Similarities Between String Grammars and Graph Grammars

Silviu Razvan Dumitrescu

***Abstract*—**In this paper, we present some studies about relations existing between well known Chomsky string grammars and graph grammars, in particularly hypergraph grammars. We are discussing about deterministic context free Lindenmayer Systems used to describe commands to a device that generates black and white digital images. Instead of well-known methods of drawing, we will paint squares, not lines. After that, we give some important properties of growth functions of D0L-systems. In addition, we turn the discussion to gray scale or color digital image generation. The second main part of the paper is about normal forms of hyperedge replacement grammars. In context freeness of these grammars, we can transform each of it into an equivalent grammar without λ-productions and without rewritings. After that, in a nondeterministic way, we will create equivalent grammars in Chomsky Normal Form or Greibach Normal Form. Both normal forms are inspired by string grammars. In the third part of this paper we illustrate some important differences between graph grammars and hypergraph grammars in context of freeness. On the other hand, we give a possibility to transform the planar structure of a hypergraph into a linear one with concern of determinism. This can create a path to transform a pushdown automaton into a generative grammar equivalent.

***Keywords*—**Chomsky, Context Freeness, Deterministic, Deterministic Context Free Lindenmayer Systems (D0L-systems), Digital Pictures, Graph Grammars, Greibach, Growth Functions, Hyperedge Replacement Grammars, Nondeterministic, Nondeterministic, Normal Form, Turtle.

I. Introduction

THE concepts and techniques of picture processing have been arisen from many different disciplines, among them mathematics, computer science [4], engineering or biology. L-systems are suitable tools for drawing images of real life structures. This is due to their ability to model biological growth. Traditionally, the strings of symbols generated by L-systems are interpreted as images either using vector interpretation, or, so-called, turtle geometry interpretation. In this paper, we consider the second approach, where the symbols are translated into commands to a 'turtle', which is a simple device moving on the plane and used in drawing pictures.

Manuscript received August 12, 2011.

Silviu Razvan Dumitrescu is from Department of Informatics, Faculty of Mathematics and Informatics, Transilvania University of Brasov, Iuliu Maniu 50, ROMANIA(silviu.dumitrescu@unitbv.ro), holds a career over 15 years in education and is interested in areas such as formal languages, modern programming languages .

On the other hand, in many fields of computer science, diagrams rather than strings represent the information. That is why a study in domain of graphs and formalizations of graphs could be very interesting. A hypergraph represents a generalized graph and consists by a number of hyperedges [1]. A hyperedge is an atomic item labeled with a label in a nonempty set, called alphabet, and a fixed number of tentacles. On each tentacle is attached a node. Nodes are involved in hyperedge replacement. With labeled hyperedges, we can define productions. Productions consist by a label in left hand side and a replacing structure in right hand side. If a labeled hyperedge, with the left hand side in the productions set, is replaced by the right hand side of the same production this is called a direct derivation. Therefore, we can define a language, which represents the set of structures derivable from the start structure.

Graphical languages are playing an important role in the definition of images or visual structures. Parts of these languages are based on hypergraph structures. Roughly speaking a hypergraph generalizes the notion of graph, and consists by a number of atomic items called hyperedges. Thus, a hyperedge generalize the notion of edge. An edge consists of two nodes (called source and destination, if oriented), but the hyperedge consists by a finite number of nodes. In the latter, some nodes have a special role in the derivation process.

In area of studying complex structures which describing images it is interesting to develop techniques and tools that allow graphical language designers to define and implement visual languages analogously to what already happens for string languages. The main problem of two-dimensional languages is the parsing process. Hypergraphs and hyperedges are complex objects and we have to provide new techniques of parsing, which involving not just deterministic chaining, given by concatenation.

A systematic development of graph grammars and graph language theory requires the notion of context-free graph grammars and languages. It is known that exist mechanisms for deriving graphs from graphs by applying productions. Hyperedge replacement works locally without any effect on the context of the hyperedge replaced. This result, formulated in [1], provides evidence that hyperedge systems represent a graph grammar version of context-freeness.

Some hyperedge grammars have only one set of labels [2]. In that case, the set of nonterminals is empty and the terminal structures are not labeled. In some of this grammars we can

consider derivations maximum parallel such as are in Lindenmayer systems. The languages generated by such grammars include visual structures like fractals.

In the first part of introduction section, we present the main concepts used in this paper: context free Lindenmayer systems, a special case of them called deterministic systems (D0L-systems), and digital pictures, which are formed by units. As we will see, a unit is a square defined by position, length and color.

We introduce the main concepts related to hyperedges, graphs and hypergraphs.

