STUDIA UNIV. BABEŞ-BOLYAI, INFORMATICA, VOLUME II, NUMBER 1, 1997

TREE GRAMMARS AND TREE DESCRIPTION GRAMMARS

VASILE PREJMEREAN SIMONA MOTOGNA

Abstract. Considering the disadvantages of the tree grammars, we introduce another class of grammars: tree description grammars, which are string grammars. We present some of their properties and, in the end, we state the conditions in which they are equivalent with the tree grammars, and prove this equivalence.

1. Introduction

In the theory of solid constructive geometry, different corps and figures can be described through strings (expressions corresponding to binary trees) or directly through trees [2,9].

In the case of description through strings, we can observe only the concatenation of figures, loosing various details [4,6].

In the case of description through trees we obtain a hierarchical structure [5,8], which emphasizes also the relations between component subcorps of a certain corp [3,7].

In order to specify the description through trees, the tree grammars have been introduced [7], whose production rules are defined (easier) graphically. These grammars define trees through trees, and that's why these grammars cannot be classified in the usual types of grammars from the formal languages theory [1].

Considering these reasons, we have considered that if we will manage to describe these trees through simpler grammars (the production rules to be of a known type) then the properties of such grammars will be easier to be stated. Another point of view which was taken into consideration was the remark that in fact we don't need the tree itself, only the way in which the corp or figure description tree is described.

This situation has the advantage that in pattern recognition [5,8] we can apply existing parsing algorithms [1].

The paper is structured as follows: in the first paragraph we will present the definition of the tree grammar, and in the next paragraph we will introduce the tree description grammar and we will prove some of its properties. A last paragraph will study the differences between these two types of grammars.

Received by the editors: July 15, 1998.
1991 Mathematics Subject Classification. 68Q45, 68Q50.
1991 CR Categories and Descriptors. F.4.3 [**Mathematical Logic And Formal Languages**]: Formal languages - *Classes defined by grammars or automata.*

2. Tree-grammars

Definition 2.1. [7] A *tree-grammar* G has the following form $G = (N, T, r, P, S)$ where:

- $N \cup T$ is the alphabet, containing the terminal and nonterminal symbols,
- $r : N \cup T \to \mathbf{N}$, $r(x)$ represents the number of successors for x,
- P is the set of productions rules, of the following form $A_1 \to A_2$ (where A_1 and A_2 are subtrees), and
- S is the start symbol.

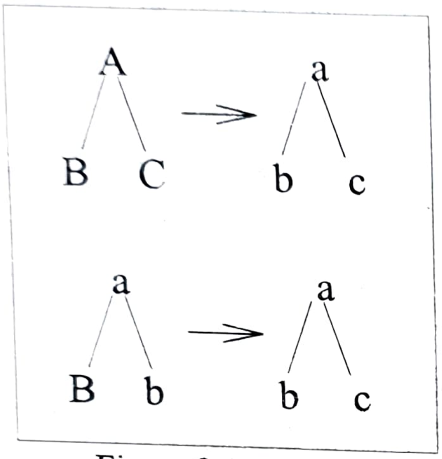

Figure 2.1.a.

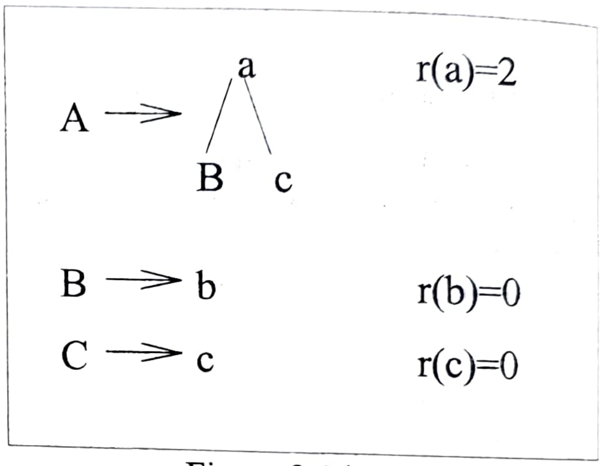

Figure 2.1.b.

