Towards a New Algebraic Approach to Graph Transformation. Basic Concepts, Sequentialization and Parallelism: Long Version

Pedro Pablo Pérez Velasco
Juan de Lara

Universidad Autónoma de Madrid
Escuela Politécnica Superior
Ingeniería Informática
pedro.perez@uam.es, jdelara@uam.es

Informe Técnico 3/2006

Towards a New Algebraic Approach to Graph Transformation. Basic Concepts, Sequentialization and Parallelism: Long Version

Pedro Pablo Pérez Velasco, Juan de Lara
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Spain
{ *pedro.perez, jdelara* }*@uam.es*

Abstract

This paper presents a new characterization of graph transformation rules for simple digraphs based on boolean matrix algebra. We introduce the concept of *coherence*, which allows the analysis of potential incompatibilities among rules that take part in a sequence of productions. Concurrency is studied under the interleaving and the explicit parallelism views. For the former, the notion of sequential independence is generalized to arbitrary permutations of rules. For the latter, rule composition is defined, which does not generate intermediate states.

Keywords: Graph Transformation, Single and Double Pushout Approaches, Boolean Matrix Algebra, Sequential Independence, Explicit Parallelism

1 Introduction

Graph Transformation [8] is becoming increasingly popular in computer science as it provides a formal basis for graph manipulation. Transformations of this data structure are central to many application areas, such as modelling with visual languages [19], visual simulation [16], picture processing and generation [5] and model transformation [12].

The classical algebraic approach to graph transformation is based on *category theory* [6], and has a rich body of theoretical results which have been developed over the last 30 years (see [8]). Thus, graph transformations expressed as graph rewriting become not only graphical and intuitive but also formal, declarative and high-level models, subject themselves to analysis [8] [9] [14]. Nonetheless, methods to increase efficiency and new analysis techniques that can be implemented in tools are needed for real industrial applications.

In contrast to the categorical-algebraic approach, we propose an algebraic characterization based on boolean matrix algebra. Thus, in the present paper we start working with simple digraphs, which can be represented as boolean matrices. In this way, a graph transformation rule can be characterized by two matrices L and R representing the left and right hand sides (LHS and RHS), together with deletion and addition matrices (specifying which edges are deleted and added respectively). Similar concepts apply to nodes.

We present the concept of *coherence* for a *sequence* of rules of arbitrary length, together with the conditions that must be fulfilled by the rules for the sequence to be applicable, in the sense that a rule does not prevent the execution of the following ones. Moreover, we introduce the complementary concept of *compatibility* by considering the application of the sequence to a minimal initial simple digraph and giving some conditions under which the resulting digraph is well defined. A third basic concept, which we call *G-applicability* examines the impact on elements used by productions inside a concatenation in case, for example, a permutation is applied.

Similar concepts are introduced for the *composition* of rules, where a single rule is obtained comprising all the rules in the sequence (thus, no intermediate states are generated, modelling explicit parallelism). We also generalize the classical concept of *sequential independence* by considering permutations in rule sequences of arbitrary lengths.

In the present work, most analysis are made independently of the host graph, that is, we do not consider the match from the rule's LHS to the host graph. However, the match can be seen as an *operator* modifying

the rule's LHS and RHS by *including* the context in which the rules are applied (induced by the match)[1]. The advantages of this approach are twofold. First, all properties under study are *inherent* to the grammar,[2]and second, it has the practical advantage that the analysis can be performed by a tool in the phase of specification of the grammar, independently of any host graph. From a practical point of view, once the grammar is specified, the calculated results for coherence, parallel independence and all information gathered can then be used when a host graph is considered, while on the theoretical side concepts introduced need not be modified.

The rest of the paper is organized as follows. Section 2 presents related approaches to graph transformation with their main results. Section 3 introduces the basic concepts of our approach and the characterization of graph transformation rules. Section 4 presents the notion of concatenation of productions and the conditions for the sequence to be applicable (that is, coherence). Section 5 defines the related concept of *composition* of a set of rules, where no intermediate steps are generated, and the conditions for such composition to be compatible (the resulting rule produces a simple digraph). Related concepts are *compatibility* and *minimal initial digraph*. Section 6 studies *permutation* of rules and sequential independence, introducing *G-applicability*. Section 7 presents the results on explicit parallelism and finally, section 8 ends with the conclusions and future work.

2 Related Work

The algebraic approach to graph transformation [8] is based on category theory [18]. There are two main approaches, the double pushout (DPO) [6] [3] and the single pushout (SPO) [17] [10]. Both approaches model direct derivations as one or two pushouts, in order to carry out the gluing and deletion of elements in the graph where the rule is applied (called *host graph*). The main ideas of both approaches are shown in subsections 2.1 and 2.2. In [20], they transform simple digraphs, but still, they follow a categorical approach by defining category **Pfn(Graph)** of simple digraphs and partial functions and using pushouts to model derivations. Other approaches (double pullback [7], logic-based [21], algebraic-logic [4], relation-algebraic [15]) are less relevant to our approach and will not be discussed.

2.1 Double Pushout Approach

In the DPO approach to graph rewriting, a direct derivation is represented by a double pushout in the **Graph** category of graphs and total graph morphisms. Productions can be defined as three graph components, separating the elements that should be preserved from the left and right hand sides of the rule. In this way, a production $p : (L \xleftarrow{l} K \xrightarrow{r} R)$ consists of a production name p and a pair of injective graph morphisms $l\colon K \to L$ and $r\colon K \to R$. Graphs L, R and K are called the left-hand side (LHS), right-hand side (RHS) and the interface of p. Morphisms l and r are usually injective and can be taken to be inclusions without loss of generality.

The interface of a production depicts the elements that should be preserved by the production application. The elements in $L - K$ are deleted, while the elements of $R - K$ are added. Figure 1 shows a simple DPO production named *del*. It can be applied if a path of three nodes is found. If this is the case, the production eliminates the last node and creates a loop edge in the second node. Morphisms l and r between L, K and R are depicted with numbers. This is the convention we use throughout the paper.

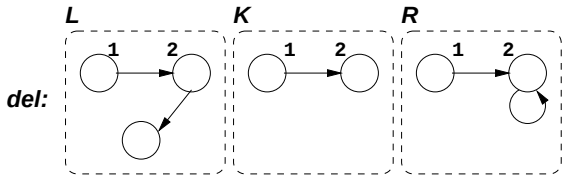

Figure 1: A simple DPO production.

[1]The idea behind this is to adapt the rule to the context on the one hand and on the other to avoid dangling edges or, using the jargon introduced in this paper, maintain compatibility.

[2]More precisely, to the graph transformation system.

A direct derivation can be defined as an application of a production to a graph through a match by constructing two pushouts. Thus, given a graph G, a production $p : (L \xleftarrow{l} K \xrightarrow{r} R)$, and a match $m\colon L \to G$, a direct derivation from G to H using p (based on m) exists iff the diagram in Figure 2 can be constructed, where both squares are required to be pushouts in **Graph**. D is called the context graph, and we write $G \overset{p,m}{\Longrightarrow} H$ or $G \overset{p}{\Longrightarrow} H$.

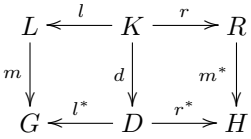

Figure 2: Direct Derivation as DPO construction.

Figure 3 shows the application of the *del* rule to a graph. Morphisms m, d and m^* are depicted by showing the correspondence of the vertices in the production and the graph.

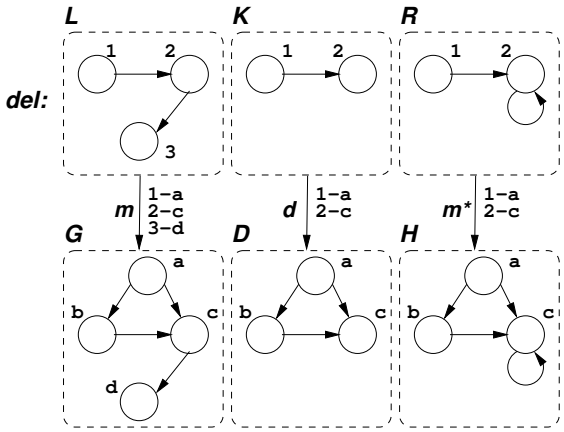

Figure 3: DPO Direct Derivation Example.

In order to apply a production to a graph G, a pushout complement has to be calculated to obtain graph D. The existence of this pushout complement is guaranteed iff the so-called *dangling* and *identification* conditions are satisfied. The first one establishes that a node in G cannot be deleted if this causes dangling edges. The second condition states that two different nodes or edges in L cannot be identified (by means of a non-injective match) as a single element in G if one of the elements is deleted and the other is preserved. Moreover, the injectivity of $l\colon K \to L$ guarantees the uniqueness of the pushout complement. In the example in Figure 3 the match $(1-a,\ 2-b,\ 3-c)$ does not fulfil the dangling condition, as the deletion of the c node would cause dangling edges. Thus, the production cannot be applied at this match.

A graph grammar can be defined as $\mathcal{G} = \langle (p : L \xleftarrow{l} K \xrightarrow{r} R)_{p\in P}, G_0\rangle$ [3], where $(p : L \xleftarrow{l} K \xrightarrow{r} R)_{p\in P}$ is a family of productions indexed by their names, and G_0 is the starting graph of the grammar.[3]The semantics of the grammar are all the reachable graphs that can be obtained by applying the rules in $\mathcal{G}$, until none of them are applicable. Note how a system state can be described with a graph. The events changing the system state can thus be modelled using graph transformation rules. In real systems, parallel actions can take place. Two main approaches can be followed in order to describe and analyze parallel computations. In the first one, parallel actions are sequentialized, giving rise to different *interleavings*. In the second approach, the actions are really simultaneous and this is called *explicit parallelism*.

In the interleaving approach, two actions (i.e. rule applications) are considered to be parallel if they can be performed in any order yielding the same result. This can be observed from two points of view. The first one is called *parallel independence*, and states that two alternative direct derivations $H_1 \overset{p_1}{\Longleftarrow} G \overset{p_2}{\Longrightarrow} H_2$ are independent if there are direct derivations such that $H_1 \overset{p_2}{\Longrightarrow} X \overset{p_1}{\Longleftarrow} H_2$. That is, both derivations are not in conflict, but one can be postponed after the other. If one element is preserved by one derivation, but deleted by

[3]If G_0 is not considered, but the family of productions alone, then it is called a *graph transformation system*.

the other one, then the latter is said to be *weakly parallel independent* of the first. Thus, parallel independence can be defined as mutual weak parallel independence. On the other hand, two direct derivations are called *sequential independent* if they can be performed in a different order without changing the result. That is, both $G \overset{p_1}{\Longrightarrow} H_1 \overset{p_2}{\Longrightarrow} X$ and $G \overset{p_2}{\Longrightarrow} H_2 \overset{p_1}{\Longrightarrow} X$ yield the same result.

The conditions for sequential and parallel independence are given in the *Local Church Rosser* theorem [3]. It says that two alternative parallel derivations are parallel independent if their matches only overlap in items that are preserved. Two consecutive direct derivations are sequentially independent if the match of the second one does not depend on elements generated by the first one, and the second derivation does not delete an item that has been accessed by the first. Moreover, if two direct alternative derivations are parallel independent, their concatenation is sequential independent and vice versa.

The explicit parallelism [3] [1] view abstracts from any application order. Thus, no intermediate states are produced. In this approach, a derivation is modelled by a single production, called *parallel production*. Given two productions, p_1 and p_2, the parallel production $p_1 + p_2$ is the disjoint union of both. The application of such production is denoted as $G \overset{p_1+p_2}{\Longrightarrow} X$. Two problems arise here: the sequentialization of a parallel production (*analysis*), and the parallelization of a derivation (*synthesis*). In the DPO, the *parallelism theorem* states that a parallel derivation $G \overset{p_1+p_2}{\Longrightarrow} X$ can be sequentialized into two derivations ($G \overset{p_1}{\Longrightarrow} H_1 \overset{p_2}{\Longrightarrow} X$ and $G \overset{p_2}{\Longrightarrow} H_2 \overset{p_1}{\Longrightarrow} X$) that are sequentially independent. Conversely, two sequential independent derivations can be put in parallel if they are sequentially independent.

2.2 Single Pushout Approach

In the single pushout approach (SPO) to graph transformation, rules are modelled with two component graphs (L and R), and direct derivations are built with one pushout (which performs the gluing and the deletion). Thus SPO works with the $\mathbf{Graph}^P$ category of graphs and partial graph morphisms. An SPO production p can be defined as $p : (L \overset{r}{\rightarrow} R)$, where r is an injective partial graph morphism. A match for a production p in a graph G is a *total* morphism $m\colon L \rightarrow G$. Given a production p and match m for p in G, the direct derivation from G is the pushout of p and m in $\mathbf{Graph}^P$.

Figure 4 shows an example of the rule in Figure 1, but expressed in the SPO approach. The rule is applied to the same graph G as in Figure 2, but at a different match. The match in the SPO example was not valid in the previous DPO example. In SPO, there is no dangling condition: if an edge dangles, it is deleted. Thus the rule can be seen as having side effects. In addition, in case of a conflict with the identification condition, because of a non-injective matching, the conflicting elements are deleted. In the example, node c is deleted and edges (a, c) and (c, d) are also erased, as they were dangling edges.

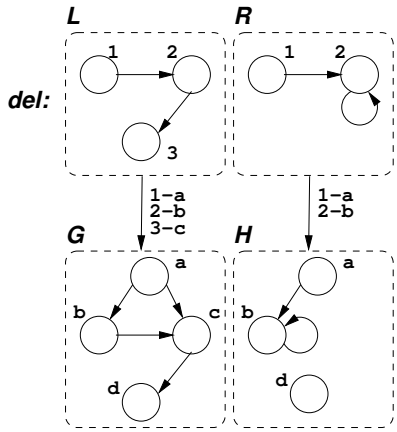

Figure 4: SPO Direct Derivation Example.

Results for explicit parallelism are slightly different in SPO. In this approach, a parallel direct derivation $G \overset{p_1+p_2}{\Longrightarrow} X$ can be sequentialized into $G \overset{p_1}{\Longrightarrow} H_1 \overset{p_2}{\Longrightarrow} X$ if $G \overset{p_2}{\Longrightarrow} H_2$ is weakly parallel independent of $G \overset{p_1}{\Longrightarrow} H_1$ (and similar for the other sequentialization). Thus, as this condition may not hold, there are parallel direct derivations that do not have an equivalent interleaving sequence.

3 Characterization And Basic Properties

This section presents the basic concepts in our approach. We start defining simple digraphs, which can be represented as boolean matrices, introduce basic operations on these matrices and show a characterization of graph transformation rules using them. We formulate the conditions for a production to be *compatible* (that is, it defines a simple digraph) and the concept of *completion*, where matrices representing graphs are modified to permit operations between them.

A graph $G = (V, E)$ consists of two sets, one of nodes $V = \{V_i \mid i \in I\}$ and one of edges $E = \{(V_i, V_j) \in V \times V\}$, that can be simply thought of as arrows connecting two nodes. If every arrow has a direction then we shall call it a *digraph*, which stands for "directed graph". In this paper we are concerned with *simple digraphs*, "simple" meaning that only one arrow is allowed between two nodes. As an example, the complete simple digraph with three vertices may be found in figure 5 below.

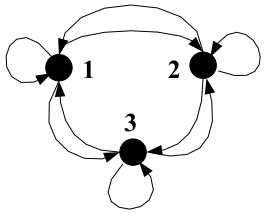

Figure 5: Complete Simple Digraph with Three Nodes

Any simple digraph G is uniquely determined through its associated matrix, known as *adjacency matrix* A_G, whose element a_{ij} is defined to be one if there exists an arrow joining vertex i with vertex j, and zero otherwise. In addition, a vector that we call *nodes vector* V_G is associated to our digraph G, with its elements equal to one if the corresponding node is present in G and zero otherwise.

