Computer Assisted Mechanics and Engineering Sciences, 3: 191–203, 1996.

Quasi context sensitive graph grammars as a formal model of FE mesh generation

Mariusz Flasiński and Robert Schaefer
Institute of Computer Science, Jagiellonian University
Nawojki 11, 30-072 Cracow, Poland

(Received July 24, 1995)

Mathematical linguistics models that can be useful for controlling FE mesh generation are presented in the paper. The problems concerning an application of formal grammars for this purpose are outlined. Advantages and drawbacks related to the use of various types of Lindenmayer string/graph systems applied recently for FE mesh generation are discussed. An efficient (parsable) class of ETPL(k) graph grammars is proposed as a formal tool in this research area.

1. Introduction

Construction of efficient algorithms for mesh generation according to a current initial/boundary value problem that could be adopted during a computational process is one of the most important problems in CAE [9]. The following three main groups of generation methods are known in the literature.

- *Structural generation*, which consists in mapping some network primitives (patterns) in a computational domain. If one needs to obtain a high quality network (in a sense of a low computational error), a class of admissible mappings is restricted severely. As the result, meshes that are quite uniform on simple domains can be accepted only.

- *Delaney's process* a of simplex triangulation (triangles in 2D, tetrahedra in 3D) under a given set of vertices, which generates an optimum network of an arbitrary distributed density in case of convex domains. Node positions are usually determined basing on a density function that is specific to a current b.v. problem.

- *Wandering front methods*, which add elements consecutively, starting from a border. Elements may be fitted to a predefined set of nodes, or generated together with their vertices. In the first case, node displacement may be arbitrary, but the topology is usually far from the optimum one. In the second case the topology is forced to be optimum (e.g. triangles are close to equilateral) but node positions are accidental.

Refinement/derefinement techniques are used, firstly, in order to remove imperfections in a network, which is generated initially by methods listed above. Secondly, they are also applied in order to adopt a network to a current error estimator displacement or to solution singularities, which may wander across a computational domain in the case of nonstationary problems.

Computational networks may be refined or derefined in the following two ways:

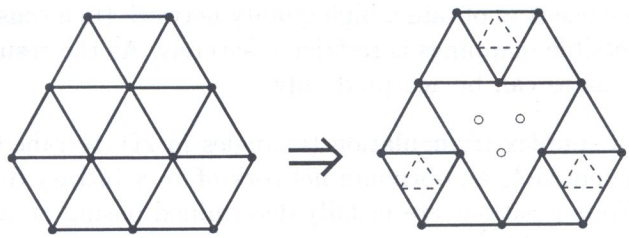
- by total or partial remeshing based on e.g. a newly assumed node density function and using one of above methods. Old nodes and degrees of freedom may be then removed in this process,

- by element breaking or gluing in order to obtain a network that is more or less dense locally.

The last technique can be formalized on the basis of the theory of formal grammars and automata (mathematical linguistics). This theory delivers suitable models for such important areas of computer science applications like image and vision analysis, pattern recognition, information coding, CAD/CAM, etc. The usefulness of its models for generation of 2D structures interpreted as FE meshes has been demonstrated recently in the literature [15]. In a cited paper, string graph Lindenmayer systems of the class 0L have been proposed as the generation formalism. Its authors have shown that a broad spectrum of 2D structures can be generated with this formalism. In the third chapter we show some power limitations of 0L systems that can make using such systems difficult in case of more complex situations encountered during FE mesh restructuring. Moreover, we discuss also computational efficiency issues related to using Lindenmayer systems. We propose two ways of enhancement of the model proposed in [15]. The first one, discussed in Chapter 4, consists in using stronger (i.e. context-sensitive) string Lindenmayer systems, namely IL systems. In the fifth chapter we introduce graph Lindenmayer systems, GL systems, as an enhanced multi-dimensional version of standard (string) Lindenmayer systems. We discuss, also, some generative power properties of GL systems in comparison with other, computationally more efficient, classes of graph generation systems (graph grammars). At the end, we propose parsable, $O(n^2)$, ETPL(k) graph grammars [4, 5, 6, 7, 8] as a tool for 2D FE mesh structure generation.

2. The use of 0L-systems for a FE mesh generation

In computer science, a structure generation problem is solved usually in a dynamic, iterative, way. That is, we begin with some axiom representing the initial structure, and then we apply a sequence of structure-modifying operators that act locally at certain substructures and rewrite them according to pre-specified rules. For example, if we want to restructure a lattice/mesh shown at the left-hand side of Fig. 1a, in order to receive the one shown at the right-hand side of the figure, we can use a rewriting rule (called a production in the theory of formal grammars and automata), which is specified in Fig. 1b.