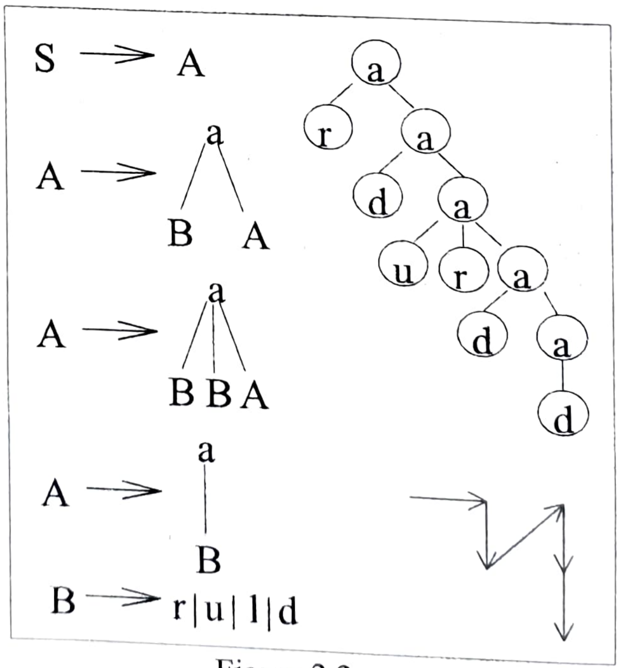

Figure 2.2.

The production rules can be defined graphically easier, since there can be productions which do not preserve the structure, as in example from figure 2.1.b. A derivation of the form $A_\alpha \Longrightarrow A_\beta$, assumes that there is a production $A_i \to A_j$ through which the subtree A_i (from A_α) is replaced by A_j obtaining the tree A_β.

Definition 2.2. [7] The *language* generated by a tree grammar G is defined as: $L(G) = \{ A / S \overset{*}{\Longrightarrow} A \}$.

Example 2.1. Figure 2.2 shows a grammar which generates a description tree for a 2-D trajectory. The production rules are written in the left side, and the right side contains the generated tree and the trajectory described by this tree.

We can notice that such a grammar (of a tree-type) allows a more general description than a string grammar, although uses only four terminals, corresponding to the four directions r,u,l,d:

$r = \rightarrow$ ("right"),
$u = \uparrow$ ("up"),
$l = \leftarrow$ ("left")
$d = \downarrow$ ("down").

3. Tree description grammars and their properties

In order to emphasize the difference between a tree grammar and a tree description grammar we will define such a grammar. The grammars to be used will be context-free, and the production rules will have the right-side starting with a terminal, namely the form $A \rightarrow a\alpha$, $\alpha \in (N \cup T)^*$, and the words (the language elements generated by these grammars) will be derivation trees (parsing trees) as presented in [1]. If in string languages, the language elements are formed only considering the derivation tree front (concatenating the tree leaves), in a tree description grammar the words will be the derivation tree (considered as a whole).

Definition 3.1. A *tree description grammar*, or simpler a *d-grammar*, $G=(N,T,P,S)$ is defined as a string grammar ([1]), modifying only the meaning of the production rules (and implicitly the meaning of the derivation) as follows:

- N and T are labels (values) for tree nodes,
- S is the label of the starting tree which has only one node: $\textcircled{S}$
- $P=\{ A \rightarrow a\alpha \, / \, A \in N,\ a \in T$ and $\alpha \in (N \cup T)^* \} \cup \{ A \rightarrow B \, / \, A, B \in N \}$;

Invoking, in a derivation, a rule of the form $A \rightarrow a\alpha$ we will achieve the transformation of the leaf A in the subtree with the root a and successors $x_1 x_2 \ldots x_n = \alpha$ ($x_i \in N \cup T$, $1 \leq i \leq n$) as depicted in figure 3.1. In addition to the production rules of the above form, we can admit in a d-grammar rules of the form $A \rightarrow B$ (A, $B \in N$), which do not change the structure at all, and only relabel the nodes.

