Electronic Communications of the EASST
Volume 2 (2006)

Proceedings of the Workshop on Petri Nets and Graph Transformation (PNGT 2006)

Petri Nets and Matrix Graph Grammars: Reachability.

Pedro Pablo Pérez Velasco, Juan de Lara

16 pages

Guest Editors: Paolo Baldan, Hartmut Ehrig, Julia Padberg, Grzegorz Rozenberg
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Petri Nets and Matrix Graph Grammars: Reachability.

Pedro Pablo Pérez Velasco[1], **Juan de Lara**[1]

[1] (pedro.perez, jdelara)@uam.es
Escuela Politécnica Superior
Universidad Autónoma de Madrid (Spain)

Abstract: This paper attempts to contribute in two directions. First, concepts and results of our Matrix Graph Grammars approach [VL06b, VL06a] such as coherence and minimal initial digraph are applied to Petri nets, especially to reachability criteria and to the state equation. Second, the state equation and related Petri nets techniques for reachability are generalized to cover a wider class of graph grammars.

Keywords: Graph Transformation, Petri Nets, Reachability, Boolean Matrix Algebra.

1 Introduction

In this paper analysis techniques from Matrix Graph Grammars (MGGs) [VL06a, VL06b] are applied to Petri nets [Mur89] and vice-versa. In MGGs, simple digraphs are represented using a boolean matrix for edges and a boolean vector for nodes. Rules are also represented with matrices and vectors, specifying the left hand side graph together with the elements that should be added and removed. Therefore graph rewriting can be represented using boolean operations only. We have developed some analysis techniques for MGGs, for example, to check whether a sequence of productions is applicable a priory (assuming a certain identification of nodes and edges between the rules), which we call *coherence*; to find if a permutation of a coherent sequence remains coherent; to verify if two sequence permutations yield the same result; or to calculate the minimal graph necessary in order to be able to apply a sequence (called *minimal initial digraph*).

The main motivation of this work is to take advantage of the similar basis of MGG and the algebraic view of Petri nets. In the first part of the work, some of the MGG concepts we have developed are applied to Petri nets. In particular it is possible to investigate for example which is the minimum marking that enables the firing of a certain transition sequence, and express the reachability problem in MGG terms (using coherence and the minimal initial digraph). On the other hand, by using tensor algebra the state equation of Petri nets has been extended for graph grammars (i.e. by replacing the incidence matrix by an incidence tensor). Both the cases of DPO-like and SPO-like graph grammars have been considered.

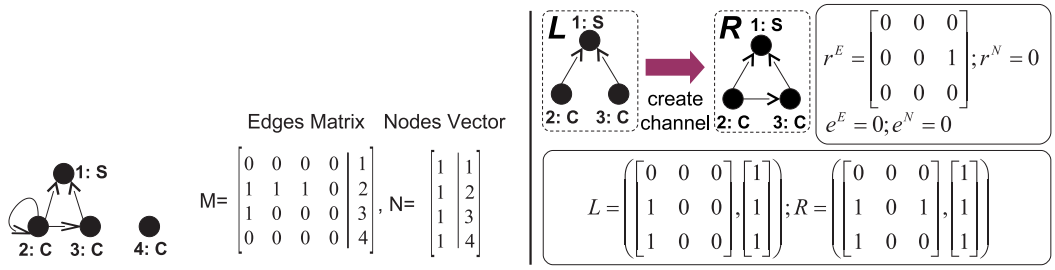
The paper is organized as follows. Section 2 gives a brief introduction to MGGs. Section 3 introduces some algebraic analysis techniques for Petri nets, in particular the state equation. In this section we discuss why the equation gives a necessary (but not sufficient) condition for reachability. Section 4 applies MGG techniques to the analysis of Petri nets. Section 5 extends the state equation for DPO-like graph grammars. Section 6 deals with the case of SPO-like graph

grammars. Section 7 compares with related work, and finally section 8 ends with the conclusions and future work.

2 Matrix Graph Grammars

In this section we review the MGG concepts which are relevant to Petri nets reachability results. For an extensive presentation, the reader is referred to [VL06a, VL06b]. The proof of the theorems can be found in [VL06b].

In our approach, we work with simple digraphs, which can be represented as a tuple (M,N) with M a boolean matrix for edges and N a boolean vector for nodes. The latter is necessary as in the rewriting nodes can be added and deleted. Figure 1(a) shows an example of a graph representing a network of three clients and a server, where messages are depicted as self-loops. Note how we can check for well-formedness of graphs (i.e. no dangling edges) by verifying that $\left\|(M \vee M^t) \odot \overline{N}\right\|_1 = 0$, where $\odot$ is the boolean matrix product (like the regular matrix product, but with **and** and **or** instead of mutiplication and addition), and $\|\cdot\|_1$ being an operation that results in the **or** of all the components of the vector. We call this property *compatibility*.



(a) (b)

Figure 1: (a) Simple Digraph Example. (b) "Localsend" Rule.

In our framework we can assign a type to each node by a function from the set of nodes V to a set of types T, $type\colon V \to T$. In Figure 1(a), types are represented as an additional column in the matrices. For edges we use the types of their source and target nodes.

A production, or grammar rule, $p : L \to R$ is a partial injective morphism of simple digraphs. Using a *static formulation*, we can represent a rule by two boolean matrices and two vectors $p = \left(L^E, R^E; L^N, R^N\right)$, (where E stands for edges and N for nodes) to characterize the left and right hand side simple digraphs (LHS and RHS). Production p is the morphism which identifies nodes (resp., edges) on the LHS with nodes (resp., edges) on the RHS. The main actions that can be performed by a rule are deletion and addition of elements. Therefore using a *dynamic formulation* a rule can be represented by $p = \left((L^E, L^N); e^E, r^E; e^N, r^N\right)$, where e^E and e^N are the deletion boolean matrix and vector, while r^E and r^N are the addition boolean matrix and vector. The output of rule p can be calculated by $R = r \vee \overline{e}L$, where the formula applies both to nodes and edges. Superindices E and N shall be omitted if the formula applies to both cases. Moreover, we usually omit the $\wedge$ (**and**) symbol. Figure 1(b) shows a rule and its associated matrices.

