Matrix Approach To Graph Transformation: Matching and Sequences

Pedro Pablo Pérez Velasco, Juan de Lara

Escuela Politécnica Superior
Universidad Autónoma de Madrid
{pedro.perez, juan.delara}@uam.es

Abstract. In this work we present our approach to (simple di-)graph transformation based on an algebra of boolean matrices. Rules are represented as boolean matrices for nodes and edges and derivations can be efficiently characterized with boolean operations only. Our objective is to analyze properties inherent to rules themselves (without considering an initial graph), so this information can be calculated at specification time. We present basic results concerning well-formedness of rules and derivations (compatibility), as well as concatenation of rules, the conditions under which they are applicable (coherence) and permutations. We introduce the *match*, which permits the identification of a grammar rule left hand side inside a graph. We follow a similar approach to the single pushout approach (SPO), where dangling edges are deleted, but we first adapt the rule in order to take into account any deleted edge. To this end, a notation borrowed from functional analysis is used. We study the conditions under which the calculated data at specification time can be used when the match is considered.

1 Introduction

Graph Transformation [11] 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 visual languages, visual simulation, picture processing and model transformation (see [5] and [11] vol.2 for some applications).

The classical algebraic approach to graph transformation is based on *category theory* [3], and provides a rich body of theoretical results(see [11] vol.1). 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 [11] [5] [6]. Nonetheless, methods to increase efficiency and new analysis techniques that can be efficiently 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. In this way, simple digraphs can be represented as boolean matrices and productions as matrices for edge and node deletion and addition, together with a graph L (also represented with matrices) that must be present in the host graph in order for the rule to be applicable. Therefore, the effects of a production $p : L \to R$ can be modelled using boolean matrix operations only. This purely algebraic approach constitutes a different perspective from algebraic-categorical approaches, as it provides an operational characterization of most concepts (closer to implementation) and has the potential for efficient implementation and parallelization.

In our work [10], most analysis is made independently of the host graph. The advantages of this approach are twofold. First, all properties under study are *inherent* to the graph transformation system 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. We present concepts such as coherence (potential applicability of a sequence), minimal initial digraph (smallest graph with enough elements to execute a sequence), rule permutation coherence and G-congruence (potential sequential independence). These concepts provide a rich amount of information about productions and how they are related to each other, including limitation in their application, dependencies and dynamical behaviour. To the best of our knowledge, some of these results are new, for example we have studied conditions for coherence of rule advancement and delay an arbitrary number of positions in a sequence. For space limitations, some proofs are omitted, but can be found in [10].

In addition, we introduce the match as an *operator* modifying the rule by *including* the context in which it is applied. We use a similar approach to SPO [4], where the dangling edges are deleted. Thus, the rule is adapted to include the edges that would become dangling and explicitly delete them. Our goal is to use the information calculated about the grammar at specification time once the initial host graph is considered. In this work, we study how this information is modified when a host graph is taken into account. We also introduce a bra-ket operational notation for rules similar to that of functional analysis for operators (also known as *Dirac Notation*) [1]. Thus, productions can be depicted as $R = \langle L, p \rangle$, splitting the static part (initial state, L) from the dynamics (element addition and deletion, p).

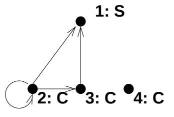
The paper is organized as follows. Section 2 presents the characterization of graphs and productions in our approach, together with rule sequences, minimal initial digraph, permutation and G-congruence. Section 3 presents our approach to handle the match. Section 4 revisits the properties calculated for rules in section 2, and study how they are affected by the match. Section 5 presents the conclusions and future work.

2 Characterization and Basic Properties

This section presents an informal introduction to the basic concepts in our approach. In subsection 2.1, 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* (i.e. it defines a simple digraph) and the concept of *completion*, where matrices representing graphs are modified – *arranged* – to permit operations between them. In subsection 2.2, we present production concatenation together with the concept of *coherence*. We present the minimal initial digraph, the conditions for sequence permutations to be coherent and the concept of potential sequential independence.

2.1 Simple Digraphs and Productions

A graph $G = (V, E)$ consists of two sets, one of nodes $V = \{V_i \mid i \in I\}$ and one of edges $E = \{(V_i, V_j) \in V \times V\}$. In this paper we are concerned with *simple digraphs,* "simple" meaning that only two arrows are allowed between two nodes (one in each direction), and "di-" because arrows have a direction. A simple digraph G is uniquely determined by its *adjacency matrix* A_G, whose element a_{ij} is one if $(i, j) \in E$, and zero otherwise. As we will delete and add edges and nodes, a *nodes vector* V_G is also associated to our digraph G, with its elements equal to one if the corresponding node is present in G and zero otherwise.



(a)

$$M^E_{C-S} = \left[\begin{array}{cccc|c} 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 2 \\ 1 & 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 & 4 \end{array}\right] \quad M^N_{C-S} = \left[\begin{array}{c|c} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{array}\right]$$

(b)

Fig. 1. (a) A Simple Digraph Representing a Client-Server System (b) Matrix Representation.

Fig. 1(a) shows a digraph representing a client-server system. Links between the clients and the server represent that the client is connected to a server. Links between clients represent a directed communication channel, while a loop link represents a message. The matrix representation of the previous graph is shown in Fig.1(b).

The boolean product between two adjacency matrices $M_G = (g_{ij})_{i,j \in \{1,\ldots,n\}}$ and $M_H = (h_{ij})_{i,j \in \{1,\ldots,n\}}$ is defined as $(M_G \odot M_H)_{ij} = \bigvee_{k=1}^{n} (g_{ik} \wedge h_{kj})$.

Next, we are interested in formulating the properties (that we call *compatibility*) that should be fulfilled by a boolean matrix and a vector of nodes to define a simple digraph. We want to forbid edges incident to nodes that do not belong to the digraph. We first define the norm $\|\cdot\|_1$ of a vector $N = (v_1, \ldots, v_n)$ as $\|N\|_1 = \bigvee_{i=1}^{n} v_i$.

Proposition 1. *A pair* (M, N)*, where* M *is an adjacency matrix and* N *a vector of nodes, is compatible if and only if they verify* $\left\|(M \vee M^t) \odot \overline{N}\right\|_1 = 0$. [1]

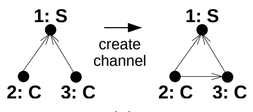
Now we consider productions and their characterization. We define a production as a morphism – in the sense of category theory – which transforms a simple digraph into another one, $p : L \to R$. We can describe a production p with two matrices for edges and two vectors for nodes. Therefore a production can be specified as functions between boolean matrices and vectors.