After that, in introduction, we consider the turtle device, which works as a drawn device. The commands for turtle are described by a D0L-system. Considering Sierpinski triangle, we give an example of its generation. The main difference between this method and the others is that instead of drawing lines we are filling units. The problem with turtle, in this case, is that we have to scale the dimension of the unit in order to draw the image, with more details, in the same frameset.

In the first main section of this paper, we will study some important properties of growth functions defined for those D0L-systems that are describing turtle movements. We discuss about a recursive formula based on the Cayley-Hamilton theorem.

After that, we extend the discussion at gray-scale or color images generation. This is doing by introducing new attributes in the tuple that defines the turtle state.

In the next part of the paper, we will consider the alphabet of labels divided into two disjoint sets: the alphabet of terminals, which labels only structures as right hand side of some productions, and the alphabet of nonterminals, which labels structures as both sides of productions, same as in string grammars.

In this paper, all of the grammars considered are context free. Therefore, it does not matter how we choose the starting hyperedge in the replacement and it is not relevant how many times we repeat the replacement, but it is important to have, in each step of the derivation, a production where the label of the replaced hyperedge exists on its left side.

We continue the main section with considering grammars without λ-productions and without rewritings. As it has shown in [8] this could be obtaining by starting from a regular grammar. The algorithm is nondeterministic, that means it does not matter how we will split the left side of the production because the choice does not influence the result.

In the last sections of the paper, we will make a survey about generative power of hyperedge replacement grammars and about some essential properties related to context freeness. After that, our discussion is about some particular hyperedge grammars called graph grammars. In the end of the paper, we turn our discussions about close relations existing with context free string grammars.

II. Problem Formulation

A. Definitions and notations

We start this section with basic definitions involved in this paper.

Definition 1: [6] An 0L-system (context free Lindenmayer system) represents an ordered tuple (V, P_0, F), where V is the finite, nonempty set of symbols, usually called alphabet, P_0 is a nonempty word over V, called initial word or axiom, and F represents a set of ordered pairs called productions.

$$F = \{(a, P) \mid a \in V, P \in V^{*\,1)}\}$$

We usually denote a production by $a \rightarrow P$.

The main three differences between L-systems and well-known string grammars are:

- there is a unique alphabet (without separation between terminal and nonterminal symbols)
- for each symbol, $a \in V$, exists at least a production in F
- a step in the derivation chain represents replacement of all symbols of current word by productions in F.

The prefix 0, in 0L-systems, denotes the property of context freeness. Thus, the replacement of a symbol does not care about the context where the symbol exists inside of the word.

In this paper, we consider a special case of 0L-systems, called D0L-systems, deterministic systems, which mean that for each symbol, we have exactly one production in F.

Definition 2: A digital image $\sum$ is a finite rectangular array whose elements are called units. Each unit, $U = (x, y, l)$, of $\sum$ is a square defined by a pair of Cartesian coordinates $(x, y) \in \mathbf{R}^2$ (down-left corner) and a length $l \in \mathbf{R}$, which is constant.

A unit, U, in a digital image $\sum$, has two types of neighbors:

- its four horizontal and vertical neighbors: $U_l = (x_l, y_l, l)$, $U_r = (x_r, y_r, l)$, $U_u = (x_u, y_u, l)$, $U_d = (x_d, y_d, l)$ such that $|x-x_i|+|y-y_i| = 1$, $i \in \{l, r, u, d\}$
- its four diagonal neighbors: $U_{ul} = (x_{ul}, y_{ul}, l)$, $U_{ur} = (x_{ur}, y_{ur}, l)$, $U_{dl} = (x_{dl}, y_{dl}, l)$, $U_{dr} = (x_{dr}, y_{dr}, l)$ such that $|x-x_i| = |y-y_i| = 1$, $i \in \{ul, ur, dl, dr\}$

We shall refer to the neighbors of first type as the 4-neighbors of U and the neighbors of both types, collectively, as the 8-neighbors of U. The former neighbors are said to be 4-adjacent to U, and the latter, 8-adjacent. Note that if U is on the border of $\sum$, some of its neighbors may not exist.

We continue with some basic definitions involved in the second part of the main section of this paper.

Definition 1: [4] A hypergraph, H, is a tuple $(V_H, E_H, att_H, lab_H, ext_H)$, where:

- V_H is the finite set of nodes

- E_H is the finite set of hyperedges
- att_H: $E_H \rightarrow V_H^{*}$ [1] is the application of attaching, which assigns a sequence of pair wise distinct nodes to every hyperedge
- lab_H: $E_H \rightarrow C$ is the application of labeling, which assigns a label to every hyperedge from arbitrary but fixed and not empty set C, and $ext_H \in V_H^{*}$ is a sequence of pair wise distinct external nodes.

Definition 2: [4] The type of a hyperedge, $e \in E$, is the application type: $C \rightarrow \mathbf{N}$ with $type(lab(e)) = |att(e)|$ [2].