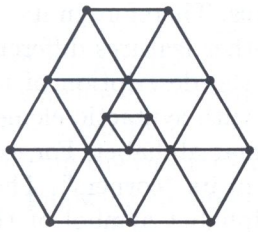
a)

b)

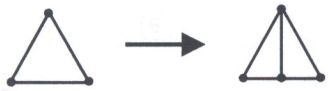
Fig. 1

As one can see in Fig. 1b, we want to restructure the mesh only at certain places, i.e. we want to apply a production only at these places. This problem is referred to as the control of a (local) applicability of a production, and it will be discussed further on. Another important

problem encountered, in case of applying formal grammars for a structure generation/modification is formulated in the following way. Restructuring a mesh in a certain place according to one rule can sometimes cause a need of restructuring the mesh in other specific places. This situation is shown in Fig. 2a, in which an application of a production defined in Fig. 1b should cause restructuring the neighbourhood cells, but according to a different rule, which is defined in Fig. 2b. This problem is very difficult to handle, and "plain" structure rewriting systems, as e.g. 0L systems used in [15] do not support its solution. We will come back to it in the next chapter.



a)



b)

Fig. 2

0L systems defined by Lindenmayer in [12] are one of the most powerful rewriting systems used for structure generation/modification. These systems have been used for FE mesh generation in [15]. We will start our considerations concerning the use of rewriting systems (formal grammars) from that point. Therefore, let us introduce 0L systems in a formal way.

Definition 1

An *0L-scheme* is a pair $S = (\Sigma, P)$, where Σ is a finite nonempty set, called the alphabet of S, P is a finite nonempty set of productions of the form $a \longrightarrow \gamma, a \in \Sigma, \gamma \in \Sigma^*$.

Definition 2

Let $S = (\Sigma, P)$ be an 0L-scheme and $x \in \Sigma^+, y \in \Sigma^*$. The word y is *directly derivable* from x, denoted $x \xrightarrow[S]{} y$, iff $x = x_1 \ldots x_n, y = \gamma_1 \ldots \gamma_n, x_i \in \Sigma, \gamma_i \in \Sigma^*$, and $x_i \longrightarrow \gamma_i \in P$ for $1 \leq i \leq n$.

We say y is *derivable* from x iff $x \xrightarrow[S]{*} y$, where $\xrightarrow[S]{*}$ is the reflexive and transitive closure of $\xrightarrow[S]{}$.

Definition 3

An *0L-system (grammar)* is a triplet $G = (\Sigma, P, \omega)$, where (Σ, P) is an (underlying) 0L-scheme, $\omega \in \Sigma^+$ is the axiom.

Definition 4

The *language generated by the 0L-system* G is the set $L(G) = \{x : x \in \Sigma^* \text{ and } \omega \xrightarrow[G]{*} x\}$.

A simple example of 0L grammar production is shown in Fig. 3a. This production is formally defined as:

$a \longrightarrow bc$

3. Basic problems of the use of linguistic formalisms for the controlled structure rewriting

After introducing an 0L grammar in a formal way, we can discuss three basic problems concerning its use for the *controlled* structure rewriting. These problems can be formulated in the following way.

1. *The problem of generated object semantics.* Formal grammars and automata were defined originally as formalisms for generation/analysis of abstract structure representations, like for example natural languages syntactic structures. Therefore in its "pure" form they describe only a syntax of structures, not their semantics (other features different from a syntactic frame only). In case of the use of these formalisms for the description of real-world phenomena/objects we need to associate additional information with syntactic elements. This can be made with attributes associated with elements of a grammar alphabet. For example, the element a shown in Fig. 3b can be attributed with coordinates of its "corners". Then, instead of using just "pure" syntax element a, we use an attributed alphabet symbol of the form $a[(x_1, y_1), (x_2, y_2), (x_3, y_3)]$. Of course, this attributing property influences a way of defining grammar productions.

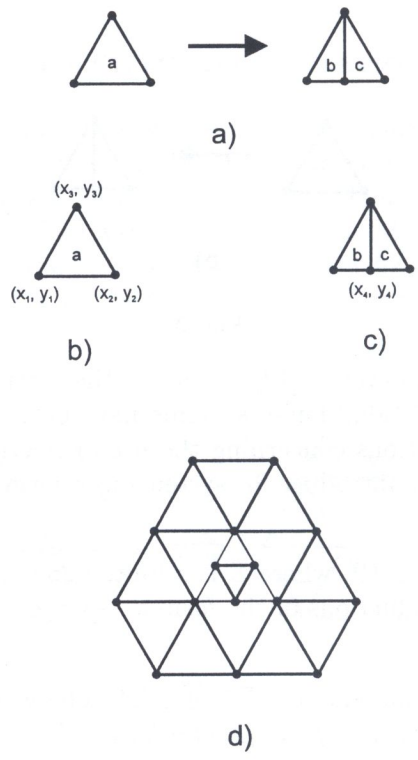


Fig. 3

Let us notice, for example, that if a production shown in Fig. 3a is concerned, we would like to be able to assign an additional attribute to a point (x_4, y_4) which is generated as the result of this production application (see Fig. 3c). Therefore, we have to modify the form of the production. Till now, we assumed that a production is a pair (a, γ) (cf. Definition 1). Attributed scheme/system/grammar production is a triple (a, γ, F), where F is the set of attributing functions. In case of our example the set F should contain the following two attributing functions computing values of coordinates x_4 and y_4:

$$x_4 := (x_1 + x_2)/2\,;$$
$$y_4 := (y_1 + y_2)/2\,.$$

2. *The problem of (local) applicability of productions.* As we have mentioned in the previous chapter, we want to be able to modify a structure only at certain places. Usually these places of a production application are just determined by attributes ascribed to them. This problem is also solved by changing the form of a grammar production. The production is then treated as a quadruple (a, γ, π, F), where π is the so-called predicate of production applicability. It is simply a conjunction of conditions defined over attributes of a substructure to be rewritten. The production is applied at the substructure occurence only if these conditions are fulfilled, i.e. the predicate equals to "true".

