 (Gluing of graphs). A *graph morphism star* $MS = (G_i \leftarrow G_{ij} \rightarrow G_j)_{1 \leq i < j \leq n}$ consists of graphs G_i, G_j and G_{ij} and graph morphisms $g_{ij} : G_{ij} \rightarrow G_i$ and $g_{ji} : G_{ij} \rightarrow G_j$ for all $1 \leq i < j \leq n$. Let $\bar{I}_G$ be the equivalence relation generated by the relation $I_G = \{(g_{ij}(x), g_{ji}(x)) \mid x \in G_{ij}, 1 \leq i < j \leq n\}$. The *gluing graph* G of MS is then defined by $G = (\biguplus_{1 \leq i \leq n} G_i)_{/\bar{I}_G}$, the quotient set of the disjoint union of all G_i. Functions $g_i : G_i \rightarrow G$ for all $1 \leq i \leq n$ send each element of G_i to its equivalence class in G.

The gluing graph G can be characterized to be the colimit object of MS in the category GRAPH.(See [EK76], [Ehr79] and [Tae92] for more details to the gluing of graphs.)

Example 3. Gluing two copies of the left hand graph in figure 6 which is solidly bordered at the graph dashed bordered inside yields the graph dotted bordered within the left hand side of figure 7.

Definition 4 (Partial dynamic covering). Given a set of productions $P = \{p_i \mid p_i = (L_i \stackrel{l_i}{\leftarrow} K_i \stackrel{r_i}{\rightarrow} R_i), 1 \leq i \leq n\}$, a set of subproductions $SP = \{s_{ij} \mid s_{ij} = (L_{ij} \stackrel{l_{ij}}{\leftarrow} K_{ij} \stackrel{r_{ij}}{\rightarrow} R_{ij}), 1 \leq i < j \leq n\}$, graphs G and L and a graph morphism $g : L \to G$ a partial dynamic covering $PDCOVERING_{P,SP,L}(G)$ consists of the diagram below if the following conditions are satisfied:

1. Each production s_{ij} is a subproduction of p_i and p_j, $\forall 1 \leq i < j \leq n$.
2. Graph L is the gluing graph of graph morphism star $(L_i \stackrel{g_{ij}}{\leftarrow} L_{ij} \stackrel{g_{ji}}{\rightarrow} L_j)_{1 \leq i < j \leq n}$.
3. $L_i \to G = L_i \to L \to G$, $\forall 1 \leq i \leq n$.

A spider covering is a partial dynamic covering where all s_{ij}, $1 \leq i < j \leq n$, are the same. A partial dynamic covering is fully synchronized if $g_i(L_i) \cap g_j(L_j) \subseteq g_i \circ g_{ij}(L_{ij}) \cup (g_i \circ l_i(K_i) \cap g_j \circ l_j(K_j)), \forall 1 \leq i < j \leq n$.

$$
\begin{tikzcd}
 & & L_i \arrow[dll] \arrow[dl] & K_i \arrow[l] \arrow[r] & R_i \\
G & L \arrow[l] & L_{ij} \arrow[u] \arrow[d] & K_{ij} \arrow[l] \arrow[r] \arrow[u] \arrow[d] & R_{ij} \arrow[u] \arrow[d] \\
 & & L_j \arrow[ull] \arrow[ul] & K_j \arrow[l] \arrow[r] & R_j
\arrow[from=1-3, to=2-4, phantom, "(=)"]
\arrow[from=1-4, to=2-5, phantom, "(=)"]
\arrow[from=2-3, to=3-4, phantom, "(=)"]
\arrow[from=2-4, to=3-5, phantom, "(=)"]
\arrow[from=1-3, to=2-1, phantom, "(=)"]
\arrow[from=3-3, to=2-1, phantom, "(=)"]
\end{tikzcd}
$$

The way the construction of a partial dynamic covering is described above is now formalized in the following covering construction.

Proposition 5 (Allquantified covering construction). *Given an operational production set $S = \{(e_k, SP_k) \mid 1 \leq k \leq m\}$ and a graph G there is a partial dynamic covering $PDCOVERING_{P,SP,L}(G)$ with*

1. $P = \{p_i \mid 1 \leq i \leq n\}$ *with* $p_i = (L_i \leftarrow K_i \rightarrow R_i) = e_k$ *for all different occurrences* $L_i \to G$ *and some* $1 \leq k \leq m$,
2. $SP = \{s_{ij} \mid 1 \leq i < j \leq n\}$ *with* $s_{ij} = (L_{ij} \leftarrow K_{ij} \rightarrow R_{ij}) \in SP_k \cap SP_{k'}$ *if* $p_i = e_k$ *and* $p_j = e_{k'}$, $1 \leq k, k' \leq m$,
3. $\forall 1 \leq i < j \leq n : L_{ij} \to L_i \to G = L_{ij} \to L_j \to G$
4. L *is the gluing graph of* $(L_i \leftarrow L_{ij} \rightarrow L_j)_{1 \leq i < j \leq n}$.