Definition 1 (Production) *A production* p *is a morphism between two simple digraphs* L *and* R*, and can be specified by the tuple* $p = \left(L^E, R^E; L^N, R^N\right)$ *where* E *stands for* edge *and* N *for* node. L *is the* left hand side *(LHS) and* R *is the* right hand side *(RHS).*

[1] where *t* denotes transposition.

A production models deletion and addition of edges and nodes, carried out in the order just mentioned, i.e., first deletion and then addition. These actions can be represented with two matrices for edges (e^E, r^E) and two vectors for nodes (e^N, r^N), which can be calculated as:[2] $e = L\,\overline{(L\,R)} = L\,\overline{R}$ and $r = R\,\overline{(L\,R)} = R\,\overline{L}$.

Fig. 2 shows a rule that creates a communication channel between two clients connected to the same server. The deletion matrix e^E (and vector e^N) is zero, while the addition matrix r^E has a unique non-zero element at position $(2, 3)$ and the addition vector for nodes is zero. From previous definitions, a number of conditions are immedi-



(a)

$$L^E_{CC} = \left[\begin{array}{ccc|c} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 2 \\ 1 & 0 & 0 & 3 \end{array}\right] \qquad R^E_{CC} = \left[\begin{array}{ccc|c} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 2 \\ 1 & 0 & 0 & 3 \end{array}\right]$$

(b)

Fig. 2. (a) Create Channel Rule (b) Matrix Representation of Rule (only for edges).

ate (see next proposition). The first two state that elements 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.[3] The last two conditions say that if an element is in the RHS, then it is not deleted, and that if the element is in the LHS, it is not created.

Proposition 2. *Let $p : L \to R$ be a production, the following identities hold for both edges and nodes: $r\,\overline{e} = r$, $e\,\overline{r} = e$, $R\,\overline{e} = R$, $L\,\overline{r} = L$.*

Finally we are ready to characterize a production $p : L \to R$ using deletion and addition matrices, starting from its LHS: $R = r \vee \overline{e}\,L$ (for both edges and nodes). It could be the case that the production erases a node but leaves some incident edges (*dangling edges*). Some conditions have to be imposed on matrices and vectors of nodes and edges to keep compatibility when a rule is applied (i.e., to avoid dangling edges):

1. An incoming edge cannot be added to a node that is going to be deleted or, using the norm, $\left\| r^E \odot e^N \right\|_1 = 0$. Similarly, for outgoing edges: $\left\| \left(r^E\right)^t \odot e^N \right\|_1 = 0$. Note how, vector e^N has a 1 in position i, if the node has to be deleted. Row i in matrix r^E depicts the outgoing edges for node i, and has a 1 in column j if edge (i, j) has to be added. Therefore vector $r^E \odot e^N$ contains elements $(\vee_{j=1}^{n} r^E_{ij} \wedge e^N_j)_{i\in\{1,\ldots,n\}}$ with a 1 in position i, if there is some newly added edge from node i to some node j which is deleted by the production. The transposition of r^E checks for new edges starting from deleted nodes.

[2] Superindices *E* and *N* shall be omitted if, for example, the formula applies to both cases or if it is clear from context which we refer to. Moreover, the **and** operator ($\wedge$) will also be omitted.

[3] This contrasts with the DPO approach, in which edges and nodes can be rewritten in a single rule. This can be useful to forbid the rule application if the dangling condition is violated. Section 3 explains how to deal with dangling edges in this approach.

2. Deleting a node with some incoming edge is forbidden, if the edge is not deleted as well: $\left\|\overline{e^E}\, L^E \odot e^N\right\|_1 = 0$. For outgoing edges: $\left\|\left(\overline{e^E}\, L^E\right)^t \odot e^N\right\|_1 = 0$. Matrix $\overline{e^E}\, L^E$ contains the edges in the rule's LHS that are not deleted, therefore $\overline{e^E}\, L^E \odot e^N$ results in a vector with a one in position i if some node j is deleted and has an incident edge coming from i (and the edge is not deleted). The transposition of $\overline{e^E}\, L^E$ checks for outgoing edges from deleted nodes.
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: $\left\|r^E \odot \left(\overline{r^N}\, \overline{L^N}\right)\right\|_1 = 0$. Similarly, for edges starting in a given node: $\left\|\left(r^E\right)^t \odot \left(\overline{r^N}\, \overline{L^N}\right)\right\|_1 = 0$. In this case, $\overline{r^N}\, \overline{L^N}$ is a vector containing a 1 in position i if node i does not belong to the LHS and is not going to be added.
4. It is not possible for an edge to reach a node which does not belong to the LHS and which is not going to be added: $\left\|\left(\overline{e^E} L^E\right) \odot \left(\overline{r^N}\, \overline{L^N}\right)\right\|_1 = 0$. For outgoing edges: $\left\|\left(\overline{e^E} L^E\right)^t \odot \left(\overline{r^N}\, \overline{L^N}\right)\right\|_1 = 0$. In this case, $\overline{e^E} L^E$ is a matrix with a 1 in the edges that are in the LHS and not deleted.

Thus we arrive naturally at the next proposition:

Proposition 3. *Let $p : L \to R$ be a production, if previous conditions in items 1-4 are fulfilled then $R^E = r^E \vee \left(\overline{e^E}\, L^E\right)$ and $R^N = r^N \vee \left(\overline{e^N}\, L^N\right)$ are compatible.*

which is easily proved, as we have to check that $\left\|\left(M \vee M^t\right) \odot \overline{N}\right\|_1 = 0$, with $M = r^E \vee \overline{e^E} L^E$ and $\overline{N} = \overline{r^N}\left(e^N \vee \overline{L^N}\right)$. Therefore,

$$\left(M \vee M^t\right) \odot \overline{N} = \left[\left(r^E \vee \overline{e^E} L^E\right) \vee \left(r^E \vee \overline{e^E} L^E\right)^t\right] \odot \left[\overline{r^N}\left(e^N \vee \overline{L^N}\right)\right] =$$
$$= \left[r^E \vee \overline{e^E} L^E \vee \left(r^E\right)^t \vee \left(\overline{e^E} L^E\right)^t\right] \odot \left(e^N \vee \overline{r^N}\, \overline{L^N}\right) \qquad (1)$$

Conditions in items 1-4 are taken from this identity.

For the rule in Fig. 2, it is easy to check that (R^E, R^N) are compatible, as vector $\overline{N}$ has all elements equal to zero (because e^N and $\overline{L^N}$ are zero).

