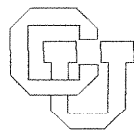


Greibach Normal Form Seen as a Transformation of Trees *

**Andrzej Ehrenfeucht
Grzegorz Rozenberg**

CU-CS-254-83



**University of Colorado at Boulder
DEPARTMENT OF COMPUTER SCIENCE**

* This research was supported by NSF grant number MCS 79-03838.

ANY OPINIONS, FINDINGS, AND CONCLUSIONS OR RECOMMENDATIONS
EXPRESSED IN THIS PUBLICATION ARE THOSE OF THE AUTHOR(S) AND DO NOT
NECESSARILY REFLECT THE VIEWS OF THE AGENCIES NAMED IN THE
ACKNOWLEDGMENTS SECTION.

GREIBACH NORMAL FORM SEEN
AS A TRANSFORMATION OF TREES

(A preliminary version)

by

Andrzej Ehrenfeucht* and Grzegorz Rozenberg**

CU-CS-254-83

April, 1983

*University of Colorado, Computer Science Department, Boulder, CO 80309

**Institute of Applied Mathematics and Computer Science, University of Leiden,
Leiden, The Netherlands

All correspondence to the second author.

This research was supported by NSF grant number MCS 79-03838.

ABSTRACT

An algorithm transforming an arbitrary context-free grammar into an equivalent context-free grammar in Greibach normal form is presented. It has two important features. (1) Given a context-free grammar in which the length of the right-hand side of any production does not exceed $k \geq 1$, the algorithm produces *directly* an equivalent context-free grammar in $(k+1)$ -Greibach normal form. (2) The transformation can be considered as a tree transformation: given a derivation tree T of a word w in the language of an "input" context-free grammar G one transforms T without having any information about G , except for T itself, into a derivation tree U of w in the "output" context-free grammar H in Greibach normal form.

INTRODUCTION

Among many normal forms for context-free grammars, Greibach normal form occupies quite a special place - it turns out to be quite important from both the theoretical and the practical points of view (see, e.g., [H], [S], and [AU]). For this reason quite a number of algorithms are available that transform an arbitrary context-free grammar into a context-free grammar in Greibach normal form.

In the present paper we present yet another algorithm for obtaining of grammars in Greibach normal form. The algorithm seems to be quite different from other algorithms presented in the literature. In particular it has two distinct features.

- (1) Given a context-free grammar in which the length of the right-hand side of any production does not exceed $k \geq 2$, the algorithm produces an equivalent context-free grammar that is in $(k+1)$ -Greibach normal form (that is the length of the right hand side of any production does not exceed $(k+2)$).
- (2) The transformation we present can be considered as a tree transformation: transforming derivation trees in the original grammar G into derivation trees in the new grammar H in Greibach normal form. This transformation is "local" in the following sense: given a derivation tree T of a word w in $L(G)$ one transforms T *without having any information about G* , except for T itself, into a derivation tree U of w in $L(H)$. Also the converse is true: given a derivation tree U of a word u in $L(H)$ one transforms U *without having any information about H* except for U itself into a derivation tree T of u in $L(G)$.

The paper is organized as follows.

In Section 1 the main construction of the paper is introduced and illustrated by an example. Then in Section 2 various basic parameters of the result-

ing context-free grammar (in Greibach normal form) are compared with (expressed in the terms of) the corresponding parameters of the originally given context-free grammar. Section 3 presents an algorithm implementing the main construction and discusses its time complexity while in Section 4 the correctness proof of the main construction is provided. In Section 5 the main construction is discussed in the case then the input grammar is a complete binary context-free grammar (that is all productions are of the form $A \rightarrow X Y$ where A is a nonterminal and X, Y are arbitrary symbols).

In Section 6 two classes of trees are considered: complete binary and left needle trees. These classes of trees are very basic in Section 7 where our main construction is considered as a tree transformation. In the last section (8) a short discussion of obtained results is given and moreover various technical short-cuts are discussed which allow one to obtain "small" context-free grammars in Greibach normal form.

0. PRELIMINARIES

In this section we recall some basic notions and settle the basic notation for this paper.

We assume the reader to be familiar with basic formal language theory—we will use many standard notions without defining them here.

For a set Z , $\#Z$ denotes its cardinality; \emptyset denotes the empty set. For sets Z_1, Z_2 , $Z_1 \setminus Z_2$ denotes their set theoretical difference. \mathbf{N} denotes the set of all nonnegative integers.

For a word x , $|x|$ denotes its length; Λ denotes the empty word. If x is a nonempty word, then $first(x)$ denotes the first letter of x .

In this paper we will consider transformations of (directed) trees where nodes and edges are transformed independently. For this reason it is convenient to specify *trees* in the form $T = (V, E, source, target)$ where V is the set of nodes of T , E is the set of edges of T , *source* and *target* are functions from E into V ; if for $e \in E$, $source(e) = v$ and $target(e) = u$ then we say that e is *leading from v* and e is *leading to u* . The components V , E , *source*, and *target* of T will be denoted by V_T , E_T , $source_T$ and $target_T$, respectively. *We consider finite trees only.*

A *node labelled tree* is specified in the form $T = (V, E, source, target, \Gamma, lab)$ where Γ is the *label alphabet*, *lab* is a function from V into Γ and other components are as above; the tree $(V, E, source, target)$ is called the *underlying tree* of T and denoted by $un(T)$. The components V , E , *source*, *target*, Γ and *lab* will be denoted by V_T , E_T , $source_T$, $target_T$, Γ_T and lab_T respectively.

As usual in the formal language theory we will considered only ordered (node labelled) trees, that is (node labelled) trees where the set of all direct descendants of every node is linearly ordered; to simplify the notation we will not specify this ordering when specifying the tree; however in the context of our considerations this should not lead to a confusion.

Let T be a (node labelled) tree.

- (i) The root of T is denoted by $root(T)$.
- (ii) The meaning of terms: *parent*, *child*, *sibling* (of a node), *inside node*, *leaf*, *frontier of T* , *yield of T* (for the node labelled case) is standard; the frontier of T is denoted by $front(T)$ and the yield of T is denoted by $yield(T)$.
- (iii) If $u \in V_T$ and $u_1, \dots, u_n \in V_T$ are all children of u (with their linear order: u_1, \dots, u_n) then we say that u is *expanded (in T)* using the production $lab_T(u) \rightarrow lab_T(u_1) \cdots lab_T(u_n)$.

(iv) If k is a positive integer such that each node of T has no more than k children, then we say that T is a k -ary tree; a 2-ary tree is called a *binary tree* and a 3-ary tree is called a *ternary tree*.

(v) For a node u we talk in the standard way about its *left(most) child* and consequently about a *left path* in T (a path involving left children only). A *complete left path* is a left path leading from a node that is not a left child to a leaf, the *maximal left path* is the complete left path starting at $root(T)$; since each path consists of directed edges we can talk about the *top* and *bottom* nodes of a path. Analogously we have a *right(most) child*, a *right path*, a *complete right path* and the *maximal right path*.

Two trees T_1 and T_2 are *disjoint* if $V_{T_1} \cap V_{T_2} = \phi$ and $E_{T_1} \cap E_{T_2} = \phi$; the set of trees is *disjoint* if any two different trees from it are disjoint. Informally speaking, the *sum* of a set of disjoint trees $\{T_1, \dots, T_n\}$, $n \geq 2$, is a graph with the set of nodes equal $V_{T_1} \cup \dots \cup V_{T_n}$, the set of edges equal $E_{T_1} \cup \dots \cup E_{T_n}$ and such that its *source*, *target* and *lab* function are specified "independently" for each component T_i in the way they were specified in T_i ; this sum is denoted by $T_1 \cup T_2 \cup \dots \cup T_n$.

A *context-free grammar* (a *cf grammar* for short) is specified in the form $G = (\Sigma, \Delta, P, S)$ where Σ is the *total alphabet* of G , Δ is the *terminal alphabet* of G , P is the set of *productions* of G and S is the *axiom* of G ; components Σ, Δ, P, S will be also denoted by Σ_G, Δ_G, P_G and S_G respectively. Often a cf grammar will be specified by giving only its axiom (S_G) and its set of productions (P_G). The *language* of a cf grammar G is denoted by $L(G)$. The *equivalence* of cf grammars is considered *modulo* Λ , i.e., we say that cf grammars G_1, G_2 are (language) *equivalent* if $L(G_1) \cup \{\Lambda\} = L(G_2) \cup \{\Lambda\}$ (in the considerations of Sections 5 and 7 we will consider even a more "liberal" definition of the equivalence of cf grammars). We write $G_1 \sim G_2$ whenever G_1 is equivalent to G_2 .

Let $G = (\Sigma, \Delta, P, S)$ be a cf grammar.

(1) A production of the form $A \rightarrow B$ where $B \in \Sigma \setminus \Delta$ is called a *chain production*, a production of the form $A \rightarrow \Lambda$ is called an *erasing production*; if G does not have chain productions then it is called *chain-free* and if G does not have erasing productions then it is called Λ -free.

(2) \Rightarrow_G , \Rightarrow_G^+ and \Rightarrow_G^* denote the direct derivation relation in G , its transitive closure and its reflexive and transitive closure respectively. For a positive integer n , \Rightarrow_G^n denotes the n -steps derivation relation in G ; for each $w \in \Sigma^*$ we have $w \Rightarrow_G^0 w$.

(3) If $A \in \Sigma \setminus \Delta$ and $w \in \Sigma^*$ are such that $A \Rightarrow_G^+ w$, then there is a *derivation of w from A (in G)*; if $A = S$ then we talk about a *derivation of w (in G)*. To each derivation of w from A (in G) there corresponds a unique *derivation tree*.