In order to operate graphs of different sizes, an operation called **completion** adds extra rows

and columns with zero elements and rearranges rows and columns so that the identified edges and nodes of the two graphs match.

Given a collection 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. Note that we may have the same production in different places of the sequence. A concatenation is said to be **coherent** if actions carried out by one production do not prevent the application of those coming afterwards. Note that we assume a certain identification of nodes and edges between productions (i.e., the matrices have been completed in some way, which assumes an overlapping of the rules in the matching in the host graph). Therefore, coherence is calculated with respect to the given identification.

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

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

where

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

Coherence allows the grammar designer to check dependencies between rules, and to realize possible conflicts, some of which can be solved if the initial host 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 sequence to be applicable.

Theorem 2 (Minimal Initial Digraph) *For a coherent concatenation of productions $s_n = p_n;\ldots;p_1$, its minimal initial digraph is given by the equation*

$$M_n = \bigtriangledown_{1}^{n}(\overline{r_x} L_y) \tag{2}$$

As in previous theorem, assume we have the coherent concatenation s_n with minimal initial digraph M_n, then its image is given by:

$$s_n(M_n) = \bigwedge_{i=1}^{n}(\overline{e_i} M_n) \vee \bigtriangleup_{1}^{n}(\overline{e_x}\, r_y) \tag{3}$$

see [VL06b] for a proof. To continue our analysis of finite sequences of productions, another useful operation is **composition** [VL06b]. The main difference between concatenation and composition is the generation of intermediate states in the former. If a concatenation is coherent, then its composition can be defined. Next, we state the conditions under which a permutation of a coherent sequence remains coherent. In particular, we focus on advancement of the last production to the front and vice-versa.

Theorem 3 (Production Advance and Delay) *Consider coherent sequences $t_n = p_\alpha; p_n; p_{n-1};\ldots;p_2;p_1$ and $s_n = p_n; p_{n-1};\ldots;p_2;p_1;p_\beta$ and permutations ϕ_n and δ_n.*

1. $\phi_{n+1}(t_n)$ *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. $\delta_{n+1}(s_n)$ *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_n advances (i.e. moves to the right inside the concatenation) one production $n-1$ positions, i.e., has associated permutation $\phi_n = [1 \quad n \quad n-1 \ldots 3 \quad 2]$. This is a notation for permutation cycles that means that rule 1 (the left-most one) is sent to position n, then rule in position n is moved to position $n-1$, and similarly until rule 3, which is moved to position 2, and this one to position 1. Operator δ_n delays one production $n-1$ positions, i.e., has $\delta_n = [1 \quad 2 \ldots n-1 \quad n]$ as associated permutation cycle (i.e. each rule is moved to the right, and rule n to position 1).

G-congruence guarantees that two coherent concatenations have the same minimal initial digraph G. The conditions to be fulfilled are known in [VL06b] as *Congruence Conditions* (CC). The key point is that a coherent concatenation s_n and a coherent permutation of it, $\sigma(s_n)$, which besides have the same minimal initial digraph G (i.e., which are *G-congruent*) are sequential independent. Next theorem present the congruence conditions for advancement and delay of productions.

Theorem 4 (G-congruence) *Given sequence s_n, the congruence conditions for rule advance (ϕ_{n-1}) and delay (δ_{n-1}) are given by:*

$$CC_n(\phi_{n-1}, s_n) = L_n \nabla^{n-1}_1 (\overline{e_x} r_y) \vee r_n \nabla^{n-1}_1 (\overline{r_x} L_y) = 0 \quad (4)$$

$$CC_n(\delta_{n-1}, s_n) = L_1 \nabla^n_2 (\overline{e_x} r_y) \vee r_1 \nabla^n_2 (\overline{r_x} L_y) = 0 \quad (5)$$

3 Algebraic Techniques for Petri Nets: The State Equation

Algebraic techniques for Petri nets are based on the representation of the net with an incidence matrix A in which columns are transitions (element A^i_j is the number of tokens that transition i removes – negative – or adds – positive – to place j). One of the problems that can be analyzed using algebraic techniques is **reachability**. Given an initial marking M_0 and a final marking M_d, a necessary condition to reach M_d from M_0 is to find a solution x for the equation $M_d = M_0 + Ax$, which can be rewritten as a linear system

$$M = Ax \quad (6)$$

Solution x – known as Parikh vector – specifies the number of times that each transition should be fired, but not the firing order. Identity (6) is known as the *state equation* [Mur89].

The state equation introduces a matrix, which conceptually can be thought of as associating a vector space to the dynamic behaviour of the Petri net. It is interesting to graphically interpret the operations involved in linear combinations: addition and multiplication by scalars, as depicted in Figure 2. The addition of two transitions is again a transition $t_k = t_i + t_j$ for which input places are the addition of input places of every transition and the same for output places. If a place

appears as input and output place in t_k, then it can be removed. Multiplication by -1 inverts the transition, i.e., input places become output places and viceversa, which in some sense is equivalent to disapplying the transition.