Up to 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 with some common subsets. Moreover, the elements in both matrices can appear in a different order. An operation called *completion* modifies matrices (and vectors) to allow some specified operation. Suppose we want to operate with two matrices representing the edges of two graphs (a similar operation can be defined for vectors of nodes). In this way, first a common subset C of elements are identified, and it is moved up in the matrices, maintaining the order. Then, the common subset is sorted in the second matrix to obtain the same order as in the first one. Then, the elements present in the first matrix but not in the second one are added to the second one (i.e. rows and columns of zeros), sorted like in the first one. Similarly, the elements present in the second matrix but not in the first one are added to the first one (i.e. rows and columns of zeros), sorted like in the second one.

For example, if we have to operate the graph in Fig. 1 with the LHS of rule in Fig. 2, then the matrix of edges and the vector of nodes of the rule have to be enlarged. If we identify nodes and edges with the same label, we get the following result:

$$L'^{E}_{CC} = \left[\begin{array}{cccc|c} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 2 \\ 1 & 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 & 4 \end{array}\right] \; ; \; L'^{N}_{CC} = \left[\begin{array}{c|c} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 0 & 4 \end{array}\right]$$

where an additional column and row has been added to the edge matrix and an additional element has been added to the nodes vector. In this case, the matrices for the graph in Fig. 1 remain the same. Note how, if we had assumed other identification of nodes in the different graphs, the completion procedure would have produced a different result. Once the matrices and vectors of the two graphs are completed, we can define any graph transformation (i.e. any morphism on simple digraphs) as two boolean functions (for the edges matrix and for the nodes vector, which we have modelled with e and r). These functions may change arbitrarily 0's and 1's in the matrix of edges and vector of nodes (and thus we have to check compatibilty after their application).

2.2 Concatenation, Permutations and Minimal Initial Digraph

It is possible to define sequences of rules and the order in which they are to be applied.

Definition 2 (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.*

A concatenation is said to be *coherent* if actions carried out by one production do not prevent[4] the application of those coming afterwards. Fig. 3 shows more rules for the example. Messages are depicted as self-loops, which can be sent through channels. For example sequence $remove\_channel; send; message\_ready; create\_channel$ is coherent, as link $(2, 3)$ is created by the first rule ($create\_channel$), used by rule $send$ and then deleted by the last rule. We assume an identification of nodes in the different rules having the same numbers, but other combinations could be studied as well.[5]

The conditions for coherence of a concatenation of two rules $s_2 = p_2; p_1$ are:

1. The first production – p_1 – does not delete any edge used by p_2: $e_1^E L_2^E = 0$.
2. p_2 does not add any edge used, but not deleted, by p_1: $r_2^E L_1^E \overline{e_1^E} = 0$.
3. No common edges are added by both productions: $r_1^E r_2^E = 0$.

[4] Potentially, because no actual application of productions to a host graph is considered.

[5] Hence, completion is not unique – there may exist several ways to identify nodes across productions – depending on how rules are defined or the operation to be performed.

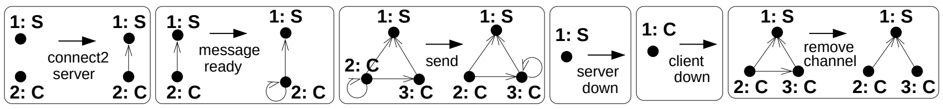


Fig. 3. Additional Rules for the Client-Server Example

The first condition is needed because if p_1 deletes one edge used by p_2, then p_2 is not applicable. The last two conditions are needed in order to obtain a simple digraph (with at most one edge in each direction between two nodes). Applying the first two identities in proposition 2, the three previous equalities can be transformed into $R_1^E\overline{e_2^E}r_2^E \vee L_2^E e_1^E\,\overline{r_1^E} = 0$ and similar for nodes.

Our objective is to obtain a closed formula to represent these conditions for the case with n productions. For this purpose, we introduce a graphical notation for boolean equations: a single arrow means $\wedge$, while a fork (more than one arrow starting in the same node) stands for $\vee$. These diagrams are useful to understand how the formulas change depending on the number of productions. As an example, the representation of coherence equations for two productions (for edges) is shown in Fig. 4(left). The figure also shows the equations for three and five productions.

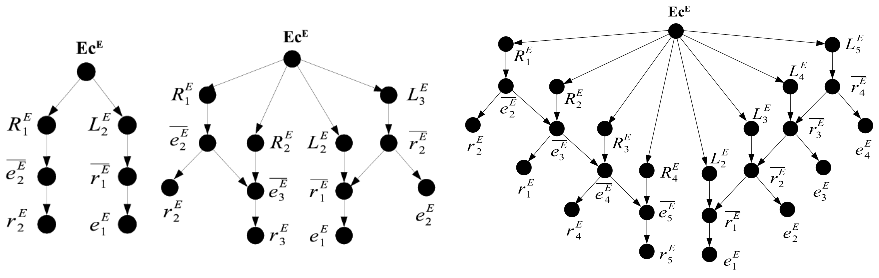


Fig. 4. Graph for Sequence of Length 2 (left), 3(middle) and 5(right).

Analysing the graphs for sequences of increasing size, we arrive at the following theorem concerning sequences of arbitrary size. The proof is not included here, it can be found at [10].

Theorem 1 (Sequence Coherence). *The concatenation $s_n = p_n; \ldots ; p_1$ is coherent if*

$$\bigvee_{i=1}^{n} \left(R_i \bigtriangledown_{i+1}^{n} \left(\overline{e_x}\, r_y\right) \vee L_i \bigtriangleup_{1}^{i-1} \left(e_y\, \overline{r_x}\right)\right) = 0 \qquad (2)$$

where

$$\bigtriangleup_{t_0}^{t_1} \left(F(x,y)\right) = \bigvee_{y=t_0}^{t_1} \left(\bigwedge_{x=y}^{t_1} \left(F(x,y)\right)\right) ; \bigtriangledown_{t_0}^{t_1} \left(G(x,y)\right) = \bigvee_{y=t_0}^{t_1} \left(\bigwedge_{x=t_0}^{y} \left(G(x,y)\right)\right)$$

E.g., sequence $s_1 = remove\_channel; send; message\_ready; create\_channel$ is coherent but $send; message\_ready; remove\_channel$ is not, because the first production ($remove\_channel$) deletes edge $(2, 3)$ needed by $send$ one step afterwards. The resulting matrix of the coherence formula has a one in such position and zeros elsewhere. In this way, the resulting matrix of the formula is useful to indicate where the potential coherence problems are. On the other hand, sequence $s_2 = remove\_channel; send;$ $create\_channel$ is coherent, but it is worth stressing that edge $(2, 2)$ needs to be supplied by the host graph, because rule $send$ needs a self loop representing a message and we know that such element is not added by any rule before $send$. Altogether, coherence allows the grammar designer to check dependencies between rules, and to realize possible conflicts, some of which can be solved if the initial graph provides enough edges and nodes. This is related to the notion of *minimal initial digraph*, which is a graph containing the necessary nodes and edges for a rule (or sequence) to be applicable.