Definition 3.1 (Boolean Matrix Product) *Let $M_G = (g_{ij})_{i,j\in\{1,\ldots,n\}}$ and $M_H = (h_{ij})_{i,j\in\{1,\ldots,n\}}$ be the adjacency matrix of digraphs G and H, respectively. Their boolean product is a matrix whose elements are defined by:*

$$(M_G \odot M_H)_{ij} = \bigvee_{k=1}^{n} (g_{ik} \wedge h_{kj}) \tag{1}$$

Element (i, j) in the boolean product matrix is one if there exists an edge joining node i in digraph G with some node k in the same digraph and another edge in digraph H starting in k and ending in j. The value will be zero otherwise. If for example we want to check whether node j is reachable starting in node i in h steps or less, we may calculate $\bigvee_{k=1}^{n} A^{(k)}$ and test if element (i, j) is one.[4]

Definition 3.2 (Compatibility) *A boolean matrix M and a vector of nodes N are* compatible *if they define a simple digraph: no edge is incident to any node that does not belong to the digraph. That is, there are no dangling edges.*

In the algebraic-categorical approach this condition is checked when building a direct derivation. The condition is known as *dangling condition* and the idea behind it is to obtain a closed set of entities. The next proposition provides a criteria for testing compatibility for simple digraphs.

Definition 3.3 (Norm of a Boolean Vector) *Let $N = (v_1, \ldots, v_n)$ be a boolean vector, then its norm $\|\cdot\|_1$ is given by*

$$\|N\|_1 = \bigvee_{i=1}^{n} v_i \tag{2}$$

[4]In order to distinguish when we are using the standard or boolean product, in the latter, exponents will be enclosed between brackets.

Proposition 3.4 *A pair (M, N), where M is an adjacency matrix and N a vector of nodes, is compatible if and only if they verify*

$$\left\| \left(M \vee M^t\right) \odot \overline{N} \right\|_1 = 0 \tag{3}$$

where t *denotes transposition.*

Proof

□In an adjacency matrix, row i represents outgoing edges from vertex i, while column j are incoming edges to vertex j. Moreover, $(M)_{ik} \wedge \left(\overline{N}\right)_k = 1$ iff $(M)_{ik} = 1$ and $(N)_k = 0$, and thus the $i-th$ element of vector $M \odot \overline{N}$ is one if and only if there is a dangling edge in row number i.

We have just considered outgoing edges, but not incoming ones. In this case we have a very similar term: $M^t \odot \overline{N}$.

To finish the sufficient part of the proof - necessity is straightforward - we OR both terms and take norms, in order to detect if there is a 1.■

Remark. We have used in the proof of proposition 3.4 distribution of $\odot$ and $\vee$, or in other words, $(M_1 \vee M_2) \odot M_3 = (M_1 \odot M_3) \vee (M_2 \odot M_3)$. In addition, we also have the distributive law on the left, i.e., $M_3 \odot (M_1 \vee M_2) = (M_3 \odot M_1) \vee (M_3 \odot M_2)$. Besides, we shall state without proof that $\|\omega_1 \vee \omega_2\|_1 = \|\omega_1\|_1 \vee \|\omega_2\|_1$.

Now we consider productions and their characterization. In this paper, we define a production as an application which transforms a simple digraph into another simple digraph, $p : L \to R$. We can describe a production p with two matrices (those with an E superindex) and two vectors (those with an N superindex). In this way, $p = (L^E, R^E; L^N, R^N)$, where the components are respectively the left hand side edges matrix and nodes vector, and the right hand side edges matrix and nodes vector. For further reference,

Definition 3.5 (Production) *A production p is a morphism between two simple digraphs L and R, and can be specified by the tuple*

$$p = \left(L^E, R^E; L^N, R^N\right) \tag{4}$$

where E *stands for* edge *and* N *for* node. L *is the* left hand side *and* R *is the* right hand side.

A production models deletion and addition actions on both edges and nodes, carried out in the order just mentioned, i.e., first deletion and then addition, so appropriate matrices are introduced to represent them and will be used throughout the paper to prove properties of the rules.

Definition 3.6 (Deletion and Addition of Edges) *Matrices for deletion and addition of edges are defined elementwise by the formulae*

$$e^E = (e)_{ij} = \begin{cases} 1 & \text{if edge } (i,j) \text{ is to be erased} \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

$$r^E = (r)_{ij} = \begin{cases} 1 & \text{if edge } (i,j) \text{ is to be added} \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

Note how given a production $p = (L^E, R^E; L^N, R^N)$, both matrices can be calculated as follows:

$$e^E = L^E \wedge \overline{(L^E \wedge R^E)} = L^E \wedge \overline{R^E} \tag{7}$$

$$r^E = R^E \wedge \overline{(L^E \wedge R^E)} = R^E \wedge \overline{L^E} \tag{8}$$

where $L^E \wedge R^E$ are the elements that are preserved by the rule application (similar to the K component in DPO rules). Thus, using the previous construction, the following two conditions hold and will be frequently used: edges can be added if they do not currently exist and may be deleted only if they are present in the left hand side of the production.

$$e^E \wedge L^E = e^E \tag{9}$$

$$r^E \wedge \overline{L^E} = r^E \tag{10}$$

In a similar way, vectors for the deletion and addition of nodes can be defined.

Definition 3.7 (Deletion and Addition of Nodes)

$$e^N = (e)_i = \begin{cases} 1 & \textit{if node } i \textit{ is to be erased} \\ 0 & \textit{otherwise} \end{cases} \tag{11}$$

$$r^N = (r)_i = \begin{cases} 1 & \textit{if node } i \textit{ is to be added} \\ 0 & \textit{otherwise} \end{cases} \tag{12}$$

The following simple conditions are apparent. The first two state that edges or nodes cannot be rewritten (erased and created or vice versa) by a rule application. This is a consequence of the way in which matrices e and r are calculated. This contrasts with the DPO approach, in which edges and nodes can be rewritten in a single rule. The latter may be useful to forbid the rule application if the dangling condition is violated. The remaining two conditions say that if a node or edge is in the RHS, then it is not deleted, and that if a node or edge is in the LHS, then it is not created.

Proposition 3.8 *Let $p : L \to R$ be a production, the following identities are immediate*

$$r^E \wedge \overline{e^E} = r^E \qquad r^N \wedge \overline{e^N} = r^N \tag{13}$$

$$e^E \wedge \overline{r^E} = e^E \qquad e^N \wedge \overline{r^N} = e^N \tag{14}$$

$$R^E \wedge \overline{e^E} = R^E \qquad R^N \wedge \overline{e^N} = R^N \tag{15}$$

$$L^E \wedge \overline{r^E} = L^E \qquad L^N \wedge \overline{r^N} = L^N \tag{16}$$

Proof
□■

Finally we are ready to characterize a production $p : L \to R$ using deletion and addition matrices, starting from its LHS:[5]

$$R^N = r^N \vee \left(\overline{e^N} \wedge L^N\right) \tag{17}$$

$$R^E = r^E \vee \left(\overline{e^E} \wedge L^E\right) \tag{18}$$

The resulting graph R is calculated by first deleting the elements in the initial graph – $\overline{e^X} \wedge L^X$ – and then adding the new elements – $r^X \vee \left(\overline{e^X} \wedge L^X\right)$ –. It can thus be demonstrated, using proposition 3.8, that in fact it doesn't matter whether deletion is carried out first and addition later or viceversa.

So there are two ways to characterize a production so far, either using its initial and final states (see definition 3.5) or the operations it specifies:

$$p = \left(e^E, r^E; e^N, r^N\right) \tag{19}$$

As a matter of fact, they are not completely equivalent. Using L and R gives more information because those elements which are present in both of them are mandatory if the production is to be applied to a host graph, but they do not appear in the e-r characterization.[6]

Some conditions have to be imposed on matrices and vectors of nodes and edges to keep compatibility when a rule is applied, that is, the following conditions are necessary in order to avoid dangling edges once the rule is applied. From a conceptual point of view, the idea is the same as that of the dangling condition[7] in DPO.

1. An incoming edge cannot be added to a node that is going to be deleted:

$$\left\| r^E \odot e^N \right\|_1 = 0. \tag{20}$$

Similarly, for outgoing edges, the condition is:

$$\left\| \left(r^E\right)^t \odot e^N \right\|_1 = 0. \tag{21}$$

[5] In the rest of the paper we shall omit $\wedge$ if possible, and avoid unnecessary parenthesis bearing in mind that $\wedge$ has priority over $\vee$. Thus, for example, formula (18) shall be written $R^E = r^E \vee \overline{e^E} L^E$.

[6] These usage of elements, whose presence is demanded but are not used, is a sort of *positive application condition* in the sense of application conditions in DPO and SPO approaches.

[7] At times referred to as *gluing condition*.

2. Another forbidden situation is deleting a node with some incoming edge, if the edge is not deleted as well:

$$\left\| \overline{e^E}\, L^E \odot e^N \right\|_1 = 0. \tag{22}$$

Similarly for outgoing edges:

$$\left\| \left(\overline{e^E}\, L^E \right)^t \odot e^N \right\|_1 = 0. \tag{23}$$

3. It is not possible to add an incoming edge to a node which is neither present in the LHS nor added by the production:

$$\left\| r^E \odot \left(\overline{r^N}\, \overline{L^N} \right) \right\|_1 = 0. \tag{24}$$

Similarly, for edges starting in a given node:

$$\left\| \left(r^E \right)^t \odot \left(\overline{r^N}\, \overline{L^N} \right) \right\|_1 = 0. \tag{25}$$

4. Finally, our last conditions state that it is not possible that an edge reaches a node which does not belong to the LHS and which is not going to be added:

$$\left\| \left(\overline{e^E} L^E \right) \odot \left(\overline{r^N}\, \overline{L^N} \right) \right\|_1 = 0. \tag{26}$$

And again, for outgoing edges:

$$\left\| \left(\overline{e^E} L^E \right)^t \odot \left(\overline{r^N}\, \overline{L^N} \right) \right\|_1 = 0. \tag{27}$$

Thus we arrive naturally at the next proposition:

Proposition 3.9 *Let $p : L \to R$ be a production, if conditions (20) – (27) are fulfilled then (17) and (18) are compatible.*

Proof

□ We have to check $\left\| (M \vee M^t) \odot \overline{N} \right\|_1 = 0$, with $M = r^E \vee \overline{e^E} L^E$ and $\overline{N} = \overline{r^N} \left(e^N \vee \overline{L^N} \right)$. Applying (14) in the second equality we have

$$\begin{aligned} \left(M \vee M^t \right) \odot \overline{N} &= \left[\left(r^E \vee \overline{e^E} L^E \right) \vee \left(r^E \vee \overline{e^E} L^E \right)^t \right] \odot \left[\overline{r^N} \left(e^N \vee \overline{L^N} \right) \right] = \\ &= \left[r^E \vee \overline{e^E} L^E \vee \left(r^E \right)^t \vee \left(\overline{e^E} L^E \right)^t \right] \odot \left(e^N \vee \overline{r^N}\, \overline{L^N} \right) \end{aligned} \tag{28}$$

Conditions (20) – (27) are taken from this identity.■

Next we introduce the concept of *completion*. Until now we have assumed that when operating with matrices and vectors these had the same size, but in general matrices and vectors represent graphs with different sets of nodes or edges, although probably there will be common subsets. *Completion* modifies matrices (and vectors) to allow some specified operation. Two problems may occur: matrices may not fully coincide with respect to the nodes under consideration and, even if they are the same, they may well not be ordered as needed.

To address the first problem, matrices and vectors are enlarged, adding the missing vertices. When matrices and vectors are defined, an ordering is automatically given, but it may be that elements are not equally sorted in different matrices. If for example an AND is specified between two matrices, say $A \wedge B$, the first thing to do is to reorder elements so it makes sense to AND element by element, i.e., elements representing the same node are operated. If we are defining a grammar on a computer, the tool or environment will automatically do it, but some procedure has to be followed. For the sake of an example, the following procedure is proposed:

1. Find the set C of common elements.

2. Move elements of C upwards by rows in A and B, maintaining the order. A similar operation must be done moving corresponding elements to the left by columns.

3. Sort common elements in B to obtain the same ordering as in A.

4. Add remaining elements in A to B sorted as in A, immediately after the elements accessed in previous step.

5. Add remaining elements in B to A sorted as in B.

Addition of elements and reordering – the operations needed for completion – extend and modify productions syntactically but not from a semantical point of view.

Example. Consider the production depicted in Figure 6. The associated matrices, where rightmost columns represent nodes, are:

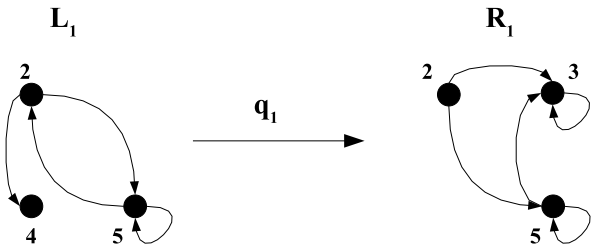

Figure 6: Example of Production

$$L_1^E = \left[\begin{array}{ccc|c} 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 4 \\ 1 & 0 & 1 & 5 \end{array}\right] \quad L_1^N = \left[\begin{array}{c|c} 1 & 2 \\ 1 & 4 \\ 1 & 5 \end{array}\right] \quad R_1^E = \left[\begin{array}{ccc|c} 0 & 1 & 1 & 2 \\ 0 & 1 & 0 & 3 \\ 0 & 1 & 1 & 5 \end{array}\right] \quad R_1^N = \left[\begin{array}{c|c} 1 & 2 \\ 1 & 3 \\ 1 & 5 \end{array}\right]$$

$$e_1^E = \left[\begin{array}{ccc|c} 0 & 1 & 0 & 2 \\ 0 & 0 & 0 & 4 \\ 1 & 0 & 0 & 5 \end{array}\right] \quad e_1^N = \left[\begin{array}{c|c} 0 & 2 \\ 1 & 4 \\ 0 & 5 \end{array}\right] \quad r_1^E = \left[\begin{array}{ccc|c} 0 & 1 & 0 & 2 \\ 0 & 1 & 0 & 3 \\ 0 & 1 & 0 & 5 \end{array}\right] \quad r_1^N = \left[\begin{array}{c|c} 0 & 2 \\ 1 & 3 \\ 0 & 5 \end{array}\right]$$

For example, if we needed to perform the operation $\overline{e_1^E}\, r_1^E$, then both matrices need to be completed. If we follow the steps described above, we obtain:

$$e_1^E = \left[\begin{array}{cccc|c} 0 & 1 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 4 \\ 1 & 0 & 0 & 0 & 5 \\ 0 & 0 & 0 & 0 & 3 \end{array}\right] \quad r_1^E = \left[\begin{array}{cccc|c} 0 & 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 0 & 4 \\ 0 & 0 & 0 & 1 & 5 \\ 0 & 0 & 0 & 1 & 3 \end{array}\right] \quad L_1^N = \left[\begin{array}{c|c} 1 & 2 \\ 1 & 4 \\ 1 & 5 \\ 0 & 3 \end{array}\right] \; R_1^N = \left[\begin{array}{c|c} 1 & 2 \\ 0 & 4 \\ 1 & 5 \\ 1 & 3 \end{array}\right]$$

where, besides the erasing and addition matrices, the completion of the nodes vectors for both left and right hand sides are displayed. Now we shall see that $r_1^N \vee \overline{e_1^N}\, L_1^N$ and $r_1^E \vee \overline{e_1^E}\, L_1^E$ are compatible, i.e., R_1^E and R_1^N define a simple digraph. Proposition (3.4) and equation (3) are used, so we need to compute (28) and, as

$$r_1^E \vee \overline{e_1^E} L_1^E = \left[\begin{array}{cccc|c} 0 & 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 0 & 4 \\ 0 & 0 & 1 & 1 & 5 \\ 0 & 0 & 0 & 1 & 3 \end{array}\right], \qquad \overline{r_1^N}\left(e_1^N \vee \overline{L_1^N}\right) = \left[\begin{array}{c|c} 0 & 2 \\ 1 & 4 \\ 0 & 5 \\ 0 & 3 \end{array}\right],$$

substituting we finally arrive at

$$(28) = \left(\left[\begin{array}{cccc|c} 0 & 0 & 1 & 1 & 2 \\ 0 & 0 & 0 & 0 & 4 \\ 0 & 0 & 1 & 1 & 5 \\ 0 & 0 & 0 & 1 & 3 \end{array}\right] \vee \left[\begin{array}{cccc|c} 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 4 \\ 1 & 0 & 1 & 0 & 5 \\ 1 & 0 & 1 & 1 & 3 \end{array}\right]\right) \odot \left[\begin{array}{c|c} 0 & 2 \\ 1 & 4 \\ 0 & 5 \\ 0 & 3 \end{array}\right] = \left[\begin{array}{c|c} 0 & 2 \\ 0 & 4 \\ 0 & 5 \\ 0 & 3 \end{array}\right]$$

as desired.