For a hypergraph, H, the type of H, type(H), is the number of external nodes, $type(H)=|ext_H|$.

Let $H = (V_H, E_H, att_H, lab_H, ext_H)$ be a hypergraph and R a hypergraph over the same set of labels as H. By $H[e|R]$ we understand the hypergraph obtained from H by replacing hyperedge e, $e \in E_H$, with the hypergraph R. The replacing process is made by cutting the hyperedge e from H and adding the hypergraph R so that the i-th external node of R is glued over the i-th attached node of e with i = 1,type(e). Moreover, $ext_{H[e|R]} = ext_{H,}$ $type(H[e|R]) = type(H)$.

Definition 3: [4] A hyperedge replacement grammar, HRG, is a tuple (N, T, P, S), where:

- N is the set of nonterminal labels
- T is the set of terminal labels, $N \cap T = \varnothing$
- P is the set of productions, $P = \{(A, R) | A \in N$, R is a hypergraph labeled in $N \cup T$ [3], with $type(A^{\bullet})$ [4]$= type(R)\}$
- $S \in N$ is the label of the starting symbol.

A direct derivation in HRG, using productions from P, $H \Rightarrow H'$, takes place if and only if exists $e \in E_H$ such as $(lab_H(e), R) \in P$ and $H' = H[e|R]$.

The language generated by the hyperedge replacement grammar HRG is $L(HRG) = \{H| \exists S^{\bullet} \Rightarrow_P^{*} H$ [5], $lab_H(e) \in T \ \forall e \in E_H\}$.

Definition 4: A graph is a system (E, V), where $E \subseteq V \times V$ is called the set of edges, and V is called the set of nodes.

A directed graph introduces two functions s, d: $E \rightarrow V$ which attaching a source and a destination to each edge. In this case, the edge (1, 2) is different by the edge (2, 1). In the case of general graphs this does not happened.

In addition to these functions, we consider the map of labeling, l: $E \rightarrow C$, which attaching a label from C to each edge of E.

Because we use graphs and edges in the process of derivation, we introduce two special nodes *begin*, *end* $\in$ V. A graph without those nodes is known as underlying graph.

In the derivation process, we consider the set of labels divided into two disjoint sets: nonterminals, N, and terminals, T.

Definition 5: [7] A production over N is an ordered pair p = (A, R), where: $A \in N$ is the label of the edge removed from the existing graph, and R is the graph which will replace the removed edge.

The process of direct derivation can be described as follows: the edge e, l(e) = A, will be removed and the replacing graph R will glue its nodes, $begin_R$ with s(e) and end_R with d(e). A derivation represents a chain of valid direct derivations.

Based on previous assertion we consider a graph grammar a tuple GG = (N, T, P, Z), where P represents the set of productions, and $Z \in N$ the axiom. In a graph grammar, a derivation starts with Z and ends with a graph labeled only in T.

Starting from graph definition, we can give the following definition of a hypergraph.

Definition 6: A hypergraph represents a generalized graph, that means a system (HE, V), where $HE \in V^{*}$ [1] and called hyperedge. The process of mapping between hyperedges and nodes is doing by the function a: $HE \rightarrow V^{*}$.

For hypergraphs we do not have the notion of directed hypergraph, but we can introduce an order over V such that, the nodes of a hyperedge having a certain order. This order will be useful in parsing process.

The process of labeling does not give any news comparing with the process described previously for graphs. Each hyperedge has a label from C, l: $HE \rightarrow C$.

For each hyperedge, we will consider a special subset of pair wise distinct nodes, named external nodes, e_H, used in the derivation process. The type of a hypergraph H represents the number of external nodes.

Let $H = (E_H, V_H, a_H, l_H, e_H)$ be a hypergraph, and R a hypergraph over the same set of labels as H. By $H[e|R]$ we understand the hypergraph obtained from H by replacing hyperedge e, $e \in E_H$, with the hypergraph R. The replacing process is made by cutting the hyperedge e from H and adding the hypergraph R, so that the i-th external node of R is glued over the i-th attached node of e, with i = 1, type(e). Moreover, $e_{H[e|R]} = e_{H,}$ $type(H[e|R]) = type(H)$.

---

[1] For a set V, V^{*} denotes the set of all strings over V, including the empty string λ; $V^{+} = V^{*}-\{\lambda\}$ denotes the set of all strings over A except the empty string λ.

[1] For a set V, V^{*} denotes the set of all strings over V, including the empty string λ; $V^{+} = V^{*}-\{\lambda\}$ denotes the set of all strings over A except the empty string λ.

[2] For $w \in V^{*}$, |w| denotes the length of w.

[3] A λ-production is defined in [3].

[4] $A^{\bullet}$ represents a hypergraph with one hyperedge labeled with A.