Theorem 2 (Minimal Initial Digraph). *Given a coherent concatenation of productions $s_n = p_n; \ldots; p_1$, its minimal initial digraph is defined by: $M_n = \bigtriangledown_1^n \left(\overline{r_x} L_y\right)$.*

One graph is easily obtained which contains enough nodes and edges to execute a coherent sequence: $\bigvee_{i=1}^n L_i$. However, this graph can be made smaller, so for example, for production p_1 we only include in M_n elements which are in the LHS, but not added. In a similar way, for p_2 we include elements in its LHS if they are not added by p_2 nor p_1. Therefore, we have $M_n = (\overline{r}_1 L_1) \vee (\overline{r}_1 L_2)(\overline{r}_2 L_2) \vee \cdots \vee (\overline{r}_1 L_n) \cdots (\overline{r}_n L_n)$, which is the expanded form of $\bigtriangledown_1^n \left(\overline{r_x} L_y\right)$. Note how, we assume a given identification of nodes and edges in the different productions of the sequence, that is, a certain way of completing each matrix. The calculation of the minimal initial digaph for sequence $s_2 = remove\_channel; send; create\_channel$ is shown in Fig.5 as an example.

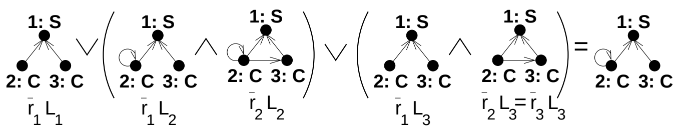


Fig. 5. Minimal Digraph for Sequence s_2.

The **image of a concatenation** $s_n = p_n; \ldots; p_1$ (please, refer to [10]) almost can be seen as a production $s_n = (r_s, e_s)$, where $r_s = \bigtriangleup_1^n \left(\overline{e_x}\, r_y\right)$ and $e_s = \bigvee_{i=1}^n e_i$, i.e.,

$$s_n (M_n) = \bigwedge_{i=1}^{n} \left(\overline{e_i} M_n\right) \vee \bigtriangleup_1^n \left(\overline{e_x}\, r_y\right) = r_s \vee \overline{e_s}\, M_n \tag{3}$$

However, in this case, it is not true that $r_s\, \overline{e_s} = r_s$, which in particular implies that it is important to *delete* elements (apply e_s) before *addition* takes place (r_s application).

The following result states conditions to keep coherence in case of permuting one production inside a sequence [10].

Theorem 3 (Production Permutations). *Consider coherent productions $t_n = p_\alpha; p_n;$ $p_{n-1}; \ldots; p_1$ and $s_n = p_n; p_{n-1}; \ldots; p_1; p_\beta$ and permutations ϕ and δ.*

1. *$\phi\left(t_n\right)$ is coherent if:* $e^E_\alpha \bigtriangledown^n_1 \left(\overline{r^E_x}\, L^E_y\right) \vee R^E_\alpha \bigtriangledown^n_1 \left(\overline{e^E_x}\, r^E_y\right) = 0.$
2. *$\delta\left(s_n\right)$ is coherent if:* $L^E_\beta \bigtriangleup^n_1 \left(\overline{r^E_x}\, e^E_y\right) \vee r^E_\beta \bigtriangleup^n_1 \left(\overline{e^E_x}\, R^E_y\right) = 0.$

where ϕ advances the last production to the front, that is, moves the left-most rule to the right $n-1$ positions in a sequence of n rules. Thus, ϕ has associated permutation $\phi = [\,1 \quad n \quad n{-}1 \ldots 3 \quad 2\,]$. In a similar way, δ delays the first production $n{-}1$ positions in a sequence of n rules, moving it to the last position. Thus, $\delta = [\,1 \quad 2\,\ldots\, n{-}1 \quad n\,]$. For sequence $t_2 = send; create\_channel; remove\_channel$, $\phi(t_2) = create\_channel; remove\_channel; send$ is coherent.

G-congruence guarantees that two coherent and compatible concatenations have the same output starting with G as minimal initial digraph. The conditions to be fulfilled are known as *Congruence Conditions* (CC). A coherent and compatible concatenation s_n and a coherent and compatible permutation of it, $\sigma\left(s_n\right)$, which besides have the same minimal initial digraph G (*G-congruent*) are *potentially sequential independent*. For advancement and delaying of productions, the congruence conditions are (see [10]):

$$CC\left(\phi, s_n\right) = L_n \nabla^{n-1}_1 \left(\overline{e_x}\, r_y\right) \vee r_n \nabla^{n-1}_1 \left(\overline{r_x}\, L_y\right) = 0 \tag{4}$$

$$CC\left(\delta, s_n\right) = L_1 \nabla^{n}_2 \left(\overline{e_x}\, r_y\right) \vee r_1 \nabla^{n}_2 \left(\overline{r_x}\, L_y\right) = 0 \tag{5}$$

For sequence $s = send; create\_channel; remove\_channel$, $CC(\phi, s) = 0$, therefore we obtain the same result by advancing $send$ twice. As s and $\phi(s)$ have the same initial digraph (the one in Fig. 5, plus edge $(2, 3)$), they are potential sequential independent. Symbol $\perp$ denotes potential sequence independence, thus we can write $send \perp (create\_channel; remove\_channel)$ in previous example. Note that it is possible to check sequential independence between a rule and a sequence, in contrast with results in the algebraic-categorical approach.

3 Match, Extended Match and Production Transformation

Matching is the operation of identifying the LHS of a rule inside a host graph. This identification is not necessarily unique, thus becoming a source of non determinism.

Definition 3 (Match) *Given a production $p : L \rightarrow R$ and a simple digraph G, any $m : L \rightarrow G$ total injective morphism is known as a match (for p in G).*

Recalling the notion of *completion*, a match can be interpreted as one of the possible ways to *complete* L in G. We do not explicitly care about types or labels in our matrices (“S” and “C” in the examples), but this can be thought as restrictions for the *completion* procedure, which cannot identify elements with different types.