Definition 3.2. A tree A is *final* relatively to a set M if all the terminal nodes (without successors) are from the set M. The set of all the final trees relatively to a set M will be denoted by A_M.

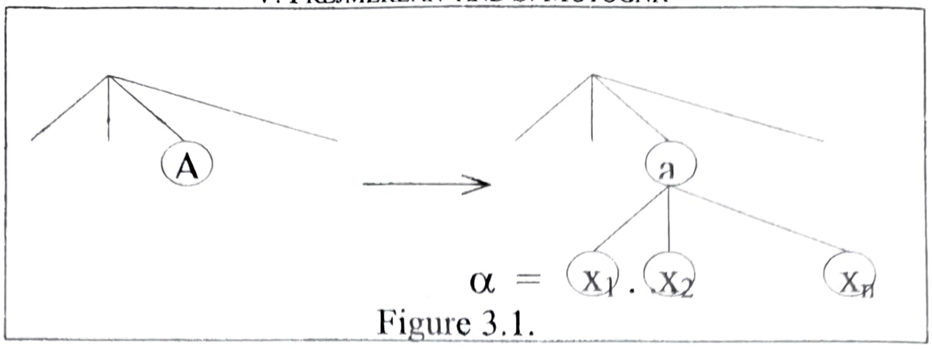

Figure 3.1.

Definition 3.3. The *language* generated by a *d*-grammar $G = (N, T, P, S)$ is defined as follows:

$$L(G) = \{ \mathrm{A} \in \mathrm{A}_T / S \overset{*}{\Longrightarrow} \mathrm{A} \}.$$

In regards to the above definitions we can notice that starting from the root (the initial symbol S) we apply the production rules only for nonterminal leaves ($\in N$), and when all the leaves are terminals, in other words the tree is final relatively to T, the derivation ends.

Remark 3.1. If all the production rules have $|\alpha| = n \in \{0, 2\}$, in the right side they have either one terminal symbol ($n = 0$) or three symbols ($n = 2$), then the generated trees will be binary trees. We have eliminated the case $n=1$, namely the rules of the form $A \to ab/aB$, because these rules do not define the subtree position (left or right), being in fact a single one. If we wish to generate binary trees with rules having $n \leq 2$, a convention will be necessary for the rules mentioned above (case $n = 1$). We have considered here that a binary tree is formed from left subtree, root and right subtree. If, in case of solving certain problems, we will consider that a binary tree is a tree with at most two successors, then we can give up the restriction $n \neq 1$.

In conclusion, we can state that if all the production rules of a d-grammar have the property $|\alpha| \leq 2$ then the grammar describes binary trees and we say that it is *d-linear*.

Example 3.1. The tree description grammar

$$G=(\{S, A, B, C\}, \{a, r, u, l, d\}, P, S),$$

with the production rules:

$$P = \{ S \to A\ ,\ A \to aBA\ /\ aCBA\ /\ aB\ ,\ B \to r\ /\ u\ /\ l\ /\ d\ ,\ C \to r\ /\ u\ /\ l\ /\ d \}$$

is equivalent with the tree grammar in example 2.1, which generates trees as in figure 2.2.

We can notice that this d-grammar can be reduced, obtaining a simplified d-grammar G', equivalent with the d-grammar G (L(G)=L(G')). The new reduced grammar G' is:

$$G' = (\{S, B\}, \{a, r, u, l, d\}, P', S),$$

with

$$P' = \{S \to aBS/aBBS/aB,\ B \to r/u/l/d\},$$

which can generate the tree from figure 3.2. We have numbered the rules of P' from 1 to 7 so we could be able to refer the production used in a derivation.