[5] $\Rightarrow_P^{*}$ represents a derivation in k steps, $k \geq 0$, with productions from P (reflexive and transitive closure)

We can consider C split into two subsets, terminal and nonterminals, and the definition of a hypergraph grammar is quite the same with the one for graph grammar, but respecting the particularities of hypergraphs.

B. The drawn device

Lets turn back to Lindenmayer systems and consider de drawn device.

As we saw, small units compose digital images and we need a device to draw them.

We call this device "turtle". A turtle has mobility to move inside of the Euclidean plan. The turtle state is given by the triple (x, y, α), where (x, y)$\in \mathbf{R}^2$ are the plan coordinates and $\alpha \in \{0, \pi/4, \pi/2, 3\pi/4, \pi, 5\pi/4, 3\pi/2, 7\pi/4\}$ is the direction of turtle movement.

A simple move of the turtle means advance one unit in the direction given by α. The simple move can be with or without drawing depends, the turtle is up or is down.

In the first step, we will draw only black and white digital images. For this, we will consider background colored in white and the image colored in black.

The movements of the turtle, in order to draw a black and white digital image, can be described by a D0L-system, D = (V, P_0, F), where V = {U, u, +, -} and P_0 = {U}. The symbols of V have the following meaning:

- U, simple move with drawing (we say that the turtle is down)
- u, simple move without drawing (we say that the turtle is up)
- +, rotate the drawing direction with angle α, counterclockwise
- -, rotate the drawing direction with angle α, clockwise.

In addition, the initial state of the turtle can be modified for each individual case, but most likely it is (0, 0, $\pi/2$).

Note that the only change to the original turtle interpretation by Prusinkiewicz is the interpretation of U: Instead of drawing lines, we paint black squares.

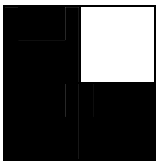
Example 1: The D0L-system that describes the Sierpinski triangle is defined by the tuple D = (V, P_0, F), where F = {(U, U-u-u++U+u-U), (u, uu), (+, +), (-, -)}. The initial state of the turtle is the usual one. A derivation in three steps using these productions looks as follows:

U =>
U-u-u++U+u-U =>
U-u-u++U+u-U-uu-uu++U-u-u++U+u-U+uu-U-u-u++U+u-U
=>U-u-u++U+u-U-uu-uu++U-u-u++U+u-U+uu-U-u-u++U+u-U-uuuu-uuuu++U-u-u++U+u-U-uu-uu++U-u-u++U+u-U+uu-U-u-u++U+u-U+uuuu-U-u-u++U+u-U-uu-uu++U-u-u++U+u-U+uu-U-u-u++U+u-U

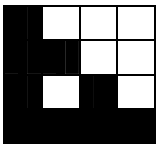
During derivation process the word w_k, obtained after $k \geq 1$ steps (w_0 = U), will be represented inside of a white square formed by $2^k \times 2^k$ units. When k increases, the image details increase. However, when k increasing the image will not be properly represented because it will have huge dimension. The simple solution is to scale the image at each step, which means reducing turtle unit dimension. The minimum dimension of the turtle unit is one pixel.

When k approaches infinity, the image is full detailed, but in real representation, this cannot be doing, so we have to consider the finite number of steps until the turtle unit approaches one pixel.

In the previous example, we start with the word w_0. This is representing by a black square inside of a frame by q × q pixels. After first derivation step, the turtle unit will be scaled such as we can draw four units inside of the frame. w_1 is represented as following:



w_2 is formed by 16 units. The process is simple. Each black unit is scaled such as instead of one initial unit we create four new small units colored by the same pattern. The new image, after two steps, is looking as following:



The scale factor is ½.

Continuing the derivation process, after k steps, initial frame is formed by $2^k \times 2^k$ turtle units.

We say about a point $(x, y) \in [0, q] \times [0, q]$ that is in the image generated if it is inside of a unit square colored in black.

We define the image generated on step k by:

$$T(w_k) = \{(x, y) \mid (x, y) \text{ is in a black unit of } 2^k \times 2^k \text{ units}\}$$

If the sequence $T(w_0)$, $T(w_1)$,… converges in the standard Haussdorf metric, the limit

$$T(D) = \lim_{k \to \infty} T(w_k)$$

is the infinite resolution image defined by the D0L-system D with scaling defined previously.

C. Future considerations about context freeness

As we mentioned, in this paper, all of the hyperedge replacement grammars are context free with any other specifications. In the context freeness, we underline two major results obtained in [3] and [4].

First, for every hyperedge replacement grammar, HRG = (N, T, P, S), without rewritings[6] and λ-free, exits an equivalent grammar, HRGNF = (N_1, T, P_1, S), in Chomsky Normal Form. That means all productions in P_1 are by the form (A, H), where $A \in N_1$ and $|E_H| = 1$, $lab(e) \in T$, $e \in E_H$ or $|E_H| = 2$, $lab(e_i) \in N$, $e_i \in E_H$, i = 1,2.