(4) For $X \in \Sigma$ and a positive integer n , we define

$$L(G, X, n) = \{w \in \Delta^* : X \Rightarrow_G^m w \text{ for some } 0 \leq m \leq n\}$$

and

$$L(G, X) = \{w \in \Delta^* : X \Rightarrow_G^* w\}.$$

Note that

$$L(G, X) = \bigcup_{n \geq 0} L(G, X, n) \text{ and } L(G, S) = L(G).$$

(5) Let $X, Y \in \Sigma$. We define binary relations \vdash_G^l , \vdash_G^{l+} , and \vdash_G^{l*} as follows:

$X \vdash_G^l Y$ if and only if $X \rightarrow \alpha$, for some $\alpha \in \Sigma^+$, is a production in P and $first(\alpha) = Y$,

$X \vdash_G^{l*} Y$ if and only if $X \Rightarrow_G^* \alpha$, for some $\alpha \in \Sigma^+$, and $first(\alpha) = Y$,

$X \vdash_G^{l+} Y$ if and only if $X \Rightarrow_G^+ \alpha$, for some $\alpha \in \Sigma^+$, and $first(\alpha) = Y$.

(6) We define $\maxr(G)$ to be the maximal length of the right hand side of any production; i.e., $\maxr(G) = \max\{|\alpha| : A \rightarrow \alpha \text{ is in } P\}$.

The size of G is defined by: $\text{size}(G) = \sum_{A \rightarrow \alpha \in P} |A\alpha|$.

The norm of G is defined by: $\text{norm}(G) = \text{size}(G) \log_2(\#\Sigma)$.

(7) We say that G is in *Greibach normal form* (abbreviated *GNF*) if for each production $A \rightarrow \alpha$ of P we have $\alpha \in \Sigma^+$ and $\text{first}(\alpha) \in \Delta$; moreover if there exists a positive integer k such that $\maxr(G) = k+1$, then we say that G is in k -GNF.

We say that G is in *pseudo Greibach normal form* (abbreviated *pseudo GNF*) if for each production $A \rightarrow \alpha$ of P we have: either it is a chain production or it is an erasing production or $\text{first}(\alpha) \in \Delta$.

Remark. Note that our definition of GNF differs from the usual one in that we do not require that all symbols at the right-hand side of any production except for the first symbol are nonterminals. We have decided to use this definition in order to avoid a number of rather trivial technicalities that obscure the real picture of the transformations that we consider in this paper. Thus whenever we say that a given construction yields a grammar in GNF it may be that a number of symbols at the right-hand side of a production may belong to a terminal alphabet. Converting such a cf grammar to a cf grammar satisfying the usual definition of GNF is really easy: for each terminal symbol a that occurs at the right-hand side of a production not as the first symbol we introduce a nonterminal N_a , replace all "not-first" occurrences of a at all the right-hand side of all productions by N_a and add a production $N_a \rightarrow a$. Note that the size of a new grammar differs from the size of the initial one by not more than $2t$ where t is the cardinality of the terminal alphabet, while $\maxr(G)$ does not change. The size of the alphabet can be increased by at most t .

The ease of this additional adjustment allows us (in our opinion) to consider the simplified version of GNF - the gains in clarity are considerable espe-

cially when we consider transformations into GNF as tree transformations. ■

We conclude this section by recalling one of the basic transformations of cf grammars.

Let $G = (\Sigma, \Delta, P, S)$ be a cf grammar. Let $A \in \Sigma \setminus \Delta$ and let $A \rightarrow \alpha_1 | \dots | \alpha_m, m \geq 1$, be all productions for A in P . Let $\pi = B \rightarrow \gamma A \beta$ be an arbitrary production of G where $\gamma, \beta \in \Sigma^*$. Let ζ be the following transformation of G (to a cf grammar $\zeta(G)$): replace π by m productions, $B \rightarrow \gamma \alpha_1 \beta, \dots, B \rightarrow \gamma \alpha_m \beta$, all other productions in P remain intact. Such a transformation is referred to as a *sub* transformation.

It is well known (and easy to prove) that the following result holds.

Lemma 0.1. $\zeta(G) \sim G$. ■

1. THE MAIN CONSTRUCTION

In this section we provide a construction which given an arbitrary Λ -free cf grammar yields an equivalent cf grammar in GNF . The construction is illustrated by an example.

Definition 1.1. Let $G = (\Sigma, \Delta, P, S)$ be a Λ -free cf grammar.

(i) Let $G' = (\Sigma', \Delta', P', S')$ be the cf grammar such that:

$$\Sigma' = \Delta \cup \{[X, Y] : X, Y \in \Sigma \text{ and } Y \vdash^{l*} X\} \cup \{[X] : X \in \Sigma\},$$

$$\Delta' = \Delta,$$

$$S' = [S] \text{ and}$$

P'' consists of productions obtained as in (1) through (4) below.

(1) For each production $\pi = A \rightarrow Y_1 \cdots Y_k, k \geq 2$, in P , where $Y_i \in \Sigma$ for $1 \leq i \leq k$, each $B \in \Sigma \setminus \Delta$ and each $\alpha \in \Delta$ such that $B \vdash^{l*} A$ and $Y_2 \vdash^{l*} \alpha$, P' contains the production

$$[Y_1, B] \rightarrow \alpha[\alpha, Y_2][Y_3] \cdots [Y_k][A, B].$$

(2) For each production $\pi = A \rightarrow Y$ in P , where $Y \in \Sigma$, and each $B \in \Sigma \setminus \Delta$ such that $B \vdash^{l*} A$, P' contains the production

$$[Y, B] \rightarrow [A, B].$$

(3) For each $\alpha \in \Delta$ and each $X \in \Sigma$ such that $X \vdash^{l*} \alpha$, P' contains the production

$$[X] \rightarrow \alpha[\alpha, X].$$

(4) For each $X \in \Sigma$, P' contains the production $[X, X] \rightarrow \Lambda$.

(ii) Now let G_1 be the Λ -free cf grammar equivalent to G' and resulting from G' by applying, e.g., the standard construction (see, e.g., [S]) for removing Λ rules.

(iii) Let H be the Λ -free, chain-free cf grammar equivalent to G_1 and resulting from G_1 by applying, e.g., the standard construction (see, e.g., [S]) for removing chain rules. ■

The productions resulting from (1) through (4) above are referred to as productions of *type 1*, *type 2*, *type 3*, *type 4* respectively; their sets are denoted by P'_1, P'_2, P'_3, P'_4 respectively. The transformation described in the above definition will be denoted by *const* (and so H will be denoted by $\text{const}(G)$).

Remark 1.1. (1) In Section 3 we will discuss a specific algorithm for removing Λ rules (i.e., for getting G_1 from G') and a specific algorithm for removing chain rules (i.e., for getting H from G_1); these algorithms are devised in the course of estimating the complexity of an algorithm implementing the construction from the above definition. In any case we always assume a fixed algorithm for removing erasing rules and a fixed algorithm for removing chain rules; for this reason we refer to G_1 as *the* grammar obtained in step (ii) above and to H as *the* grammar resulting from the whole construction.

(2) Usually we will consider not H itself but its reduced version, i.e., the cf grammar resulting from H by removing useless nonterminals and productions for them. ■

Example 1.1.

Let $G = (\Sigma, \Delta, P, S)$ be the cf grammar, where

$$\Sigma = \{A, B, a, b, c, d\},$$

$$\Delta = \{a, b, c, d\},$$

$$S = A, \text{ and}$$

$$P \text{ consists of the following productions: } A \rightarrow AaB \mid BB \mid b,$$

$$B \rightarrow aA \mid BAa \mid Bd \mid c.$$

(This example is from [AU] p. 159, vol. 1).

Then G' constructed as in Definition 1.1. is the grammar with the axiom $[A]$ and the following productions.

$$\begin{aligned}
[A,A] &\rightarrow a[a,a][B][A,A], \\
[B,A] &\rightarrow a[a,B][A,A] \mid c[c,B][A,A] \mid d[d,d][B,A] \mid d[d,d][B,B], \\
[a,A] &\rightarrow a[a,A][B,A] \mid b[b,A][B,A] \mid c[c,A][B,A], \\
[a,B] &\rightarrow a[a,A][B,B] \mid b[b,A][B,B] \mid c[c,A][B,B], \\
[B,A] &\rightarrow a[a,A][a][B,A] \mid b[b,A][a][B,A] \mid c[c,A][a][B,A], \\
[B,B] &\rightarrow a[a,A][a][B,B] \mid b[a,A][a][B,B] \mid c[a,A][a][B,B], \\
[b,A] &\rightarrow [A,A], \\
[c,B] &\rightarrow [B,B], \\
[c,A] &\rightarrow [B,A], \\
[A] &\rightarrow a[a,A] \mid b[b,A] \mid c[c,A], \\
[B] &\rightarrow a[a,B] \mid c[c,B], \\
[t] &\rightarrow t[t,t], \text{ for all } t \in \Delta, \\
[X,X] &\rightarrow \Lambda, \text{ for all } X \in \Sigma.
\end{aligned}$$

Then $H = \text{const}(G)$ after removing useless symbols (and productions for them) yields the following grammar H' .

Axiom: $[A]$,

Productions:

$$\begin{aligned}
[A,A] &\rightarrow a[B][A,A] \mid a[B], \\
[a,A] &\rightarrow a[a,A][B,A] \mid b[A,A][B,A] \mid b[B,A] \mid c[B,A][B,A], \\
[a,B] &\rightarrow a[a,A][B,B] \mid a[a,A] \mid b[A,A][B,B] \mid b \mid b[A,A] \mid \\
&\quad b[B,B] \mid c[B,A][B,B] \mid c[B,A], \\
[B,A] &\rightarrow a[a,B][A,A] \mid a[a,B] \mid c[B,B][A,A] \mid c[A,A] \mid \\
&\quad c[B,B] \mid c \mid a[a,A][a][B,A] \mid [b[A,A][a][B,A] \mid b[a][B,A] \mid \\
&\quad c[B,A][a][B,A] \mid d[B,A] \mid d[B,B] \mid d, \\
[B,B] &\rightarrow a[a,A][a][B,B] \mid a[a,A][a] \mid b[A,A][a][B,B] \mid b[a] \\
&\quad \mid b[A,A][a] \mid b[a][B,B] \mid c[B,A][a][B,B] \mid c[B,A][a],
\end{aligned}$$

$$[A] \rightarrow a[a,A] \mid b[A,A] \mid c[B,A],$$

$$[B] \rightarrow a[a,B] \mid c[B,A],$$

$$[a] \rightarrow a.$$

H' has the following parameters: 8 nonterminals, 41 productions, $\text{maxr}(H') = 4$ and $\text{size}(H') = 144$.