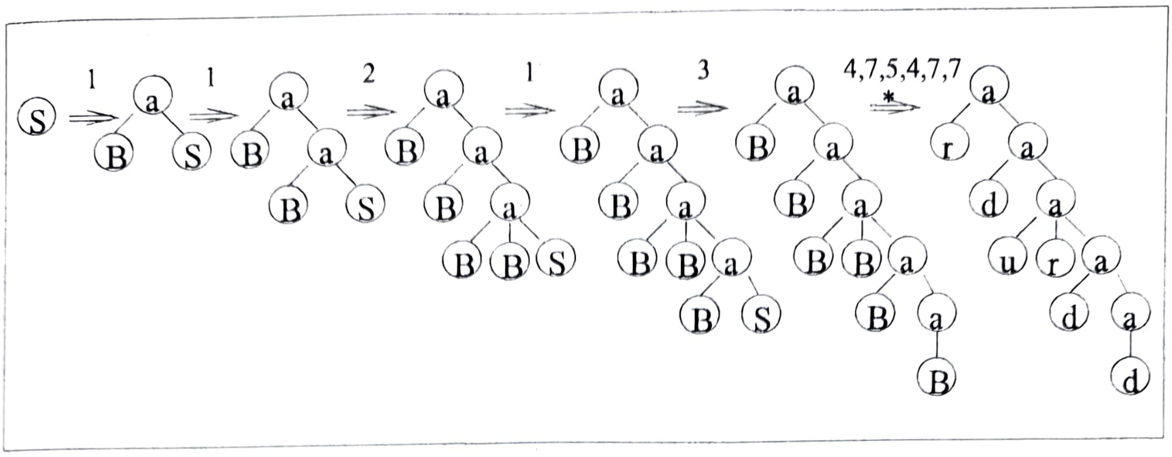

Figure 3.2.

Theorem 3.1. Let $G=(N,T,P,S)$ be a d-grammar and $L(G)$ be the tree family described by G. Any tree $\mathsf{A}\in$L(G) has all the node labels (internal nodes and leaves) from the terminal alphabet T (if we denote by V the vertex set of A, then $V\cap N=\varnothing$ or $V\subseteq T$).

Proof. Since the production rules have the form $A\rightarrow a\alpha$ (definition 3.1.), this means that when a nonterminal node ($A\in N$) is transformed it becomes an internal node, which is in T, so any internal node is from the terminal alphabet. On the other hand if $\mathsf{A}\in L(G)$ it means that $\mathsf{A}\in\mathsf{A}_T$ ($L(G)\subseteq\mathsf{A}_T$, from definition 3.2.), so it has the front formed from symbols which belongs to the terminal alphabet T (definition 3.1). Even if we use rules such as $A\rightarrow B$, these do not change at all the above situation, since from this point of view the tree structure remains unchanged in a derivation which involves such a rule.

Lemma 3.1. For any tree A, which has the nodes labeled with values from a set T, there exists a tree description grammar G=(N,T,P,S) which will generate this tree ($\mathsf{A}\in$L(G)).

Proof: Since the proof is constructive, we will explain the algorithm which builds the d-grammar $G=(N,T,P,S)$, starting from $G=(\{S\},T,\varnothing,S)$ until we reach the required final form, transforming the tree A, which will be reduced until it will have only one node labeled with S. The nonterminals X will be chosen from the capital letters set (A, $B=Succ(A)$, $C=Succ(B)$, ...).

The algorithm is:

- Initially $N=\{S\}$, $P=\varnothing$, $X:=A$;
- While A is a tree of depth > 1 do
  - For each subtree of depth 1 (has only the root = a and front = α)
    - Replace the subtree with a node labeled with X,
    - If there isn't any rule in P which contains $a\alpha$ in the right side then

      $N:=N\cup\{X\};\ P:=P\cup\{X\rightarrow a\alpha\};\ \ X:=Succ(X).\quad (*)$
- A (which now is a tree of depth 1, with root a and front α) is replaced $\textcircled{S}$ by and $P:=P\cup\{S\rightarrow a\alpha\}$.

It can be observed that the d-grammar builds the tree applying the reverse derivations and that this d-grammar generates only one tree ($|L(G)| = 1$).