It should be recalled that the numbering used so far in graphs for nodes – and edges – is merely a means to distinguish one among the rest and does not represent types of elements (we are not dealing with *typed graphs* yet). In particular it is not possible to have two nodes with the same "name" and thus completion does not decide on any kind of matching between elements when more than one production is considered, id est, elements with the same name are in fact the same element.

Regarding graphs as typed graphs would only complicate matters in the sense that inside one graph, for example the left hand side of a production, several nodes can bear the same label. Completion of two matrices would not be two matrices anymore, but the set of matrices in which all possible combinations would be considered.[8]

This paper introduces concepts which apply to concrete productions and hence, if several nodes can play the same role, it must be determined beforehand "who is who", for example as commented above by considering all possible combinations. In this case, every node is clearly identified and distinguishable from the rest, independent of the type or whatever.

Finally, it should be noted that up to this point only the production itself has been taken into account. Matching is not considered as part of the grammar rules (graph transformation system), in contrast with DPO or SPO approaches.

4 Concatenation and Coherence

Once we are able to characterize a single production, we are interested in studying finite sets of them. Two main operations, *composition* and *concatenation*, which are in fact closely related, are introduced in this section and the next one, along with notions that make it possible to speak of "potential definability": coherence and compatibility.

Definition 4.1 (Concatenation) *Given a set of productions $\{p_1, \ldots, p_n\}$, the notation $s_n = p_n; p_{n-1}; \ldots; p_1$ defines a sequence of productions establishing an order in their application, starting with p_1 and ending with p_n.*

In the literature of graph transformation, the concatenation operator is defined back to front, this is, in the sequence $p_2; p_1$, production p_2 should be applied first and p_1 right afterwards [3]. The ordering already introduced is preferred because it follows the mathematical way in which composition is defined and represented.

It is worth stressing that there exists a total order of productions in concatenation, one production being applied after the previous has finished, and thus intermediate states are generated. These "intermediate states" are indeed the difference between concatenation and composition of productions. The study of concatenation is related to the *interleaving* approach to concurrency, while composition is related to the *explicit parallelism* approach.

A production is *moved forward*, *moved to the front* or *advanced* if it is shifted one or more positions to the right inside a sequence of productions, either in a composition or a concatenation. On the contrary, *move backwards* or *delay* means shifting the production to the left, which implies delaying its application.

Definition 4.2 (Coherence) *Given a finite set of productions $\{p_1, \ldots, p_n\}$, the sequence $s_n = p_n; p_{n-1}; \ldots; p_1$ is called coherent if actions of any production do not prevent the application of the productions that follow it, taking into account the effects of intermediate productions.*

Coherence is a concept that deals with potential applicability to a host graph of a sequence s_n of productions. It does not guarantee that the application of s_n and a coherent reordering of s_n, say $\sigma(s_n)$ leads to the same result. This latter case is a sort of generalization[9]of sequential independence applied to sequences, which will be studied in section 6.

Example. We extend the previous example with two more productions. Recall that our first production q_1 deletes edge $(5, 2)$, which starts in vertex 5 and ends in vertex 2. As depicted in Figure 7, production q_2 adds this edge and q_3 uses it. If only this vertex was to be considered, then $s_3 = q_3; q_2; q_1$ would be coherent.

Now we study the conditions that have to be satisfied by the matrices associated with a coherent sequence of productions. Instead of stating a result concerning conditions on coherence and proving it immediately afterwards, we begin by discussing the case of two productions in full detail, we continue with three and we finally set a theorem – theorem 4.5 – for a finite number of them.

Let's consider the concatenation $s_1 = p_2; p_1$. In order to decide whether these two productions do not exclude each other, we impose three conditions on edges:

[8]Assuming it is not possible to know a priori which elements are related one with another, all combinations do not need to be considered. For example, if we are studying coherence of a concatenation (refer to section 4) it may be possible that conditions imposed by previous productions discard some arrangements.

[9]in the sense that, a priori, we are considering any kind of permutation.

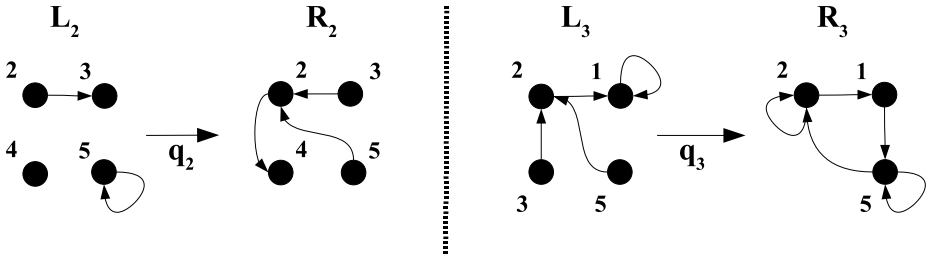

Figure 7: Some More Productions.

1. The first production – p_1 – does not delete any edge used by p_2:

$$e_1^E L_2^E = 0 \tag{29}$$

2. p_2 does not add any edge used, but not deleted, by p_1:

$$r_2^E L_1^E \overline{e_1^E} = 0 \tag{30}$$

3. No common edges are added by both productions:

$$r_1^E r_2^E = 0 \tag{31}$$

The first condition is needed because if p_1 deletes an edge used by p_2, then p_2 would not be applicable. The last two conditions are mandatory in order to obtain a simple digraph (with at most one edge between two nodes). Conditions (30) and (31) are equivalent to $r_2^E R_1^E = 0$ because, as both are equal to zero, we can do

$$0 = r_2^E L_1^E \overline{e_1^E} \vee r_2^E r_1^E = r_2^E \left(r_1^E \vee \overline{e_1^E} L_1^E \right) = r_2^E R_1^E$$

which may be read "p_2 does not add any edge that comes out from p_1's application". All conditions can be synthesized in the following identity:

$$r_2^E R_1^E \vee e_1^E L_2^E = 0 \tag{32}$$

Our immediate target is to obtain a closed formula to represent these conditions for the case with n productions. Applying (13) and (14), equation (32) can be transformed to get

$$R_1^E \overline{e_2^E} r_2^E \vee L_2^E e_1^E \, \overline{r_1^E} = 0 \tag{33}$$

A similar reasoning gives the corresponding formula for nodes:

$$R_1^N \overline{e_2^N} r_2^N \vee L_2^N e_1^N \, \overline{r_1^N} = 0 \tag{34}$$

Next we introduce a graphical notation for boolean equations: a vertical arrow means $\wedge$, while a fork stands for $\vee$. We use these diagrams because formulae grow very fast with the number of nodes. As an example, the representation of equations (33) and (34) is shown in Figure 8.

Lemma 4.3 *Let $s_2 = p_2; p_1$. If equations (33) and (34) hold, then s_2 is coherent.*

Proof
□ Only edges are considered because a symmetrical reasoning sets the result for nodes. Call D the action of deleting an edge, A its addition and U its use, i.e., the edge appears in both LHS and RHS. The following table comprises all nine possibilities for two productions

$D_2; D_1$	(29)	$D_2; U_1$	$\surd$	$D_2; A_1$	$\surd$
$U_2; D_1$	(29)	$U_2; U_1$	$\surd$	$U_2; A_1$	$\surd$
$A_2; D_1$	$\surd$	$A_2; U_1$	(30)	$A_2; A_1$	(31)

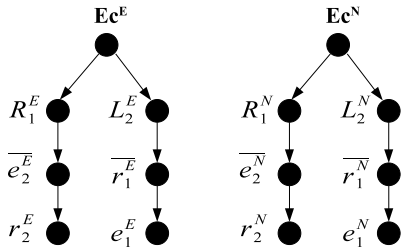

Figure 8: Coherence for Two Productions

A tick means that the action is allowed, while a number refers to the condition that prohibits the action.■

Now we proceed with three productions. We must check that p_2 does not disturb p_3 and that p_1 does not prevent the application of p_2. Notice that both of them are covered in our previous explanation (in the two productions case), and thus we just need to ensure that p_1 does not exclude p_3, taking into account that p_2 is in the middle.

1. p_1 does not delete any edge used by p_3 and not added by p_2:

$$e_1^E L_3^E \overline{r_2^E} = 0 \tag{35}$$

2. p_3 does not add any edge stemming from p_1 and not deleted by p_2:

$$r_3^E R_1^E \overline{e_2^E} = 0 \tag{36}$$

Again, the last condition is needed in order to obtain a simple digraph. The full condition for s_3 is given by the equation:

$$L_2^E \overline{e_1^E} \vee L_3^E \left(e_1^E \, \overline{r_2^E} \vee e_2^E \right) \vee R_1^E \left(\overline{e_2^E} r_3^E \vee r_2^E \right) \vee R_2^E r_3^E = 0 \tag{37}$$

Proceeding as before, identity (37) is completed:

$$\begin{aligned} L_2^E e_1^E \, \overline{r_1^E} \quad &\vee \quad L_3^E \overline{r_2^E} \left(e_1^E \, \overline{r_1^E} \vee e_2^E \right) \vee \\ &\vee \quad R_1^E \overline{e_2^E} \left(r_2^E \vee \overline{e_3^E} r_3^E \right) \vee R_2^E \overline{e_3^E} r_3^E = 0 \end{aligned} \tag{38}$$

Its representation is shown in Figure 9 for both nodes and edges.

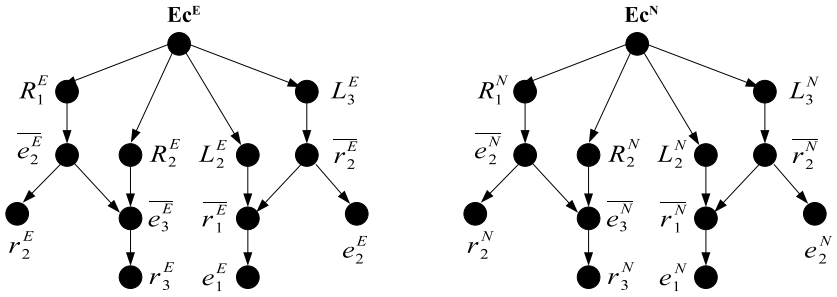

Figure 9: Coherence. Three Productions.

Remark. Lemma 4.3 can be extended slightly to include three productions in an obvious way, but we shall not discuss this further because the generalization to cover n productions is theorem 4.5.

Example. Recall productions q_1, q_2 and q_3 introduced in Figures 6 and 7. Sequences $q_3; q_2; q_1$ and $q_1; q_3; q_2$ are coherent, while $q_3; q_1; q_2$ is not. The latter is due to the fact that edge $(5, 5)$ is deleted (D) by q_2, used (U) by q_1 and added (A) by q_3, being two pairs of forbidden actions. For the former sequences, we have to check all actions performed on all edges and nodes by the productions in the order specified by the concatenation, verifying that they do not exclude each other.

Before generalizing to n productions, we define two useful operators.

Definition 4.4 *Let $F(x,y)$ and $G(x,y)$ be two boolean functions dependant on parameters $x,y \in I$ in some index set I. Operators delta $\triangle$ and nabla $\triangledown$ are defined through the equations:*

$$\triangle_{t_0}^{t_1}\left(F(x,y)\right) = \bigvee_{y=t_0}^{t_1}\left(\bigwedge_{x=y}^{t_1}\left(F(x,y)\right)\right) \tag{39}$$

$$\triangledown_{t_0}^{t_1}\left(G(x,y)\right) = \bigvee_{y=t_0}^{t_1}\left(\bigwedge_{x=t_0}^{y}\left(G(x,y)\right)\right) \tag{40}$$

In order to justify theorem 4.5, Figure 10 includes the edges digraphs for $s_4 = p_4; p_3; p_2; p_1$ and $s_5 = p_5; p_4; p_3; p_2; p_1$.

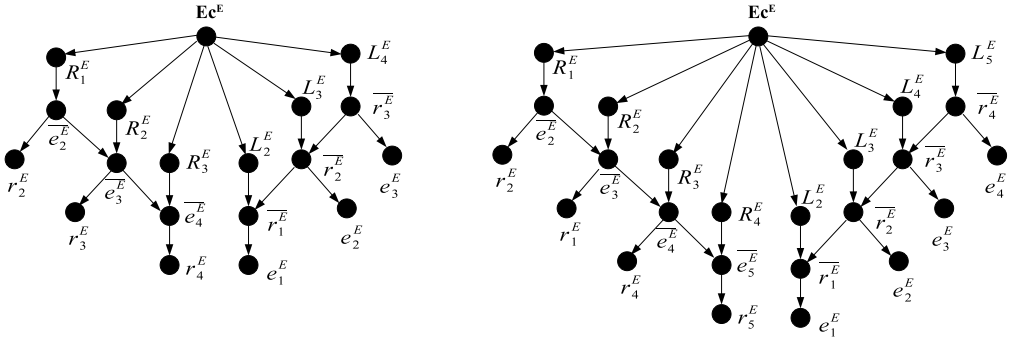

Figure 10: Coherence. Four and Five Productions.

Now we are ready to characterize coherent sequences of arbitrary finite size.