2. ON THE BASIC PARAMETERS OF $\text{const}(G)$

In this section we estimate various parameters of the resulting of grammar $H = \text{const}(G)$ in terms of the corresponding parameters of a given of grammar G . These parameters are also estimated for the "intermediate" grammars (G' and G_1) introduced in the construction of H .

(i) *The cardinality of the alphabet Σ_H .*

Note that $\Sigma' \subseteq \Delta \cup \{[X]: X \in \Sigma\} \cup \{[X, Y]: X, Y \in \Sigma\}$ and so $\#\Sigma' \leq \#\Delta + \#\Sigma + (\#\Sigma)^2$.

Since in transformations leading from G' to G_1 and from G_1 to H the size of the alphabet does not change,

$$\#\Sigma_H \leq \#\Delta + \#\Sigma + (\#\Sigma)^2.$$

(ii) *The maximal length of the right hand side of a production in P_H .*

From the construction of G' it follows directly that $\text{maxr}(G') = \text{maxr}(G) + 1$.

Since in transformations leading from G' to G_1 and from G_1 to H the maximal length of the right hand side of a production does not change,

$$\text{maxr}(H) = \text{maxr}(G) + 1.$$

(iii) *The cardinality of the set of productions P_H .*

(iii.1) For each production $A \rightarrow Y_1 \cdots Y_k$, $k \leq 2$, in P where $Y_i \in \Sigma$ for $1 \leq i \leq k$,

we have introduced at most $\#\Delta \cdot \#(\Sigma \setminus \Delta)$ productions in P'_1 .

(iii.2) For each production $A \rightarrow Y$ in P , where $Y \in \Sigma$, we have introduced at most

$\#(\Sigma \setminus \Delta)$ productions in P'_2 .

(iii.3) P'_3 contains at most $\#\Sigma \cdot \#\Delta$ productions.

(iii.4) P'_4 contains $\#\Sigma$ productions.

Hence $\#P' \leq P_G \cdot \#\Delta \cdot \#(\Sigma \setminus \Delta) + P_G \cdot \#(\Sigma \setminus \Delta) + \#\Sigma \cdot \#\Delta + \#\Sigma$.

$$\leq \#P_G \cdot 2 \cdot \#\Sigma \cdot (\#\Delta + 1) \leq \#P_G \cdot 4 \cdot \#\Sigma \cdot \#\Delta.$$

Transforming G' to G_1 increases the number of productions by at most the factor of 4 because in each production of G' the right hand side contains at most two (occurrences of) letters that can derive Λ in G' .

Hence $\#P_{G_1} \leq \#P' \cdot 4 \leq \#P_G \cdot 16 \cdot \#\Sigma \cdot \#\Delta$.

In transforming G_1 to H we replace each chain production of G_1 by no more than $\#(\Sigma \setminus \Delta) \cdot \#P_{G_1}$ productions and consequently

$$\begin{aligned} \#P_H &\leq \#P_{G_1} + \#(\Sigma \setminus \Delta) \#P_{G_1} \leq \#P_{G_1} \cdot 2 \cdot \#\Sigma \\ &\leq \#P_G \cdot 32 \cdot (\#\Sigma)^2 \cdot \#\Delta. \end{aligned}$$

(iv) *The size of H .*

Note that in constructing G' from G the only case when we increase the length of the right-hand size of a production is when we construct productions in P'_1 - the size increases then by 1.

Since $\#P' \leq \#P_G \cdot 4 \cdot \#\Sigma \cdot \#\Delta$, we have

$$\text{size}(G') \leq \text{size}(G) \cdot 4 \cdot \#\Sigma \cdot \#\Delta + \#P_G \cdot 4 \cdot \#\Sigma \cdot \#\Delta \leq \text{size}(G) \cdot 8 \cdot \#\Sigma \cdot \#\Delta.$$

In constructing G_1 from G' the length of the right-hand size of a production does not increase.

Since $\#P_{G_1} \leq \#P' \cdot 4$, we have

$$\text{size}(G_1) \leq \text{size}(G') \cdot 4 \leq \text{size}(G) \cdot 32 \cdot \#\Sigma \cdot \#\Delta.$$

In constructing H from G_1 the length of the right-hand size of a production does not increase.

Since $\#P_H \leq \#P_{G_1} \cdot 2 \cdot \#\Sigma$, we have

$$\text{size}(H) \leq \text{size}(G_1) \cdot 2 \cdot \#\Sigma \leq \text{size}(G) \cdot 64 \cdot (\#\Sigma)^2 \cdot \#\Delta.$$

(v) *The norm of H .*

Recall that $\#\Sigma' \leq \#\Delta + \#\Sigma + (\#\Sigma)^2$. Hence

$$(\#\Sigma') = (\#\Sigma)^2 \left(1 + \frac{\#\Delta + \#\Sigma}{(\#\Sigma)^2}\right) \leq 2(\#\Sigma)^2. \text{ Then we have}$$

$$\begin{aligned} \text{norm}(G') &= \text{size}(G') \cdot \log_2 \# \Sigma' \leq \text{size}(G') (2 \log_2 \# \Sigma + 1) \\ &\leq \text{size}(G') \cdot 3 \log_2 \# \Sigma . \end{aligned}$$

Since $\text{size}(G') \leq \text{size}(G) \cdot 8 \cdot \# \Sigma \cdot \# \Delta$, we have

$$\text{norm}(G') \leq \text{size}(G) \cdot \log_2 \# \Sigma \cdot 24 \cdot \# \Sigma \cdot \# \Delta = \text{norm}(G) \cdot 24 \cdot \# \Sigma \cdot \# \Delta .$$

Since in transforming G' into G_1 we do not change the alphabet, we have

$$\begin{aligned} \text{norm}(G_1) &= \text{size}(G_1) \cdot \log_2 \# \Sigma_1 = \text{size}(G_1) \cdot \log_2 \# \Sigma' \\ &\leq \text{size}(G') \cdot 4 \cdot \log_2 \# \Sigma' = 4 \text{norm}(G') \leq \text{norm}(G) \cdot 96 \cdot \# \Sigma \cdot \# \Delta . \end{aligned}$$

Since in transforming G_1 into H we do not change the alphabet, we have

$$\begin{aligned} \text{norm}(H) &= \text{size}(H) \cdot \log_2 \# \Sigma_2 = \text{size}(H) \cdot \log_2 \# \Sigma_1 \\ &\leq \text{size}(G_1) \cdot 2 \cdot \# \Sigma \cdot \log_2 \# \Sigma_1 = \text{norm}(G_1) \cdot 2 \cdot \# \Sigma \\ &\leq \text{norm}(G) \cdot 192 \cdot (\# \Sigma)^2 \cdot \# \Delta . \end{aligned}$$

3. AN ALGORITHM IMPLEMENTING *const*

In this section we describe an algorithm implementing *const* and discuss the time complexity of the algorithm.

We will use the notation from Definition 1.1, i.e., we start with a Λ -free of grammar $G = (\Sigma, \Delta, P, S)$, then we construct $G' = (\Sigma', \Delta', P', S')$ then we construct $G_1 = (\Sigma_1, \Delta_1, P_1, S_1)$ and finally we obtain H in *GNF*.

We assume that the alphabet Σ is linearly ordered and so $\Sigma = (\sigma_1, \dots, \sigma_m)$ for some $m \geq 2$. Let $\#\Delta = e$. We assume that a list $\rho(P)$ of productions from P is given.

We give now the description of our algorithm which consists of eight steps.
STEP 1.

Construct the incidence matrix M_{\vdash^l} for the relation \vdash^l .

This is done as follows.

First we construct the $m \times m$ matrix M^0 with m rows indexed by $\sigma_1, \sigma_2, \dots, \sigma_m$ and m columns indexed by $\sigma_1, \dots, \sigma_m$ and such that for each $1 \leq i, j \leq m$, $M^0(\sigma_i, \sigma_j) = 0$.

Then we read the list of all productions from P and when we read off a production of the form $A \rightarrow \alpha$ with $first(\alpha) = Y$ then we set the entry $M_{\vdash^l}(A, Y)$ to 1; hence if we start with M^0 and go through all productions from P then we get the incidence matrix M_{\vdash^l} .

Consequently the time complexity of STEP 1 is bounded by $O(size(G) + m^2)$.
STEP 2.

Construct the incidence matrix $M_{\vdash^l_}$ for the relation \vdash^{l_*} .*

Obviously M_{\vdash^*} is the reflexive and transitive closure of $M_{\vdash'}$. Using, e.g., the Warshall algorithm (see, e.g., [AU]) we can accomplish STEP 2 in no more than $O(m^3)$ steps.

STEP 3

Construct P' .

This is done as follows.

We read the list $\rho(P)$ from left to right, production after production. Assume that we currently scan production π . We distinguish several cases.

(1) If π is of type 1, then for each pair of symbols $a \in \Delta$ and $B \in \Sigma \setminus \Delta$ we check (using M_{\vdash^*}) whether or not $Y_2 \vdash^{l*} a$ and $B \vdash^{l*} A$. If the answer is positive, then we output the production

$$[Y_1, B] \rightarrow a[a, Y_2][Y_3] \cdots [Y_k][A, B].$$

If the answer is negative, then we do not have an output for this production π .

The processing of all productions π of type 1 will take no more than $O(\text{size}(G)m \cdot e)$ steps.