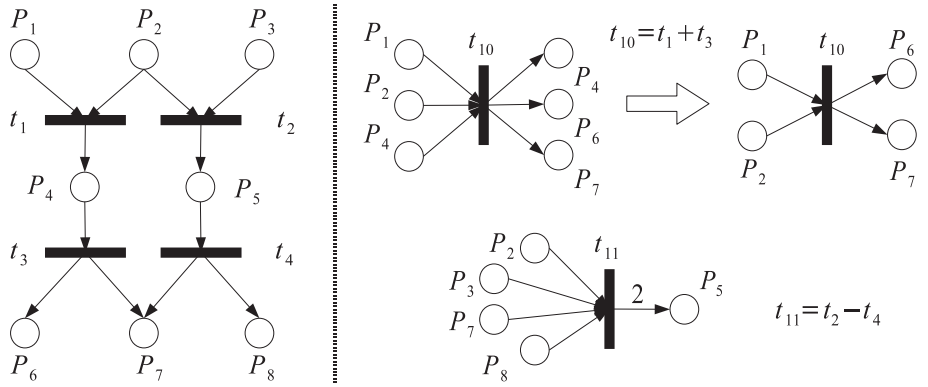


Figure 2: Linear Combinations in the Context of Petri Nets.

One important issue is that of notation. Linear algebra uses an additive notation (addition and substraction) which is normally employed when an abelian structure is under consideration. For non-commutative structures, such as permutation groups, the multiplicative notation (composition and inverses) is preferred. The basic operation with productions is the definition of sequences (concatenation), for which historically a multiplicative notation has been chosen, but substituting composition "$\circ$" by the concatenation ";" operation.[1]

From a conceptual point of view, we are interested in relating linear combinations and sequences of productions.[2] Note that, due to commutativity, linear combinations do not have an associated notion of ordering, e.g., linear combination $PV_1 = p_1 + 2p_2 + p_3$ coming from Parikh vector $[1,2,1]$ can for example represent sequences $p_1; p_2; p_3; p_2$ or $p_2; p_2; p_3; p_1$, which can be quite different. The fundamental concept that deals with commutativity is precisely sequential independence.

Following this reasoning, we can find the problem that makes the state equation a necessary but not a sufficient condition: some transition can temporarily owe some tokens to the net. The Parikh vector specifies a linear combination of transitions and thus, negatives are temporarily allowed (substraction).

Proposition 1 *Suffiency of the state equation can only be ruined by transitions firing without being enabled (i.e., temporarily borrowing tokens from the Petri net).*

This proposition does not provide any criteria based on the topology of the Petri net, as theorems 16, 17, 18 and corollaries 2 and 3 in [Mur89], but contains the essential idea in their proofs: the hypothesis in previously mentioned theorems guarantee that cycles in the Petri net will not

[1] This is the reason why [VL06b] introduces ";" to be read from right to left, contrary to the literature. Composition "$\circ$" has the effect of *simultaneous* application, which is different to sequential application. This is important for example to differentiate sequential and parallel application.

[2] Linear combinations are the building blocks of vector spaces, and the structure to be kept by matrix application.

ruin coherence.

4 Using Matrix Graph Grammars Techniques for Petri Nets

Given a Petri net, we shall consider it as the initial host graph in our matrix graph grammar. One production is associated to every transition in which places and tokens are nodes and there is an arrow joining each token to its place. In fact, we represent places for illustrative purposes only, as they are not strictly necessary, including tokens alone is enough. Figure 3 shows an example in which each production corresponds to a transition. The firing of a transition corresponds to the application of a rule.

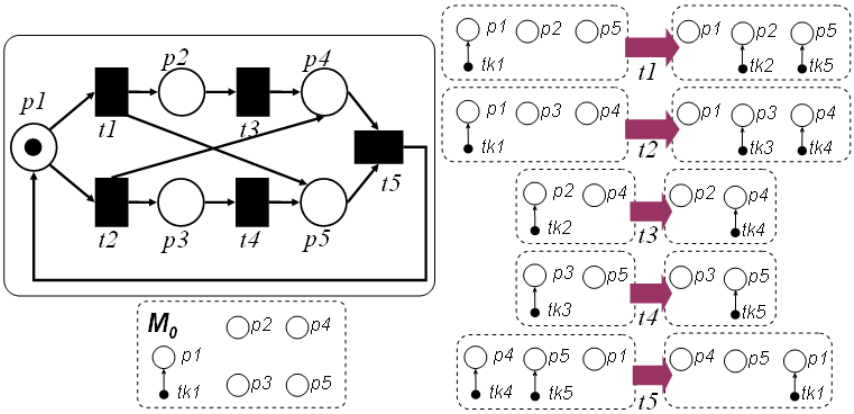


Figure 3: Petri Net with Related Production set.

Thus, Petri nets can be considered a proper subset of graph grammars with two important properties:

1. There are no dangling edges when applying productions (firing transitions).

2. Every production can be applied only in one part of the host graph.

Properties (1) and (2) somehow allow us to safely "ignore" matchings as introduced in [VL06a]. In addition, we consider Petri nets with no self-loops (i.e., so called *pure Petri nets*), that is, one production either adds or deletes nodes of a concrete type, but there is never a simultaneous addition and deletion of nodes of the same type. This agrees with the expected behaviour of MGG productions with respect to nodes (which is the behaviour of edges as well, see [VL06b]) and will be kept throughout the present work, mainly because rules in SPO-like grammars are adapted depending on whether a given production deletes nodes or not (refer to section 6). Hence, it is necessary that elements in matrices are not relative integers, i.e., a number four must mean that production x adds four nodes of type t and not that x adds four nodes more than it deletes of type t. If we had one such production p, a possible way to proceed is to split p into two, one owning the addition actions, p_r, and the other the deletion ones, p_e. Sequentially, p should be decomposed as $p = p_r; p_e$.

Minimal Marking. The concept of minimal initial digraph can be used to find the minimum marking able to fire a given transition sequence. For example, Figure 4 shows the calculation of the minimal marking able to fire transition sequence $t_5;t_3;t_1$ (from right to left). Notice that $(\bar{r}_1 L_1) \vee (\bar{r}_1 L_2)(\bar{r}_2 L_2) \vee \cdots \vee (\bar{r}_1 L_n) \cdots (\bar{r}_n L_n)$ is the expanded form of equation 2.