Theorem 4.5 *The concatenation $s_n = p_n; p_{n-1}; \ldots; p_2; p_1$ is coherent if for edges and nodes we have*

$$\bigvee_{i=1}^{n}\left(R_i^E \triangledown_{i+1}^{n}\left(\overline{e_x^E}\, r_y^E\right) \vee L_i^E \triangle_1^{i-1}\left(e_y^E\, \overline{r_x^E}\right)\right) = 0 \tag{41}$$

$$\bigvee_{i=1}^{n}\left(R_i^N \triangledown_{i+1}^{n}\left(\overline{e_x^N}\, r_y^N\right) \vee L_i^N \triangle_1^{i-1}\left(e_y^N\, \overline{r_x^N}\right)\right) = 0 \tag{42}$$

Proof

□ An induction argument similar to what we have done for s_2 proves the result.■

Example. We are going to verify that $s_1 = q_1; q_3; q_2$ is coherent (only for edges), where q_i are the productions introduced in previous examples. We start expanding formula (41) for $n = 3$:

$$\bigvee_{i=1}^{3}\left(R_i^E \triangledown_{i+1}^{3}\left(\overline{e_x^E}\, r_y^E\right) \vee L_i^E \triangle_1^{i-1}\left(e_y^E\, \overline{r_x^E}\right)\right) =$$

$$= R_1^E\left(r_2^E \vee \overline{e_2^E}\, r_3^E\right) \vee R_2^E r_3^E \vee L_2^E e_1^E \vee L_3^E\left(e_1^E\, \overline{r_2^E} \vee e_2^E\right)$$

which should be zero. Note that this equation applies to concatenation $s = q_3; q_2; q_1$ and thus we have to map $(1,2,3) \to (2,3,1)$ to obtain

$$\underbrace{R_2^E\left(r_3^E \vee \overline{e_3^E}\, r_1^E\right)}_{(*)} \vee \underbrace{R_3^E r_1^E \vee L_3^E e_2^E}_{(**)} \vee \underbrace{L_1^E\left(e_2^E\, \overline{r_3^E} \vee e_3^E\right)}_{(***)} = 0 \tag{43}$$

Before checking whether these expressions are zero or not, we have to complete the involved matrices. All calculations have been divided into three steps and, as they are operated with "or", if one fails to be zero, the

result should not be null.

$$(*) = \left[\begin{array}{ccccc|c} 0&0&0&1&0&2\\ 1&0&0&0&0&3\\ 1&0&0&0&0&5\\ 0&0&0&0&0&1\\ 0&0&0&0&0&4 \end{array}\right] \left(\left[\begin{array}{ccccc|c} 1&0&0&0&0&2\\ 0&0&0&0&0&3\\ 0&0&1&0&0&5\\ 0&0&1&0&0&1\\ 0&0&0&0&0&4 \end{array}\right] \vee \left[\begin{array}{ccccc|c} 1&1&1&1&1&2\\ 0&1&1&1&1&3\\ 1&1&1&1&1&5\\ 1&1&1&0&1&1\\ 1&1&1&1&1&4 \end{array}\right] \left[\begin{array}{ccccc|c} 0&1&0&0&0&2\\ 0&1&0&0&0&3\\ 0&1&0&0&0&5\\ 0&0&0&0&0&1\\ 0&0&0&0&0&4 \end{array}\right] \right) = 0$$

$$(**) = \left[\begin{array}{ccccc|c} 1&0&0&1&0&2\\ 0&0&0&0&0&3\\ 1&0&1&0&0&5\\ 0&0&1&0&0&1\\ 0&0&0&0&0&4 \end{array}\right] \left[\begin{array}{ccccc|c} 0&1&0&0&0&2\\ 0&1&0&0&0&3\\ 0&1&0&0&0&5\\ 0&0&0&0&0&1\\ 0&0&0&0&0&4 \end{array}\right] \vee \left[\begin{array}{ccccc|c} 0&0&0&1&0&2\\ 1&0&0&0&0&3\\ 1&0&0&0&0&5\\ 0&0&0&1&0&1\\ 0&0&0&0&0&4 \end{array}\right] \left[\begin{array}{ccccc|c} 0&1&0&0&0&2\\ 0&0&0&0&0&3\\ 0&0&1&0&0&5\\ 0&0&0&0&0&1\\ 0&0&0&0&0&4 \end{array}\right] = 0$$

$$(***) = \left[\begin{array}{ccccc|c} 0&0&0&0&1&2\\ 0&0&0&0&0&3\\ 1&0&1&0&0&5\\ 0&0&0&0&0&1\\ 0&0&0&0&0&4 \end{array}\right] \left(\left[\begin{array}{ccccc|c} 0&1&0&0&0&2\\ 0&0&0&0&0&3\\ 0&0&1&0&0&5\\ 0&0&0&0&0&1\\ 0&0&0&0&0&4 \end{array}\right] \left[\begin{array}{ccccc|c} 0&1&1&1&1&2\\ 1&1&1&1&1&3\\ 1&1&0&1&1&5\\ 1&1&0&1&1&1\\ 1&1&1&1&1&4 \end{array}\right] \vee \left[\begin{array}{ccccc|c} 0&0&0&0&0&2\\ 1&0&0&0&0&3\\ 0&0&0&0&0&5\\ 0&0&0&1&0&1\\ 0&0&0&0&0&4 \end{array}\right] \right) = 0$$

where, as usual, a matrix filled up with zeros is represented by 0.

Now consider sequence $s_2 = q_2; q_3; q_1$. The condition for its coherence is

$$\underbrace{R_1^E \left(r_3^E \vee \overline{e_3^E}\, r_2^E \right)}_{(*)} \vee \underbrace{R_3^E r_2^E \vee L_3^E e_1^E}_{(**)} \vee \underbrace{L_2^E \left(e_1^E\, \overline{r_3^E} \vee e_3^E \right)}_{(***)} = 0 \tag{44}$$

If we focus just on the first term in equation (44)

$$(*) = \left[\begin{array}{ccccc|c} 0&1&1&0&0&2\\ 0&1&0&0&0&3\\ 0&1&1&0&0&5\\ 0&0&0&0&0&1\\ 0&0&0&0&0&4 \end{array}\right] \left(\left[\begin{array}{ccccc|c} 1&0&0&0&0&2\\ 0&0&0&0&0&3\\ 0&0&1&0&0&5\\ 0&0&1&0&0&1\\ 0&0&0&0&0&4 \end{array}\right] \vee \left[\begin{array}{ccccc|c} 1&1&1&1&1&2\\ 0&1&1&1&1&3\\ 1&1&1&1&1&5\\ 1&1&1&0&1&1\\ 1&1&1&1&1&4 \end{array}\right] \left[\begin{array}{ccccc|c} 0&0&0&0&1&2\\ 1&0&0&0&0&3\\ 1&0&0&0&0&5\\ 0&0&0&0&0&1\\ 0&0&0&0&0&4 \end{array}\right] \right)$$

we obtain a matrix filled up with zeros except in position (3,3) which corresponds to an edge that starts and ends in node 5. In this way, we realize that the sequence is not coherent, and in addition we obtain information on which node or edge may present problems when applied to an actual host graph.

Note that a sequence not being coherent does not necessarily mean that the grammar is not well defined, but that we have to be especially careful when applying it to a host graph because it is mandatory for the match to identify all problematic parts in different places. Somehow, matches should spread doubtful subsets of the concatenation across the host graph.

This information could be used when actually finding the match; a possible strategy, if parallel matching for different productions is required, is to start with those elements which may present a problem. The same applies to *G-congruence*, to be studied in section 6.

5 Compatibility, Composition and Minimal Initial Digraph

In this section we deal with compatibility and introduce composition of a sequence of productions and its associated minimal initial digraph.

Example. We start considering productions u and v defined in Figure 11. It is easy to see that $v; u$ is coherent, but it seems a little more difficult to define their composition $v \circ u$, as if they were applied to the same nodes we would get a dangling edge. Although coherence itself does not guarantee applicability of a sequence, we shall see that compatibility is sufficient (generalized to consider concatenations, not only single productions as in definition 3.2).

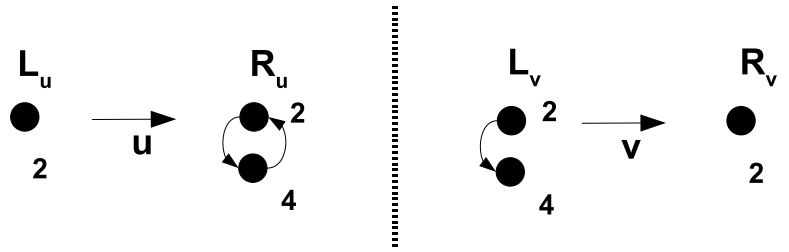

Figure 11: Non-Compatible Productions.

Defining compatibility for a sequence of productions is not straightforward, nor its composition starting with a coherent concatenation, and it will be necessary to bring in the concept of *minimal initial digraph*.

The example shows a problem that led us to consider not only productions, but also the context in which they are to be applied. This is something we try to avoid as we want to study productions alone without host graphs, matches, etcetera, so please allow us to explain.

Two possibilities are found in the literature in order to define a match, depending whether DPO or SPO is followed [8]. In the latter, deletion prevails, and thus in the present example production v would be "enlarged" in order to delete edge $(4, 2)$. Our approximation to the match of a production will follow this basically but be slightly different, considering it as an operator that acts on a space whose elements are productions.

Compatibility is determined by the result of applying a production to an initial graph and checking nodes and edges of the result. If we try to define compatibility for a concatenation or composition, we have to decide which is the initial graph, but as earlier mentioned before we prefer not to begin our analysis of matches yet. This situation might be overcome if we are able to define a minimal and unique host graph which permits the application of a given concatenation or composition of productions. We call it a *minimal initial digraph*. Note that we were able to set when a single production is compatible in definition 3.2 because it is clear (so obvious that we did not mention it) which one is the minimal initial digraph: its left hand side.

One graph is known which permits the application of $p_n; \ldots; p_1$ – supposing it is coherent –, namely $\bigvee_{i=1}^{n} L_i$, in the sense that it has enough elements to carry out all operations. We shall only consider coherent productions, so it is not necessary to repeat nodes in this minimal host graph.

Theorem 5.1 *Given a coherent concatenation of productions $s_n = p_n; \ldots; p_1$, the minimal initial digraph is defined by the equation*

$$M_n = \bigtriangledown_1^n \left(\overline{r_x} L_y\right) \tag{45}$$

Superscripts have been omitted in order to read the formulae more easily. In figure 12, formula (45) and its negation (55) are expanded for four productions.

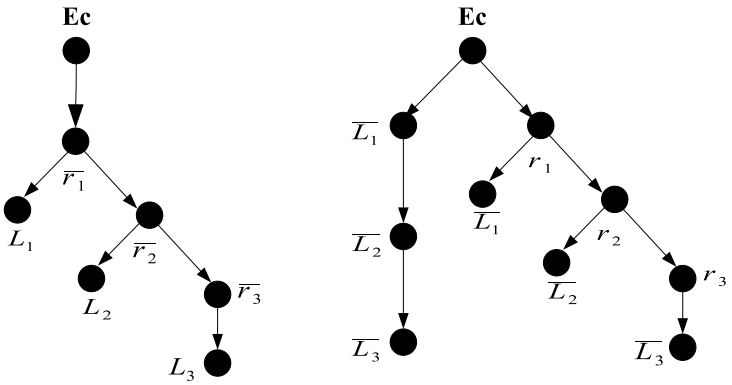

Figure 12: Minimal Initial Digraph (45) and Its Negation (55) for Four Productions.

Recall our previous comments about completion, where it was pointed out that if several nodes are of the same "type", completion of the concatenation will result in the set of all (or part of all) possible combinations

such that every node[10]can be uniquely identified, i.e., if there are nodes labeled with a 1, it is decided which ones continue to be 1 (are the same), which are called $1'$ (and thus are the same), and so on. It is not possible, once the process of completion has finished, to have two nodes with the same label inside the same production because from an operational point of view it is mandatory to know all relations between nodes.[11]

Proof of Theorem 5.1

□To properly demonstrate this theorem we have to prove that M_n has enough edges and nodes in order to apply all productions in the specified order, that it is minimal and finally that it is unique (up to isomorphisms). We shall proceed by induction on the number of productions.

By hypothesis we know that the concatenation is coherent and thus the application of one production does not exclude the ones coming after it. In order to see that there are sufficient nodes and edges, it is enough to check that $s_n\left(\bigvee_{i=1}^{n} L_i\right) = s_n\left(M_n\right)$, as the most complete digraph to start with is $\bigvee_{i=1}^{n} L_i$, which has enough elements due to coherence.

If we had a sequence consisting of only one production $s_1 = p_1$, then it should be obvious that the minimal digraph needed to apply the concatenation is L_1.

In the case of a sequence of two productions, say $s_2 = p_2; p_1$, what p_1 uses is again needed. All edges that p_2 uses, except those added by p_1, are also mandatory. Note that the elements produced by p_1 are not considered in the minimal initial digraph. If an element is used and not erased by p_1, then it should not be taken into account either:

$$L_1 \vee L_2\overline{r_1}\,\overline{(\overline{e_1}L_1)} = L_1 \vee L_2\overline{r_1}\left(e_1 \vee \overline{L_1}\right) = L_1 \vee L_2\overline{R_1} \tag{46}$$

This formula can be paraphrased as "elements used by p_1 plus those needed by p_2's left hand side, except the ones resulting from p_1 application". It provides enough elements to s_2:

$$\begin{aligned} p_2; p_1\left(L_1 \vee L_2\overline{R_1}\right) &= r_2 \vee \overline{e_2}\left(r_1 \vee \overline{e_1}\left(L_1 \vee L_2\overline{R_1}\right)\right) = \\ &= r_2 \vee \overline{e_2}\left(R_1 \vee r_1\overline{R_1}L_2 \vee \overline{e_1}\,\overline{R_1}L_2\right) = \\ &= r_2 \vee \overline{e_2}\left(r_1 \vee \overline{e_1}\left(L_1 \vee L_2\right)\right) = p_2; p_1\left(L_1 \vee L_2\right) \end{aligned}$$

Let's move one step forward with the sequence of three productions $s_3 = p_3; p_2; p_1$. The minimal digraph needs what s_2 needed, but even more so. We have to add what the third production uses, except what comes out from p_1 and is not deleted by p_2, and finally remove what comes out from p_2.

$$L_1 \vee L_2\overline{R_1} \vee L_3\overline{(\overline{e_2}\,R_1)}\,\overline{R_2} = L_1 \vee L_2\overline{R_1} \vee L_3\overline{R_2}\left(e_2 \vee \overline{R_1}\right) \tag{47}$$

Similarly to what has already been done for s_2, we check that the minimal initial digraph has enough elements such that it is possible to apply p_1, p_2 and p_3.

$$\begin{aligned} &p_3; p_2; p_1\left(M_3\right) = r_3 \vee \overline{e_3}\left(r_2 \vee \overline{e_2}\left(r_1 \vee \overline{e_1}\left(L_1 \vee L_2\overline{R_1} \vee L_3\overline{R_2}\left(e_2 \vee \overline{R_1}\right)\right)\right)\right) = \\ &= r_3 \vee \overline{e_3}\left(r_2 \vee \overline{e_2}\left(\overline{e_1}L_2 \vee \overline{e_1}e_2L_3\overline{R_2} \vee \underbrace{R_1 \vee L_3\overline{e_1}\,\overline{R_1}R_2}_{R_1 \vee L_3\overline{e_1}R_2}\right)\right) = \\ &= r_3 \vee \overline{e_3}\left(\overline{e_2}r_1 \vee \overline{e_2}\,\overline{e_1}L_1 \vee \overline{e_2}\,\overline{e_1}L_2 \vee \underbrace{r_2 \vee L_3\overline{e_1}\,\overline{e_2}\,\overline{r_2}\,\overline{L_2}}_{r_2 \vee L_3\overline{e_1}\,\overline{e_2}\,\overline{L_2}}\right) = \\ &= r_3 \vee \overline{e_3}\left(r_2 \vee \overline{e_2}\left(r_1 \vee \overline{e_1}\left(L_1 \vee L_2 \vee L_3\right)\right)\right) = \\ &= p_3; p_2; p_1\left(L_1 \vee L_2 \vee L_3\right) \end{aligned}$$

Reasoning for the case of four productions, the condition derived is $L_1 \vee L_2\overline{R_1} \vee L_3\overline{(\overline{e_2}\,R_1)}\,\overline{R_2} \vee L_4\overline{(\overline{e_3}\,\overline{e_2}R_1)}\,\overline{(\overline{e_3}R_2)}\,\overline{R_3}$. Minimality is inferred by construction, because for each L_i all elements added and not deleted by any production p_j, $j < i$, are removed. If any other element is erased from the minimal initial digraph, some production in s_n would no longer be applicable.[12]

Now we want to express previous formulae using operators ∇ and Δ. The expression

$$L_1^E \vee \bigvee_{i=2}^{n}\left[L_i^E \triangle_1^{i-1}\left(\overline{R_x^E}\,e_y^E\right)\right] \tag{48}$$

[10]and by extension every edge.