The existence of a partial dynamic covering can be proven easily. (See [Tae92] for more details.)

In general, a partial dynamic covering need not cover the whole part of a mother graph G that *can* be covered by given productions. The allquantified construction of a covering is only one way of covering the mother graph. Other covering constructions are imaginable.

The allquantified covering construction is not unique in general, but in the special case that the given operational production set is simple, as this holds in all examples of section 3, uniqueness can be proven.

Proposition 6 (Uniqueness of special allquantified covering construction). *Given a simple operational production set $S = \{(e_k, SP_k) | k \in I\!N\}$ there is a unique partial dynamic covering as constructed in proposition 5.*

If all subproductions in $SP_k, 1 \leq k \leq n$, are the same the resulting covering is a spider covering.

This result follows directly from the assumptions chosen above.

Example 4. All AGG-operations presented in section 3 can be described by simple operational production sets where all subproductions are the same for each operation. Therefore, all resulting coverings are spiders.

Given a partial dynamic covering a graph transformation is performed in two steps. First a new production is generated that models the synchronization of all extended actions at their interface actions. This is done by gluing the left hand sides, right hand sides and gluing graphs of the productions at their interfaces in order to get the new so-called amalgamated production. This production is then applied to the actual graph by constructing the usual double pushout as described in [Ehr79], [EHKP92], etc.

Definition 7 (Amalgamated graph transformation). Given a partial dynamic covering $PDCOVERING_{P,SP,L}(G)$ with sets of productions $P = \{p_i | p_i = (L_i \leftarrow K_i \rightarrow R_i), 1 \leq i \leq n\}$, $SP = \{s_{ij} | s_{ij} = (L_{ij} \leftarrow K_{ij} \rightarrow R_{ij}), 1 \leq i < j \leq n\}$ and morphism $g : L \rightarrow G$, an amalgamated graph transformation[4] $G \Longrightarrow G'$ over $PDCOVERING_{P,SP,L}(G)$ consists of the following two steps:

1. Construction of an amalgamated production $(L \xleftarrow{l} K \xrightarrow{r} R)$ by generating the gluing graphs L of $(L_i \leftarrow L_{ij} \rightarrow L_j)_{1 \leq i < j \leq n}$, K of $(K_i \leftarrow K_{ij} \rightarrow K_j)_{1 \leq i < j \leq n}$ and R of $(R_i \leftarrow R_{ij} \rightarrow R_j)_{1 \leq i < j \leq n}$. Morphism $K \xrightarrow{l} L$ $(K \xrightarrow{r} R)$ is the unique morphism such that

   $$K_i \rightarrow K \xrightarrow{l} L = K_i \rightarrow L_i \rightarrow L (K_i \rightarrow K \xrightarrow{r} R = K_i \rightarrow R_i \rightarrow R), \forall 1 \leq i < j \leq n$$

   $$\begin{tikzcd}
   & L_{ij} \arrow[dl] \arrow[dd] & & K_{ij} \arrow[ll] \arrow[rr] \arrow[dl] \arrow[dd] & & R_{ij} \arrow[dl] \arrow[dd] \\
   L_i \arrow[dd] & & K_i \arrow[ll] \arrow[rr] \arrow[dd] & & R_i \arrow[dd] & \\
   & L_j \arrow[dl] & & K_j \arrow[ll] \arrow[rr] \arrow[dl] & & R_j \arrow[dl] \\
   L & & K \arrow[ll] \arrow[rr] & & R &
   \end{tikzcd}$$

[4] Note that the kind of transformation just defined is called amalgamated *two-level derivation* in [ET92] and [Tae92].

2. Application of the amalgamated production $p = (L \xleftarrow{l} K \xrightarrow{r} R)$ to graph G with occurrence $g : L \rightarrow G$ by a double pushout, i.e. constructing first the pushout-complement graph $D = G - g(L - l(K))$. The gluing graph G' of $(D \leftarrow K \xrightarrow{r} R)$ is then the result of the amalgamated graph transformation.