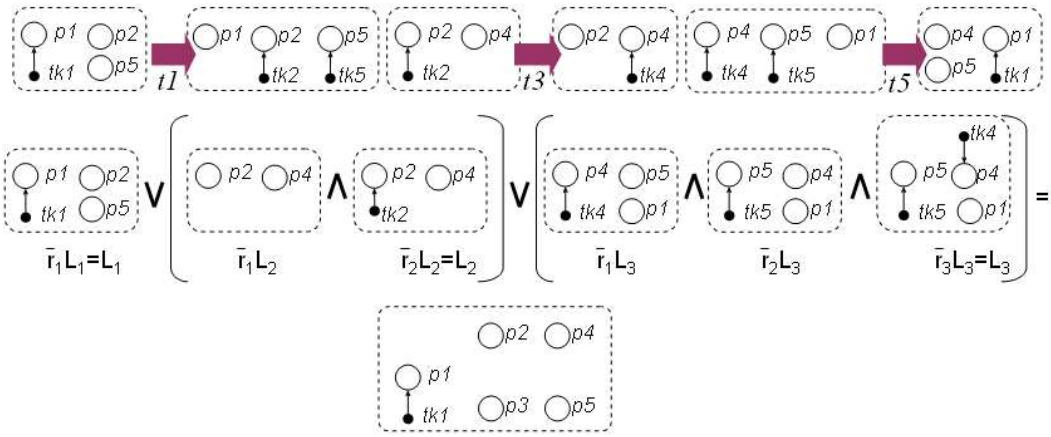


Figure 4: Minimal Marking Firing Sequence $t_5;t_3;t_1$.

Reachability. The reachability problem can also be expressed using MGG concepts, as the following definition shows.

Definition 1 (Reachability) Given a grammar $\mathfrak{G} = (M_0, \{p_1, \ldots, p_n\})$, a state M_d is called *reachable* starting in state M_0, if there exists a coherent concatenation made up of productions $p_i \in \mathfrak{G}$ with minimal initial digraph contained in M_0 and image in M_d.

5 Reachability: DPO-Like Matrix Graph Grammars

In this and next sections we shall be concerned with the generalization of the state equation to wider types of grammars. By a *DPO-like* matrix graph grammar we understand a grammar as introduced in [VL06b], but in which rule applications do not generate dangling edges. That is, in any reachable graph from the initial one, no rule application can generate a dangling edge (i.e. the dangling condition in classical Double Pushout graph grammars [EEPT06] can be safely ignored as it can never happen). Property 2 in section 4 of Petri nets is relaxed because now a single production may eventually be applied in several different places of the host graph. However, following the discussion in previous section, we restrict to DPO rules in which nodes (or edges) of the same type are not rewritten (deleted and created) in the same rule.

In order to perform an *a priori* analysis it is mandatory to get rid of matches. To this end, either an approach as proposed in [VL06a, VL06b] is followed (as we did in this paper in section 4), or types of nodes are taken into account instead of nodes themselves. Here, we take this second alternative, therefore productions, initial state and final state are transformed such that types of elements are considered, obtaining matrices with elements in $\mathbb{Z}$.

Tensor notation [Sok51] will be used in the rest of the paper to extend the state equation.

Though it shall be avoided whenever possible, four index may be used simultaneously, ${}^{E}_{0}\mathrm{A}^{i}_{j}$. Top left index indicates whether we are working with nodes (N) or edges (E). Bottom left index specifies the position inside a sequence, if any. Top right and bottom right are contravariant and covariant indices, respectively.

Definition 2 Let $\mathfrak{G} = ({}_0M, \{p_1, \ldots, p_n\})$ be a DPO-like graph grammar and m the number of different types of nodes in $\mathfrak{G}$. The incidence matrix for nodes ${}^{N}\mathrm{A} = \left(\mathrm{A}^{i}_{k}\right)$ where $i \in \{1, \ldots, n\}$ and $k \in \{1, \ldots, m\}$ is defined by the identity

$$\mathrm{A}^{i}_{k} = \begin{cases} +r & \text{if production } k \text{ adds } r \text{ nodes of type } i \\ -r & \text{if production } k \text{ deletes } r \text{ nodes of type } i \end{cases} \tag{7}$$

It is straightforward to deduce for nodes an equation similar to (6):

$${}^{N}_{d}\mathrm{M}^{i} = {}^{N}_{0}\mathrm{M}^{i} + \sum_{k=1}^{n} {}^{N}\mathrm{A}^{i}_{k}\mathrm{x}^{k} \tag{8}$$

The case for edges is similar, with the peculiarity that edges are represented by matrices instead of vectors and thus the incidence matrix becomes the *incidence tensor* ${}^{E}\mathrm{A}^{i}_{jk}$. Again, only types of edges, and not edges themselves, are taken into account. Two edges e_1 and e_2 are of the same type if their starting and final nodes are of the same type. Initial nodes of edges will be assumed to have a contravariant behaviour (index on top, i) while terminal nodes (first index, j) and productions (second index, k) will behave covariantly (index on bottom). See diagram in the center of Figure 6.

Example.□Some rules for a simple client-server system adapted from [VL06a] are defined in Figure 5. There are three types of nodes: clients (C), servers (S) and routers (R). Messages (self-loops in clients) can only be broadcasted. In the MGG approach, this transformation system will behave SPO or DPO-like depending on the initial state. Note that production p_4 adds and deletes edges of the same type (C,C). By now, the rule will not be split into its addition and deletion components as suggested in 4. See section 6.1 for an example.

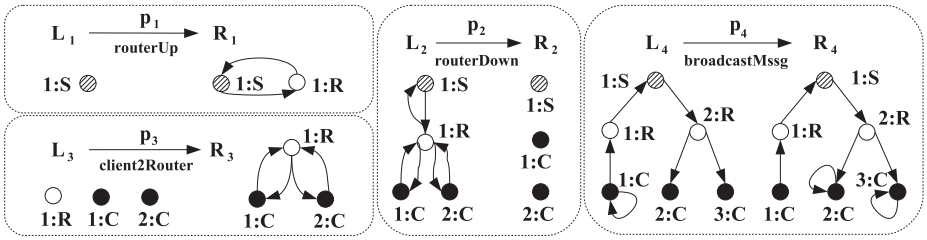


Figure 5: Rules for a Client-Server Broadcast-Limited System.