Fig.6(a) displays a production p and a match m for p in G. It is possible to close the diagram, making it commutative $(m^* \circ p = p^* \circ m)$, using the pushout construction [5] on category **Pfn(Graph)** of simple digraphs and partial functions (see [9]). This categorical construction for relational graph rewiting is carried out in [9] in their Theorem 3.2 and Corollary 3.3. Proposition 3.5 in [9] gives a sufficient condition to decide if a given rewriting square like the one in Fig.6(a) can be closed.

$$\begin{tikzcd}
L \arrow[r, "p"] \arrow[d, "m"'] & R \arrow[d, dotted, "m*"] \\
G \arrow[r, dotted, "p*"'] & H
\end{tikzcd}
\qquad
\begin{tikzcd}
L \arrow[r, "p"] \arrow[d, "m"'] & R \arrow[d, dotted, "m^*"] \\
G \arrow[r, dotted, "p*"'] \arrow[d, "m_\varepsilon"'] & H \arrow[d, dotted, "m_\varepsilon^*"] \\
G \arrow[r, dotted, "\widehat{p}^*"'] & H
\end{tikzcd}
\qquad
\begin{tikzcd}
L \arrow[rr, "p"] \arrow[dr, "m"] \arrow[dd, "i_L"'] \arrow[dddr, "m_G"] & & R \arrow[dr, "m^*"] \arrow[dd, dotted, "i_R"] & \\
& G \arrow[rr, "p^*"] \arrow[dl, "i_G"'] \arrow[dd, "m_\varepsilon"] & & H \arrow[dl, dotted, "i_H"] \arrow[dd, "m_\varepsilon^*"] \\
L \times G \arrow[rr, dotted, "\widehat{p}"] \arrow[dr, "\widehat{m}"'] & & R \times H \arrow[dr, dotted, "\widehat{m}^*"] & \\
& G \arrow[rr, "\widehat{p}^*"] & & H
\end{tikzcd}$$

Fig. 6. (a) Production plus Match. (b) Neighbourhood. (c) Extended Match and Production.

Definition 4 (Direct Derivation) *Given $p : L \to R$ and $m : L \to G$ as in Fig.6(a), $d = (p, m)$ is called a direct derivation with result $H = p^* (G)$.*

If a concatenation $s_n = p_n; \ldots ; p_1$ is considered together with the set of matchings $m_n = \{m_1, \ldots , m_n\}$, then $d_n = (s_n, m_n)$ is a **derivation**.

When applying a rule to a host graph, the main problem to concentrate on is that of so-called *dangling edges*, which is differently addressed in SPO and DPO. In DPO, if an edge comes to be dangling then the rule is not applicable (for that match), while SPO allows the production to be applied, deleting any dangling edge. In this paper we propose an SPO-like behaviour. Fig.6(b) shows our strategy to handle dangling edges:

1. Morphism m shall identify rule's left hand side in the host graph.
2. A neighbourhood of $m(L) \subseteq G$ covering all relevant extra elements is selected (performed by m_ε[6]), taking into account all dangling edges not considered by match m with their corresponding source and target nodes.
3. Finally, p is enlarged (through operator T_ε, see definition below) erasing any otherwise dangling edge.

Definition 5 (Extended Match) *Given a production $p : L \to R$, a host graph G and a match $m : L \to G$, the extended match $\widehat{m} : L \times G \to G$ is a morphism whose image is $m(L) \bigcup \varepsilon$, where ε is the set of dangling edges and their source and target nodes.*

Coproduct (see Fig.6(c)) is used for coupling L and G, being the first embedded into the second by morphism m. We use the notation $\underline{L} \overset{def}{=} m_G(L) \overset{def}{=} (m_\varepsilon \circ m)(L)$ i.e., extended digraphs are underlined and defined by composing m and m_ε.

Example.□Consider the digraph L, the host graph G and the morphism match depicted on the left side of Fig. 7. On the top right side in the same figure, $m(L)$ is drawn, and $m_G(L)$ on the bottom right side. Nodes 2 and 3 and edges $(2, 1)$, $(2, 3)$ and $(2, 2)$ have been added to $m_G(L)$. The edges would become dangling in the image "graph" of G by p, $p(G)$. Note how this composition is possible, as m and m_ε are functions between boolean matrices which have been completed. ■

Once we are able to complete the rule's LHS, we have to do the same for the rest of the rule. To this end we define an operator $T_\varepsilon : \mathfrak{G} \to \mathfrak{G}'$, where $\mathfrak{G}$ is the original

[6] Recall that morphisms are functions on boolean matrices and vectors.

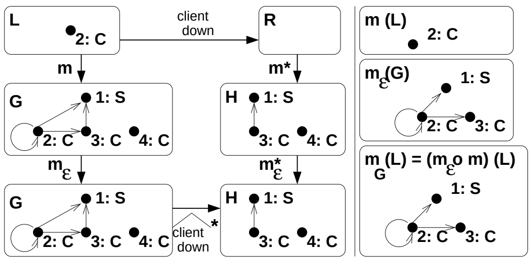


Fig. 7. Matching and Extended Match.

grammar and $\mathfrak{G}'$ is the grammar transformed once T_ε has modified the production. The notation that we use from now on is borrowed from functional analysis [1]. Bringing this notation to graph grammar rules, a rule is written as $R = \langle L, p \rangle$ (separating the static and dynamic parts of the production) while the grammar rule transformation including matchings is: $\underline{R} = \langle m_G(L), T_\varepsilon p \rangle$.

Proposition 4. *With notation as above, production p can be extended to consider any dangling edge, $\underline{R} = \langle m_G(L), T_\varepsilon p \rangle$.*

Proof
□What we do is to split the identity operator in such a way that any problematic element is taken into account (erased) by the production. In some sense, we first add elements to p's LHS and afterwards enlarge p to erase them. Otherwise stated, $m_G^* = T_\varepsilon^{-1}$ and $T_\varepsilon^* = m_G^{-1}$, so in fact we have $R = \langle L, p \rangle = \left\langle L, \left(T_\varepsilon^{-1} \circ T_\varepsilon\right) p \right\rangle = \langle m_G(L), T_\varepsilon(p) \rangle = \underline{R}$. The equality $\underline{R} = R$ is valid strictly for edges. ■

The effect of considering a match can be interpreted as a new production concatenated to the original production. Let $p_\varepsilon \overset{def}{=} T_\varepsilon^*$,

$$\begin{aligned} \underline{R} = \langle m_G(L), T_\varepsilon(p) \rangle &= \langle T_\varepsilon^*(m_G(L)), (p) \rangle = \\ &= p\left(T_\varepsilon^*(m_G(L))\right) = p\,;\, p_\varepsilon\,;\, m_G(L) = p\,;\, p_\varepsilon(\underline{L}) \end{aligned} \tag{6}$$

Considering the match can be interpreted as a temporary modification of the grammar, so it can be said that the grammar modifies the host graph and – temporarily – the host graph interacts with the grammar.