(2) If π is of type 2, then for each $B \in \Sigma \setminus \Delta$ we check (using M_{\vdash^*}) whether or not $B \vdash^{l*} A$. If the answer is positive then we output production $[Y, B] \rightarrow [A, B]$; otherwise we do not have an output.

The processing of all productions π of type 2 will take no more than $O(\text{size}(G) \cdot m)$ steps.

After the reading of $\rho(P)$ is completed (according to rules (1) and (2) above) we do the following.

(3) For each $a \in \Delta$ and each $X \in \Sigma$ we check, using M_{\vdash^*} , whether or not $X \vdash^{l*} a$.

If the answer is positive, then we output the production $[X] \rightarrow a[a, X]$; otherwise

we do not have an output.

The action (3) above can be accomplished in no more than $O(me)$ steps.

(4) For each $X \in \Sigma$ we output the production $[X, X] \rightarrow \Lambda$.

This action can be completed in no more than $O(m)$ steps.

Hence the time complexity of STEP 3 (constructing P' from P) is bounded from above by:

$$O(\text{size}(G)me) + O(\text{size}(G) \cdot m) + O(me) + O(m) = O(\text{size}(G)me) \text{ steps.}$$

Thus the time complexity of our algorithm for constructing P' from P equals

$$\begin{aligned} &O(\text{size}(G) + m^2) + O(m^3) + O(\text{size}(G)me) \\ &O(\text{size}(G)me + m^3). \end{aligned}$$

We can assume now that a list $\rho(P')$ of all productions from P' is given.

STEP 4

For each $X \in \Sigma$ construct the $m \times m$ boolean matrix M_X with rows and columns indexed by the elements of Σ and such that, for all $\tau, \sigma \in \Sigma$,

$$M_X(\tau, \sigma) = \begin{cases} 1 & \text{if } [\tau, x] \rightarrow [\sigma, x] \text{ is in } P', \\ 0 & \text{otherwise.} \end{cases}$$

This is done as follows.

Given $X \in \Sigma$ we construct the $m \times m$ boolean matrix M_X^0 with rows and columns indexed by the elements of Σ and such that, for all $\tau, \sigma \in \Sigma$, $M_X^0(\xi) = 0$.

We can construct all M_X^0 matrices in no more than $O(m^3)$ steps.

Now we scan the list $\rho(P')$ and whenever we encounter a chain production $[\tau, x] \rightarrow [\sigma, x]$ we change the entry (τ, σ) of partially constructed (starting with M_X^0) matrix M_X to 1.

Thus to construct all M_X matrices starting from M_X^0 takes no more than $O(\text{size}(G'))$ steps.

Hence the time complexity of STEP 4 is bounded from above by $O(\text{size}(G') + m^3) = O(\text{size}(G)me + m^3)$; recall from Section 2 that $\text{size}(G') \leq O(\text{size}(G)me)$.

STEP 5

For each $X \in \Sigma$ construct matrix M_X^+ equal to the transitive and reflexive closure of matrix M_X .

Since for each X the construction of M_X^+ takes no more than $O(m^3)$ steps (see STEP 2), the time complexity of STEP 5 does not exceed $O(m^4)$.

STEP 6

Construct boolean arrays $A_0, A_{1,2}, A_{2,3}, A_1, A_2, A_3$ defined below.

To define these arrays we need a number of definitions first.

Let $\Sigma'' = \Sigma' \setminus (\Delta \cup \{[X] : X \in \Sigma\})$; hence Σ'' is the set of all letters from Σ' of the type $[X, Y]$ where $X, Y \in \Sigma$ and $Y \vdash^{l*} X$. We will partition Σ'' into 3 subsets as follows:

Σ_1'' consists of all letters σ from Σ'' such that $L(G', \sigma) = \{\Lambda\}$,

Σ_2'' consists of all letters σ from Σ'' such that $L(G', \sigma) \neq \{\Lambda\}$ and $\Lambda \in L(G', \sigma)$,

$\Sigma_3'' = \Sigma'' \setminus (\Sigma_1'' \cup \Sigma_2'')$.

The following result follows directly from the definitions of $P', \Sigma_1'', \Sigma_2''$ and Σ_3'' .

Lemma 3.1. Let $\sigma = [X, Y] \in \Sigma''$. Then

(1) $\sigma \in \Sigma_1'' \cup \Sigma_2''$ if and only if $M_Y^*(X, Y) = 1$.

(2) $\sigma \in \Sigma_2'' \cup \Sigma_3''$ if and only if there exists a $Z \in \Sigma$ such that $M_Y^*(X, Z) = 1$ and either P' contains a production for $[Z, Y]$ of type 1 or P' contains a production for $[Z, Y]$ of type 3. ■

Now we are ready to define boolean arrays $A_0, A_{1,2}, A_{2,3}, A_1, A_2, A_3$ as follows. Each of them has $\#\Sigma''$ entries indexed by elements of Σ'' (we assume a fixed order of elements of Σ''). The boolean value of an entry $\sigma = [X, Y]$ is defined by:

$A_0(\sigma) = 1$ if and only if either P' has a production of type 1 for σ or

P' has a production of type 3 for σ ,

$A_{1,2}(\sigma) = 1$ if and only if $\sigma \in \Sigma_1'' \cup \Sigma_2''$,

$A_{2,3}(\sigma) = 1$ if and only if $\sigma \in \Sigma_2'' \cup \Sigma_3''$,

$A_1(\sigma) = 1$ if and only if $\sigma \in \Sigma_1''$,

$A_2(\sigma) = 1$ if and only if $\sigma \in \Sigma_2''$,

$A_3(\sigma) = 1$ if and only if $\sigma \in \Sigma_3''$.

To construct A_0 we fill all its entries initially with 0's and then we make a scan of $\rho(P')$ and each time that we meet a production for $\sigma \in \Sigma''$ that is either of type 1 or of type 3 we fill in the σ -entry in A_0 with 1.

Hence we construct A_0 in no more than

$$O(m^2 + \text{size}(G')) = O(m^2 + \text{size}(G)me) \text{ steps;}$$

recall (see Section 2) that $\text{size}(G') \leq O(\text{size}(G)me)$.

To construct $A_{1,2}$ we fill all its entries initially with 0's and then for each $[X, Y] \in \Sigma''$ we check whether or not $M_Y^*(X, Y) = 1$. If $M_Y^*(X, Y) = 1$, then we set $A_{1,2}([X, Y]) = 1$ (see Lemma 3.1.(1)). Hence, the construction of $A_{1,2}$ can be accomplished in no more than $O(m^2 + m^2) = O(m^2)$ steps.

To construct $A_{2,3}$ we fill all its entries initially with 0's and then for each $[X, Y] \in \Sigma''$ we check for each $Z \in \Sigma$ whether or not $M_Y^*(X, Z) = 1$ and $A_0([Z, Y]) = 1$. If both answers are positive, then we set $A_{2,3}([X, Y]) = 1$. Hence, the construction of $A_{2,3}$ can be accomplished in no more than $O(m^2 + m^3) = O(m^3)$ steps.

To construct A_1 we notice that, for each $[X, Y] \in \Sigma''$, $A_1([X, Y]) = A_{1,2}([X, Y]) \dot{-} A_{2,3}([X, Y])$, where $\dot{-}$ denotes the boolean substraction. Hence A_1 can be constructed in no more than $O(m^2 + m^2) = O(m^2)$ steps.

To construct A_2 we notice that, for each $[X, Y] \in \Sigma''$, $A_2([X, Y])$ is the conjunction of $A_{1,2}([X, Y])$ and $A_{2,3}([X, Y])$.

Hence A_2 can be constructed in no more than $O(m^2 + m^2) = O(m^2)$ steps.

To construct A_3 we notice that, for each $[X, Y] \in \Sigma''$, $A_3([X, Y]) = A_{2,3}([X, Y]) \dot{-} A_{1,2}([X, Y])$.

Hence A_3 can be constructed in no more than $O(m^2 + m^2) = O(m^2)$ steps.

Consequently STEP 6 of our algorithm has time complexity bounded from above by

$$O(m^2 + \text{size}(G)me) + O(m^2) + O(m^3) + O(m^2) + O(m^2) + O(m^2) = O(m^3 + \text{size}(G)me).$$

STEP 7

Construct G_1 .

This is done as follows.

We scan the list $\rho(P')$ and when reading a production π we do the following.

(1) If π is of type 1,

$$\pi = [Y_1, B] \rightarrow a[a, Y_2][Y_3] \cdots [Y_k][A, B]$$

(see point (1) of Definition 1.1) then we distinguish nine cases.

(1.1) If $A_1([a, Y_2]) = 1$ and $A_1([A, B]) = 1$, then we output the production $[Y_1, B] \rightarrow a[Y_3] \cdots [Y_k]$.

(1.2) If $A_1([a, Y_2]) = 1$ and $A_2([A, B]) = 1$, then we output productions $[Y_1, B] \rightarrow a[Y_3] \cdots [Y_k]$ and $[Y_1, B] \rightarrow a[Y_3] \cdots [Y_k][A, B]$.

(1.3) If $A_1([a, Y_2]) = 1$ and $A_3([A, B]) = 1$, then we output the production $[Y_1, B] \rightarrow a[Y_3] \cdots [Y_k][A, B]$.

(1.4) If $A_2([a, Y_2]) = 1$ and $A_1([A, B]) = 1$, then we output productions $[Y_1, B] \rightarrow a[Y_3] \cdots [Y_k]$ and $[Y_1, B] \rightarrow a[a, Y_2][Y_3] \cdots [Y_k]$.