For two problems discussed above, there are simple and effective solutions described. They have been applied successfully in [15] to enhance a generative power of 0L systems applied for FE mesh generation. However, the third problem, which will be discussed below, is much more difficult. There are no ready solutions delivered for applications by the theory of formal grammars and automata. What is worse, the problem is still open from the theoretical point of view. Before we propose some solutions for FE mesh generation in the following chapters, let us now discuss this problem.

3. *The problem of context-sensitiveness.* Modification of a structure at a certain occurence of a substructure should sometimes cause its further modifications in a context (neighbourhood) of an already modified substructure. Such a situation is schematically shown in Fig. 3d, in which restructuring of a central cell should force restructuring its context. Modifying a context should be made either with the same production or not. The problem is crucial, because without solving it we are, in fact, unable to control a rectructuring process, that is we cannot restructure meshes in a way we would like to. The theory of formal grammars and automata does not provide satisfactory solutions of this problem.
However, it points out three directions of enhancing generative power, namely:

- context-sensitive rewriting systems/grammars,
- multidimensional systems/grammars,
- quasi context-free systems/grammars.

In the following chapters, we discuss these solution in more detail.

4. IL-SYSTEMS AS A ONE-DIMENSIONAL FORMALISM INCREASING THE STRUCTURE GENERATIVE POWER

Increasing generative power in a way that it allows one to take into account the context, and in this way to control a restructuring process is the simplest solution. In case of using Lindenmayer formalisms, the problem reduces to the application of (context-sensitive) IL schemes, which are defined in the following way.

Definition 5

Let $k, l \in \mathcal{N}_0$. A (k,l)-*scheme* be a triple $S = (\Sigma, P, g)$, where Σ is an alphabet, $g \notin \Sigma$ is a special symbol, called the environmental symbol, P is a finite nonempty set of productions of the form $(\alpha, a, \beta) \longrightarrow \gamma$, in which $a \in \Sigma$, $\alpha \in \{g\}^*\Sigma^*$, $|\alpha| = k$, $\beta \in \Sigma^*\{g\}^*$, $|\beta| = l$, $\gamma \in \Sigma^*$.

Definition 6

Let $S = (\Sigma, P, g)$ be a (k,l)-scheme and $x \in \Sigma^+, y \in \Sigma^*$. The word y is *directly derivable* from x, denoted $x \xrightarrow[S]{} y$, iff $x = x_1 \dots x_n$, $y = \gamma_1 \dots \gamma_n$, $x_i \in \Sigma$, $\gamma_i \in \Sigma^*$, and $\alpha_i x_i \beta_i \longrightarrow \gamma_i \in P$ for $1 \leq i \leq n$, where $\alpha_i(\beta_i)$ is the adjacent subword to the left (right) of length $k(l)$ of x_i within the word $g^k x_1 \dots x_n g^l$.

We say y is *derivable* from x iff $x \xrightarrow[S]{*} y$, where $\xrightarrow[S]{*}$ is the reflexive and transitive closure of $\xrightarrow[S]{}$.

The notions of: indirect derivation, (k,l)-system and (k,l)-language are defined in an analogical way as for 0L-systems. The following naming convention is used for IL-systems:

- $(0,0)$ — systems correspond to 0L-systems,
- $(1,0)$ — systems correspond to 1L-systems,
- $(1,1)$ — systems correspond to 2L-systems, etc.

We call a Lindenmayer system a (proper) IL-system, if $\mathrm{I} > 0$. If with $\mathcal{L}(X)$ we denote a class of all the languages generated by the grammars/systems of the type X, then the following holds:

$$\mathcal{L}(0L) \subsetneq \mathcal{L}(IL)\,.$$

Let us notice, that with extending an I parameter of IL-systems, we extend a context of an occurence of a substructure, which is to be modified. This production can be applied only in case of occurrence of a specific context of a substructure to be replaced by the right-hand side of a production. As we have mentioned, this solution is simple from the theoretical point of view. However, from the practical point of view, it is usually unacceptable, since algorithms for context-sensitive systems/grammars are of non-polynomial time complexity. Therefore, for lattices/meshes with a large number of nodes they are very inefficient.