Secondly, for every hyperedge replacement grammar in Chomsky Normal Form, HRGNF = (N_1, T, P_1, S), exits an equivalent grammar, GNF = (N_2, T, P_2, S), in Greibach Normal Form. That means all productions in P_2 are by the form (A, H), where $A \in N_2$ and lab_H: $E_H \rightarrow T \cup N_2$, $|E_H| \geq 1$ with exactly one hyperedge labeled in T.

The hierarchies of nonterminal and production sets are: $N \subseteq N_1 \subseteq N_2$, $P \subseteq P_1 \subseteq P_2$

We continue with some parallel discussions about graph grammars freeness and hypergraph grammars freeness.

In the graph case, the replaced object is an edge and the replacing object is a graph. In the hypergraph case, the replaced object is a hyperedge and the replacing object is a hypergraph.

As is proof in [7] we can consider the Context-Freeness Lemma, which proofs that under an isomorphism, the derivation steps in context free string grammars can be simulate by direct derivations in context free graph grammars. The same consideration can be find in [8] related to hypergraphs.

In previous work ([3], [4], [5], [6]) we proved some special properties for context free hyperedge replacement grammars: Chomsky Normal Form, Greibach Normal Form, and relation between this grammar and pushdown automata.

In [7] is proved that properties are not true for graphs grammar without loosing the generative power.

In the main part of the next section, we prove that the generative power is not affected using the structure defined by a hyperedge replacement grammars

III. Growth Functions Of Systems That Generates Images

An important question when we deal with the D0L-systems is the length of their words. The growth function $f_D: \mathbf{N} \rightarrow \mathbf{N}$ of the D0L-system D is defined by:

$$f_D(k) = |w_k|$$

and means the length of the word w_k.

By studying growth functions, we determine which types of biological growth D0L-system is capable of modeling.

Here we discuss a special matrix representation of homomorphism introduced [8] to help growth function calculation.

Definition 7: The incidence matrix of a homomorphism F is defined by the following square matrix (the dimension of the matrix is equal to number of symbols in alphabet):

$$M(F) = (m_{a,b})_{a,b \in V},\ m_{a,b} = |F(b)|_a$$

The element $m_{a,b}$ represents the number of occurrences of symbol a in the production where b is the left hand side member.

In *Example 1*, we have matrix M defined as following:

$$M(F) = \begin{pmatrix} 3 & 0 & 0 & 0 \\ 3 & 2 & 0 & 0 \\ 3 & 0 & 1 & 0 \\ 3 & 0 & 0 & 1 \end{pmatrix}$$

Recall now the Parikh vector, which describes the letter distribution of a word. Let η be a column vector with all elements 1 and Π the Parikh vector of the axiom P_0. We can now discuss an important identity regarding the growth function of D0L-systems:

$$f_D(k) = \eta^T M(F)^k \Pi^T \quad (1)$$

that calculates the length of the word w_k, after k steps, starting with the axiom.

Remark: in the previous formula we used an important result proofed in [7]: $M(F^k) = M(F)^k$, which means that the incidence matrix after k derivations equals the incidence matrix after one derivation on power k.

Now we define a recursive formula for the growth function starting from the Cayley-Hamilton theorem. This says that every square matrix over the real or complex field satisfies its own characteristic equation.

Therefore, for a matrix of dimension n, we can calculate de coefficients, $c_i \in \mathbf{R}$, i = 1, n, such as the Cayley-Hamilton formula is:

$$M^n = c_1 * M^{n-1} + \ldots + c_{n-1} * M + c_n * M^0, \text{ where } M^0 = I_n,$$

If we consider that M is the incidence matrix, we apply previous formula to (1)

$$\eta^T M(F)^n \Pi^T = c_1 \eta^T M(F)^{n-1} \Pi^T + \ldots + c_{n-1} \eta^T M(F) \Pi^T + c_n \eta^T M(F)^0 \Pi^T$$

that means:

$$f_D(n) = c_1 f_D(n-1) + \ldots + c_{n-1} f_D(1) + c_n f_D(0), \text{ where } n = |V|.$$

We conclude the previous results into next theorem.

[6] A rewriting is defined in [3].

Theorem 1: For the growth function of D0L-systems, D = (V, P_0, F), we can find a recursive formula given by:

$f_D(n+i) = c_1 f_D(n+i-1) + \ldots + c_{n-1} f_D(i+1) + c_n f_D(i)$, where n=|V|, i≥0.