[11]It is compulsory to clearly specify matrices e and r.

[12]Matches for this approach will be introduced in a future contribution, together with the notion of *initial digraph set* which can be used to properly prove minimality.

is close but we would be adding terms that include $\overline{R_1^E}e_1^E$, and clearly $\overline{R_1^E}e_1^E \neq \overline{R_1^E}$, which is what we have in the minimal initial digraph.[13]Thus, considering the fact that $\overline{a}b \vee \overline{a}\,\overline{b} = \overline{a}$, we eliminate them by performing "or" operations:

$$\overline{e_1^E} \bigtriangledown_1^{n-1} \left(\overline{R_x^E} L_{y+1}\right) \tag{49}$$

Please refer to figure 13, where on the right side expression (50) is represented while on the left the same equation, but simplified, is depicted for $n = 4$.

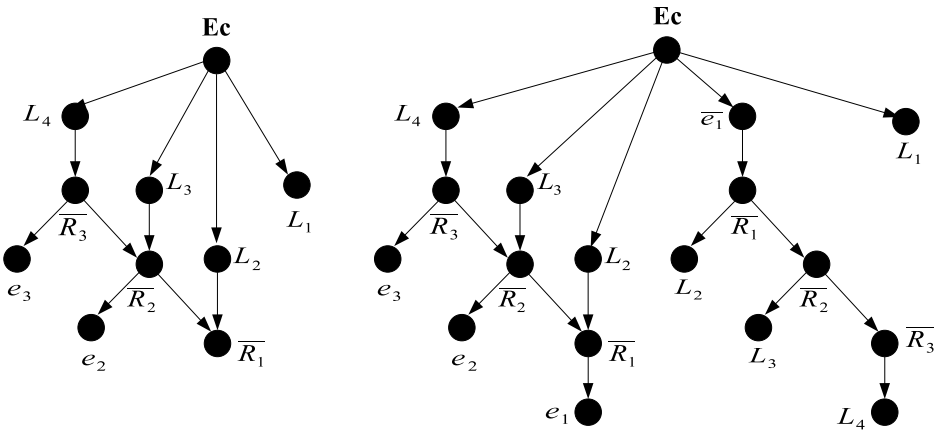

Figure 13: Minimal Initial Digraph (Intermediate Expression). Four Productions.

Thus we have a formula for the minimal initial digraph which is slightly different from that in the theorem:

$$M_n = L_1 \vee \overline{e_1} \bigtriangledown_1^{n-1} \left(\overline{R_x} L_{y+1}\right) \vee \bigvee_{i=2}^{n} \left[L_i \bigtriangleup_1^{i-1} \left(\overline{R_x}\, e_y\right)\right] \tag{50}$$

Our next step is to show that previous identity is equivalent to

$$M_n = L_1 \vee \overline{e_1} \bigtriangledown_1^{n-1} \left(\overline{r_x} L_{y+1}\right) \vee \bigvee_{i=2}^{n} \left[L_i \bigtriangleup_1^{i-1} \left(\overline{r_x}\, e_y\right)\right] \tag{51}$$

illustrating the way to proceed for $n = 3$. To this end, equation (16) is used as well as the fact that $a \vee \overline{a}b = a \vee b$.

$L_1 \vee L_2\overline{R_1} \vee L_3\overline{R_2}\left(e_2 \vee \overline{R_1}\right) =$

$= L_1 \vee L_2\overline{r_1}\left(e_1 \vee \overline{L_1}\right) \vee \left(L_3\overline{r_2}e_2 \vee L_3\overline{r_2}\,\overline{L_2}\right)\left(e_2 \vee \overline{r_1}e_1\overline{r_1}\,\overline{L_1}\right) =$

$= L_1 \vee L_2\overline{r_1}\,\overline{L_1} \vee L_2e_1 \vee L_3e_2 \vee \underbrace{L_3e_2e_1 \vee L_3e_2\overline{r_1}\,\overline{L_1} \vee L_3e_2\overline{L_2}}_{\text{disappears due to } L_3e_2} \vee$

$\vee\, L_3\overline{r_2}\,\overline{L_2}\overline{r_1}\,\overline{L_1} \vee L_3\overline{r_2}\,\overline{L_2}e_1 =$

$= L_1 \vee L_2\left(\overline{r_1} \vee e_1\right) \vee L_3\overline{L_2}\overline{r_2}\,\overline{r_1} \vee L_3e_2 \vee L_3\overline{L_2}\overline{r_2}e_1 =$

$= L_1 \vee L_2\overline{r_1} \vee L_3\overline{r_2}\left(e_2 \vee \overline{r_1}\right)$

But (51) is what we have in the theorem, because as the concatenation is coherent, the third term in (51) is zero:

$$\bigvee_{i=2}^{n} \left[L_i \bigtriangleup_1^{i-1} \left(\overline{r_x}\, e_y\right)\right] = 0 \tag{52}$$

Finally, as $L_1 = L_1 \vee e_1$, it is possible to omit $\overline{e_1}$ and obtain (45), recalling that $\overline{r}L = L$.■

Example. Let $s_2 = u; v$ and $s_2' = v; u$ (see Figure 11 for the definition of the productions). Minimal initial digraphs for these productions are represented in Figure 14. In the way we have introduced the concept of minimal initial digraph, M_2 (which allows the application of $v; u$) cannot be considered as such. In the same figure the minimal initial digraphs for productions $q_3; q_2; q_1$ and $q_1; q_3; q_2$ are also represented.

[13]Not in (45), but in expressions derived up to now for minimal initial digraph: (46) and (47).

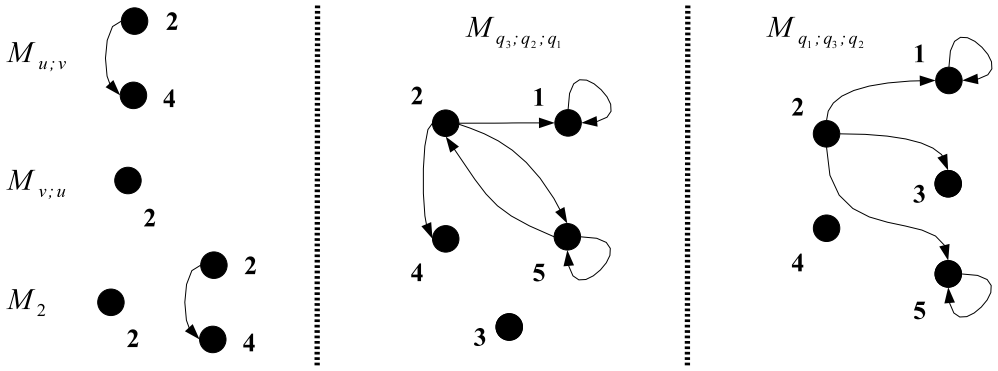

Figure 14: Minimal Initial Digraph. Examples and Counterexample.

We shall explicitly compute the minimal initial digraph for the concatenation $q_3; q_2; q_1$. In this example, and in order to illustrate some of the steps used to prove the previous theorem, formula (50) is used. Once simplified it lays the equation:

$$\underbrace{L_1^E \vee L_2^E \overline{R_1^E}}_{(*)} \vee \underbrace{L_3^E \overline{R_2^E} \left(e_2^E \vee \overline{R_1^E}\right)}_{(**)}$$

$$(*) = \left[\begin{array}{ccccc|c} 0&0&1&0&1&2\\ 0&0&0&0&0&3\\ 1&0&1&0&0&5\\ 0&0&0&0&0&1\\ 0&0&0&0&0&4 \end{array}\right] \vee \left[\begin{array}{ccccc|c} 0&1&0&0&0&2\\ 0&0&0&0&0&3\\ 0&0&1&0&0&5\\ 0&0&0&0&0&1\\ 0&0&0&0&0&4 \end{array}\right] \left[\begin{array}{ccccc|c} 1&0&0&1&1&2\\ 1&1&0&1&1&3\\ 1&0&0&1&1&5\\ 1&1&1&1&1&1\\ 1&1&1&1&1&4 \end{array}\right] = \left[\begin{array}{ccccc|c} 0&0&1&0&1&2\\ 0&0&0&0&0&3\\ 1&0&1&0&0&5\\ 0&0&0&0&0&1\\ 0&0&0&0&0&4 \end{array}\right]$$

$$(**) = \left[\begin{array}{ccccc|c} 0&0&0&1&0&2\\ 1&0&0&0&0&3\\ 1&0&1&0&0&5\\ 0&0&0&1&0&1\\ 0&0&0&0&0&4 \end{array}\right] \left[\begin{array}{ccccc|c} 1&1&1&1&0&2\\ 0&1&1&1&1&3\\ 1&1&1&1&0&5\\ 1&1&1&1&1&1\\ 1&1&1&1&1&4 \end{array}\right] \left(\left[\begin{array}{ccccc|c} 0&1&0&0&0&2\\ 0&0&0&0&0&3\\ 0&0&1&0&0&5\\ 0&0&0&0&0&1\\ 0&0&0&0&0&4 \end{array}\right] \vee \left[\begin{array}{ccccc|c} 1&0&0&1&1&2\\ 1&1&0&1&1&3\\ 1&0&0&1&1&5\\ 1&1&1&1&1&1\\ 1&1&1&1&1&4 \end{array}\right] \right)$$

$$(*) \vee (**) = \left[\begin{array}{ccccc|c} 0&0&1&0&1&2\\ 0&0&0&0&0&3\\ 1&0&1&0&0&5\\ 0&0&0&0&0&1\\ 0&0&0&0&0&4 \end{array}\right] \vee \left[\begin{array}{ccccc|c} 0&0&0&1&0&2\\ 0&0&0&0&0&3\\ 1&0&0&0&0&5\\ 0&0&0&1&0&1\\ 0&0&0&0&0&4 \end{array}\right] = \left[\begin{array}{ccccc|c} 0&0&1&1&1&2\\ 0&0&0&0&0&3\\ 1&0&1&0&0&5\\ 0&0&0&1&0&1\\ 0&0&0&0&0&4 \end{array}\right]$$

A closed formula for the effect of the application of a coherent concatenation might be useful if we want to operate in the general case. This is where next corollary comes in.

Corollary 5.2 *Let $s_n = p_n; \ldots; p_1$ be a coherent concatenation of productions, and M_n its minimal initial digraph, as defined in (45). Then,*

$$s_n\left(M_n^E\right) = \bigwedge_{i=1}^{n} \left(\overline{e_i^E} M_n^E\right) \vee \triangle_1^n \left(\overline{e_x^E}\, r_y^E\right) \tag{53}$$

$$\overline{s_n\left(M_n^E\right)} = \bigwedge_{i=1}^{n} \left(\overline{r_i^E}\, \overline{M_n^E}\right) \vee \triangle_1^n \left(\overline{r_x^E}\, e_y^E\right) \tag{54}$$

Proof

□Theorem 45 proves that $s_n\left(M_n^E\right) = s_n\left(\bigvee_{i=1}^{n} L_i\right)$. To derive the formulae apply induction on the number of productions and (13).■

The negation of the minimal initial digraph which appears in identity (54) – seen in figure 12 – can be explicitly calculated in terms of *nabla*:

$$\overline{M_n} = \nabla_1^{n-1}\left(\overline{L_x}\, r_y\right) \vee \bigwedge_{i=1}^{n} \overline{L_i} \tag{55}$$

For the sake of curiosity, if we used formula (51) to calculate the minimal initial digraph, the representation of its negation is included in figure 15 for $n = 3$ and $n = 4$. It might be useful to find an expression using operators Δ and ∇ for these digraphs.

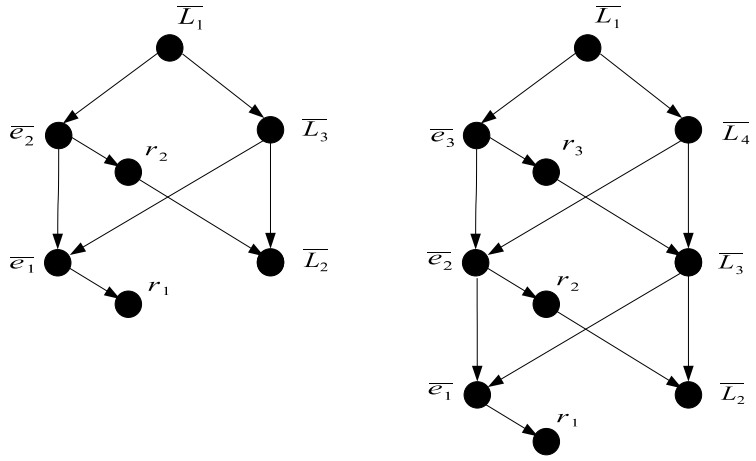

Figure 15: Negation of Minimal Initial Digraph (Intermediate Expression (51)) for Three and Four productions.

Next we are going to introduce compatibility for production sequences. When a concatenation of productions is considered, we are not only concerned with the final result but also with intermediate states – or partial results – of the sequence. Compatibility should take this into account and thus a concatenation is said to be compatible if the overall effect on its minimal initial digraph gives as result a compatible digraph starting from the first production and increasing the sequence until we get the full concatenation. We should then test compatibility for the growing sequence of concatenations $S = \{s_1, s_2, \ldots, s_n\}$ where $s_m = q_m; q_{m-1}; \ldots; q_1$, $1 \leq m \leq n$.

Definition 5.3 *Coherent concatenation s_n is said to be compatible if $\forall m \in \{1, \ldots, n\}$ the following identities are verified:*

$$\left\| \left[s_m\left(M_m^E\right) \vee \left(s_m\left(M_m^E\right)\right)^t \right] \odot \overline{s_m\left(M_m^N\right)} \right\|_1 = 0 \tag{56}$$

Coherence examines whether actions specified by a sequence of productions are feasible. It warns us if one production adds or deletes an element which it should not, as some later production might need to carry out an operation that becomes impossible. Compatibility is a more *basic* concept because it examines if the result is a digraph, that is, if the class of all digraphs is closed under the operations defined in the sequence.

So far we have presented compatibility and the minimal initial digraph and shall finish studying composition and the circumstances under which it is possible to define a composition starting with a coherent concatenation.

When we introduced the notion of production, we first defined its LHS and RHS and then we associated some matrices (e and r) with them. The situation for defining composition is similar, but this time we first observe the overall effect of the production and then decide its LHS and RHS. Assume $s_n = p_n; \ldots; p_1$ is coherent. The composition of its productions is again a production defined by the rule $c = p_n \circ p_{n-1} \circ \ldots \circ p_1$.[14]To describe its erasing matrix e and its addition matrix r, consider matrices

$$S^E = \sum_{i=1}^{n} \left(r_i^E - e_i^E\right). \tag{57}$$

$$S^N = \sum_{i=1}^{n} \left(r_i^N - e_i^N\right). \tag{58}$$

[14]The concept and notation are those commonly used in mathematics.

Due to coherence we know that elements of S^E, S^N are either $+1$, 0 or -1, so they can be split in their positive and negative parts, $S^E = r_+^E - e_-^E$, $S^N = r_+^N - e_-^N$, where all r_+ and e_- elements are either zero or one. We thus have:

Proposition 5.4 *Let $s_n = p_n; p_{n-1}; \ldots; p_1$ be a coherent and compatible concatenation of productions. Then, the composition $c = p_n \circ p_{n-1} \circ \ldots \circ p_1$ defines a production with matrices $r^E = r_+^E$, $r^N = r_+^N$ and $e^E = -e_-^E$, $e^N = -e_-^N$.*

Proof

□■

The LHS is the minimal digraph necessary to carry out all operations specified by the composition. As it is only one production, its LHS equals its erasing matrix and its right hand side is just the image.