5. Graph parallel-rewriting systems as a multidimensional tool for controlled structure rewriting

Multidimensional (tree and graph) grammars have been introduced in the theory of formal grammars and automata as a natural way to represent 2D and 3D structures. From their definitions, they are more "context-sensitive" than standard (string) grammars, since apart from the "normal" elements of an alphabet in the form of labelled tree/graph nodes, they have also edges that create a kind of links to a context. Graph Lindenmayer systems were introduced in 1975 by Nagl [13]. Let us start with formulating the following definition:

Definition 7

A *GL-scheme (Graph Lindenmayer scheme)* is a pair $G = (\Sigma, P)$, where Σ is a finite, nonempty set of node labels, P is a finite nonempty set of productions of the form (d_l, d_r, E), in which d_l is the left-hand side graph, d_r is the right-hand side graph, E is the embedding transformation of the form $E = (l_1, r_1, \ldots, l_n, r_n), l_i = \bigcup_{m=1}^{q} A_m(v_l) \times \{v_m\}, r_i = \bigcup_{m=1}^{p} \{v_m\} \times A_m(v_l)$, v_l is the node of d_l, v_m is the node of d_r, A_m is the Nagl operator.

The notions of: GL-system and language generated by GL-system are defined in an analogical way as it has been done for 0L-systems.

GL-systems are strong enough as a tool for controlling contex-sensitive rewriting in the sense understood in this paper. However, we met here a similar problem to that discussed in a previous chapter. Algorithms of controlled rewriting for GL-systems are also inefficient. However, in this case we can use a formalism of a similar power, for which very efficient algorithms are defined.

We will start our considerations leading to this formalism, from rewriting systems introduced by Culik and Lindenmayer in 1975 [3]. Culik–Lindenmayer graph grammars of the class edSCp (abbrev. from a graph grammar generating edge-labelled directed graphs with the Stencil Controlled embedding transformation and parallel derivation) are, in fact, another form of extending string Lindenmayer systems to graph languages than the form proposed by Nagl. What is interesting for us is that Stencil Controlled graph grammars are of a similar generative power as the Janssens–Rozenberg Node Label Controlled graph grammars [10]. For the latter class, we are able to define the mentioned efficient algorithms. Let us formalize our considerations by defining the sequential form of Node Label Controlled graph grammars [10].

Definition 8

An *edNLC graph grammar* (abbrev. from a graph grammar generating edge-labelled directed graphs with the Node Label Controlled embedding transformation) is a quintuple $G = (\Sigma, \Delta, \Gamma, P, Z)$, where Σ is a finite, nonempty set of node labels, $\Delta \subseteq \Sigma$ is a set of terminal node labels, Γ is a finite, nonempty set of edge labels, P is a finite set of productions of the form (l, D, C), in which $l \in \Sigma$, $D \in EDG_{\Sigma,\Gamma}$ (i.e. D is a directed graph with nodes labelled with Σ and edges labelled with Γ), $C : \Gamma \times \{in, out\} \longrightarrow 2^{\Sigma \times \Sigma \times \Gamma \times \{in,out\}}$ is the Rozenberg embedding transformation, $Z \in EDG_{\Sigma,\Gamma}$.

The edNLC and edSC graph grammars are of similar generative power. In [10] it is shown that:

$\mathcal{L}(edNLC) = \mathcal{L}(edSC)$,

$\mathcal{L}(edNLCp) \subseteq \mathcal{L}(edSCp)$.

In the next chapter, we will define the computationally efficient subclass of edNLC graph grammars.

6. Parsable ETPL(k) graph grammars for FE mesh generation

In this chapter we present definitions introduced in [4, 5, 6, 7] concerning an ETPL(k) subclass of edNLC graph grammars, for which an efficient parsing algorithm, $O(n^2)$, has been defined in [7]. We discuss neither the restrictions imposed by succeeding definitions nor the auxiliary notions in a detailed way, since it has been done in above mentioned papers. In the second section of this chapter we show how to enhance the generative power of ETPL(k) graph grammars.

6.1. Basic definitions concerning ETPL(k) grammars

We will start with definitions concerning IE graphs generated by ETPL(k) grammars.

Definition 9

An *indexed edge-unambiguous graph*, *IE* graph, over Σ and Γ is a quintuple $g = (V, E, \Sigma, \Gamma, \phi)$, where V is a *finite, non-empty set of nodes* to which indices have been ascribed in an unambiguous way, Σ is a *finite, non-empty set of node labels*, Γ is a *finite, non-empty set of edge labels*, E is a *set of edges* of the form (v, λ, w), where $v, w \in V, \lambda \in \Gamma$, such that index of v is less than index of w, $\phi : V \longrightarrow \Sigma$ is a *node-labelling function.* The *family of all the IE graphs over* Σ and Γ is denoted by $IE_{\Sigma,\Gamma}$.

Definition 10

Let $k \in V$ be the node having an index k of an *IE* graph $g = (V, E, \Sigma, \Gamma, \phi)$. A *characteristic description* $n(k), r, (e_1 \ldots e_r), (i_1 \ldots i_r)$, where n is the label of the node k, i.e. $\phi(k) = n$, r is an out-degree of k (out-degree of node designates the number of edges going out from this node), $(i_1 \ldots i_r)$ is a string of node indices, to which edges going out from k come (in increasing order), $(e_1 \ldots e_r)$ is a string of edge labels ordered in such a way that an edge having the label e_x comes into a node having the index i_x.