For *Example 1* the previous formula coefficients are $c_1 = -7$, $c_2 = 17$, $c_3 = -17$, $c_4 = 6$ and the recursive formula is:

$f_D(i+4) = -7f_D(i+3) + 17f_D(i+2) - 17f_D(i+1) + 6f_D(i)$, i≥0 with $f_D(3)$, $f_D(2)$, $f_D(1)$, $f_D(0)$ given values.

To draw gray-tone images using L-systems we can modify the turtle state by adding the weight, g. The weight can be an arbitrary real number, initially 1. Instead of painting always a black square, the turtle paints a grey square, whose darkness is giving by the current weight of the turtle. The local grayness function f: $\mathbf{R}^2 \rightarrow \mathbf{R}$ is describing the darkness of every point of the plan. Initially the plane is completely white, that is, the local grayness function is f(x, y) = 0, for all, x, y∈**R.** The weight of the turtle is simply adding to the darkness of the part f'(x, y) = f(x, y) + g. All concepts described for black and white images remain unchanged.

IV. Pushdown Automata For Hyperedge Replacement Languages

In string grammars, we have pushdown automata as counterpart of context free string grammars.

Inspired by this device we introduce in this paper an automaton to analyze a hyperedge replacement language. We proof that for each hyperedge replacement grammar we can construct a nondeterministic pushdown automaton and reverse for each pushdown automata we can build a context free hyperedge replacement grammar. After that, we proof the equivalence between the language accepted by pushdown automata and the language generated by hyperedge replacement grammar.

First, we transform the planar structure of the hypergraph into a linear one. For this we use an algorithm which scans the hypergraph. It starts with source hyperedge, and using the list of attached nodes, continues with adjacent unvisited hyperedge. Every time when we find an unvisited hyperedge we add its label into a list of visited labels. The algorithm stops when all hyperedges were visited.

Let H = (V_H, E_H, att_H, lab_H, ext_H) be a hypergraph. We consider each attached node marked with i, where i=1, $|att_H|$. We use next algorithm to obtain the linear parsing of its labels, $LIST_H$.

```
1. LINEAR(H, LIST_H)
2.    LIST_H←∅
3.    for each hyperedge e∈E_H do
4.       visit[e]←false
5.    endfor
6.    Q←∅
7.    source←first(E_H)
8.    visit[source]←true
9.    add(LIST_H, lab(source))
10.   for each node x∈att_lab(source)• do
11.      ENQUEUE(Q,x)
12.      att_H←att_H-{x}
13.   endfor
14.   while Q and att_H are not empty do
15.      if Q is empty
16.         source←next_unvisited(E_H)
17.         visit[source]←true
18.         add(LIST_H, lab(source))
19.         for each node x∈att_lab(source)• do
20.            ENQUEUE(Q,x)
21.            att_H←att_H-{x}
22.         endfor
23.      endif
24.      x←head(Q)
25.      DEQUEUE(Q)
26.      for each hyperedge e∈E_H do
27.         if not visit[e] and x∈E_lab(e)•
28.            add(LIST_H, lab(e))
29.            for each node y∈att_lab(e)• do
30.               if y∈att_H
31.                  ENQUEUE(Q,y)
32.                  att_H←att_H-{y}
33.               endif
34.            endfor
35.         endif
36.      endfor
37.   endwhile
```

Procedure LINEAR works as follows. In lines 3-4 assign the value false, in vector visit, to every hyperedge. Q is the queue where are stored parsed nodes. Initial Q is empty (line 6). Variable source denote the entrance of the hypergraph. We can consider the entrance of the hypergraph one of the hyperedges adjacent with the node marked with 1 (line 7). Corresponding to this adjust the value of the visit with true (line 8), add its label to the list, $LIST_H$ (line 9), insert all attached nodes in Q and remove them from att_H (lines 10 -13).

The main loop of the algorithm is contained in lines 14 -37. The loop iterates as long as are nodes in Q and the set of attached nodes is not empty. The first condition is necessary to test if all nodes of current hypergraph were visited and the second condition is used in case of multiple hypergraph connected components (exits nodes unreachable from current hyperedges). When Q is empty, it searches for a new component (line 16). The new source can be one of the hyperedges, which has attached the first unvisited node marked with higher index. In lines, 17-23 follows the same steps as it did in lines 8 - 13. The loop continues with extracting the head of the Q (line 24). In lines 26-36 takes all hyperedges adjacent with the node x. If the hyperedges are not yet visited inserts

them labels into $LIST_H$ (line 28) and all them attached nodes, which are still in att_H inserts into Q.

Next, we analyze the running time on the input hypergraph H. After initialization, no visit element is ever false, and thus the test in line 27 ensures that each hyperedge is visited only once. So, the total time to visit all hyperedges is $O(|E_H|)$. Every hyperedge brings with it all its attached nodes, which are in att_H in the same time. Those nodes are enqueued in Q and removed from att_H. The operation of enqueuing and dequeuing take O(1) time. Because a node is only ones enqueued and dequeued the total time devoted to queue operations is $O(|att_H|)$. Thus, the total running time of LINEAR is $O(|E_H|\cdot|att_H|)$ which means square time complexity.