Incidence tensor (edges) for these rules can be represented componentwise, each component being the matrix associated to the corresponding production.

$${}^{E}A^{i}_{j1} = \left[\begin{array}{ccc|c} 0 & 0 & 0 & C \\ 0 & 0 & 1 & R \\ 0 & 1 & 0 & S \end{array}\right]; {}^{E}A^{i}_{j2} = \begin{bmatrix} 0 & -2 & 0 \\ -2 & 0 & -1 \\ 0 & -1 & 0 \end{bmatrix}; {}^{E}A^{i}_{j3} = \begin{bmatrix} 0 & 2 & 0 \\ 2 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}; {}^{E}A^{i}_{j4} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

For space limitations, only in ${}^{E}A^{i}_{j1}$ we have specified which type each row belongs to. Columns follow the same ordering. ■

Lemma 1 *With notation as above, a necessary condition for state* ${}_{d}\mathrm{M}$ *to be reachable from state* ${}_{0}\mathrm{M}$ *is*

$$ {}_{d}\mathrm{M} - {}_{0}\mathrm{M} = {}^{E}\mathrm{M} = {}^{E}\mathrm{M}^{i}_{j} = \sum_{k=1}^{n} {}^{E}\mathrm{A}^{i}_{jk} x^{k}_{j} = \sum_{k=1, p=k}^{n} \left({}^{E}\mathrm{A} \otimes x\right)^{ip}_{jk} \tag{9} $$

where $i, j \in \{1, \ldots, m\}$.

Proof
□Consider the construction depicted in the center of figure 6 in which tensor A^{i}_{jk} is represented as a cube. A product for this object is informally defined in the following way: every vector in the cube perpendicular to matrix x acts on the corresponding row of the matrix in the usual way, i.e., for every fixed $i = i_0$ and $j = j_0$ in (9):

$$ {}^{E}_{d}\mathrm{M}^{i_0}_{j_0} = {}^{E}_{0}\mathrm{M}^{i_0}_{j_0} + \sum_{k=1}^{n} {}^{E}\mathrm{A}^{i_0}_{j_0 k} \mathrm{x}^{k}_{j_0} \tag{10} $$

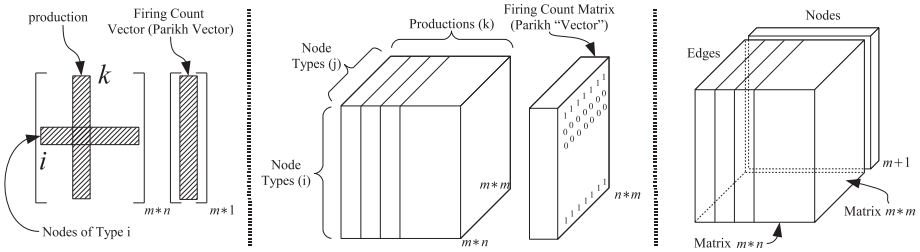


Figure 6: Matrix Representation for Nodes, Tensor for Edges and Their Coupling.

Every column in matrix x is a Parikh vector as defined for Petri nets. Its elements specify the amount of times that every production must be applied, so all rows must be equal and hence, equation (10) needs to be enlarged with some additional identities

$$ \begin{cases} \mathrm{M}^{i}_{j} & = & \sum_{k=1}^{n} {}^{E}\mathrm{A}^{i}_{jk} \mathrm{x}^{k}_{j} \\ x^{k}_{p} & = & x^{k}_{q} \end{cases} \tag{11} $$

with $p, q \in \{1, \ldots, m\}$. This uniqueness together with previous equations provide the intuition to raise (9).

Informally, we are enlarging the space of possible solutions and then projecting according to some restrictions. To see that it is a necessary condition, suppose that there exists a sequence s_n such that $s_n({}_{0}\mathrm{M}) = {}_{d}\mathrm{M}$ and that equation (10) does not provide any solution. Without loss of generality we may assume that the first column fails (the one corresponding to nodes emerging

from the first node), which produces an equation completely analogous to the state equation for Petri nets, deriving a contradiction. ■

Example (Cont'd).□Let's test whether it is possible to move from state S_0 to state S_d (see Figure 7) with the productions defined in previous example.

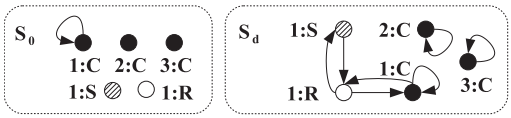


Figure 7: Initial and Final States for Productions in Fig.5.

Matrices for the states (edges only) and their difference are:

$$ {}^{E}S_0 = \left[\begin{array}{ccc|c} 1 & 0 & 0 & C \\ 0 & 0 & 0 & R \\ 0 & 0 & 0 & S \end{array}\right]; \quad {}^{E}S_d = \left[\begin{array}{ccc|c} 3 & 1 & 0 & C \\ 1 & 0 & 1 & R \\ 0 & 1 & 0 & S \end{array}\right]; \quad {}^{E}S = {}^{E}S_d - {}^{E}S_0 = \left[\begin{array}{ccc|c} 2 & 1 & 0 & C \\ 1 & 0 & 1 & R \\ 0 & 1 & 0 & S \end{array}\right] $$