Definition 11

Let $g = (V, E, \Sigma, \Gamma, \phi)$ be an *IE* graph, where $V = \{1, \ldots, k\}$ is a set of its nodes, $I(i), i = 1, \ldots, k$ is a characteristic description of the form of a quadruple defined above, of a node i. A string $I(1) \ldots I(k)$ is called a *characteristic description* of a graph g.

A derivation in ETPL(k) grammars is determined by a traversal of a derivation tree, which is spanned in an IE graph. This traversal is performed according to the well-known Breadth First Search (BFS) rule (i.e. we traverse a graph level-by-level). Therefore, let us introduce a notion of a level in an IE graph.

Definition 12

Let $g = (V, E, \Sigma, \Gamma, \phi)$ be an indexed edge-unambiguous *EDG* graph. A node having an index 1 is called a *node of the first level.* Some node v is called a *node of the n level,* if: there exists $(w, \lambda, v) \in E : w$ is a node of the $n-1$ level, and for each $[(u, \lambda, v) \in E$ or $(v, \lambda, u) \in E] : u$ is a node of at least the $n-1$ level.

Now, we impose several restrictions on the form of the right-hand side graphs of grammar productions in order to obtain an efficient parsing algorithm.

Definition 13

Let $G = (\Sigma, \Delta, \Gamma, P, Z)$ be an *edNLC* graph grammar. Grammar G is called a *TLP graph grammar* (abbrev. from two-level productions), if the following conditions are fulfilled.

1. P is a finite set of productions of the form (l, D, C), where:
   a) $l \in \Sigma$,
   b) D is the *IE* graph having the characteristic description:

$n_1(1)$	$n_2(2)$	...	$n_m(m)$	*or* $n_1(1)$,	*where* $n_i(i)$
r_1	r_2	...	r_m	0	r_i
E_1	E_2	...	E_m	–	E_i
I_1	I_2	...	I_m	–	I_i

   is a characteristic description of the node $i, i = 1, \ldots, m, n_1 \in \Delta$ (i.e. n_1 is a terminal label), $i, i = 2, \ldots, m$ is the node of the second level,
   c) $C : \Gamma \times \{in, out\} \longrightarrow 2^{\Sigma \times \Sigma \times \Gamma \times \{in,out\}}$ is the embedding transformation.
2. Z is an *IE* graph such that its characteristic description satisfies the condition defined in point 1(b).

Since we want to define a parsing scheme without backtracking, we have to restrict the way of deriving a graph to the one determined by the linear ordering imposed on the resulting graph.

Definition 14

A TLP graph grammar G is called a *closed TLP graph grammar* G if for each derivation of this grammar

$$Z = g_0 \xrightarrow[G]{} g_1 \xrightarrow[G]{} \cdots \xrightarrow[G]{} g_n$$

a graph $g_i, i = 0, \ldots, n$ is an *IE* graph.

Definition 15

Let there be given a derivation of a closed TLP graph grammar G:

$$Z = g_0 \xrightarrow[G]{} g_1 \xrightarrow[G]{} \cdots \xrightarrow[G]{} g_n.$$

This derivation is called a *regular left-hand side derivation* (denoted $\xrightarrow[r(G)]{}$) if:

(1) for each $i = 0, \ldots, n-1$ we apply a production for a node having the least index in a graph g_i,

(2) node indices do not change during derivation.

A closed TLP graph grammar rewriting graphs according to the regular left-hand side derivation is called a *closed TLPO graph grammar* (abbrev. from Two-Level Production-Ordered).

Now, we introduce two auxiliary notions allowing us to extract handles during syntax analysis. A limitation introduced by Definition 18 enables us to construct an efficient, non-backtracking, top-down parser.

Definition 16

Let g be an *IE* graph, l some node of g defined by a characteristic description $n(l), r, e_1 \dots e_r, i_1 \dots i_r$. A subgraph h of the graph g consisting of node l, nodes having indices $i_{a+1}, i_{a+2}, \dots, i_{a+m}, a \geq 0, a+m \leq r$, and edges connecting the nodes: $l, i_{a+1}, i_{a+2}, \dots, i_{a+m}$ is called an *m-successors two-level graph originated in the node l and beginning with the* (i_{a+1})th *successor*. The subgraph h is denoted $h = m - TL(g, l, i_{a+1})$. By $0 - TL(g, l, -)$ we denote the subgraph of g consisting only of node l.

Definition 17

Let g be an *IE* graph, l some node defined by the characteristic description $n(l), r, e_1 \dots e_r, i_1 \dots i_r$. The subgraph h of graph g consisting of node l, nodes having indices $i_{a+1}, i_{a+2}, \dots, i_r, a \geq 0$, and edges connecting the nodes $l, i_{a+1}, i_{a+2}, \dots, i_r$ is called a *complete two-level graph originated in node l and beginning with the* (i_{a+1})th *successor*. The subgraph h is denoted

$$h = CTL(g, l, i_{a+1}).$$

Definition 18

Let $G = (\Sigma, \Delta, \Gamma, P, Z)$ be a closed TLPO graph grammar. The grammar G is called a *PL(k)*, production-ordered k-left nodes unambiguous, *graph grammar* if the following condition is fulfilled.