If we think of m_G and T_ε^* as productions respectively applied to L and $m_G(L)$, it is necessary to specify their erasing and addition matrices. To this end, we introduce matrix ε, with elements in row i and column i equal to one if node i is to be erased by p, and zero otherwise (see definition 5). This matrix considers any potential dangling edge.

For m_G we have that $\underline{e}^N = \underline{e}^E = 0$, and $\underline{r} = \underline{L}\,\overline{L}$ (for both nodes and edges), as the production has to add the elements in $\underline{L}$ that are not present in L. Let $p_\varepsilon = \left(e^E_{T_\varepsilon}, r^E_{T_\varepsilon}; e^N_{T_\varepsilon}, r^N_{T_\varepsilon}\right)$, then $e^N_{T_\varepsilon} = r^E_{T_\varepsilon} = r^N_{T_\varepsilon} = 0$ and $e^E_{T_\varepsilon} = \varepsilon \wedge \underline{L}^E$.

Example.□Consider rules depicted in Fig. 8, in which $server\_down$ is applied to model a server failure. We have

$$e^E = r^E = L^E = \left[\,0 \middle|\, 1\,\right] e^N = \left[\,1 \middle|\, 1\,\right]\,;\; r^N = \left[\,0 \middle|\, 1\,\right]\,;\; L^N = \left[\,1 \middle|\, 1\,\right]\,;\; R^E = R^N = \emptyset$$

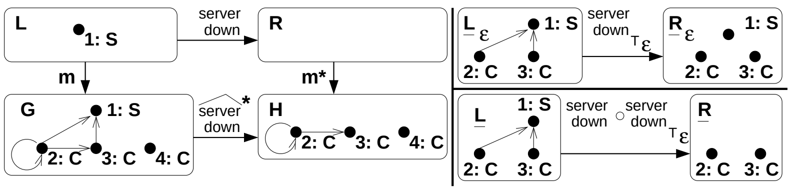


Fig. 8. Full Production and Application.

Once m_G and operator T_ε have been applied, the resulting matrices are

$$\underline{r}^E = \left[\begin{array}{ccc|c} 0&0&0&1\\ 1&0&0&2\\ 1&0&0&3 \end{array}\right] ; \ \underline{L}^E = \left[\begin{array}{ccc|c} 0&0&0&1\\ 1&0&0&2\\ 1&0&0&3 \end{array}\right] ; \ \underline{R}^E = \left[\begin{array}{cc|c} 0&0&2\\ 0&0&3 \end{array}\right] ; \ e^E_{T_\varepsilon} = \left[\begin{array}{ccc|c} 0&0&0&1\\ 1&0&0&2\\ 1&0&0&3 \end{array}\right]$$

Matrix $\underline{r}^E$, besides edges added by the production, specifies those to be added by m_G to the LHS in order to consider any potential dangling edge (in this case $(2,1)$ and $(3,1)$). As neither m_G nor production $server\_down$ delete any element, $\underline{e}^E = 0$. Finally, p_ε removes all potential dangling edges (check out matrix $e^E_{T_\varepsilon}$) but it does not add any, so $r^E_{T_\varepsilon} = 0$. Vectors for nodes have been omitted.■

Let $T^*_\varepsilon = \left(T^{*N}_\varepsilon, T^{*E}_\varepsilon\right)$ be the adjoint operator of T_ε. Define e^E_ε and r^E_ε respectively as the erasing and addition matrices of $T_\varepsilon(p)$. It is clear that $r^E_\varepsilon = \underline{r}^E = r^E$ and $e^E_\varepsilon = e^E \vee \varepsilon\, \underline{L}^E$, so

$$\begin{aligned} \underline{R}^E = \left\langle\, \underline{L}^E, T_\varepsilon\,(p)\,\right\rangle = r^E_\varepsilon \vee \overline{e^E_\varepsilon}\, \underline{L}^E = r^E \vee \overline{\left(e^E \vee \varepsilon\, \underline{L}^E\right)}\, \underline{L}^E = \\ = r^E \vee \left(\overline{\varepsilon} \vee \overline{\underline{L}^E}\right) \overline{e^E} \underline{L}^E = r^E \vee \overline{e^E}\; \overline{\varepsilon}\; \underline{L}^E \end{aligned}$$

The previous identities show that $\underline{R}^E = \left\langle \underline{L}^E, T^E_\varepsilon\left(p^E\right)\right\rangle = \left\langle \overline{\varepsilon}\, \underline{L}^E, p^E \right\rangle$, which proves that $T^*_\varepsilon = \left(T^{*N}_\varepsilon, T^{*E}_\varepsilon\right) = (id, \overline{\varepsilon})$.

Summarizing, when a given match m is considered for a production p, the production itself is first modified in order to consider all potential dangling edges. m is automatically transformed into a match which is free from any dangling element and, in a second step, a pre-production p_ε is appended to form the concatenation $\widehat{p}^* = p^*\,; p^*_\varepsilon$

4 Revision and Extension of Basic Concepts

In this section we brush over all concepts and theorems introduced in section 2, completing them by considering matchings.

Let $s_n = p_n; \ldots ; p_1$ be a concatenation. As there is a match for every production in the sequence, it is eventually transformed into $s^*_n = p_n\,; p_{\varepsilon,n}; \ldots ; p_1; p_{\varepsilon,1}$. Fig.9 displays the corresponding derivation. For **compatibility**, the main difference when considering matchings is that the sequence is increased in the number of productions so it shall be necessary to check more conditions.

$$\begin{tikzcd}
\underline{L_1} \arrow[r, "p_{\varepsilon,1}"] \arrow[d, "m_{G,1}"] & L_1 \arrow[dr] & L_1 \arrow[r, "p_1"] \arrow[d, "m_1"] & R_1 \arrow[dr] & \underline{L_2} \arrow[r, "p_{\varepsilon,2}"] \arrow[d, "m_{G,2}"] & L_2 \arrow[dr] & L_2 \arrow[r, dotted, "p_2"] \arrow[d, "m_2"] & \cdots \\
G \arrow[rr, "p^*_{\varepsilon,1}"'] & & G_{\varepsilon,1} \arrow[rr, "p^*_1"'] & & G_1 \arrow[rr, "p^*_{\varepsilon,2}"'] & & G_{\varepsilon,2} \arrow[r, dotted, "p^*_2"'] & \cdots
\end{tikzcd}$$

Fig. 9. Productions and ε-productions in a Concatenation.