The proof of proposition 2 poses the following matrices:

$$ {}^{E}A^i_{1k} = \left[\begin{array}{cccc|c} 0 & 0 & 0 & 1 & C \\ 0 & -2 & 2 & 0 & R \\ 0 & 0 & 0 & 0 & S \end{array}\right]; {}^{E}A^i_{2k} = \left[\begin{array}{cccc|c} 0 & -2 & 2 & 0 & C \\ 0 & 0 & 0 & 0 & R \\ 1 & -1 & 0 & 0 & S \end{array}\right]; {}^{E}A^i_{3k} = \left[\begin{array}{cccc|c} 0 & 0 & 0 & 0 & C \\ 1 & -1 & 0 & 0 & R \\ 0 & 0 & 0 & 0 & S \end{array}\right] $$

These matrices act on matrix $x = \left(x^p_q\right)$, $p \in \{1,2,3,4\}$, $q \in \{1,2,3\}$ to obtain

$$ \begin{aligned} {}^{E}S_1 &= \sum_{k=1}^{4} {}^{E}A_{1k}x^k_1 = \left[\begin{array}{c} x^4_1 \\ -2x^2_1 + 2x^3_1 \\ 0 \end{array}\right] \\ {}^{E}S_2 &= \sum_{k=1}^{4} {}^{E}A_{2k}x^k_2 = \left[\begin{array}{c} -2x^2_2 + 2x^3_2 \\ 0 \\ x^1_2 - x^2_2 \end{array}\right] \\ {}^{E}S_3 &= \sum_{k=1}^{4} {}^{E}A_{3k}x^k_3 = \left[\begin{array}{c} 0 \\ x^2_3 - x^3_3 \\ 0 \end{array}\right] \end{aligned} $$

Recall that x must satisfy

$$ x^1_1 = x^1_2 = x^1_3; \quad x^2_1 = x^2_2 = x^2_3; \quad x^3_1 = x^3_2 = x^3_3; \quad x^4_1 = x^4_2 = x^4_3. $$

A contradiction is derived for example with equations $x^2_2 = x^3_2$, $-2x^2_1 + 2x^3_1 = 1$, $x^2_1 = x^2_2$ and $x^3_1 = x^3_2$. ■

It is straightforward to derive a unique equation for reachability which considers both nodes and edges, i.e., equations (8) plus (9). This is accomplished extending the incidence matrix M from $M : E \to E$ to $M : E \times N \to E$ (from $M_{m\times m}$ to $M_{m\times(m+1)}$), where column $m+1$ corresponds to nodes.

Definition 3 (Incidence Tensor) Given a grammar $\mathfrak{G} = ({}_0M, \{p_1, \ldots, p_n\})$, the incidence tensor A^i_{jk} with $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, m+1\}$ is defined by (9) if $1 \leq j \leq m$ and by (8) if $j = m+1$.

Note that top left index in our notation works as follows: ${}^NA$ refers to nodes, ${}^EA$ to edges and A to their coupling. An immediate extension of lemmata 1 is:

Proposition 2 (State Equation for DPO-like Matrix Grammars) *With notation as above, a necessary condition for state ${}_d$M to be reachable (from state ${}_0$M) is*

$$M^j_i = \sum_{k=1}^{n} A^i_{jk} x^k \tag{12}$$

Notice that equation (12) is a generalization of the state equation (6) for Petri nets.

6 Reachability: SPO-like matrix graph grammars

Our intention now is to relax the second property of Petri nets and allow production application even though some dangling edge might appear (see [VL06a, VL06b]). The plan is carried out in two stages which correspond to the subsections that follow, according to the classification of ε-productions in [VL06a].

In our approach, if applying a production p_0 causes dangling edges, the production can be applied but a new production (a so-called ε-production) is created and applied first. In this way, we obtain a sequence $p_0; p_{\varepsilon 0}$, with the restriction that $p_{\varepsilon 0}$ is applied at a match that includes all nodes deleted by p_0 that produce dangling edges. Production $p_{\varepsilon 0}$ deletes the dangling edges produced by p_0, and then p_0 can be applied, without producing dangling edges (see [VL06a] for details).

Inside a sequence, a production p_0 that deletes an edge or node can have an *external* or *internal* behaviour, depending on the identification carried out by the match. Following [VL06a], if the deleted element was added or used by a previous production, we shall label the production as *internal* (according to the sequence). On the other hand, if the deleted element is provided by the host graph and it is not used until p_0's turn, then p_0 is an external production. In some sense, their properties are complementary: while external ε-productions can be advanced and composed to eventually get a single initial production which adapts the host graph to the sequence, internal ε-productions are more *static*[3] in nature. On the other hand, internal ε-productions depend on productions themselves and are somewhat independent of the host graph, opposite to external ε-productions. Note however that internal nodes can be unrelated if, for example, matchings identify them in different parts of the host graph, thus becoming external.

6.1 External ε-productions

The main property of external ε-productions, with respect to internal ε-productions, is that they act only on edges that appear in the initial state, so their application can be advanced to the beginning of the sequence. In this situation, the first thing to know for a given matrix graph

[3] Maybe it is possible to advance their application but, for sure, not to the beginning of the sequence.

grammar $\mathfrak{G} = ({}_0\mathrm{M}, \{p_1, \ldots, p_n\})$ with at most external ε-productions when applied to ${}_0\mathrm{M}$, is the maximum number of edges that can be erased from its initial state. The potential dangling edges (those with any incident node to be erased) are given by:

$$\mathfrak{e} = \bigvee_{k=1}^{n} \left(\overline{\overline{{}_k^N e} \otimes \overline{{}_k^N e}} \right) \tag{13}$$

Proposition 3 *Given a grammar $\mathfrak{G} = ({}_0M, \{p_1, \ldots, p_n\})$ with external ε-productions only, a necessary condition for state ${}_d\mathrm{M}$ to be reachable (from state ${}_0\mathrm{M}$) is*

$$M_j^i = \sum_{k=1}^{n} \left(A_{jk}^i x^k \right) + b_j^i \tag{14}$$

with the restriction ${}_0M\mathfrak{e} \leq b_j^i \leq 0$.

Note that equation (12) in proposition 2 is recovered from (14) if there are no external ε-productions.