Let

$$Z \xrightarrow[r(G)]{*} X_1 A X_2 \xrightarrow[r(G)]{} g_1 \xrightarrow[r(G)]{*} h_1$$

and

$$Z \xrightarrow[r(G)]{*} X_1 A X_2 \xrightarrow[r(G)]{} g_2 \xrightarrow[r(G)]{*} h_2,$$

where $\xrightarrow[r(G)]{*}$ is the transitive and reflexive closure of $\xrightarrow[r(G)]{}$, be two regular left-hand side derivations, such that A is a characteristic description of a node indexed with l, and X_1 and X_2 are substrings. Let max be a number of nodes of the graph $X_1 A X_2$. If

$$k - TL(h_1, l, max + 1) \overset{isom}{=} k - TL(h_2, l, max + 1)$$

then

$$CTL(g_1, l, max + 1) \overset{isom}{=} CTL(g_2, l, max + 1).$$

The last two definitions of this section, allow us to restrain potential possibilities of the embedding transformation of edNLC grammars. This will make possible to define a polynomial parsing algorithm for the resulting subclass of edNLC graph grammars.

Definition 19

Let $G = (\Sigma, \Delta, \Gamma, P, Z)$ be a PL(k) graph grammar. A pair $(b, x), b \in \Delta, x \in \Gamma$, is called a *potential previous context* for a node label $a \in \Sigma$, if there exists *IE* graph $g = (V, E, \Sigma, \Gamma, \phi)$ belonging to a certain regular left-hand side derivation in G that: $(k, x, l) \in E, \phi(k) = b$, and $\phi(l) = a$.

Definition 20

A PL(k) graph grammar G is called an *ETPL(k)*, embedding transformation - preserving production-ordered k-left nodes unambiguous, *graph grammar*, if: for each production of the form

$$(l) \quad A \longrightarrow \begin{array}{cccc} X_1(1) & X_2(2) & \dots & X_m(m) \\ r_1 & r_2 & \dots & r_m \\ E_1 & E_2 & \dots & E_m \\ I_1 & I_2 & \dots & I_m \end{array}$$

where $X_a \neq X_b, a, b = 1, \ldots, m$, if (b, y) is a potential previous context for A, then there exists only one $(X_i, b, z, in) \in C_l(y, in), i \in \{1, \ldots, m\}$, where C_l is the embedding transformation of the lth production. If $i = 1$, then $z = y$, i.e. $(X_1, b, y, in) \in C_l(y, in)$.

For the ETPL(k) graph grammars, the efficient parsing algorithm has been defined in [7]. It means that we are able to control graph rewriting with efficient procedures, that is we are able to restructure meshes in a controlled manner and in an efficient way. Moreover, as we have mentioned it in a previous chapter, graph grammars allow us to simulate a context-sensitiveness (to some extent, of course). This simulation is possible due to a great generative power of the embedding transformation. Let us come back to our example from chapter 3 (Fig. 3d). In case of string grammars the only way to generate additional edges of a mesh (shown with dashed arrows) was to use three productions in neighbourhood cells. With ETPL(k) graph grammars we can make it with one production, because these context-bordered edges can be generated with the embedding transformation.

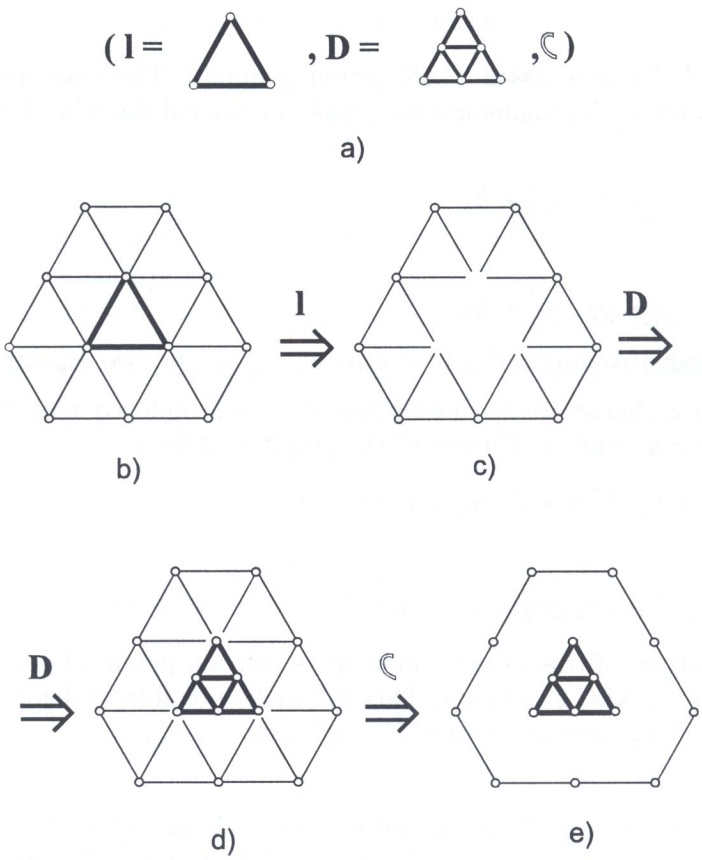


Fig. 4

The form of this production is shown in Fig. 4a, whereas a derivation process in Figs. 4b-e. First of all we identify the left-hand side graph l (see Fig. 4b), and we remove it as shown in Fig. 4c. At the next stage of the derivation step we put the right-hand side graph D at the place from which the graph l was removed (see Fig. 4d). At the last stage, shown in Fig. 4e, we generate with the embedding transformation all the needed edges (i.e. we restore the "old" edges and add the "new" ones). As a result we obtain a mesh restructured in the required way.