Corollary 5.5 *With the above notation, $c(M_n) = s_n(M_n)$.*

Composition is helpful when we have a coherent concatenation and intermediate states are useless or undesired. It will be used in sequential independence and explicit parallelism.

Example. We finish this section considering sequence $s_{321} = q_3; q_2; q_1$, calculating its composition c_{321} and comparing its result with that of s_{321}. Recall that $S^E(s_{321}) = \sum_{i=1}^{3} \left(r_i^E - e_i^E\right) = r_+^E - e_-^E$.

$$\sum_{i=1}^{3} \left(r_i^E\right) = \left[\begin{array}{ccccc|c} 0&1&0&0&0&2\\ 0&1&0&0&0&3\\ 0&1&0&0&0&5\\ 0&0&0&0&0&1\\ 0&0&0&0&0&4 \end{array}\right] + \left[\begin{array}{ccccc|c} 0&0&0&0&1&2\\ 1&0&0&0&0&3\\ 1&0&0&0&0&5\\ 0&0&0&0&0&1\\ 0&0&0&0&0&4 \end{array}\right] + \left[\begin{array}{ccccc|c} 1&0&0&0&0&2\\ 0&0&0&0&0&3\\ 0&0&1&0&0&5\\ 0&0&1&0&0&1\\ 0&0&0&0&0&4 \end{array}\right] = \left[\begin{array}{ccccc|c} 1&1&0&0&1&2\\ 1&1&0&0&0&3\\ 1&1&1&0&0&5\\ 0&0&0&0&0&1\\ 0&0&0&0&0&4 \end{array}\right]$$

$$\sum_{i=1}^{3} \left(e_i^E\right) = \left[\begin{array}{ccccc|c} 0&0&0&0&1&2\\ 0&0&0&0&0&3\\ 1&0&0&0&0&5\\ 0&0&0&0&0&1\\ 0&0&0&0&0&4 \end{array}\right] + \left[\begin{array}{ccccc|c} 0&1&0&0&0&2\\ 0&0&0&0&0&3\\ 0&0&1&0&0&5\\ 0&0&0&0&0&1\\ 0&0&0&0&0&4 \end{array}\right] + \left[\begin{array}{ccccc|c} 0&0&0&0&0&2\\ 1&0&0&0&0&3\\ 0&0&0&0&0&5\\ 0&0&0&1&0&1\\ 0&0&0&0&0&4 \end{array}\right] = \left[\begin{array}{ccccc|c} 0&1&0&0&1&2\\ 1&0&0&0&0&3\\ 1&0&1&0&0&5\\ 0&0&0&1&0&1\\ 0&0&0&0&0&4 \end{array}\right]$$

$$S^E(s_{321}) = \left[\begin{array}{ccccc|c} 1&0&0&0&0&2\\ 0&1&0&0&0&3\\ 0&1&0&0&0&5\\ 0&0&0&-1&0&1\\ 0&0&0&0&0&4 \end{array}\right] = \left[\begin{array}{ccccc|c} 1&0&0&0&0&2\\ 0&1&0&0&0&3\\ 0&1&0&0&0&5\\ 0&0&0&0&0&1\\ 0&0&0&0&0&4 \end{array}\right] - \left[\begin{array}{ccccc|c} 0&0&0&0&0&2\\ 0&0&0&0&0&3\\ 0&0&0&0&0&5\\ 0&0&0&1&0&1\\ 0&0&0&0&0&4 \end{array}\right] = r_+^E - e_-^E$$

Sequence s_{321} has been chosen not only to illustrate composition, but also compatibility and the sort of problems that may arise if it is not fulfilled. In this case, q_3 deletes node 3 and edge (3,2) but does not specify anything about edges (3,3) and (3,5) – the dotted elements in figure 16.

The previous example provides us with some clues as to the way the match could be defined. The basic idea is to introduce an operator over the set of productions, so once a match (an injective map from the LHS of the production) identifies a place in the host graph where the rule may be applied, the operator modifies the rule enlarging the deletion matrix so no dangling edge may occur (it should enlarge the grammar rule to include the context of the original rule in the graph, adding all elements on both LHS and RHS). Essentially, a match should be an injective morphism plus an operator. Pre-calculated information for coherence, sequentialization, and the like, should help and hopefully reduce the amount of calculation during runtime.

This section ends noting that, in this approach, one production is a morphism between two simple digraphs and thus it may carry out just one action on each element. When the composition of a concatenation is performed we get a single production. Suppose one production specifies the deletion of an element and another its addition, the overall mathematical result of the composition should leave the element unaltered. When a match is considered, depending on the chosen approach, all dangling edges incident to those erased nodes should be removed, establishing an important difference between a sequence and its composition.

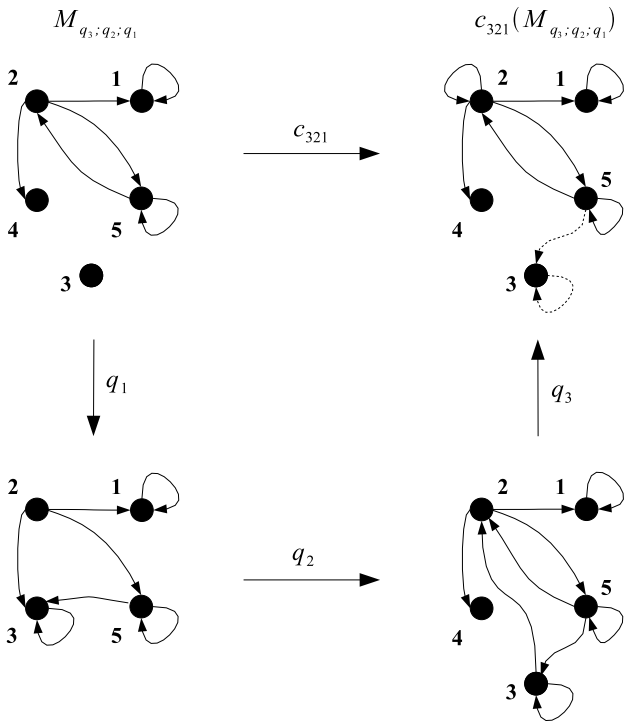

Figure 16: Composition and Concatenation of a non-Compatible Sequence.

6 Permutations and Sequential Independence

Once we know what a sequence of productions is and when it is potentially safe to define them, we are ready to deal with position interchange inside a concatenation. Let $s_3 = p_3; p_2; p_1$ be a coherent concatenation made up of three productions and suppose we want to move p_3 forward one position to obtain $\sigma(s_3) = p_2; p_3; p_1$. This may be thought of as a permutation σ acting on s_3 represented via a matrix or a vector[15]

$$\sigma = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \end{pmatrix} = [1\,2]$$

to be read as *element on first position goes to second position and element on second position goes to first position, while element on third position remains unaffected.* The idea behind sequential independence is that changes of order in the position of productions inside a sequence do not alter the result of their application.

Definition 6.1 *Let $s_n = p_n; \ldots; p_1$ be a concatenation and let σ be a permutation. Then, s_n and $\sigma(s_n)$ are said to be sequentially independent if both add and remove the same elements and have the same minimal initial digraph.*

If we have compatibility and coherence, then automatically we have sequential independence if it is possible to guarantee the same minimal initial digraph for both sequences.

Theorem 6.2 *With the notation as above, if s_n is compatible and coherent and $\sigma(s_n)$ is compatible and coherent and both have the same minimal initial digraph, then they are sequentially independent.*

□By hypothesis we can define two productions c_s, $c_{\sigma(s)}$ which are respectively the compositions coming from s_n and $\sigma(s_n)$. Using commutativity of sum in formulae (57) and (58) – i.e., the order in which elements are added does not matter – we directly see that s_n and $\sigma(s_n)$ add and remove the same elements.■

Note that, even though the final result is the same when moving sequential independent productions inside a given concatenation, intermediate states can be very different. By hypothesis, minimal initial digraphs are equal: later in this section we shall call it *G-congruence.*

[15]Vector representation is possible only if the permutation is a cycle.

In the rest of this section we shall discuss permutations that move one production forward or backward a certain number of positions, maintaining the output of the sequence. We are going to investigate, assuming compatibility, the conditions needed to move one production without changing the result. That is, using theorem 6.2, the conditions to be satisfied such that starting with a coherent concatenation we again get a coherent concatenation after applying the permutation, and besides both the original sequence and the permuted one have the same minimal initial digraph.

First, we recall a simple notation for cycles moving forward and backward a production:

1. advance production *n-1* positions: $\phi_n = [\,1 \quad n \quad n-1 \ldots 3 \quad 2\,]$
2. delay production *n-1* positions: $\delta_n = [\,1 \quad 2 \ldots n-1 \quad n\,]$

Example. Consider advancing three positions the production p_5 in $s_5 = p_5; p_4; p_3; p_2; p_1$ to get $\sigma(s_5) = p_4; p_3; p_2; p_5; p_1$, where $\sigma = [\,1\,4\,3\,2\,]$. To illustrate the way in which we represent delaying a production, moving backwards production p_2 two places $p_5; p_4; p_3; p_2; p_1 \longmapsto p_5; p_2; p_4; p_3; p_1$ has as associated cycle $[\,2\,3\,4\,]$.[16]

Theorem 6.3 *Consider coherent productions $t_n = p_\alpha; p_n; p_{n-1}; \ldots; p_2; p_1$ and $s_n = p_n; p_{n-1}; \ldots; p_2; p_1; p_\beta$ and permutations ϕ_{n+1} and δ_{n+1}.*

1. *$\phi_{n+1}(t_n)$ is coherent if*

$$e_\alpha^E \bigtriangledown_1^n \left(\overline{r_x^E}\, L_y^E\right) \vee R_\alpha^E \bigtriangledown_1^n \left(\overline{e_x^E}\, r_y^E\right) = 0 \tag{59}$$

2. *$\delta_{n+1}(s_n)$ is coherent if*

$$L_\beta^E \bigtriangleup_1^n \left(\overline{r_x^E}\, e_y^E\right) \vee r_\beta^E \bigtriangleup_1^n \left(\overline{e_x^E}\, R_y^E\right) = 0 \tag{60}$$

Proof

□Both cases have a very similar demonstration so only production advancement is included. The way to proceed is to check differences between the original sequence t_n and the swapped one, $\phi_n(t_n)$, discarding conditions already imposed by t_n.

We start with $t_2 = p_\alpha; p_2; p_1 \longmapsto \phi_3(t_2) = p_2; p_1; p_\alpha$, where $\phi_3 = [\,1\,3\,2\,]$. Coherence of both sequences specify several conditions to be fulfilled, included in the following table:

Coherence of $p_\alpha; p_2; p_1$		Coherence of $p_2; p_1; p_\alpha$	
$e_2^E\, L_\alpha^E = 0$	$(t.1.1)$	$e_1^E\, L_2^E = 0$	$(t.1.7)$
$e_1^E\, L_2^E = 0$	$(t.1.2)$	$e_\alpha^E\, L_1^E = 0$	$(t.1.8)$
$e_1^E\, L_\alpha^E\, \overline{r_2^E} = 0$	$(t.1.3)$	$e_\alpha^E\, L_2^E\, \overline{r_1^E} = 0$	$(t.1.9)$
$r_\alpha^E R_2^E = 0$	$(t.1.4)$	$r_2^E R_1^E = 0$	$(t.1.10)$
$r_2^E R_1^E = 0$	$(t.1.5)$	$r_1^E R_\alpha^E = 0$	$(t.1.11)$
$r_\alpha^E R_1^E \overline{e_2^E} = 0$	$(t.1.6)$	$r_2^E R_\alpha^E \overline{e_1^E} = 0$	$(t.1.12)$

Conditions (t.1.7) and (t.1.10) can be found in the original sequence – (t.1.2) and (t.1.5) – so they can be disregarded. Now we want to set these identities expressed using operators delta (39) and nabla (40). In addition, we would like the matrices that are dependant on the production moved forward to be clearly apart from the rest of elements in the equation. We make use of (16) on (t.1.8) and (t.1.9) to get:

$$e_\alpha^E\, L_1^E\, \overline{r_1^E} = 0 \tag{61}$$

$$e_\alpha^E\, L_2^E\, \overline{r_2^E}\, \overline{r_1^E} = 0 \tag{62}$$

For the same reason, applying (13) to conditions (t.1.11) and (t.1.12):

$$r_1^E \overline{e_1^E} R_\alpha^E = 0 \tag{63}$$

$$r_2^E \overline{e_2^E} R_\alpha^E \overline{e_1^E} = 0 \tag{64}$$

[16]Note that numbers in the permutation refer to the place the production occupies in the sequence, numbering from left to right, and not to its subindex.

Condition (t.1.4) may be split in two parts – recall (30), (31) and the remark right afterwards – being $r_2^E r_3^E = 0$ one of them. Doing the same operation on (t.1.12), $r_2^E r_3^E \overline{e_1^E} = 0$ is obtained, which is automatically verified and therefore should not be considered. It is not ruled out since, as stated above, we want to get formulae expressible using operators delta and nabla. Finally we get the equation:

$$R_\alpha^E \overline{e_1^E} \left(r_1^E \vee \overline{e_2^E} r_2^E \right) \vee e_\alpha^E \overline{r_1^E} \left(L_1^E \vee \overline{r_2^E} L_2^E \right) = 0 \tag{65}$$

Performing similar manipulations on the concatenation $t_3 = p_\alpha; p_3; p_2; p_1 \longmapsto \phi_4(t_3) = p_3; p_2; p_1; p_\alpha$, where $\phi_4 = [\,1\ 4\ 3\ 2\,]$, we find out that the condition to be satisfied is:

$$\begin{aligned} R_\alpha^E \overline{e_1^E} \Bigg(r_1^E \quad \vee \quad & \overline{e_2^E} \left[r_2^E \vee \overline{e_3^E} r_3^E \right] \Bigg) \vee \\ \vee \quad & e_\alpha^E \overline{r_1^E} \Bigg(L_1^E \vee \overline{r_2^E} \left[L_2^E \vee \overline{r_3^E} L_3^E \right] \Bigg) = 0 \end{aligned} \tag{66}$$

Figure 17 includes the associated graphs to these last examples and to $n = 4$, where in both cases a dashed box isolates the advanced production. The proof can be finished by induction.■

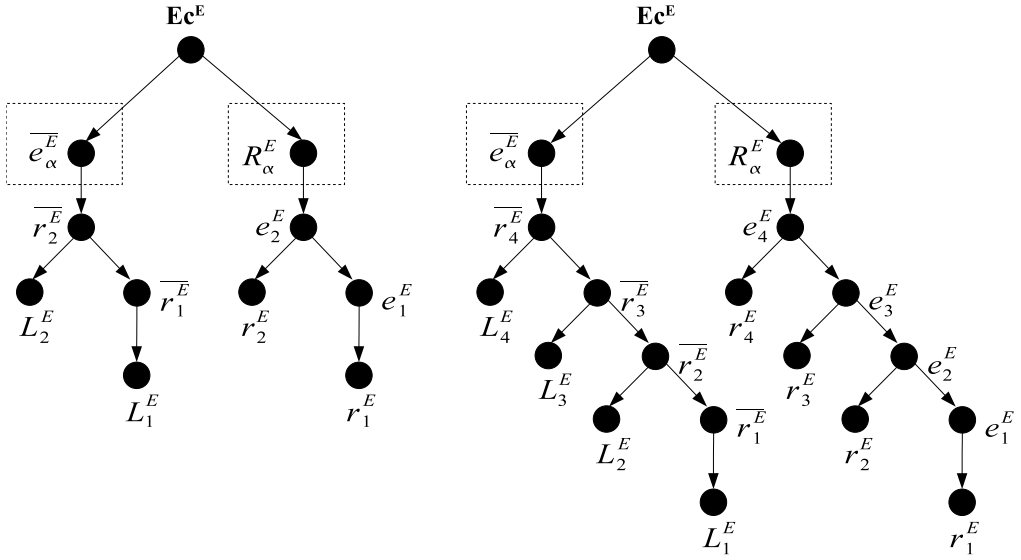

Figure 17: Advancing Productions. Three and Five Productions.