Morphisms $K \xrightarrow{l} L$ and $K \xrightarrow{r} R$ always exist in a unique way because they are the universal morphisms out of the colimit object K. Obviously, p is a production in the usual sense as defined above.

Example 5. In figure 7 all parts which lie inside the dotted lines belong to the left hand and right hand side of the new amalgamated production, resp. The gluing graph of the new production is the same as the left hand side and the morphisms in between are just inclusions.

Proposition 8 (Injectivity of amalgamated production). *If all subproductions s_{ij} in SP of a spider covering $PDCOVERING_{P,SP,L}(G)$ are proper w.r.t. their productions p_i and p_j in P the resulting production p of step 1 in definition 7 is injective.*

Proofsketch: The gluing of all left hand side graphs of a spider covering can be characterized as a special colimit called multiple pushout. The same holds for the right hand sides and gluing graphs. Multiple pushouts inherit injective morphisms similar to simple pushouts. Furthermore, all s in SP are proper w.r.t. their corresponding productions. Therefore, $K \xrightarrow{l} L$ and $K \xrightarrow{r} R$ are injective.

Proposition 9 (Applicability of amalgamated production). *If all productions p_i in P of a spider covering $PDCOVERING_{P,SP,L}(G)$ are applicable to G and $PDCOVERING_{P,SP,L}(G)$ is fully synchronized the amalgamated production p in definition 7 is applicable to G, too.*

Proofsketch: Consider the interesting case of two different productions p_i and p_j in P being applicable to G. Their occurrences g_i and g_j can overlap either in the occurrence of their subproduction or their gluing graphs. First $g_i \circ g_{ij} = g_j \circ g_{ji}$ and the second case does not destroy the applicability, anyway. (For the proofs of these two propositions see [Bet92].)

The amalgamated graph transformations can be used to describe production sets which have a certain regularity. Using elementary productions for the generation of a production suitable for the given graph large sets of similar productions building a production family can be avoided. (See also [Göt87b].)

Amalgamated graph transformations combine concepts of parallel and sequential transformations by generating a new production out of elementary productions in parallel and applying this production in the usual sequential way of graph rewriting. Thus, on one hand we have the advantage of using the compact notion of operational production sets to describe complex actions, on the other hand all results which are known for sequential transformations (like Church-Rosser-Properties, Parallelism Theorem, Concurrency Theorem, Embedding Theorem, Amalgamation Theorem, etc.) hold also for the sequential part of this type of transformation.

For the specification of the AGG-system these theoretical results can give information about which AGG-operations are independent of each other and how they can be synchronized. Furthermore, the construction of operations in the transformation component from different editor operations can be supported using concepts of concurrency.

5 Outlook

Extensions of the AGG-system planned next include labels, finding of occurrences, completion of partial occurrences and triggering and execution of transformation sequences. Further extensions would be the implementation of the double pushout approach to graph rewriting, layouting of graphs, and more. AGG has been used in courses on the algebraic approach to graph transformation at the Technical University of Berlin. It is intended to be used also as a tool for case studies on the algebraic approach to graph transformation.

Within this paper we presented first ideas for a graph specification of internal structures of the AGG-system. It would be nice to continue this work to get a feeling how amalgamated graph transformations can be used for specifying complex operations. For a graph specification of the entire AGG-system it seems to be necessary to extend the kind of graph transformation presented to attributed transformations. The attributes would comprise all graphical information of graphs.

AGG-operations are specified by amalgamated graph transformations over partial dynamic coverings constructed in an allquantified way. For special cases uniqueness of such a construction can be shown. Moreover, other covering constructions are imaginable dependent on the problem to be specified.

Restricting the definition of subproductions to that in [BFH87] would lead to further results concerning sequentialization of amalgamated graph transformations extended to more than two productions.

Due to the two-step mechanism of the graph transformation presented it is possible to get use of the whole field of theoretical results for sequential graph transformations. Doing this efficiently, conditions which have to be checked for occurrences of productions have to be adapted to partial dynamic coverings.

Acknowledgements: We thank Ralph Betschko and Hartmut Ehrig for interesting discussions and the referees for valuable comments.

References

[BCF91] D. A. Bailey, J. E. Cuny, and C. D. Fisher, *Programming with very large graphs*, In Ehrig et al. [EKR91].

[Bet92] R. Betschko, *Parallele Graphgrammatiken mit Synchronisation*, Studienarbeit, Technical University of Berlin, Dep. of Comp. Sci., 1992.