6.2. Programmed ETPL(k) graph grammars

Although we have just shown how ETPL(k) graph grammars can support context-sensitiveness, we will enhance them even more. In the last example we generated additionally a context that consisted only of mesh edges. In order to support modifying a context in the full meaning of this notion, that is edges and nodes of a graph, we have to define quasi-context ETPL(k) graph grammars. In case of string structures two ways of enhancing context-free grammars are known from the literature: Aho indexed grammars [1] and Rosenkrantz programmed grammars [14]. For graph grammars, a programming concept has been introduced by Bunke [2]. For ETPL(k) graph grammars a kind of programming has been introduced in [8] in order to increase the generative power of the ETPL(k) graph grammar generating structures representing CAD solid models for CAD/CAM integration purposes. Let us introduce a programming mechanism for our purpose of FE mesh generation.

Definition 21

A *programmed ETPL(k) graph grammar* is a six-tuple $G = (\Sigma, \Delta, \Gamma, P, Z, O)$, where $\Sigma, \Delta, \Gamma, Z$ are defined as for an ETPL(k) graph grammar, P is a finite set of productions of the form (l, D, C, π, F), in which (l, D, C) is an (underlying) ETPL(k)-type production, π is the predicate of a production applicability, F is the set of attributing functions, and O is the control automaton.

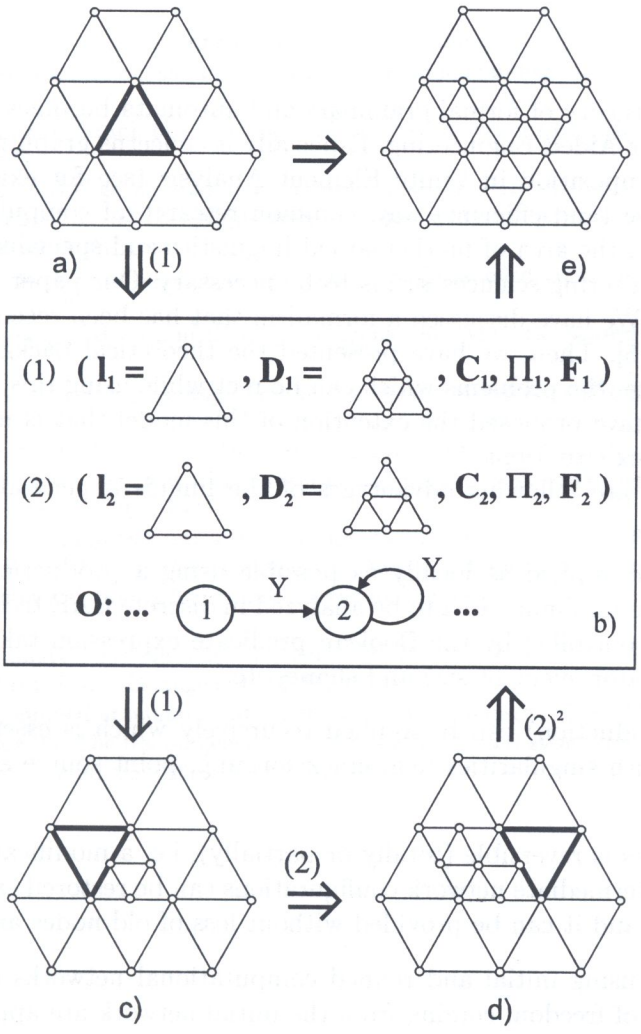


Fig. 5

We discuss the definition for an example shown in Fig. 5. Let us assume that we start with a mesh shown in Fig. 5a, and we want to control its restructuring with grammar productions in such a way that we receive exactly a structure shown in Fig. 5e. In a box we see two productions containing additionally: the predicate of production applicability and a set of attributing functions (both discussed in Chapter 3). These two additional elements of productions allow us to control applicability of productions locally. The control automaton, the part of which is shown in Fig. 5 enables us, however, to control a derivation (or rather to force a way of rewriting) in a global way. The part of the automaton shown in the figure can be interpreted in the following way.

If you have succeeded in applying production indexed with 1, i.e. you have exited a node indexed with 1 of the automaton with a transition labelled with "Y" (Yes — means a success in applying the production; contrary to a label "N" — No: meaning the predicate of applicability has not been fulfilled, so a production has not been applied), you have to try to apply the production indexed with 2 (only this one — not another). Then, let us notice that a success in applying a production 2 forces us to try this production once more (a transition labelled with "Y" comes back to the same node indexed with 2). Only if the predicate of applicability of the production 2 disables us to apply this production, we can go to other (invisible in the figure) parts of the control automaton, that is we can try to apply other productions.