Considering the tree from figure 3.2, the d-grammar will be:

$$G=(\{S,A,B,C,D\},\{a,r,u,l,d\},\{S\to arD,\ D\to adC,\ C\to aurB,\ B\to adA,\ A\to ad\},\ S).$$

Although the algorithm tries to reduce the repeated production rules (*), we still face the problem of reducing this d-grammar (to a simpler *d*-grammar G') such that $L(G)\subset L(G')$. For the d-grammar which we have built the rule $D\to adC$ is almost identical with the rule $B\to adA$, which leads us to the idea of *unifying* them, changing D with B. The reduced d-grammar G' will be:

$$G'=(\{S,A,B,C\},\{a,r,u,l,d\},\{S\to arB,\ B\to adC,\ C\to aurB,\ B\to adA,\ A\to ad\},\ S).$$

In this *d*-grammar G' the nonterminal C cannot be eliminated. If in the case of traditional string grammar, applying this elimination we will obtain the grammar $G''=(\{S,A,B\},\ \{a,r,u,l,d\},\ \{S\to arB,\ B\to adaurB,\ B\to adA,\ A\to ad\},\ S)$ equivalent to G' ($L(G)=L(G')$), in the case of tree description grammars this situation is no longer possible, because two trees with the same front may not be necessary identical.

The new *d*-grammar G' describes now a tree family, including the tree described by the *d*-grammar G (if in a derivation we use all the rules from P in the order given above). If in a derivation we use only the rules $S\to arB$, $B\to adA$ and $A\to ad$ (1,4, and 5 from P) we will obtain the tree from the figure 3.3.

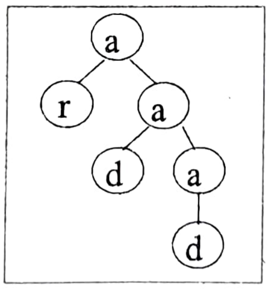

Figure 3.3.

Definition 3.4.

- A set of trees with nodes labeled is called *d-language*,
- A *d*-language L where the labels are from a set T, will be denoted L_T.

Theorem 3.2. For any finite *d*-language L_T, there exists (it can be constructed) a *d*-grammar $G=(N,T,P,S)$ such that $\mathsf{L}_T = L(G)$.

The proof is based on lemma 3.1., which for any tree $\mathsf{A}_i \in \mathsf{L}_T$ provides a grammar $G_i=(N_i,T,P_i,S_i)$ which will generate it ($1\le i\le n=|\mathsf{L}_T|$). Replacing all the rules $S_i\to a_i\alpha_i$ with $S\to a_i\alpha_i$ we obtain the desired d-grammar $G=(N_1\cup\ldots\cup N_n, T,\ P_1\cup\ldots\cup P_n, S)$. The nonterminals N_i will be denoted (in the construction of the grammar G_i) by A_i, B_i, C_i, ...

.

Definition 3.5. A d-language is context-free, linear or regular if there is a d-grammar context-free, d-linear or regular, respectively, which will generate it.

Lemma 3.2. The following statements are true:

- a family of chained lists is a regular d-language,
- a family of binary trees is a linear d-language, and
- a family of (n-ary) trees is a context-free d-language.

Proof: We will take into consideration the length of the α sequence, from a production rule $A\to a\alpha$ (Definition 3.1):

If $|\alpha| \leq 1$, then the d-language is regular, and describes chained lists. If $|\alpha| \leq 2$, then then the d-language is linear, and describes binary trees (see also the Remark 3.1). If $|\alpha| > 2$, then then the d-language is context-free, and describes n-ary trees.

Theorem 3.3. $L_{Reg} \subset L_{Lin} \subset L_{Cf}$, where L_{Reg}, L_{Lin} and L_{Cf} represent the corresponding classes of d-languages.

The proof is obvious either from formal languages point of view (Chomsky hierarchy [1]), or directly through data structures referred in lemma 3.2.