Example. Reusing some productions introduced in previous examples (q_1, q_2 and q_3), we are going to check coherence for a sequence of three productions in which one is directly delayed two positions. As commented in preceeding example, it is mandatory to change q_3 in order to keep compatibility, so a new production q_3' is introduced, depicted in Figure 18.

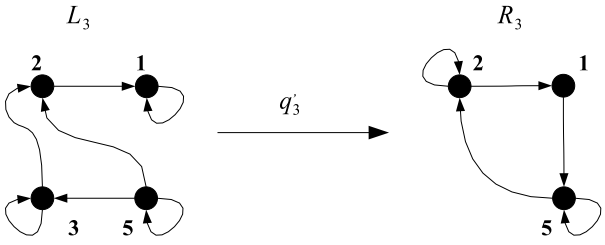

Figure 18: New production q_3'.

The minimal initial digraph for $q_3';q_2;q_1$ remains unaltered, i.e. $M_{q_3';q_2;q_1} = M_{q_3;q_2;q_1}$, but the one for $q_1;q_3';q_2$ is slightly different and can be found in Figure 19 along with the concatenation $s'_{123} = q_1;q_3';q_2$ and its intermediate states.

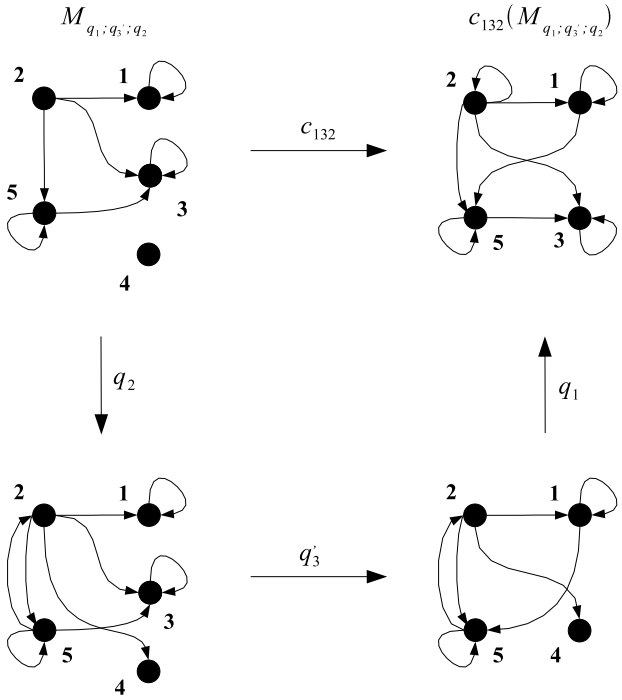

Figure 19: Composition and Concatenation. Three Productions.

In this example, production q_1 is delayed two positions inside $q_3';q_2;q_1$ to obtain $q_1;q_3';q_2$. We can express such permutation as $\delta_3 = [1\ 2\ 3]$. Formula (59) expanded, simplified and adapted for this case is:

$$\underbrace{L_1^E\left(e_2^E\overline{r_3^E} \vee e_3^E\right)}_{(*)} \vee \underbrace{r_1^E\left(\overline{e_3^E}R_2^E \vee R_3^E\right)}_{(**)} \tag{67}$$

Finally, all elements are substituted and the operations are performed, checking that the result is the null matrix.

$$(*) = \left[\begin{array}{ccccc|c} 0&0&0&0&1&2\\ 0&0&0&0&0&3\\ 1&0&1&0&0&5\\ 0&0&0&0&0&1\\ 0&0&0&0&0&4 \end{array}\right]\left(\left[\begin{array}{ccccc|c} 0&1&0&0&1&2\\ 0&0&0&0&0&3\\ 0&0&1&0&0&5\\ 0&0&0&0&0&1\\ 0&0&0&0&0&4 \end{array}\right]\left[\begin{array}{ccccc|c} 0&1&1&1&1&2\\ 1&1&1&1&1&3\\ 1&1&0&1&1&5\\ 1&1&0&1&1&1\\ 1&1&1&1&1&4 \end{array}\right] \vee \left[\begin{array}{ccccc|c} 0&0&0&0&0&2\\ 1&1&0&0&0&3\\ 0&1&0&0&0&5\\ 0&0&0&1&0&1\\ 0&0&0&0&0&4 \end{array}\right]\right) = 0$$

$$(**) = \left[\begin{array}{ccccc|c} 0&1&0&0&1&2\\ 0&1&0&0&0&3\\ 0&1&0&0&0&5\\ 0&0&0&0&0&1\\ 0&0&0&0&0&4 \end{array}\right]\left(\left[\begin{array}{ccccc|c} 1&1&1&1&1&2\\ 0&0&1&1&1&3\\ 1&0&1&1&1&5\\ 1&1&1&0&1&1\\ 1&1&1&1&1&4 \end{array}\right]\left[\begin{array}{ccccc|c} 0&0&0&1&0&2\\ 1&0&0&0&0&3\\ 1&0&0&0&0&5\\ 0&0&0&0&0&1\\ 0&0&0&0&0&4 \end{array}\right] \vee \left[\begin{array}{ccccc|c} 1&0&0&1&0&2\\ 0&0&0&0&0&3\\ 1&0&1&0&0&5\\ 0&0&1&0&0&1\\ 0&0&0&0&0&4 \end{array}\right]\right) = 0$$

Now we seek the conditions that must be fulfilled to guarantee sameness of the minimal initial digraphs,[17] first for productions advancement and then for delaying. As before, we start with two productions, continue with three, four and set the theorem for the general case.

Suppose we have a coherent sequence made up of two productions $s_2 = p_2;p_1$ with minimal initial digraph M_2 and applying the (only possible) permutation σ_2 get another coherent concatenation $s_2' = p_1;p_2$ with minimal initial digraph M_2'.

[17] Sameness with respect to a digraph G will be defined as *G-applicability* later in this section.

Production p_1 does not delete any element added by p_2 because, otherwise, if p_1 in s_2 deleted something, it should mean that it already existed (as p_1 is applied first in s_2) while p_2 adding that same element in s_2' should mean that this element wasn't present (because p_2 is applied first in s_2'). This condition can be written as

$$e_1 r_2 = 0 \tag{68}$$

A similar reasoning states that p_1 can't add any element that p_2 is going to use:

$$r_1 L_2 = 0 \tag{69}$$

Analogously for p_2 against p_1, we have

$$e_2 r_1 = 0 \tag{70}$$

$$r_2 L_1 = 0 \tag{71}$$

As a matter of fact two equations are redundant because they are already contained in the other two. Note that $e_i L_i = e_i$, i.e., in some sense $e_i \subset L_i$, so it is enough to ask for:

$$r_1 L_2 \vee r_2 L_1 = 0 \tag{72}$$

It would be necessary to check that $M_2 = M_2'$, but we are not going to include these calculations because they are going to be carefully justified for three and four productions. It seems more interesting to compare conditions for two productions with those of the SPO categorical-algebraic approach.

In references [8] [9], sequential independence is defined and categorically charaterized. It is not very difficult to translate those conditions to our "matrix language":

$$r_1 L_2 \quad = \quad 0 \tag{73}$$

$$e_2 R_1 \quad \equiv \quad e_2 r_1 \vee e_2 \, \overline{e_1} \, L_1 = 0 \tag{74}$$

First condition is (69) and, as mentioned above, first part of second condition – $e_2 r_1 = 0$ – is already considered in (69) or in (73). Second part of second equation – $e_2 \, \overline{e_1} \, L_1 = 0$ – is demanded for coherence.[18]

We shall proceed with three productions, so following a consistent notation we set $s_3 = p_3; p_2; p_1$, $s_3' = p_2; p_1; p_3$ with permutation $\sigma_3 = [\,1\ 3\ 2\,]$ and their corresponding minimal initial digraphs $M_3 = L_1 \vee \overline{r_1} \, L_2 \vee \overline{r_1} \, \overline{r_2} \, L_3$ and $M_3' = \overline{r_3} \, L_1 \vee \overline{r_3} \, \overline{r_2} \, L_2 \vee L_3$. Conditions are deduced similarly to the two productions case and the reader may find interesting to interpret them:

$$r_3 L_1 = 0 \qquad r_3 L_2 \overline{r_1} = 0 \qquad r_1 L_3 = 0 \qquad r_2 L_3 \overline{e_1} = 0$$

which can be put altogether in a single expression

$$L_3 \, (r_1 \vee \overline{e_1} \, r_2) \vee r_3 \, (L_1 \vee \overline{r_1} \, L_2) = 0 \tag{75}$$

Moving one production three positions forward in a sequence of four productions, i.e., $p_4; p_3; p_2; p_1 \rightarrow p_3; p_2; p_1; p_4$, while maintaining the minimal initial digraph has as associated conditions the equation

$$L_4 \, (r_1 \vee \overline{e_1} \, r_2 \vee \overline{e_1} \, \overline{e_2} \, r_3) \vee r_4 \, (L_1 \vee \overline{r_1} \, L_2 \vee \overline{r_1} \, \overline{r_2} \, L_3) = 0 \tag{76}$$

Definition 6.4 (G-congruence) *Two sequences s_n and $\sigma\,(s_n)$, where σ is a permutation, are called* G-congruent *if and only if they have the same minimal initial digraph, $M_{s_n} = M_{\sigma(s_n)}$*

Conditions that must be fulfilled in order to maintain the minimal initial digraph will be called **congruence conditions** and will be abbreviated as **CC**. By induction it can be proved that for advancement of one production $n-1$ positions inside the sequence of n productions $s_n = p_n; \ldots; p_1$, the equation which contains all CC can be expressed in terms of operator ∇ and has the form

$$CC_n \, (\phi_{n-1}, s_n) = L_n \nabla_1^{n-1} \, (\overline{e_x} \, r_y) \vee r_n \nabla_1^{n-1} \, (\overline{r_x} \, L_y) \tag{77}$$

[18] In fact, what is mandatory for coherence is a bit stronger: $e_2 L_1 = 0$.

Delaying a production $n-1$ positions has a very similar associated formula. Suppose that we are moving backwards production p_1 in concatenation s_n to get $s_n'' = p_1; p_n; \ldots; p_2$, i.e., we are applying δ_{n-1}

$$CC_n\left(\delta_{n-1}, s_n\right) = L_1 \nabla_2^n \left(\overline{e_x}\, r_y\right) \vee r_1 \nabla_2^n \left(\overline{r_x}\, L_y\right) \tag{78}$$

It is necessary to show that these conditions guarantee sameness of minimal initial digraphs, but first we need a technical lemma that provides us with some identities used to transform the minimal initial digraphs. Both, in the lemma and in the theorem, $n = 3$ and $n = 4$ are demonstrated in full detail, leaving the end of the proofs to induction. Advancement and delaying are very similar so only advancement is considered in the rest of the section.

Lemma 6.5 *Suppose $s_n = p_n; \ldots; p_1$ and $s_n' = \sigma\left(s_n\right) = p_{n-1}; \ldots; p_1; p_n$ and that $CC_n\left(\phi\right)$ is satisfied. Then the following identity may be or'ed to M_n without changing it.*

$$DC_n\left(\phi_{n-1}, s_n\right) = L_n \nabla_{i=1}^{n-2} \left(\overline{r_x}\, e_y\right) \tag{79}$$

Proof
□Let's start with three productions. Recall that $M_3 = L_1 \vee \ldots$ and that $L_1 = L_1 \vee e_1 = L_1 \vee e_1 \vee e_1 L_3$. Note that $e_1 L_3$ is (79) for $n = 3$.

For $n = 4$ apart from $e_1 L_4$, we need to get $e_2 \overline{r_1} L_4$. Recall again the minimal initial digraph for four productions whose two first terms are $M_4 = L_1 \vee \overline{r_1} L_1 \vee \ldots = (L_1 \vee e_1) \vee (\overline{r_1} L_1 \vee \overline{r_1} e_1) \vee \ldots = (L_1 \vee e_1 \vee e_1 L_1) \vee (\overline{r_1} L_2 \vee \overline{r_1} e_2 \vee \overline{r_1} e_2 L_4) \vee \ldots$. Last term $\overline{r_1} e_2 L_4$ is (79) for $n = 4$. The proof can be ended by induction. ■

Theorem 6.6 *With notation as above, if s_n and s_n' are coherent and condition $CC_n\left(\phi_{n-1}, s_n\right)$ is satisfied then they are G-congruent, $M_n = M_n'$.*

Proof
□It will be shown for three and five productions, using CC_i and DC_i, that $M_i = M_i'$. The identities $a \vee \overline{a}\, b = a \vee b$ and $\overline{a} \vee a\, b = \overline{a} \vee b$ will be used in this demonstration.

$$\begin{aligned} M_3 \vee CC_3 \vee DC_3 &= L_1 \vee \overline{r_1}\, L_2 \vee \overline{r_1}\, \overline{r_2}\, L_3 \vee r_1 L_3 \vee \overline{e_1}\, r_2 L_3 \vee r_3 L_1 \vee \\ &\vee \quad \overline{r_1}\, r_3 L_2 \vee e_1 L_3 = L_1 \vee \overline{r_1}\, L_2 \vee \not{r_1}\, \overline{r_2}\, L_3 \vee r_1 L_3 \vee \\ &\vee \quad \not{e_1}\, r_2 L_3 \vee e_1 L_3 = L_1 \vee \overline{r_1}\, L_2 \vee \not{r_2}\, L_3 \vee r_2 L_3 = \\ &= \quad L_1 \vee \overline{r_1}\, L_2 \vee L_3 \end{aligned}$$

In our first step, as neither $r_3 L_1$ nor $\overline{r_1}\, r_3 L_2$ are applied to M_3, they have been omitted. Once $r_1 L_3$ and $e_1 L_3$ have been used, they are omitted as well.

Let's check out M_3', where in second equality $r_1 L_3$ and $r_2\, \overline{e_1}\, L_3$ are ruled out since they are not used.

$$\begin{aligned} M_3' \vee CC_3 &= \overline{r_3}\, L_1 \vee \overline{r_1}\, \overline{r_3}\, L_2 \vee L_3 \vee r_1 L_3 \vee r_2\, \overline{e_1}\, L_3 \vee r_3 L_1 \vee \overline{r_1}\, r_3 L_2 = \\ &= \not{r_3}\, L_1 \vee \overline{r_1}\ \not{r_3}\, L_2 \vee L_3 \vee r_3 L_1 \vee \overline{r_1}\, r_3 L_2 = \\ &= L_1 \vee \overline{r_1}\, L_2 \vee L_3 \end{aligned}$$

The case for five productions will illustrate in detail the procedure to follow and how CC_5 and DC_5 are used to prove that $M_5 = M_5'$. The key point is the transformation $\overline{r_1}\, \overline{r_2}\, \overline{r_3}\, \overline{r_4}\, L_5 \to L_5$ and the followig identities show in detail the way to proceed.