Let us notice that the part of the automaton really controlls a derivation in the way that we receive the required restructured mesh shown in Fig. 5e.

7. Conclusions

The formalisms of the theory of formal grammars and automata becomes more and more popular in the area of Computer Aided Engineering. Especially it concerns graph models, which have been recently used for decomposition in Finite Element Analysis (see for example [11]). In order to use them in an effective (and efficient) way, common research of computer scientists conducting fundamental research in the area of mathematical linguistics and specialists in computer methods for mechanics and engineering sciences seems to be necessary. Our paper includes the first results of such a cooperation. We have discussed a formalism that has been recently used successfully for FE mesh generation [15]. Then, we have presented the theoretical background of this formalism pointing out, in advance, the problems which can be met while using this model for more complex cases. At the end, we have proposed the extention of this model that is more powerful, i.e. it can be used in more complex situations.

Summing up, there are following advantages of the linguistic generation/adoption formalism proposed in this paper.

- Modification can be applied as locally as possible using a production to a particular single element, which is the minimum area to be analyzed in discrete CAE technique. The applicability of a production is controlled by the Boolean predicate expression taking into account a local value of error estimator, element size and shape, etc.
- A modification (production) can be applied recursively which is essentially important in the case of problems with singularities (e.g. crack forming, point source and loading, plastic zone determination).
- A refinement process is reversible (totally or partially), i.e. a modification path is stored automatically and all intermediate network configurations can be restored easily. A derefined process is then very simple and it can be provided without loss of old nodes and degrees of freedom.
- Solutions obtained using initial and refined computational networks can be compared easily, because all degrees of freedom coming from the initial network are applied in the refined one.
- The adoption technique keeps global network features (e.g. the Delaney's feature) or it can control global features (e.g. a minimal angle in case of triangle networks).

- Grammar reductions are context-sensitive. It means, that breaking or gluing of each single element introduces a topological perturbation in its neighborhood. A topological context of a production to be applied is automatically recognized and additional proper productions are forced to the neighbourig elements.
- The formalism proposed is computationally efficient, which has been verified practically in such applicational areas as: image analysis, syntactic pattern recognition, and CAD/CAM [4, 5, 6, 7, 8].

These numerous advantages of the method proposed encourage the Authors to continue their research into the use of mathematical linguistics models for Computer Aided Engineering. Future results will be the subject of further publications.

Acknowledgment

This work was partially supported by the Polish State Committee for Scientific Research under Grant No. 8 T11C 036 08.

References

[1] A.V. Aho. Indexed grammars — an extension of context-free grammars. *J. ACM* **15**: 647–671, 1968.
[2] H.O. Bunke. Graph grammars as a generative tool in image understanding. *Lecture Notes in Comp. Sci.* **153**: 8–19, 1982.
[3] K. Culik II, A. Lindenmayer. Graph 0L - systems and recurrence systems on graphs. *Proc. 8th Conf. Systems Science*, Hawaii, Jan. 1975.
[4] M. Flasiński. Parsing of edNLC-graph grammars for scene analysis. *Pattern Recognition* **21**: 623–629, 1988.
[5] M. Flasiński. Characteristics of edNLC-graph grammars for syntactic pattern recognition. *Comput. Vision Graphics Image Process.* **47**: 1–21, 1989.
[6] M. Flasiński. Distorted pattern analysis with the help of node labelled controlled graph languages. *Pattern Recognition* **23**: 765–774, 1990.
[7] M. Flasiński. On the parsing of deterministic graph languages for syntactic pattern recognition. *Pattern Recognition* **26**: 1–16, 1993.
[8] M. Flasiński. On the use of graph grammars for the description of mechanical parts. *Computer Aided Design* **27**, 1995.
[9] P.L. George. *Automatic Mesh Generation. Application to Finite Element Methods*, Wiley, 1991.
[10] D. Janssens, G. Rozenberg, R. Verraedt. On sequential and parallel node-rewriting graph grammars. *Comput. Graphics Image Process.* **18**: 279–304, 1982.
[11] A. Kaveh, C.R. Roosta. A graph-theoretical method for decomposition in Finite Element Analysis. *Proc. Advances in Parallel and Vector Process. for Structural Mechanics.* Edinburg, Scotland: 35–42, 1994.
[12] A. Lindenmayer. Mathematical models for cellular interactions in development. *Journ. Theor. Biol.* **18**: 280–315, 1968.
[13] M. Nagl. Graph - Lindenmayer - systems and languages. *Arbeitsberichte des Instituts für Mathematische Maschinen und Datenverarbeitung.* Uniw. Erlangen, **1**: 16–63, 1975.
[14] D.J. Rosenkrantz. Programmed grammars and classes of formal languages. *J. ACM* **16**: 107–131, 1969.
[15] T. Łukasiak, K.H. Żmijewski. The fractal generator of FE meshes for 2D structures. *Proc. XI Pol. Conf. Comp. Meth. Mechanics.* Kielce, Poland: 551–560, 11–14 May 1993.