(1.5) If $A_2([a, Y_2]) = 1$ and $A_2([A, B]) = 1$, then we output productions

$$[Y_1, B] \rightarrow a[Y_3] \cdots [Y_k], \quad [Y_1, B] \rightarrow a[a, Y_2][Y_3] \cdots [Y_k],$$

$$[Y_1, B] \rightarrow a[a, Y_2][Y_3] \cdots [Y_k] \text{ and } [Y_1, B] \rightarrow a[a, Y_2][Y_3] \cdots [Y_k][A, B].$$

(1.6) If $A_2([a, Y_2]) = 1$ and $A_3([A, B]) = 1$, then we output productions

$$[Y_1, B] \rightarrow a[Y_3] \cdots [Y_k][A, B] \text{ and } [Y_1, B] \rightarrow a[a, Y_2][Y_3] \cdots [Y_k][A, B].$$

(1.7) If $A_3([a, Y_2]) = 1$ and $A_1([A, B]) = 1$, then we output the production

$$[Y_1, B] \rightarrow a[a, Y_2][Y_3] \cdots [Y_k].$$

(1.8) If $A_3([a, Y_2]) = 1$ and $A_2([A, B]) = 1$, then we output productions

$$[Y_1, B] \rightarrow a[a, Y_2][Y_3] \cdots [Y_k] \text{ and } [Y_1, B] \rightarrow a[a, Y_2][Y_3] \cdots [Y_k][A, B].$$

(1.9) If $A_3([a, Y_2]) = 1$ and $A_3([A, B]) = 1$, then we output the production

$$[Y_1, B] \rightarrow a[a, Y_2][Y_3] \cdots [Y_k][A, B].$$

(2) If π is a production of type 2,

$$\pi = [Y, B] \rightarrow [A, B]$$

(see point (2) of Definition 1.1), then we distinguish two cases.

(1.1) If $A_1([A, B]) = 1$, then we do not have an output.

(1.2) If $A_1([A, B]) = 0$, then we output π itself.

(3) If π is a production of type 3,

$$\pi = [X] \rightarrow a[a, X]$$

(see point (3) of Definition 1.1), then we distinguish three cases.

(3.1) If $A_1([a, X]) = 1$, then we output the production $[X] \rightarrow a$.

(3.2) If $A_2([a, X]) = 1$, then we output productions $[X] \rightarrow a$ and π .

(3.3) If $A_3([a, X]) = 1$, then we output π .

(4) If π is a production of type 4, then we do not have an output.

Hence in executing STEP 7 while reading a production π from $\rho(P')$ we will output at most 4 productions for it (see case 1.5 above). Consequently STEP 7 may be accomplished in no more than $O(\text{size}(G')) = O(\text{size}(G)me)$ steps.

Thus the time complexity of constructing G_1 from G' equals the time complexity of steps 4 through 7. Consequently it can be accomplished in no more than

$$O(\text{size}(G)me + m^3) + O(m^4) + O(\text{size}(G)me + m^3) + (O(\text{size}(G)me) = O(\text{size}(G)me + m^4) \text{ steps.}$$

We can assume now that a list $\rho(P_1)$ of all productions from P_1 is given.

STEP 8

Construct H from G_1 .

This is done as follows.

We scan the list $\rho(P_1)$ and when reading off a production π we do the following

(1) If π is a production of type 1,

$$\pi = [Y_1, B] \rightarrow a[a, Y_2][Y_3] \cdots [Y_k][A, B]$$

(see point (1) of Definition 1.1) then:

for each pair $X, Y \in \Sigma$ we check whether or not

$$M_{Y_2}^*(a, X) = 1 \text{ and } A_0(X, Y_2) = 1 \text{ and } M_B^*(A, Y) = 1 \text{ and } A_0(Y, B) = 1.$$

If the answer is positive then we output the production

$$[Y_1, B] \rightarrow a[X, Y_2][Y_3] \cdots [Y_k][Y, B], \text{ otherwise we do not have an output.}$$

(2) If π is a production from P_1 resulting from a production of type 1 in P' and of the form

$$\pi = [Y_1, B] \rightarrow a[Y_3] \cdots [Y_k][A, B]$$

(see point (1) of Definition 1.1 and STEP 7 of our algorithm)

then for each $Y \in \Sigma$ we check whether or not

$$M_B^*(A, Y) = 1 \text{ and } A_0(Y, B) = 1.$$

If the answer is positive then we output the production

$$[Y_1, B] \rightarrow a[Y_3] \cdots [Y_k][Y, B],$$

otherwise there is no output.

(3) If π is a production from P_1 resulting from a production of type 1 in P' and of the form

$$\pi = [Y_1, B] \rightarrow a[a, Y_2][Y_3] \cdots [Y_k]$$

(see point (1) of Definition 1.1 and STEP 7 of our algorithm)

then for $X \in \Sigma$, we check whether or not

$$M_{Y_2}^*(a, X) = 1 \text{ and } A_0(X, Y_2) = 1.$$

If the answer is positive then we output the production

$$[Y_1, B] \rightarrow a[X, Y_2][Y_3] \cdots [Y_k].$$

(4) If π is a chain production from P_1 then we do not have an output.

(5) If π is a production from P_1 obtained in point (3) of STEP 7, then we distinguish two cases.

$$(5.1) \quad \pi = [X] \rightarrow a[a, X] \text{ for an } X \in \Sigma \text{ and } a \in \Delta.$$

Then for each $Y \in \Sigma$ we check whether or not

$$M_X^*(a, Y) = 1 \text{ and } A_0(Y, X) = 1.$$

If the answer is positive then we output the production $[X] \rightarrow a[Y, X]$, otherwise we do not have an output.

$$(5.2) \quad \pi = [X] \rightarrow a \text{ for an } X \in \Sigma \text{ and } a \in \Delta.$$

Then we output π .

In this way $H = \text{const}(G)$ have been constructed.

Hence in executing STEP 8 while reading a production π from $\rho(P_1)$ we will output no more than m^2 productions for it. Consequently STEP 8 may be accomplished in no more than

$$O(\text{size}(G_1) \cdot m^2) = O(\text{size}(G) \cdot m^3 e) \text{ steps - recall that } \text{size}(G_1) \leq O(\text{size}(G) \cdot m e).$$

Thus the total time complexity of our algorithm is bounded from above by

$$\begin{aligned} &O(\text{size}(G)m e + m^3) + O(\text{size}(G)m e + m^4) + O(\text{size}(G)m^3 e) = \\ &O(\text{size}(G)m^3 e + m^4). \end{aligned}$$

4. THE CORRECTNESS OF THE MAIN CONSTRUCTION

In this section we will prove that $\text{const}(G)$, which is obviously in GNF , is equivalent to G . Let G be an arbitrary Λ -free cf grammar and let $H = \text{const}(G)$.

Theorem 4.1. H is in GNF and $H \sim G$.

Proof.

It follows directly from the construction of H from G that H is in GNF .

Clearly, to prove that $H \sim G$ it suffices to prove that $G' \sim G$. In order to prove this we introduce the "auxiliary" cf grammar $G'' = (\Sigma'', \Delta'', P'', S'')$ where $\Sigma'' = \Sigma'$, $\Delta'' = \Delta$, $S'' = S'$ and $P'' = \bar{P}_1 \cup P_2 \cup P_3 \cup P_4$, where \bar{P}_1 is the set of productions obtained as follows:

for each production $\pi = A \rightarrow Y_1 \cdots Y_k$, $k \geq 2$, in P , each $B \in \Sigma \setminus \Delta$ and each $\alpha \in \Delta$

such that $B \vdash^{l*} A$ and $Y_2 \vdash^{l*} \alpha$, \bar{P}_1 contains the production

$$[Y_1, B] \rightarrow [Y_2][Y_3] \cdots [Y_k][A, B].$$

Lemma 4.1. $G' \sim G''$.

Proof of Lemma 4.1

Note that G' results from G'' by the finite number of applications of the following *sub* transformation.

Take a production $\pi = [Y_1, B] \rightarrow [Y_2] \cdots [Y_k][A, B]$ from \bar{P}_1 and consider all productions for $[Y_2]$ in P'' , i.e., the set of productions from P'' of the form $[Y_2] \rightarrow \alpha[\alpha, Y_2]$ where $\alpha \in \Delta$. Now replace π by the set of productions

$\{[Y_1, B] \rightarrow \alpha[\alpha, Y_2][Y_3] \cdots [Y_k][A, B] : [Y_2] \rightarrow \alpha[\alpha, Y_2] \in P''\}$. Clearly this is a

sub transformation.

Now the result follows from Lemma 0.1. ■

Lemma 4.2. For each $X \in \Sigma$, $L(G, X) = L(G'', [X])$.

Proof of Lemma 4.2

(i) For each $X \in \Sigma$, $L(G, X) \subseteq L(G'', [X])$.

To prove this statement we notice that it is equivalent to the statement:

(i') For each $X \in \Sigma$ and each $n \in \mathbf{N}$, $L(G, X, n) \subseteq L(G'', [X])$.

This is proved by induction on n as follows.

Basis: $n = 0$. Let $X \in \Sigma$ and consider $L(G, X, 0)$. If $X \in \Sigma \setminus \Delta$, then $L(G, X, 0) = \emptyset$ and the statement trivially holds. If $X \in \Delta$, then $L(G, X, 0) = \{X\}$. But in G'' we have $[X] \underset{G''}{=} X[X, X] \underset{G''}{=} X$ and so $X \in L(G'', [X])$.

Induction step: assume that (i') holds for all $0 \leq m < n$.

Now let $X \in \Sigma$ and consider $L(G, X, n)$. If $X \in \Delta$ then $L(G, X, 0) = \{X\}$ and, by the argument as above, the statement holds. Assume then that $X \in \Sigma \setminus \Delta$. Let $w \in L(G, X, n)$ and let us consider a derivation tree T in G corresponding to a derivation of w in G from X in no more than n steps.

T has the following form:

Figure 4.1.

where we use, in the obvious way, vectors of nonnegative integers as nodes, X is the label of the root of T , symbols A_1, \dots, A_t label all other inner nodes on the maximal left path of T (that is depicted in the bold face), a is the label of the leaf on the maximal left path and symbols $A_{i,j}$ label all nodes that are not on the maximal left path but are children of nodes on the maximal left path. For $1 \leq r \leq t+1$ and $1 \leq s \leq u_r$, $T_{r,s}$ is the subtree of T rooted at $\langle r, s \rangle$.