$\not{r_1}\, \overline{r_2}\, \overline{r_3}\, \overline{r_4}\, L_5 \vee r_1 L_5 = \overline{r_2}\, \overline{r_3}\, \overline{r_4}\, L_5$
$\not{r_2}\, \overline{r_3}\, \overline{r_4}\, L_5 \vee \not{e_1}\, r_2 L_5 \vee e_1 L_5 = \overline{r_3}\, \overline{r_4}\, L_5$
$\not{r_3}\, \overline{r_4}\, L_5 \vee \not{e_1}\ \not{e_2} r_3 L_5 \vee e_1 L_5 \vee \not{r_1} e_2 L_5 \vee r_1 L_5 = \overline{r_4}\, L_5$
$\not{r_4} L_5 \vee \not{e_1}\ \not{e_2}\ \not{e_3} r_4 L_5 \vee e_1 L_5 \vee \not{r_1} e_2 L_5 \vee r_1 L_5 \vee \not{r_1}\ \not{r_2} e_3 L_5 \vee \not{e_1} r_2 L_5 = L_5$ ■

Previous theorems foster the following notation: if (59) is satisfied and we have sequential independence, we shall write $p_\alpha \perp \left(p_n; \ldots; p_1\right)$ whereas if equation (60) is true and again they are sequential independent, it shall be represented by $\left(p_n; \ldots; p_1\right) \perp p_\beta$. For example, if we have the coherent sequence made up of two productions $p_2; p_1$ and we have that $p_1; p_2$ is coherent we can write $p_2 \perp p_1$ to mean that either p_2 may be moved to the front or p_1 to the back.

Example. It is not difficult to put an example of three productions $t_3 = w_3; w_2; w_1$ where the advancement of the third production two positions to get $t_3' = w_2; w_1; w_3$ has the following properties: their associated

minimal initial digraphs – N_3 and N_3', respectively – coincide, they are both coherent (and thus sequential independent in the sense introduced in this article) but $t_3'' = w_2; w_3; w_1$ can not be performed, so it is not possible to advance w_3 one position and, right afterwards, another one, i.e., the advancement of two places must be carried out in a single step.

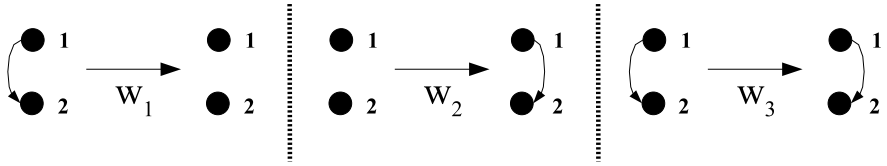

Figure 20: Three Very simple Productions.

Using the notation already introduced, this is an example where $w_3 \perp (w_2; w_1)$ but $w_3 \not\perp w_2$. As far as we know, in SPO or DPO approaches, testing whether $w_3 \perp (w_2; w_1)$ or not has to be performed in two steps: $w_3 \perp w_2$, that would allow for $w_3; w_2; w_1 \rightarrow w_2; w_3; w_1$, and $w_3 \perp w_1$ to get the desired result.

As drawn in figure 20, w_1 deletes edge $(1, 2)$, w_2 adds it while it is just used by w_3 (appears on its left hand side but it is not deleted).

7 Explicit Parallelism

This paper finishes analyzing which productions or group of productions can be computed in parallel and what conditions guarantee this operation.

In the categorical-algebraic approach, the definition for two productions is settled considering the two alternative sequential ways in which they can be composed, and looking for sameness in their final state. Intermediate states are disregarded using categorical coproduct of the involved productions. Then, the main difference between sequential and parallel execution is the existence of intermediate states in the former, as seen in Figure 21.

As stated in section 2.1, the parallel rule can only be constructed if the two sequences ($G \overset{p_1}{\Longrightarrow} X_1 \overset{p_2}{\Longrightarrow} H$ and $G \overset{p_2}{\Longrightarrow} X_2 \overset{p_1}{\Longrightarrow} H$) are sequential independent.

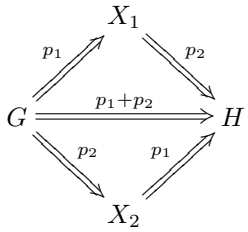

Figure 21: Parallel Execution.

We follow the same approach, but as host graphs to which apply rules are not considered, we shall say that it is possible to execute two productions in parallel if the result does not depend on generated intermediate states.

Definition 7.1 *Two productions p_1 and p_2 are said to be truly concurrent if it is possible to define their composition and it does not depend on the order:*

$$p_2 \circ p_1 = p_1 \circ p_2 \tag{80}$$

We use the notation $p_1 \parallel p_2$ to denote true concurrency. True concurrency defines a symmetric relation so it does not matter whether $p_1 \parallel p_2$ or $p_2 \parallel p_1$ is written.

Next proposition compares true concurrency and sequential independence for two productions, which is the parallelism theorem [3]. The proof is straightforward in our case and is not included.

Proposition 7.2 *Suppose that a coherent concatenation $p_2; p_1$ is given and if $p_2 \perp p_1$, they are G-congruent. Then, $p_1 \parallel p_2$ if and only if $p_2 \perp p_1$.*

So far we have just considered one production per branch when parallelizing, as represented on the left of figure 22. One way to deal with more general schemes – centre and right of the same figure – is to test for example if $p_i \parallel p_j$, $\forall i \in \{3,4,5\}$, $\forall j \in \{1,2,3\}$, for the middle scheme in figure 22.

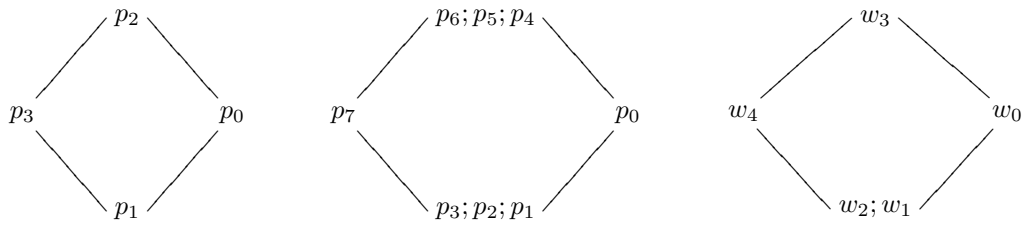

Figure 22: Examples of Parallel Execution.

Although it is not true in general, in many cases it is not necessary to check true concurrency for every two productions. The following example illustrates the idea.

Example. Suppose a given concatenation with three productions $w_3; w_2; w_1$, as those depicted in Figure 20. w_1 deletes one edge, w_2 adds the same edge while w_3 uses it.

We have already seen that $w_3; w_2; w_1$ is compatible and coherent and that $w_3 \perp (w_2; w_1)$. Both have the same minimal initial digraph. Following our previous study for two productions we would like to put w_3 and $w_2; w_1$ in parallel, as depicted on the right of Figure 22.

From a sequential point of view this diagram can be interpreted in different ways, depending on how they are computed. There are three dissimilar interleavings:

1. $w_3; w_2; w_1$
2. $w_2; w_1; w_3$
3. $w_2; w_3; w_1$

Any problem involving the first two possibilities is ruled out by coherence. As a matter of fact w_3 and $w_2; w_1$ can not be parallelized because it could be the case that w_3 is using edge $(1,2)$ when w_1 has just deleted it and before w_2 adds it, which is what the third case tries to express, leaving the system in an inconsistent state. Thus, we do not have $w_3 \parallel w_2$ nor $w_3 \parallel w_1$ – we do not have sequential independence – but both $w_3; w_2; w_1$ and $w_2; w_1; w_3$ are coherent. Next theorem tries to solve this problem.

Theorem 7.3 *Let $s_n = p_n; \ldots; p_1$ and $t_m = q_m; \ldots; q_1$ be two compatible and coherent sequences wtih the same minimal initial digraph, where either $n = 1$ or $m = 1$. Suppose $r_{m+n} = t_m; s_n$ is compatible and coherent and either $t_m \perp s_n$ or $s_n \perp t_m$. Then, $t_m \parallel s_n$ through composition.*

Proof
□Using proposition (7.2)■

In the last sentence, "through composition" means that the concatenation with length greater than one must be transformed into a single production using composition. This is possible because it is coherent and we are assuming compatibility in this section – please, refer to proposition 5.4 –. In fact it should not be necessary to transform the whole concatenation using composition, but only the parts that present a problem.

Setting $n = 1$ corresponds to advancing a production in sequential independence, while $m = 1$ to moving a production backwards inside a concatenation. In addition, in the hypothesis we ask for coherence of r_n and either $t_m \perp s_n$ or $s_m \perp t_n$. In fact, if r_{m+n} is coherent and $t_m \perp s_n$, then $s_n \perp t_m$. It is also true that if r_{m+n} is coherent and $s_n \perp t_m$, then $t_m \perp s_n$, by contradiction.

Although the idea behind theorem 7.3 is to erase intermediate states through composition, in a real system this is not always possible or desirable if, for example, these states were used for synchronization of productions or states.

8 Conclusions and Future Work

In this paper, a new, purely algebraic approach to graph transformation based on matrix algebra has been proposed. A characterization of productions by means of matrices as well as the concepts of compatibility, coherence and G-congruence have been introduced, applying them to the study of sequentialization and parallelism. The latter has been approached both from the sequential point of view (by considering sequence interleavings) as well as from the explicit parallelism view (by defining rule composition, which does not generate intermediate states).

In our opinion, the approach is interesting because it provides a new viewpoint to graph transformation. As it is purely algebraic, based on matrix manipulation, it has the potential to be efficiently implemented on computers. The approach has permitted the generalization in some sense of known results in the categorical-algebraic approach. For example the one for sequential independence which were defined for two productions. In particular, we consider permutations of rule sequences of arbitrary length. Other known facts in the categorical framework have a clear correspondence in our approach, and can be studied from another perspective, such as the dangling edge condition (compatibility), sequential independence, parallel production, etc.

Another interesting issue is the fact that we can perform the analysis independent of the host graph. This has the practical advantage that some information can be gathered a priori, when the graph transformation rules are being specified.

With respect to future work, we will start defining the match and its characterization. The match will be considered as an operator modifying the rule's LHS and RHS. As stated throughout the paper, two options are available here. In the style of SPO, the match operator could complete the rule, explictly erasing the dangling edges. If the DPO style is followed, then a match would not be valid if some edge would dangle. In both cases, the theory introduced so far would be applicable.

Additionally, we want to consider other kinds of digraphs, not only the family of simple digraphs. A lot of concepts are not even mentioned, but are interesting to consider, such as application conditions (specially negative application conditions) [13], type graphs [2], attributes [11], amalgamation [22], etc. Following with the study of parallel independence, critical pair analysis [14] could also be considered.

Staying at the level introduced in this paper, it is also possible to study the algebra associated to the operators introduced so far, or generalizations of them.

References

[1] Baldan, P., Corradini, A., Ehrig, H., Löwe, M., Montanari, U. and Rossi, F., 1999. *Concurrent Semantics of Algebraic Graph Transformations.* In [9], pp.: 107-187.

[2] Corradini, A., Montanari, U., Rossi, F. 1996. *Graph processes.* Fundamenta Informaticae, 26(3-4), pp.: 241 - 265.

[3] Corradini, A., Montanari, U., Rossi, F., Ehrig, H., Heckel, R., Löwe, M. 1999. *Algebraic Approaches to Graph Transformation - Part I: Basic Concepts and Double Pushout Approach.* In [8], pp.: 163-246

[4] Courcelle, B. 1990. *Graph Rewriting: An Algebraic and Logic Approach* Handbook of Theoretical Computer Science, Vol. B. pp.: 193-242.

[5] Drewes, F., Habel, A., Kreowski, H.-J., Taubenberger, S. 1995. *Generating self-affine fractals by collage grammars.* Theoretical Computer Science 145:159-187, 1995.

[6] Ehrig, H. 1979. *Introduction to the Algebraic Theory of Graph Grammars.* In V. Claus, H. Ehrig, and G. Rozenberg (eds.), 1st Graph Grammar Workshop, pages 1-69. Springer LNCS 73, 1979.

[7] Ehrig, H., Heckel, R., Llabres, M., Orejas, F., Padberg, J., Rozenberg, G. 1998. *Double-Pullback Graph Transitions: A Rule-Based Framework with Incomplete Information.* In Proc. Theory and Application of Graph Transformations: 6th International Workshop, TAGT'98, LNCS 1764. Springer, pp.: 85-102.

[8] Ehrig, H., Engels, G., Kreowski, H.-J., Rozenberg, G. 1999. *Handbook of Graph Grammars and Computing by Graph Transformation. Vol 1. Foundations.* World Scientific.

[9] Ehrig, H., Kreowski, H.-J., Montanari, U., Rozenberg, G. 1999. *Handbook of Graph Grammars and Computing by Graph Transformation. Vol.3., Concurrency, Parallelism and Distribution.* World Scientific.

[10] Ehrig, H., Heckel, R., Korff, M., Löwe, M., Ribeiro, L., Wagner, A., Corradini, A. 1999. *Algebraic Approaches to Graph Transformation - Part II: Single Pushout Approach and Comparison with Double Pushout Approach.* In [8], pp.: 247-312.

[11] Ehrig, H., Prange, U., Taentzer, G. 2004. *Fundamental Theory for Typed Attributed Graph Transformation.* Proceedings of International Conference of Graph Transformation (ICGT'04). Lecture Notes in Computer Science 3256 pp.: 161-177. Springer.

[12] Ehrig, H., Ehrig, K., de Lara, J., Taentzer, T., Varr, D., Varr-Gyapay, S. 2005. *Termination Criteria for Model Transformation.* Proceedings of Fundamental Approaches to Software Engineering FASE05 (ETAPS'05). Lecture Notes in Computer Science 3442 pp.: 49-63. Springer.

[13] Heckel, R., Wagner, A. 1995. *Ensuring consistency of conditional graph rewriting - a constructive approach.* Proc. of SEGRAGRA 1995, Joint COMPUGRAPH/SEMAGRAPH Workshop on Graph Rewriting and Compu-tation, In ENTCS Vol 2, 1995.

[14] Heckel, R., Küster, J. M., Taentzer, G. 2002. *Confluence of Typed Attributed Graph Transformation Systems.* In ICGT'2002. LNCS 2505, pp.: 161-176. Springer.

[15] Kahl, W. 2002. *A Relational Algebraic Approach to Graph Structure Transformation* Tech.Rep. 2002-03. Universität der Bundeswehr München.

[16] de Lara, J., Vangheluwe, H., 2004. *Defining Visual Notations and Their Manipulation Through Meta-Modelling and Graph Transformation.* Journal of Visual Languages and Computing. Special issue on "Domain-Specific Modeling with Visual Languages", Vol 15(3-4), pp.: 309-330. 2004. Elsevier Science

[17] Löwe, M. 1993. *Algebraic Approach to Single-Pushout Graph Transformation.* Theoretical Computer Science, 109:181-224.

[18] Mac Lane, S. 1997. *Categories for the Working Mathematician* . Volume 5 of Graduate Texts in Mathematics. Springer-Verlag, Berlin, 2nd. Edition (1st ed., 1971).

[19] Minas, M. 2002. *Concepts and realization of a diagram editor generator based on hypergraph transformation* Science of Computer Programming, Vol. 44(2), pp: 157 - 180.

[20] Mizoguchi, Y., Kuwahara, Y. 1995. Relational Graph Rewritings. Theoretical Computer Science, Vol 141, pp. 311-328.

[21] Schürr, A. *Programmed Graph Replacement Systems.* In [8], pp.: 479 - 546.

[22] Taentzer, G. 1996. *Parallel and Distributed Graph Transformation: Formal Description and Application to Communication-Based Systems.* Ph.D.Thesis, TU Berlin, Shaker Verlag, Germany.