[Bey91] M. Beyer, *GAG: Ein graphischer Editor für algebraische Graphgrammatiksysteme*, Diplomarbeit, Technical University of Berlin, Dep. of Comp. Sci., 1991.

[Bey92] M. Beyer, *AGG1.0 - Tutorial*, Technical University of Berlin, Department of Computer Science, 1992.

[BFH87] P. Böhm, H.-R. Fonio, and A. Habel, *Amalgamation of graph transformations: a synchronization mechanism*, J. of Comp. and Syst. Sci. **34** (1987), 377–408.

[Bra91] F. Brandenburg, *Layout Graph Grammars: The Placement Approach*, In Ehrig et al. [EKR91], pp. 144–156.

[DDK93] G. David, F. Drewes, and H.-J. Kreowski, *Hyperedge Replacement with Rendezvous*, 1993, to appear in proc. of 12th conf. of FST and TCS'92.

[EHKP92] H. Ehrig, A. Habel, H.-J. Kreowski, and F. Parisi-Presicce, *Parallelism and concurrency in High Level Replacement Systems*, Math. Struct. in Comp. Sci. **1** (1992), 361–404.

[Ehr79] H. Ehrig, *Introduction to the algebraic theory of graph grammars*, 1st Int. Workshop on Graph Grammars and their Application to Computer Science and Biology, LNCS 73, Springer, 1979, pp. 1–69.

[EK76] H. Ehrig and H.-J. Kreowski, *Parallel Graph Grammars*, Automata,Languages, Development (A. Lindenmayer and G. Rozenberg, eds.), Amsterdam: North Holland, 1976, pp. 425–447.

[EKR91] H. Ehrig, H.-J. Kreowski, and G. Rozenberg (eds.), *4th Int. Workshop on Graph Grammars and Their Application to Computer Science, LNCS 532*, Springer, 1991.

[ET92] H. Ehrig and G. Taentzer, *From Parallel Graph Grammars to Parallel High- Lev el Replacement Systems*, Lindenmayer Systems, Springer, 1992, pp. 283–303.

[Göt87a] H. Göttler, *Graph grammars and diagram editing*, 3rd Int. Workshop on Graph Grammars and Their Application to Computer Science, LNCS 291, Springer, 1987, pp. 216–231.

[Göt87b] H. Göttler (ed.), *Graphgrammatiken in Softwareengineering*, Universität Erlangen, 1987.

[Him91] M. Himsolt, *GraphEd: An interactive tool for developing graph grammars*, In Ehrig et al. [EKR91], pp. 61–65.

[JRV82] D. Janssens, G. Rozenberg, and R. Verraedt, *On sequential and parallel node-rewriting graph grammars*, Computer Vision, Graphics and Image Processing **18** (1982), 279–304.

[Kre92] H.-J. Kreowski, *Parallel Hyperedge Replacement*, Lindenmayer Systems, Springer, 1992, pp. 271–282.

[LB93] M. Löwe and M. Beyer, *AGG — An Implementation of Algebraic Graph Rewriting*, LNCS 690, Springer, 1993, Rewriting Techniques and Applications, Fifth Int. Conf., RTA'93.

[Löw93] M. Löwe, *Algebraic approach to single-pushout graph transformation*, Theoretical Computer Science **109** (1993), 181–224.

[NA83] A. Nakamura and K. Aizawa, *On a relationship between graph L-systems and picture languages*, Theoretical Computer Science **24** (1983), 161–177.

[Nag87] M. Nagl, *A software development environment based on graph technology*, 3rd Int. Workshop on Graph Grammars and Their Application to Computer Science, LNCS 291, Springer, 1987, pp. 458–478.

[NP91] F. Newbery Paulisch, *The Design of an Extendible Graph Editor*, Ph.D. thesis, University of Karlsruhe, Department of Informatics, March 1991.

[RS86] G. Rozenberg and A. Salomaa, *The Book of L*, Springer, Berlin, 1986.

[RS92] G. Rozenberg and A. Salomaa, *Lindenmayer Systems*, Springer, 1992.

[Sch91] A. Schürr, *Operationales Spezifizieren mit programmierten Graphersetzungssystemen*, Deutscher Universitätsverlag GmbH, Wiesbaden, 1991.

[Tae92] G. Taentzer, *Parallel High-Level Replacement Systems*, Tech. Report 92/10, Technical University of Berlin, Dep. of Comp. Sci., 1992.