We will construct now a derivation tree T'' in G'' corresponding to a derivation of w in G'' from $[X]$. We start by considering the tree (not a derivation

tree!) T_1 constructed on the set of (unlabelled) nodes

$$\{ \langle 0,0 \rangle, \langle 1,0 \rangle, \dots, \langle 1,u_1 \rangle, \dots, \langle t+1,0 \rangle, \dots, \langle t+1,u_{t+1} \rangle \}$$

from T (that is nodes that either lie on the leftmost maximal path in T or are children of nodes on this path) and two additional nodes v and \bar{v} ; the original nodes from T get in T_1 new labels. T_1 looks as follows:

Figure 4.2.

where the maximal right path is depicted in bold face.

Now notice that all the expansions of nodes on the maximal right path of T_1 (except for v which is not expanded) correspond to productions from P'' : v is expanded using a production from P_3 and all other nodes (except for \bar{v}) on this path are expanded using productions from \bar{P}_1 (*)

It is also easily proved (by induction on the length of the maximal left path) that if \bar{T} denotes the tree obtained from T by deleting all subtrees $T_{1,1}, \dots, T_{1,u_1}, \dots, T_{t+1,1}, \dots, T_{t+1,u_{t+1}}$ except for their roots then
 $(yield(\bar{T}))[X,X] = yield(T_1)$ (**)

Next, starting from T_1 we will obtain the desired tree T'' as follows.

- (a) Expand the node \bar{v} by using production $[X,X] \rightarrow \Lambda$.
- (b) Consider the node $\langle r,s \rangle$ where $1 \leq r \leq t+1$ and $1 \leq s \leq u_r$. Notice that this node contributes in T a subword $w_{\langle r,s \rangle}$ to w , where $w_{\langle r,s \rangle} \in L(G, A_{r,s}, f)$ for some $f < n$. Thus by the inductive assumption $w_{\langle r,s \rangle} \in L(G'', [A_{r,s}])$ and consequently using productions from P'' we can construct a subtree of T_1 rooted in $\langle r,s \rangle$ such that its yield equals $w_{\langle r,s \rangle}$.

After the above construction has been applied to all nodes $\langle r,s \rangle$ in T_1 we obtain the tree T'' .

Now: $yield(T'') = yield(T)$ follows from (**); the fact that T'' is a derivation tree of w from $[X]$ in G'' follows from the construction of T'' and the observation (*).

Thus (i') and hence (i) holds.

(ii) For each $X \in \Sigma$, $L(G'', [X]) \subseteq L(G, X)$.

To prove this statement we notice that it is equivalent to the statement:

(ii') For each $X \in \Sigma$ and each $n \in \mathbf{N}$, $L(G'', [X], n) \subseteq L(G, X)$.

This statement is proved by induction on n as follows.

Basis: note that for $n = 0$ and $n = 1$, $L(G'', [X], n) = \phi$ for each $X \in \Sigma$. Hence we may start our induction from $n = 2$. Let $X \in \Sigma$. If $X \in \Sigma \setminus \Delta$, then $L(G'', [X], 2) = \phi$ and so $L(G'', [X], 2) \subseteq L(G, X)$. If $X \in \Delta$ then $L(G'', [X], 2) = \{X\}$. But $X \in L(G, X)$ and so $L(G'', [X], 2) \subseteq L(G, X)$.

Induction step: assume that (ii') holds for all $2 \leq m < n$.

Now let $X \in \Sigma$ and consider $L(G'', [X], n)$. If $X \in \Delta$, then $L(G'', [X], n) = \{X\}$ and so (by the argument as above) the statement holds. Assume then that $X \in \Sigma \setminus \Delta$ and consider a word $w \in L(G'', [X], n)$. Let us consider a derivation tree U'' in G'' corresponding to a derivation in G'' of w from $[X]$ in no more than n steps. By the form of productions in P'' , U'' must have the following form:

Figure 4.3.

where we use, in the obvious way, vectors of nonnegative integers as nodes, $[X]$ is the label of the root of U'' and the labelling of nodes follows from productions used to expand nodes in U'' . For $2 \leq i \leq t$ and $1 \leq j \leq u_i$, $U''_{i,j}$ is the subtree of U'' rooted at $\langle i, j \rangle$. The maximal right path of U'' is depicted in the bold

face.

Now if \bar{U}'' is the tree resulting from U'' by removing subtrees $U_{2,1}'', \dots, U_2'', U_{2,1}'', \dots, U_{t,1}'', \dots, U_{t,u_t}''$ except for their roots, then starting from \bar{U}'' , the reader should have no difficulties in "reversing" the construction from the proof of (i') - this will yield a derivation tree U corresponding to a derivation of w from X in G . We leave this construction to the reader.

Hence $w \in L(G, X)$.

Consequently (ii') and hence (ii) holds.

Now Lemma 4.2 follows from (i) and (ii). ■

If in the statement of Lemma 4.2 we set $X = S$, then we get $L(G) = L(G'')$.

Thus by Lemma 4.1, $G \sim G'$ and consequently $G \sim H$.

Thus the theorem holds. ■

5. COMPLETE BINARY CF GRAMMARS

In this section we will consider the effect that our construction has for complete binary cf grammars.

Definition 5.1. A cf grammar $G = (\Sigma, \Delta, P, S)$ is called a *complete binary* cf grammar (abbreviated *cb grammar*) if each production of P is of the form $A \rightarrow XY$, where $A \in \Sigma \setminus \Delta$ and $X, Y \in \Sigma$. ■

Remark 5.1. In all considerations in the sequel of this paper where we deal with converting a cb grammar G to an equivalent cf grammar \bar{H} in GNF we will neglect all words of length 1 generated by G . This will allow us to skip a number of technicalities that obscure the total picture of the situations encountered. Clearly, once we get \bar{H} , it suffices to add to it a finite number of productions (not exceeding the sum of the number of productions in \bar{H} for its axiom and the number of one-letter words in $L(G)$) so that the resulting grammar will be equivalent to G and it will also be in GNF. ■

The reason that we consider cb grammars separately is that they obviously generate all context free languages (see the remark above) and the derivation trees in cb grammars are complete binary trees (their definition is recalled in Section 6) - a very natural and often considered class of trees. Later in the paper we will consider our construction of converting a cf grammar to a GNF as a tree transformation - in the case when an initial cf grammar is a cb grammar we will deal with transformations of complete binary trees.

Assume that G is a cb grammar. Then it is easily seen that (with obvious modifications) $\text{const}(G)$ can be written as the following grammar \bar{H} .

Definition 5.2. Let $G = (\Sigma, \Delta, P, S)$ be a *cb* grammar. Let \bar{H} be the cf grammar defined as follows.

$$\bar{\Sigma} = \Delta \cup \{\bar{S}\} \cup \bar{\Sigma}_1, \text{ where } \bar{\Sigma}_1 = \{[Y, B] : Y \in \Sigma, B \in \Sigma \setminus \Delta \text{ and } B \vdash^{l_+} Y\},$$

$$\bar{\Delta} = \Delta, \bar{S} = \$ \text{ where } \$ \notin \Delta \cup \Sigma_1,$$

\bar{P} is obtained as in (i) and (ii) below.

(i) Let $[Y, B] \in \bar{\Sigma}_1$.

(i.1) For all $[\alpha, C], [A, B] \in \bar{\Sigma}_1$, such that $A \in \Sigma \setminus \Delta$ and $\alpha \in \Delta$,

$[Y, B] \rightarrow \alpha[\alpha, C][A, B] \in \bar{P}$ whenever $A \rightarrow YC \in P$.

(i.2) For all $[A, B] \in \bar{\Sigma}_1$ such that $A \in \Sigma \setminus \Delta$ and all $\alpha \in \Delta$,

$[Y, B] \rightarrow \alpha[A, B] \in \bar{P}$ whenever $A \rightarrow Y\alpha \in P$.

(i.3) For all $[\alpha, C] \in \bar{\Sigma}_1$ such that $\alpha \in \Delta$,

$[Y, B] \rightarrow \alpha[\alpha, C] \in \bar{P}$ whenever $B \rightarrow YC \in P$.

(i.4) For all $\alpha \in \Delta$,

$[Y, B] \rightarrow \alpha \in \bar{P}$ whenever $B \rightarrow Y\alpha \in P$.

(ii) \bar{P} contains the following productions for $\$$:

for all $\alpha \in \Delta$, $\$ \rightarrow \alpha[\alpha, S] \in \bar{P}$. ■

The transformation described in the above definition will be denoted by const' (and so \bar{H} will be denoted by $\text{const}'(G)$). Note that this transformation yields directly a grammar in *GNF* : neither chain nor erasing productions are introduced.

The above description of the construction of \bar{H} (based on the division of productions in P into $A \rightarrow YC$ and $A \rightarrow Y\alpha$ types) will be quite useful in the proof of the main theorem of Section 7.

From Definition 5.1 and Theorem 4.1 we have immediately the following result.

Theorem 5.1. Let G be a cb grammar. Then $const'(G)$ is in GNF and $const'(G) \sim G$. ■

Example 5.1.

Let G be the cb grammar with the axiom A_1 and the following productions:

$$A_1 \rightarrow A_2 A_3 \mid 1 A_3 \mid A_2 0 \mid 1 0,$$

$$A_2 \rightarrow A_1 A_2 \mid A_1 1,$$

$$A_3 \rightarrow A_1 A_3 \mid A_1 0$$

(This is the grammar from [H] p. 113 modified in the obvious way to an equivalent cb grammar).