4. Comparing tree grammars and tree description grammars

We will start giving another definition to the tree description grammars, which is more general, and we will prove that if a tree grammar is context-free then there is an equivalence between them and d-grammars. This result has its own importance, since, so far, only tree grammars have been used in this domain, and we can use their properties for our d-grammars. However, we have proved that d-grammars are easier to use, because they are string grammars and we can follow traditional parsing algorithm in pattern recognition.

Definition 4.1. A *tree description grammar*, or simpler *d-grammar*, $G=(N,T,P,S)$ is defined as a context-free string grammar [1], with the remark that the production rules meaning (and implicitly the derivation meaning) is as follows:

a) N and T are labels (values) of the tree nodes
b) S is the label of the starting (initial) tree, which has only one node: $\textcircled{S}$
c) P consists of production rules of the second type ($P = \{A\to\alpha \mid A\in N$ and $\alpha\in(N\cup T)^*\}$).

Remark 4.1.

- The condition c) assures that the grammar is context free.
- In order to make a clear distinction between the two definitions given for tree description grammars (definition 3.1. and definition 4.1), we will denote the grammar from the last definition by d*-grammar.

If we call in a derivation a rule of the form $A\to\alpha$ $(\alpha=x_0x_1x_2...x_n)$ we achieve the transformation of the leaf A in the subtree with the root x_0 and successors $x_1x_2...x_n$ $(x_i\in N\cup T,\ 0\leq i\leq n)$, as shown in figure 4.1. If we apply a rule of the form $A\to B\alpha$ then a rule of the form $B \to a$ must exist and must be applied in the derivation. Such a rule does not change at all the tree structure, only relabels a node, such that the described tree would be *final* (definition 3.2.).

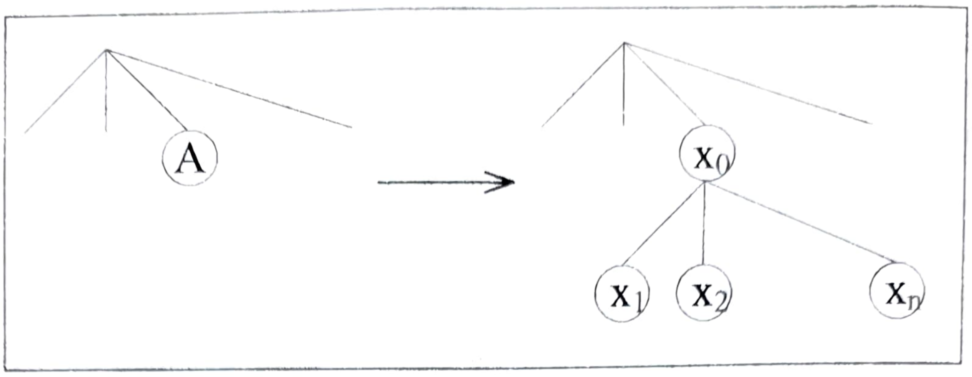

Figure 4.1.

Definition 4.2. The *language* generated by a d*-grammar $G = (N,T,P,S)$ is defined analogous (Definition 3.3):

$$L(G) = \{ \mathsf{A} \in \mathsf{A}_T / S \overset{*}{\Longrightarrow} \mathsf{A} \}.$$

Remark 4.2. All the previous results presented for d-grammars are valid for d*-grammars, since d*-grammars can be viewed as a generalization of d-grammars.

Theorem 4.1. $\mathsf{L}_d = \mathsf{L}_{d*}$

Proof.

a) The inclusion $\subseteq$ is obvious.
b) In order to prove the inclusion $\supseteq$ we proceed as follows.

For any language, there exists a *d**-grammar, such that we can built a *d*-grammar equivalent with it, in the following way:

if $G_{d*} = (N_{d*}, T_{d*}, P_{d*}, S_{d*})$

then $G_d = (N_{d*}, T_{d*}, P_d, S_{d*})$

(the alphabet and the start symbol of the two grammars are identical, only the set of the production rules are different)

For any $A \to B\alpha \in P_{d*}$ there is $B \to a \in P_{d*}$, such that

$$P_d = \{A \to a\alpha / A \to B\alpha \text{ and } B \to a \in P_{d*}\} \cup \{A \to a\alpha / A \to a\alpha \in P_{d*}\}.$$

Lemma 4.1. Let G_t be a context-free tree grammar. We denote by L_t the tree family generated by this grammar. Then there exists (it can be built) a *d*-grammar G_d, equivalent to G_t.