With previous procedure we can transform into a linear structure every hypergraph even it's connected or not.

Definition 8: A nondeterministic pushdown automaton for hyperedge replacement languages, PDAH, is a system $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, where:

- Q is a finite set of states;
- Σ is an alphabet, called the input alphabet
- Γ is an alphabet, called the stack alphabet
- $q_0 \in Q$ is the initial state
- $Z_0 \in \Gamma$ is the start symbol
- $F \subseteq Q$ is the set of final states
- δ: $Q\times(\Sigma\cup\{\varepsilon\})\times\Gamma\rightarrow\wp(Q\times\Gamma^*)$ is the transition function.

Instantaneous description of a PDAH is a triple (q, w, γ), where $q\in Q$, w is a string of input symbols and γ is a string of stack symbols.

A transition of PDAH from $(q, w, Z\alpha)$ to $(p, w', \beta\alpha)$, denoted by $(q, aw, Z\alpha) \vdash (p, w, \beta\alpha)$, exists if and only if $(p, \beta)\in\delta(q, a, Z)$, where a may be ε or an input symbol.

We define the language accepted by empty stack of PDAH to be:

$$\{w|(q_0, w, Z_0) \vdash^* (p, \varepsilon, \varepsilon), p\in Q\}$$

Let now GNF = (N, T, P, S), be a hyperedge replacement grammar in Greibach normal form. We transform derivations, which use productions of P, into transitions, which use δ function from PDAH.

V. Generative Power Of Hyperedge Replacement Languages

Definition 9: A graph grammar is called m-bounded, $m \in \mathbf{N}$, if the right-hand side of each production (A, R) has at most m nodes.

Theorem 2: [7] For each $m \in \mathbf{N}$ there exists a graph with nodes of connection, G, such that the language generated by G cannot be generated by an m-bounded graph grammar.

Because finite graph languages are context-free, the unbounded theorem says that the number of nodes in right side of productions cannot be bounded without reducing the generative power of replacement system. On the other hand is known that each edge in a graph has exactly two nodes. Keeping in mind the first assertion, means that each production has an unbounded number of labeled edge on right side without reducing the generative power. The immediate conclusion is we cannot transform a graph context free grammar into a normal form. This is the first main difference between graph grammars and string grammars.

In [3] we gave an example, which show the generative power of hyperedge replacement grammar. The language $L = \{(a^n b^n c^n), n>=1\}$. This language cannot be generated by a string grammar context free.

Definition 10: A hypergraph grammar is called m-bounded, $m \in \mathbf{N}$, if the right-hand side of each production (A, R) has at most m labeled hyperedges.

The previous definition takes care of the fact that each hyperedge in a hypergraph can have a finite number of nodes.

Theorem 3: In case of hypergraphs, grammars are bounded.

Proof: With respect of generative power, each context free hypergraph grammar uses a production, which increases the number of nodes. Let $H[e|R]$ that one, where e is a hyperedge and R a hypergraph. The only condition necessary to respect in the derivation process is that $e_{H[e|R]} = e_H$. So, we can consider that type(e) < type(R). In such a way, the total number of nodes increases.

With this theorem we proved that we can introduce the Chomsky normal form and in relation with this, the Greibach normal form.

Next step usually done next with string grammars is to find a push down automata, which recognized the language generated by a free context grammar. We gave an algorithm about that in [5]. The main part of the algorithm is parsing the hyperedges such that transform the planar structure into linear one. We assumed there that the solution of algorithm is not deterministic.

In this paper we consider the indexing mapping for nodes, which attach to each node a certain order in set of nodes. Let us say that we have the edge e with attached nodes $v_i v_j \ldots v_k$. Using the indexing mapping, we have v_i of index less than v_j in previous list of attached nodes.

In the same way, we consider the indexing mapping for labels, which attach to each label a certain order in set of labels.