Then the grammar $\bar{H} = const'(G)$ has the axiom $\$$ and the following productions:

$$\$ \rightarrow 1[1, A_1],$$

$$[A_1, A_2] \rightarrow 1[1, A_2][A_2, A_1] \mid 1[A_2, A_1] \mid 1[1, A_2][A_2, A_3] \mid 1[A_2, A_2] \mid 1[1, A_2] \mid 1,$$

$$[A_2, A_1] \rightarrow 1[1, A_3][A_1, A_1] \mid 0[A_1, A_1] \mid 1[1, A_3] \mid 0,$$

$$[A_2, A_2] \rightarrow 1[1, A_3][A_1, A_2] \mid 0[A_1, A_2],$$

$$[A_1, A_3] \rightarrow 1[1, A_2][A_2, A_3] \mid 1[A_2, A_3] \mid 1[1, A_3] \mid 0,$$

$$[A_2, A_3] \rightarrow 1[1, A_3][A_1, A_3] \mid 0[A_1, A_3],$$

$$[1, A_1] \rightarrow 1[1, A_3][A_1, A_1] \mid 0[A_1, A_1] \mid 1[1, A_3] \mid 0,$$

$$[1, A_2] \rightarrow 1[1, A_3][A_1, A_2] \mid 0[A_1, A_2],$$

$$[1, A_3] \rightarrow 1[1, A_3][A_1, A_3] \mid 0[A_1, A_3].$$

Hence \bar{H} has the following parameters: 10 nonterminals, 27 productions, $maxr(\bar{H}) = 3$ and $size(\bar{H}) = 86$. ■

We conclude this section by estimating basic parameters of $\bar{H} = const'(G)$ in terms of the corresponding parameters for G .

(i) *The cardinality of the alphabet $\Sigma_{\bar{H}}$.*

It follows directly from the definition of \bar{H} that

$$\#\Sigma_H \leq \#\Sigma \cdot \#(\Sigma \setminus \Delta) + 1 + \#\Sigma \leq (\#\Sigma)^2.$$

(ii) *The maximal length of the right hand side of a production in P_H .*

It follows directly from the definition of \bar{H} that $\max(\bar{H}) = 3$.

(iii) *The cardinality of the set of productions P_H .*

We note that, for each production $\pi \in P$:

(1) points (i.1) and (i.2) of Definition 5.2 will yield no more than $\#(\Sigma \setminus \Delta) \cdot \#\Delta$ productions, and

(2) points (i.3) and (i.4) of Definition 5.2 will yield no more than $\#\Delta$ productions.

For the axiom $\$$ we have $\#\Delta$ productions in P_H .

Hence

$$\#P_H \leq \#P \cdot \#(\Sigma \setminus \Delta) \cdot \#\Delta + \#P \cdot \#\Delta + \#\Delta \leq 2\#P\#(\Sigma \setminus \Delta)\#\Delta.$$

(iv) *The size of \bar{H} .*

Since for each $\pi = A \rightarrow \alpha \in P_H$, $|A\alpha| \leq 4$, we have

$$\text{size}(\bar{H}) \leq 4\#P_H \leq 8\#P\#(\Sigma \setminus \Delta)\#\Delta.$$

But G is a binary cf grammar and so we have

$$\text{size}(G) = 3\#P_G.$$

Consequently

$$\text{size}(\bar{H}) \leq \frac{8}{3}\text{size}(G) \cdot \#(\Sigma \setminus \Delta) \cdot \#\Delta.$$

(v) *The norm of \bar{H} .*

$$\text{norm}(\bar{H}) = \text{size}(\bar{H}) \cdot \log_2(\#\Sigma_H).$$

Since $\#\Sigma_H \leq (\#\Sigma)^2$,

$$\begin{aligned} \text{norm}(\bar{H}) &\leq 2\text{size}(\bar{H})\log_2(\#\Sigma) \leq \frac{16}{3}\text{size}(G)\#(\Sigma \setminus \Delta)\#\Delta \\ &\leq \frac{16}{3}\text{size}(G)\#(\Sigma \setminus \Delta)\#\Delta \cdot \log_2(\#\Sigma) = \\ &= \frac{16}{3}\text{norm}(G)\#(\Sigma \setminus \Delta)\#\Delta. \end{aligned}$$

We conclude this section by discussing an algorithm implementing *const*.

This algorithm consists of three steps.

We will use the notation from Definition 5.2, i.e., we start with a *cb* grammar $G = (\Sigma, \Delta, P, S)$ and we construct a cf grammar $\bar{H} = (\bar{\Sigma}, \Delta, \bar{P}, \bar{S})$ in GNF.

We assume that the alphabet Σ is linearly ordered and so $\Sigma = (\sigma_1, \dots, \sigma_m)$ for some $m \geq 2$. Let $\#\Delta = e$. We assume that a list $\rho(P)$ of all productions in P is available.

STEP 1

Construct the incidence matrix M_{\vdash^l} for the relation \vdash^l .

This is done exactly as in Section 3. The time complexity of this algorithm is bounded from above by $O(\text{size}(G) + m^2)$.

STEP 2

Construct the incidence matrix M_{\vdash^+} for the relation \vdash^+ .

Again, as in Section 3, we can use here the Warshall algorithm and accomplish this step in no more $O(m^3)$ steps.

STEP 3

Construct $P_{\bar{H}}$.

This is done as follows.

We scan the list $\rho(P)$ and when reading a production π we apply one of the rules (i.1) through (i.4) from Definition 5.2 to output a production in \bar{P} . Finally we output productions for \bar{S} . Clearly, the most work is required when following rule (i.1) of Definition 5.2 and so it suffices to estimate the number of steps for doing it.

To implement rule (i.1) we consider all pairs $B \in \Sigma \setminus \Delta$ and $a \in \Delta$ and for each such pair we check using M_{\vdash^+} whether or not $B \vdash^+ A$ and $C \vdash^+ a$, where $\pi = A \rightarrow YC$ in the notation of Definition 5.2. If the answer is positive, then we output productions

$[Y, B] \rightarrow a[a, C][A, B]$ and $[Y, B] \rightarrow a[a, C]$

(the second one satisfies point (i.3) of Definition 5.2).

If the answer is negative but $C \vdash^{\iota_+} a$ then we output only the production $[Y, B] \rightarrow a[a, C]$.

Hence STEP 3 can be accomplished in no more than $O(\text{size}(G)me)$ steps.

Consequently the time complexity of our algorithm is bounded from above by $O(\text{size}(G) + m^2) + O(m^3) + O(\text{size}(G)me) = O(\text{size}(G)me + m^3)$.

6. COMPLETE BINARY AND LEFT NEEDLE TREES

This section considers two classes of trees: complete binary trees and left needle trees. The first class arises when derivation trees in *cb* grammars are considered and the second class arises when derivation trees in grammars resulting from converting *cb* grammars to *cf* grammars in *GNF* (using our method) are considered. In this and the next section it will be very convenient to consider trees with edge labelling by labels l, m, r (corresponding to *left*, *middle* and *right*, respectively). This will simplify considerably the description of the transformation of trees that we will introduce.

The main aim of this section is to consider specific transformations of complete binary trees into left needle trees; these transformations will form an important tool in the considerations of the next section.

We start by recalling the definition of a complete binary tree.

Definition 6.1. A *complete binary tree* (a *cb tree* for short) is defined inductively as follows:

(0) A tree consisting of one node only is a *cb tree*.

(1) Let T be a *cb tree* and let v be a leaf of T .

Let T_1, T_2 be disjoint *cb trees* such that their sum is disjoint with T . Let T' be a tree resulting from taking $T \cup T_1 \cup T_2$ and adding two new edges: one leading from v to $root(T_1)$ labelled l and the other one leading from v to $root(T_2)$ labelled r . Then T' is a *cb tree*. ■

The only difference with the standard definition of a *cb tree* is that we have now the labelling of edges by l and r symbols. This labelling automatically assigns the linear order among children of any node: the *left child* (the node to which an edge labelled l is leading) precedes the *right child* (the node to which

an edge labelled τ is leading). Note that according to the above definition each inside node has exactly one left and one right child. Hence (taking into the account the consistency of labelling by l and τ with the ordering of children of each node as out lined above) the usual notion of a left (right) path coincides here with the notion of a path going through left (right) labelled edges only.

In a more informal way the above definition can be paraphrased as follows.

- (0) A tree of the form $\begin{array}{c} T \\ \tau \\ T_1 \end{array}$ is a *cb* tree.
- (1) A tree of the form $\begin{array}{c} T \\ l \\ T_1 \end{array}$ is a *cb* tree.

Figure 6.1.

where T, T_1, T_2 are *cb* trees and v is a leaf of T , is a *cb* tree.

Example 6.1. The following tree is a *cb* tree:

Figure 6.2.

We introduce now the class of left needle trees.

Definition 6.2. A *left needle tree* (a *ln* tree for short) is a ternary tree defined inductively as follows.

- (0) A tree consisting of two nodes connected by an edge labelled l is a *ln* tree.
- (1) Let $n \geq 1, T_1, \dots, T_n$ be pairwise disjoint *ln* trees and let v, \bar{v} be two different nodes that do not belong to the set of nodes of $\bar{T} = T_1 \cup \dots \cup T_n$. Let U be the tree constructed as follows.
 - (i) Let T' be the tree consisting of two nodes, v and \bar{v} , and an edge labelled l leading from v to \bar{v} .
 - (ii) Let U be the tree resulting from $T' \cup \bar{T}$ by adding an edge labelled m lead-

ing from v to $root(T_1)$ and , for $1 \leq i \leq n-1$, an edge labelled r leading from $root(T_i)$ to $root(T_{i+1})$; $root(u) = v$. ■

In a more informal way the above definition can be paraphrased as follows.

(0) A tree of the form is a *ln* tree.

(1) A tree of the form

Figure 6.3.

where $n \geq 1$ and T_1, \dots, T_n are *ln* trees, is a *ln* tree.

In a *ln* tree we use labels l , m and r to label edges. Hence for each inside node we can classify each child of it as either *left*, or *right* or a *middle child* (depending on whether an edge labelled l , r or m respectively is leading to it). Again this labelling induces the linear order among children of each node if we assume that "*left precedes middle*" and "*middle precedes right*". Note that the above definition guarantees that among all children of a given node at most one can be left, at most one can be right and at most one can be middle child.