According to [VL06a], all ε-productions can be advanced to the beginning of the sequence and be composed to obtain a single production, adapting the initial digraph before applying the sequence, which in some sense interprets matrix b as *the* production number $n+1$ in the sequence (i.e. the first to be applied). Because it is not possible to know in advance the order of application of productions, all we can do is to provide bounds for the number of edges to be erased.

Example. Consider the initial and final states shown in Figure 8. Productions of previous examples are used, but two of them are modified (p_2 and p_3).

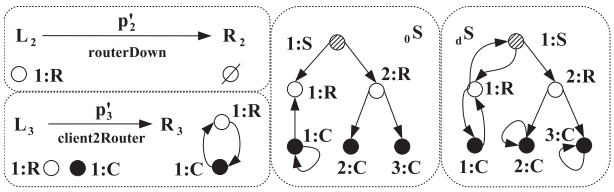


Figure 8: Initial and Final States with Amendments to Some Productions of Fig. 5.

In this case there are sequences that transform state ${}_0S$ in ${}_dS$, for example, $s_4 = p_4; p_1; p_3'; p_2'$. Note that the problem is in edge $(1:S, 1:R)$ of the intial state: router 1 is able to receive packets from server 1, but not to send them.

Next, matrices for the states and their difference are calculated. The first three columns correspond to edges (first to clients, second to routers and third to servers), fourth to nodes and fifth specifies types.

$${}_0S = \left[\begin{array}{ccc|c|c} 1 & 1 & 0 & 3 & C \\ 2 & 0 & 0 & 2 & R \\ 0 & 2 & 0 & 1 & S \end{array}\right]; \quad {}_dS = \left[\begin{array}{ccc|c|c} 2 & 1 & 0 & 3 & C \\ 3 & 0 & 1 & 2 & R \\ 0 & 2 & 0 & 1 & S \end{array}\right]; \quad S = {}_dS - {}_0S = \left[\begin{array}{ccc|c|c} 1 & 0 & 0 & 0 & C \\ 1 & 0 & 1 & 0 & R \\ 0 & 0 & 0 & 0 & S \end{array}\right]$$

The incidence tensors for every production (recall that p_2 and p_3 are as in Figure 8) have the form

$$A^i_{j1} = \left[\begin{array}{ccc|c} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \end{array}\right]; A^i_{j2} = \left[\begin{array}{c|c} 0 & 0 \\ & -1 \\ & 0 \end{array}\right]; A^i_{j3} = \left[\begin{array}{ccc|c} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right]; A^i_{j4} = \left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right]$$

Though it does not seem to be strictly necessary here, more information is kept and calculations are more flexible if production p_4 is split in the part that deletes messages and the part that adds them, $p_4 = p_{4r}; p_{4e}$. See comments about this in section 4

$$A^i_{j4e} = \left[\begin{array}{ccc|c|cc} -1 & 0 & 0 & 0 & C \\ 0 & 0 & 0 & 0 & R \\ 0 & 0 & 0 & 0 & S \end{array}\right]; \quad A^i_{j4r} = \left[\begin{array}{ccc|c|cc} 2 & 0 & 0 & 0 & C \\ 0 & 0 & 0 & 0 & R \\ 0 & 0 & 0 & 0 & S \end{array}\right]$$

As in the example of section 5, the following matrices are more appropriate for calculations:

$$A^i_{1k} = \left[\begin{array}{ccccc} 0 & 0 & 0 & -1 & 2 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right]; A^i_{2k} = \left[\begin{array}{ccccc} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{array}\right]; A^i_{3k} = \left[\begin{array}{ccccc} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right];$$

$$A^i_{4k} = \left[\begin{array}{ccccc} 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

If (12) is directly applied, we obtain $x^1 = 0$ and $x^1 = 1$ (third row of A^i_{2k} and second of A^i_{3k}) deriving a contradiction. The variations permitted for the initial state are given by the matrix

$$_0M\mathfrak{e} = \left[\begin{array}{cccc} 0 & \alpha^1_2 & 0 & 0 \\ \alpha^2_1 & 0 & 0 & 0 \\ 0 & \alpha^3_2 & 0 & 0 \end{array}\right] \tag{15}$$

With $\alpha^1_2 \in \{0,-1\}$, $\alpha^2_1, \alpha^3_2 \in \{0,-1,-2\}$. Setting $b^1_2 = -1$ and $b^3_2 = -1$ (one edge (S,R) and one edge (C,R) removed) the system to be solved is

$$\left[\begin{array}{cccc} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{array}\right] = \left[\begin{array}{cccc} -x^4 + 2x^4 & x^3 & 0 & 0 \\ x^3 & 0 & x^1 & x^1 - x^2 \\ 0 & x^1 & 0 & 0 \end{array}\right]$$

with solution $x^1 = x^2 = x^3 = x^4 = 1$, s_4 being one of its associated sequences. Note that the restriction in proposition 3 is fulfilled, see (15).

In previous example, as we knew a sequence (s_4) answer to the reachability problem, we have fixed matrix b directly to show how proposition 3 works. Although this will not be normally the case, the way to proceed is very similar: relax matrix M by substracting b, find a set of solutions $\{x, b\}$ and check whether the restriction for matrix b is fulfilled or not.

6.2 Internal ε-productions

Internal ε-productions delete edges appended or used by productions preceding it in the sequence. In this subsection we first limit to sequences which may have only internal ε-productions and, by the end of the section, we shall put together proposition 3 from subsection 6.1 with results derived here to state theorem 5 for SPO-like matrix graph grammars.

The way to proceed is completely analogous to that of external ε-productions. The idea is to allow some variation in the amount of edges erased by every production, but this variation is constrained depending on the behaviour (definition) of the rest of the rules. Unfortunately, not so much information is gathered in this case and what we are basically doing is ignoring this part of the state equation.