4.1 Initial Digraph Set

Concerning the *minimal initial digraph*, one may have different ways of completing the rule matrices, depending on the matches. Therefore, we no longer have a unique initial digraph, but a set.

Definition 6 (Initial Digraph Set) *Given s_n a sequence, its associated initial digraph set $\mathfrak{M}\left(s_n\right)$ is the set of simple digraphs M_i such that*

1. *M_i has enough nodes and edges for every production of the concatenation to be applied in the specified order, and*
2. *M_i has no proper subgraph with previous property*

$\forall M_i \in \mathfrak{M}\left(s_n\right)$. *Every element $M_i \in \mathfrak{M}\left(s_n\right)$ is said to be an initial digraph for s_n.*

It is easy to see that $\mathfrak{M}\left(s_n\right) \neq \emptyset$, $\forall s_n$ finite sequence of productions. The initial digraph set contains all graphs that can potentially be identified by matches in concrete host graphs. In section 2.1, coherence was used in an absolute way but now, due to matching, coherence is a property depending on the given initial digraph. Hence, we now say that s_n is coherent with respect to initial digraph M_i.

For the initial digraph set, we can define the *maximal initial digraph* as the element $M_n \in \mathfrak{M}\left(s_n\right)$ which considers all nodes in p_i to be different. This element is unique up to isomorphism, and corresponds to considering the parallel application of every production in the sequence. In a similar way, $M_i \in \mathfrak{M}\left(s_n\right)$ in which all possible identifications are performed are known as *minimal initial digraphs*, which in general are not unique. As an example, left of Fig. 10 shows the minimal digraph set for sequence $s_2 = remove\_channel; remove\_channel$, which is not coherent, as the link between two clients is deleted twice. In this way, the initial digraphs should provide two links. It is possible to provide some structure $\mathfrak{T}\left(s_n\right)$ to set $\mathfrak{M}\left(s_n\right)$ (see the right of Fig. 10). Every node in $\mathfrak{T}$ represents an element of $\mathfrak{M}$, and a directed edge from one node to another stands for one operation of identification between corresponding nodes in LHS and RHS of productions of the sequence s_n. Node M_7 is the maximal initial digraph, as it only has outgoing edges. The structure $\mathfrak{T}$ is known as graph-structured stack, in our case with single root node.

4.2 Coherence

Coherence formulas do not change, except that now there are conditions for all ε-productions. When considering the match, coherence is similar to conflict detection in

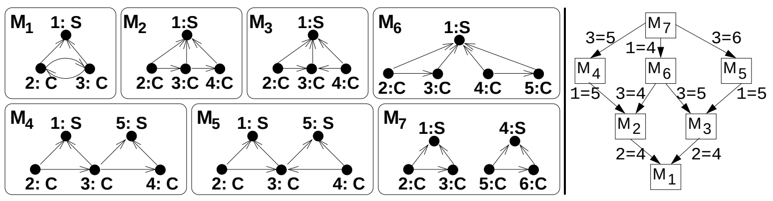


Fig. 10. Initial Digraph Set for $s_2 = remove\_channel; remove\_channel$.

critical pairs [5] [6], where an important issue is efficiency [8]. We believe our approach is a contribution in improving the efficiency in finding this kind of conflicts.

The functional notation introduced so far can be used to re-enunciate Theorem 1 for coherence, deriving conditions which resemble those of perpendicular vectors and kernel of a function. Let $q_{L_i} = \triangle_1^{n-1} \left(\overline{r_x}\, e_y\right)$ and $q_{R_i} = \nabla_{i+1}^{n} \left(\overline{e_x}\, r_y\right)$, then $s_n = p_n; \ldots; p_1$ is coherent if $\langle L_i, q_{L_i} \rangle = \langle R_i, q_{R_i} \rangle = 0$.

In addition, when the host graph is not considered, if nodes are identified across rules, it can be the case that some dangling edge appears in the concatenation. For example, given $p_2; p_1$, suppose that rule p_1 uses but does not delete edge $(4, 1)$, that rule p_2 specifies the deletion of node 1 and that we have identified both nodes 1. It is mandatory to add one ε-production $p_{\varepsilon,2}$ to the grammar, which conceptually is of a different nature than those previously discussed. The latter dangling edges appear in the context where the rule is applied, but not in other rules. We have an unavoidable problem of coherence between p_1 and $p_{\varepsilon,2}$ if we wanted to advance the application of $p_{\varepsilon,2}$ to p_1. Hence, we split the set of edges deleted by ε-productions into two disjoint classes:

– **External**. Any edge not appearing explicitly in the grammar rules, i.e., edges of the host graph “in the surroundings” of the actual initial digraph. Examples are edges $(2, 1)$ and $(3, 1)$ in Fig.8.
– **Internal**. Any edge used or appended by a previous production in the concatenation. One example is the previously mentioned edge $(4, 1)$.

ε-productions can be classified accordingly in **internal ε-productions** if any of its edges is internal and **external ε-production** otherwise. External ε-productions cannot be considered during rule specification which, in turn, may spoil coherence, compatibility, etc. One way to handle this problem is to check the conditions under which all ε-productions can be advanced to the front of the sequence. Given a host graph G in which s_n – coherent and compatible – is to be applied, and assuming a match which identifies s_n's actual initial digraph (M_n) in G, we check whether for some $\widehat{m}$ and $\widehat{T_\varepsilon}$, which respectively represent all changes to be done to M_n and all modifications to s_n, it is correct to write $H_n = \left\langle \widehat{m}\left(M_n\right), \widehat{T_\varepsilon}\left(s_n\right) \right\rangle$, where H_n would be the piece of the final state graph H corresponding to the image of M_n.