The algorithm described in [5] will produce one deterministic linear solution. Therefore, we can generate one and just one push down automata, which recognizing the language generated by a hypergraph context free grammar with conditions: hypergraph be connected and establish one hyperedge to be the first one. The algorithm is presented bellow:

```
1. LinearDeterministic(H, LIST_H)
2.   LIST_H←∅
3.   for each hyperedge e∈E_H do
4.     visit[e]←false
5.   endfor
6.   Q←∅
7.   source←first(E_H)
8.   visit[source]←true
9.   add(LIST_H, lab(source))
10.    while att_lab(source)• has nodes do
11.      x = next node of att_lab(source)•
12.      ENQUEUE(Q,x)
13.    endwhile
14.    while Q is not empty do
15.      x←head(Q)
16.      DEQUEUE(Q)
17.      while E_H has labels do
18.        e = next label of E_H
19.        if not visit[e] and x∈E_lab(e)•
20.          add(LIST_H, lab(e))
21.          while att_lab(e)• has nodes do
22.            y = next node of att_lab(e)•
23.            if y∈att_H
24.              ENQUEUE(Q,y)
25.              att_H←att_H-{y}
26.            endif
27.          endwhile
28.        endif
29.      endwhile
30.    endwhile
```

The algorithm is a version of that presented in [5] with the specification of unique solution. This will lead us to the unique push down automata and after that to the reverse construction: having a push down automata to create a unique generative grammar.

VI. Conclusion

Digital images can be decomposing in units, and can be drawn with special devices. The movements of those devices can be described by D0L-systems. We can find a recursive formula for growth functions that describes the word generated during the derivation. We can draw not only black and white images, but grayscale or colored images. Drawing of color images is based on weight turtles. We can introduce the forth member in the tuple that describes the state of the turtle for grayscale images and two more attributes to describe color images, corresponding to RGB codification of colors.

Context free hyperedge replacement grammars have a behavior very much like context-free Chomsky grammars. The important difference is related to transformation of a planar structure into a linear one. All the algorithms involved in transformations are nondeterministic.

In the end of the paper, we gave a possibility to create an equivalent generative grammar starting from a pushdown automaton.

References

[1] S. Dumitrescu**,** About Normal Forms for Hyperedge Replacement Grammars, *New Aspects of Computers, Proceedings of the 12th WSEAS International Conference on Computers, July 23-25, Heraklion, Greece*, 2008, ISSN 1790-5109, ISBN 978-960-6766-85-5, pp. 208-211

[2] S. Dumitrescu**,** *Hyperedge Replacement Languages and Pushdown Automata*, Recent Advances in Mathematical and Computational Methods in Science and Engineering, Proceedings of the 10th WSEAS International Conference on Mathematical and Computational Methods in Science and Engineering, nov 7-9, Bucharest, Romania, 2008, ISSN 1790-2769, ISBN 978-960-474-019-2, pp. 487-490

[3] S. Dumitrescu, *Aspects of Context Freeness for Hyperedge Replacement Grammars*, Foundations of Computing and Decision Sciences, Vol. 30, No. 2, pg. 91 – 101, 2005

[4] S. Dumitrescu**,** *Structuri vizuale si Limbaje Formale*, Editura Universitatii Transilvania din Brasov, 2006, ISBN (10) 973-635-857-7; (13) 978-973-635-857-9

[5] S. Dumitrescu, *Several aspects of context freeness for hyperedge replacement grammar*s, WSEAS Transactions on Computers, 2008, vol. 7, Issue 10, ISSN 1109-2750, pp. 1594-1604.

[6] S. Dumitrescu**,** *About Graph and Hypergraph Context Free Grammars*, Recent Researches in Computer Science, Proceedings of the 15th WSEAS International Conference on Computers, July 15-17, Corfu Island, Greece, 2011, ISSN 1792-4251, ISBN 978-1-61804-019-0, pp. 481-484

[7] S. Dumitrescu**,** *About Digital Images and Lindenmayer Systems*, Mathematical Methods, Computational Techniques, Intelligent Systems, Proceedings of the 12th WSEAS International Conference on Mathematical, Computational Techniques, Intelligent Systems, may 3-6, Kantaoui, Sousse, Tunisia, 2010, ISSN 1790-2769, ISBN 978-960-474-188-5, pp. 48-51

[8] A. Paz and A. Salomaa, *Integral sequential word functions and growth equivalence of Lindenmayer systems*, Information and Control, 23:313-343, 1973

[9] A. Salomaa, *Formal Language*, Academic Press, 1973

[10] J. Dassow, A. Habel, St. Taubenger, *Chain-Code Pictures and Collages Generated by Hyperedge Replacement, Graph Grammars and Their Applications to Computer Science*, Springer-Verlag, 1996.

[11] F. Drewes, Tree Based Picture Generation, *Theoretical Computer Science*, No. 246, 2000, pg. 1 – 51

[12] G. Rozenberg, *Handbook of Graph Grammars and Computing by Graph Transformation*, World Scientific, 1997

[13] A. Habel, H. Kreowski, Characteristics of Graph Languages Generated by Edge Replacement, *Theoretical Computer Science*, No. 51, pg. 81-115, 1987

Silviu Razvan Dumitrescu, Department of Informatics, Faculty of Mathematics and Informatics, Transilvania University of Brasov, Iuliu Maniu 50, ROMANIA
silviu.dumitrescu@unitbv.ro