Define $h^i_{jk} = \left[A^i_{jk}\left(\mathfrak{e}\otimes\mathbb{I}_k\right)\right]_+ = \max\left(A(\mathfrak{e}\otimes\mathbb{I}),0\right)$, where $\mathbb{I}_k = [1,\ldots,1]_{(1,k)}$.[4]

Proposition 4 *Given a grammar $\mathfrak{G} = ({}_0M, \{p_1,\ldots,p_n\})$ with internal ε-productions only, a necessary condition for state ${}_d$M to be reachable (from state ${}_0$M) is*

$$M^i_j = \sum_{k=1}^{n}\left(A^i_{jk}+V\right)x^k \tag{16}$$

with the restriction $h^i_{jk} \le V^i_{jk} \le 0$.

In some sense, external productions are the limiting case of internal productions and can be seen almost as a particular case: as ε-productions do not interfere with previous productions they have to act exclusively on the host graph. The full generalization of the state equation for matrix graph grammars is:

Theorem 5 (State Equation) *With previous notation, a necessary condition for state ${}_d$M to be reachable (from state ${}_0$M) is*

$$M^i_j = \sum_{k=1}^{n}\left(A^i_{jk}+V\right)x^k + b^i_j \tag{17}$$

b^i_j must satisfy restrictions specified in proposition 3 and V those in proposition 4.

Strengthening hypothesis, formula (5) becomes those already studied for SPO with internal ε-productions ($b=0$), with external ε-productions ($V=0$), DPO-like (from multilinear to linear transformations) or Petri nets, fully recovering the original form of the state equation.

7 Related Work

Concerning our MGG approach to graph rewriting, in [Val98] an implementation of the DPO categorical approach to graph transformation was implemented using Mathematica. In that work, (simple) digraphs were represented with their boolean adjacency matrices. This is the only similarity with our work, as our goal is to develop a theory for (simple) graph rewriting based on

[4] $\mathfrak{e}\otimes\mathbb{I}(k)$ defines a tensor of type (1,2) wich "repeats" matrix $\mathfrak{e}$ k times.

boolean matrix algebra. Other somehow related approach is the relational approaches of [MK95] and [Kah02]. However, they rely on using category theory for expressing the rewriting.

The recent work [VVE+06] shows a means to encode a graph transformation system into a Petri net. Then algebraic approaches based on the state equation can be used for analysis. The approach is similar to ours, as they perform the same abstraction (taking node and edge types). However, on the one hand, they consider negative application conditions in rules. On the other hand, we consider both DPO and SPO-like grammars, and we extend the state equation using tensors, instead of first encoding the transformation system as a Petri net.

8 Conclusions and Future Work.

The starting point of the present paper is the study of Petri nets as a particular case of MGGs. Next, reachability and the state equation have been reformulated and extended with the language of this new approach, trying to provide tools for grammars as general as possible. The objective is almost fulfilled, bearing in mind that the more general the grammar, the less information the state equation provides. For example, equation (12) is more accurate as long as the rate of the amount of types of nodes with respect to the amount of nodes approaches one. Hence, in general, it will be of little practical use if there are many nodes but few types.

Nevertheless, we are in a much better position than we were before. Although the use of vector spaces (as in Petri nets) and multilinear algebra is almost straightforward, many other algebraic structures are available to improve the results herein presented. For example, Lie algebras seem a good candidate if we think of the Lie bracket as a measure of commutativity (recall subsection 3 in which we saw that this is one of the main problems of using linear combinations).

Other concepts and techniques from Petri nets such as invariants, boundedness, liveness, reversibility, persistence, etc., can also be extended to more general grammars. Besides, it is possible as well to get a better insight and understanding of matters by applying matrix graph grammars techniques (as those in [VL06a, VL06b] and the present paper) to Petri nets. We are also extending our matrix graph grammars approach to work with multi-graphs. A first line of attack is to model edges as special nodes (with exactly one incoming and outgoing edge).

Acknowledgements: This work has been sponsored by the Spanish Ministry of Science and Education, project MOSAIC (TSI2005-08225-C07-06). The authors would like to thank the referees, the workshop participants and Alejandro Pérez Velasco for their useful suggestions.

References

[EEPT06] H. Ehrig, K. Ehrig, U. Prange, G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. Monographs in Theoretical Computer Science, an EATCS series. Springer, Berlin, Heidelberg, New York, 2006.

[Kah02] W. Kahl. A Relational Algebraic Approach to Graph Structure Transformation. Technical report 2002-03, Universitat der Bundeswehr Munchen, 2002.

[MK95] Y. Mizoguchi, Y. Kuwahara. Relational Graph Rewritings. *Theoretical Computer Science* 141:311–328, 1995.

[Mur89] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE* 77(4):541–580, April 1989.

[Sok51] I. S. Sokolnikoff. *Tensor Analysis, Theory and Applications*. Applied Mathematics series. John Wiley and Sons, New York, 1951.

[Val98] G. Valiente. Grammatica: An Implementation of Algebraic Graph Transformation on Mathematica. In *Proc. Sixth Workshop on Theory and Application of Graph Transformations*. Pp. 261–267. 1998.

[VL06a] P. P. P. Velasco, J. de Lara. Matrix Approach To Graph Transformation: Matching and Sequences. In *Proc. ICGT'06*. Lecture Notes in Computer Science 4178, pp. 122–137. Springer-Verlag, 2006.

[VL06b] P. P. P. Velasco, J. de Lara. Towards a New Algebraic Approach to Graph Transformation: Long Version. Technical report, Universidad Autónoma de Madrid, 2006. http://www.ii.uam.es/jlara/investigacion/techrep_03_06.pdf

[VVE+06] D. Varro, S. Varro-Gyapay, H. Ehrig, U. Prange, G. Taentzer. Termination Analysis of Model Transformations by Petri Nets. In *Proc. ICGT'06*. Lecture Notes in Computer Science 4178, pp. 260–274. Springer-Verlag, 2006.