Proof.

a) $G_t \subseteq G_d$

Tree grammars are defined in paragraph 2 (Definition 2.1.). A context-free tree grammar will have to obey the following condition: the left side of a production rule must be a nonterminal symbol. Therefore, the rules will be only of the form from figure 2.1.b (such rules as in figure 2.1.a don't respect this condition).

The proof is constructive. Figure 4.2 shows how the equivalent production rules are built for $A \rightarrow \alpha$, $\alpha = x_0 x_1 x_2 \ldots x_n$, $x_i \in N \cup T$, $0 \leq i \leq n$.

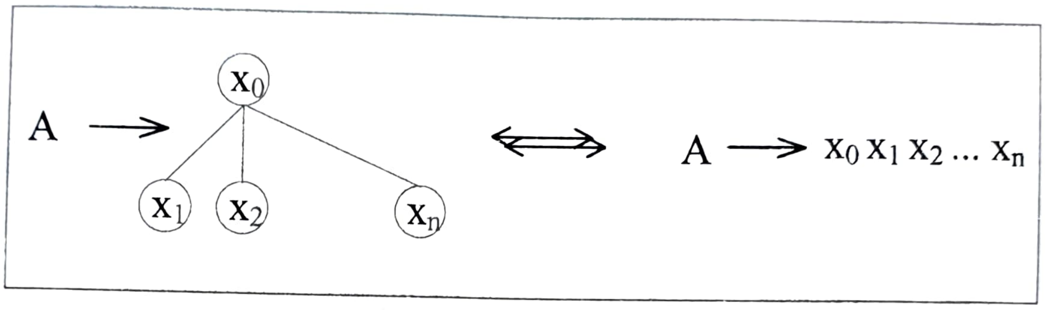

Figure 4.2

b) $G_d \subseteq G_t$ - obviously, if we represent the production rules graphically.

Theorem 4.2.

$$L_{t\_Cf} = L_{d^*}$$

where L_{t_Cf} represents the class of context-free tree languages.

The proof is obvious using lemma 4.1 and theorem 4.1.

REFERENCES

[1] Aho, J.D. Ullman, *The Theory of Parsing, Translation and Compiling*, Prentice-Hall, 1972.

[2] J.D. Foley, A.V. Dam, *Fundamentals of Interactive Computer Graphics*, Addison Wesley, Londra, 1982.

[3] K.S. Fu, *Tree Languages and Syntactic Pattern Recognition and Artificial Intelligence*, Academic Press, New York,1976.

[4] H. A. Maurer, G. Rozenberg, E. Welzl, *Using String Languages to Describe Picture Languages*, *Information and Control*, Vol.54, Nr.3, 1982, pp.115-185.

[5] T. Pavlidis, *Structural Pattern Recognition*, Academic Press, New York, 1972.

[6] Rosenfeld, *Picture Languages - Formal Models for Picture Recognitions*, Academic Press, 1979.

[7] R.J. Schakoff, *Pattern Recognition: Statistical, Structural and Neural Approaches*, (Part.3 Syntactic Pattern Recognition), John Wiley & Sons, Inc., New York, 1992.

[8] R. Vancea, S. Holban, D. Ciubotariu, *Recunoasterea Formelor, Aplicatii*, Editura Academiei, Bucuresti 1989.

[9] Watt, *3D Computer Graphics*, Addison-Wesley, Great Britain, 1993.

Babeş-Bolyai University, Faculty of Mathematics and Informatics, RO 3400 Cluj-Napoca, str. Kogalniceanu 1, România.

E-mail address: {per,motogna}@cs.ubbcluj.ro