Example.□Let $s_2 = p_2; p_1$ be a coherent and compatible concatenation. Using operators we can write $H = \langle m_{G,2}\left(\langle m_{G,1}\left(M_2\right), T_{\varepsilon,1}\left(p_1\right)\rangle\right), T_{\varepsilon,2}\left(p_2\right)\rangle$, which is equivalent to $H = p_2;\, p_{\varepsilon,2}; p_1;\, p_{\varepsilon,1}\left(\underline{M_2}\right)$, with actual initial digraph twice modified $\underline{M_2} = m_{G,2}\left(m_{G,1}\left(M_2\right)\right) = \left(m_{G,2} \circ m_{G,1}\right)\left(M_2\right)$.■

Definition 7 (Exact Derivation) *Let $d_n = (s_n, m_n)$ be a derivation with actual initial digraph M_n, concatenation $s_n = p_n; \ldots; p_1$, matches $m_n = \{m_{G,1}, \ldots, m_{G,n}\}$ and ε-productions $\{p_{\varepsilon,1}, \ldots, p_{\varepsilon,n}\}$. It is an* exact *derivation if there exist $\widehat{m}$ and $\widehat{T_\varepsilon}$ such that $H_n = d_n(M_n) = \left\langle \widehat{m}(M_n), \widehat{T_\varepsilon}(s_n) \right\rangle$.*

Previous equation might be satisfied if once all matches are calculated, the following identity holds: $p_n; p_{\varepsilon,n}; \ldots; p_1; p_{\varepsilon,1} = p_n; \ldots; p_1; p_{\varepsilon,n}; \ldots; p_{\varepsilon,1}$. Equation (3) allows us to consider a concatenation almost as a production, justifying operators $\widehat{T_\varepsilon}$ and $\widehat{m}$ and our abuse of the notation (recall that brakets apply to productions and not to sequences).

Proposition 5. *With notation as before, if $p_{\varepsilon,j} \perp (p_{j-1}; \ldots; p_1)$, $\forall j$, then d_n is exact.*

Proof
□Operator $\widehat{T_\varepsilon}$ modifies the sequence adding a unique ε-production, the composition [7] of all ε-productions $p_{\varepsilon,i}$. To see this, if one edge is to dangle, it should be eliminated by the corresponding ε-production, so no other ε-production deletes it unless it is added by a subsequent production. But by hypothesis there is sequential independence of every $p_{\varepsilon,j}$ with respect to all preceeding productions and hence $p_{\varepsilon,j}$ does not delete any edge used by $p_{j-1}, \ldots, p_1$. In particular no edge added by any of these productions is erased.

In definition 7, $\widehat{m}$ is the extension of the match m which identifies the actual initial digraph in the host graph, so it adds to $m(M_n)$ all nodes and edges to distance one to nodes that are going to be erased. A symmetrical reasoning to that of $\widehat{T_\varepsilon}$ shows that $\widehat{m}$ is the composition of all $m_{G,i}$.■

With definition 7 and proposition 5 it is feasible to get a concatenation where all ε-productions are applied first, and all grammar rules afterwards, recovering the original concatenation. Despite some obvious advantages, all dangling edges are deleted at the beginning, which may be counterintuitive or even undesired. For example, if the deletion of a particular edge is used for synchronization purposes. The following corollary states that exactness can only be ruined by internal ε-productions. Let s_n be a sequence to be applied to a host graph G and $M_k \in \mathfrak{M}(s_n)$.

Corollary 1. *With notation as above, assume there exists at least one match in G for M_k that does not add any internal ε-production. Then, d_n is exact.*

Proof (sketch)
□All potential dangling elements are edges surrounding the actual initial digraph. It is thus possible to adapt the part of the host graph modified by the sequence at the beginning, so applying proposition 5 we get exacteness.■

5 Conclusions and Future Work

In this paper we have presented a new approach to simple digraph transformation based on an algebra of boolean matrices. We have shown some results (coherence, minimal

[7] Given a sequence of productions, their composition is one production which performs the same operations, see [10] for the formal definition.

initial digraphs, permutation, G-congruence) that can be calculated on the graph transformation system, independent of the host graph. We have introduced the match, and how to handle dangling edges by generating ε-productions which are applied previous to the original rule in order to delete dangling edges.

We believe that the main difference of our approach with respect to others is that we use boolean operators to represent graph manipulations. Other approaches such as DPO and SPO use a categorical representation of the operations, which, on the one hand makes the approach more general, but on the other, makes bigger the gap between specification and implementation on tools. In addition, we believe that concepts like initial digraph, coherence, arbitrary sequences of finite length are easier to express and study in our framework than using category theory. Concerning additional related work, the relational approach of [9] uses also exclusively a categorical approach for operations. Other approaches such as logic-based [12], algebraic-logic [2], relation-algebraic [7] are more distant from ours.

With respect to future work, we are working on application conditions, studying the structure of $\mathfrak{M}(s_n)$, bringing to our framework techinques from Petri nets, considering more general types of graphs and implementing the current concepts in a tool.

Acknowledgements: This work has been sponsored by the Spanish Ministry of Science and Education, project TSI2005-08225-C07-06. The authors would like to thank the referees for their useful comments.

References

1. Braket notation intro: `http://en.wikipedia.org/wiki/Bra-ket_notation`.
2. Courcelle, B. 1990. *Graph Rewriting: An Algebraic and Logic Approach* Handbook of Theoretical Computer Science, Vol. B. pp.: 193-242.
3. 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. LNCS 73.
4. 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 [11] Vol.1, pp.: 247-312.
5. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G. 2006. *Fundamentals of Algebraic Graph Transformation*. Springer.
6. Heckel, R., Küster, J. M., Taentzer, G. 2002. *Confluence of Typed Attributed Graph Transformation Systems*. Proc. ICGT'2002. LNCS 2505, pp.: 161-176. Springer.
7. Kahl, W. 2002. *A Relational Algebraic Approach to Graph Structure Transformation* Tech.Rep. 2002-03. Universität der Bundeswehr München.
8. Lambers, L., Ehrig, H., Orejas, F. 2006. *Efficient Conflict Detection in Graph Transformation Systems by Essential Critical Pairs*. Proc. GT-VMT'06, to appear in ENTCS (Elsevier).
9. Mizoguchi, Y., Kuwahara, Y. 1995. Relational Graph Rewritings. Theoretical Computer Science, Vol 141, pp. 311-328.
10. Pérez Velasco, P. P., de Lara, J. 2006. *Towards a New Algebraic Approach to Graph Transformation: Long Version*. Tech. Rep. of the School of Comp. Sci., Univ. Autónoma Madrid. `http://www.ii.uam.es/~jlara/investigacion/techrep_03_06.pdf`.
11. Rozenberg, G. (managing ed.) 1999. *Handbook of Graph Grammars and Computing by Graph Transformation. Vol.1 (Foundations), Vol.2(Applications, Languages and Tools), Vol.3., (Concurrency, Parallelism and Distribution).* World Scientific.
12. Schürr, A. *Programmed Graph Replacement Systems.* In [11], Vol.1, pp.: 479 - 546.