Thus now we talk about a *left (right) path* in a *ln* tree as a path going through l labelled (r labelled) edges. Then all other related notions such as a *complete left (right) path* and the *maximal left (right) path* carry over accordingly.

Example 6.2.

The following three trees are *ln* trees:

Figure 6.4.

■

We will describe now a transformation from the class of *cb* trees into the class of *ln* trees; for a *cb* tree T this transformation is denoted by δ_T .

Definition 6.3. Let T be a *cb* tree.

- (0) If T is a one node tree, then $\delta_T(T)$ is a *ln* tree consisting of two nodes connected by an edge labelled l .
- (1) Let T contain at least three nodes and let $v_1, v_2, \dots, v_n, v_{n+1}$, $n \geq 1$, be the sequence of nodes corresponding to the maximal left path of T , where $v_1 = \text{root}(T)$, v_{n+1} is a leaf, and T_1, \dots, T_n are subtrees of T rooted at the right children of v_1, \dots, v_n respectively. Let us assume that $\delta_{T_1}(T_1), \dots, \delta_{T_n}(T_n)$ are pairwise disjoint, v is a node different from v_{n+1} and neither v nor v_1 is included in the set of nodes of $\bar{T} = \delta_{T_1}(T_1) \cup \dots \cup \delta_{T_n}(T_n)$. Let T' be the *ln* tree consisting of nodes v, v_{n+1} and an edge labelled l leading from v to v_{n+1} . Let $\delta_T(T)$ be the tree resulting from $T' \cup \bar{T}$ by adding an edge labelled m leading from v to $\text{root}(T_n)$ and, for $2 \leq i \leq n$, an edge labelled r leading from $\text{root}(T_i)$ to $\text{root}(T_{i-1})$. ■

In a more informal way the above definition can be paraphrased as follows.




- (0) If T is of the form , then $\delta_T(T)$ is of the form .
- (1) If T is of the form 

Figure 6.5.

where the maximal left path is depicted in the bold face, then $\delta_T(T)$ is of the form

Figure 6.6.

Now we will make our mappings δ_T more specific (detailed) by providing, for each *cb* tree T , the correspondence between V_T and $V_{\delta_T(T)}$ and the correspondence between E_T and $E_{\delta_T(T)}$. We need some additional terminology first.

Let T be a *cb* tree. An edge of T incident with a leaf is called an *outside edge* (of T); all other edges are called *inside edges* (of T). An inside edge labelled by l is called a *left inside edge* and an inside edge labelled by r is called a *right inside edge*. The sets of outside, left inside and right inside edges of T are denoted by $out(T)$, $lin(T)$ and $rin(T)$ respectively.

Definition 6.4. Let T be a *cb* tree.

(1) μ_T is the function from V_T into $V_{\delta_T(T)}$ defined as follows.

(1.0) If T is a one node tree, say $V_T = \{v\}$, then $\mu_T(v)$ is the leaf of $\delta_T(T)$.

(1.1) Let T contain at least three nodes. In the notation from the definition of $\delta_T(T)$ we have:

$$\mu_T(v_{n+1}) = v_{n+1},$$

for $1 \leq i \leq n$, $\mu_T(v_i) = \text{root}(\delta_{T_i}(T_i))$ and

for $u \in V_{T_i}$, $\mu_T(u) = \mu_{T_i}(u)$.

(2) ψ_T is the function from E_T into $E_{\delta_T(T)}$ defined as follows. Let T be a *cb* tree with at least three nodes.

(i) If $e \in out(T)$ is incident with a leaf u , then $\psi_T(e)$ is the edge (labelled by l) of $\delta_T(T)$ leading to $\mu_T(u)$.

(ii) If $e \in lin(T)$ is leading to a node u , then $\psi_T(e)$ is the edge of $\delta_T(T)$ labelled by r and leading from $\mu_T(u)$.

(iii) If $e \in \text{rin}(T)$ is leading from a node u , then $\psi_T(e)$ is the edge of $\delta_T(T)$ labelled by m and leading from $\mu_T(u)$. ■

It is easily proved by induction (following the inductive definition of $\delta_T(T)$) that functions μ_T and ψ_T are well defined, i.e., for each $u \in V_T$, $\mu_T(u)$ exist (and is unique) and for each $e \in E_T$, $\psi_T(e)$ exists (and is unique).

In the sequel we will write δ , μ and ψ rather than δ_T , μ_T and ψ_T whenever it will not lead to confusion.

Example 6.3.

Let T be the following *cb* tree:

Figure 6.7.

where we have identified each node by a positive integer (from 1 through 15) and each edge also by a positive integer (from 1 through 14). This will allow us to demonstrate easily functions μ_T and ψ_T .

Then $\delta_T(T)$ is the following *ln* tree:

Figure 6.8.

Here, again, nodes (except for the root) and edges (except for the edge labelled m leaving the root) are identified by positive integers. In this way this figure gives functions μ_T and ψ_T if we assume the following equalities:

for $1 \leq i \leq 15$, $\mu_T(i) = i$, and

for $1 \leq j \leq 14$, $\psi_T(j) = j$.

The following two properties of transformations δ follow easily from the definitions.

Lemma 6.1. If T is a *cb* tree with $front(T) = u_1 \cdots u_k$, where $k \geq 1$, and u_1, \dots, u_k leafs of T , then $front(\delta(T)) = \mu(u_1) \cdots \mu(u_k)$. ■

Lemma 6.2. Let T be a *cb* tree. Let $E' = E_{\delta_T(T)} \setminus \{e_m\}$ where e_m is the edge of $\delta(T)$ labelled m and leading from $root(\delta(T))$ and let $V' = V_{\delta_T(T)} \setminus \{root(\delta(T))\}$. Let $\psi': E_T \rightarrow E'$ be such that, for each $e \in T$, $\psi'(e) = \psi(e)$ and let $\mu': V_T \rightarrow V'$ be such that, for each $u \in V_T$, $\mu'(u) = \mu(u)$.

Then ψ' and μ' are bijections (injective onto functions.) ■

Now one can define, in the obvious way, the inverse transformation that for a given *ln* tree U yields the *cb* tree $\delta^{-1}(U)$. It is easily seen that the following result holds.

Lemma 6.3. (1) Let T be a *cb* tree and let $U = \delta(T)$. Then $\delta^{-1}(u) = T$. (2) Let U be a *ln* tree and let $T = \delta^{-1}(U)$. Then $\delta(T) = U$. ■

Finally we extend our transformation to the case of node labelled *cb* trees (*nlcb tree* for short).

We remark here that *we consider only such nlcb trees where the set of labels used to label leafs (called leaf labels) is disjoint with the set of labels used to label insider nodes (called inside labels).*

Definition 6.5. Let T be a *nlcb* tree. Then $\delta(T)$ is the node labelled *ln* tree (*nl_{ln} tree* for short) such that

- (1) $un(\delta(T)) = \delta(un(T))$,
- (2) $\Gamma_{\delta_T(T)} = \Gamma' \cup \{\$ \}$ where $\$ \notin \Gamma'$, $lab_{\delta_T(T)}(root(\delta(T))) = \$$ and, for each $u \in V_{\delta_T(T)} \setminus \{root(\delta(T))\}$, $lab_{\delta_T(T)}(u) \neq \$$,
- (3) for each $u \in V_T$, $lab_T(u) = lab_{\delta_T(T)}(\mu(u))$. ■

The symbol $\$$ above is referred to as the *root symbol* and it is a reserved symbol used to label roots of the trees of the form $\delta(T)$ where T is a *nlcb* tree (recall also that this symbol is the axiom of *const*(G) - see Definition 5.2).

Note that Lemma 6.2 and the definition of $lab_{\delta_T(T)}$ yield the following result.

Lemma 6.4. Let T be a *nlcb* tree. Then $yield(T) = yield(\delta(T))$. ■

Obviously Lemmas 6.2, 6.3 and 6.4 carry over to *nlcb* and *nltn* trees.

Now we define a transformation which relabels *nlcb* trees.

For an alphabet Σ , $\Sigma^{(2)}$ denotes the alphabet $\{[X, Y] : X, Y \in \Sigma\}$. If $\sigma = [X, Y] \in \Sigma^{(2)}$ then we refer to X as the *first component* of σ and denote it by $fic(\sigma)$, and we refer to Y as the *second component* of σ and denote it by $sec(\sigma)$.

Definition 6.6. Let T be a *nlcb* tree.

(1) The *pair labelling function induced by T* , denoted by $lab_T^{(2)}$, is the function from V_T into $\Gamma_T^{(2)}$ defined by:

for each leaf u of T , $lab_T^{(2)}(u) = lab_T(u)$,

for each inside node u of T , $lab_T^{(2)}(u) = [lab_T(u_1), lab_T(u_2)]$ where u_1 is the left child of u and u_2 is the top node of the complete left path on which u lies.

(2) The *pair version of T* , denoted $T^{(2)}$, is the *nlcb* tree such that $un(T^{(2)}) = un(T)$, $\Gamma_{T^{(2)}} = \Gamma_T^{(2)}$ and $lab_{T^{(2)}} = lab_T^{(2)}$. ■

For a *nlcb* tree T , $pair_T$ will denote the transformation leading from T to $T^{(2)}$.

Example 6.4. Let T be the following *nlcb* tree:

Figure 6.9.

(Note that $un(T)$ is the cb tree from Example 6.3). Then $T^{(2)}$ is the following $nldb$ tree:

Figure 6.10.

■

Lemma 6.5. Let T be a $nldb$ tree.

- (1) If u_1, u_2 are inside nodes of $\delta(T^{(2)})$ that lie on the same right path and σ_1, σ_2 are labels of u_1, u_2 respectively, then $\sec(\sigma_1) = \sec(\sigma_2)$.
- (2) If u_1 is a middle node of $\delta(T^{(2)})$, u_2 is the left sibling of u_1 and σ_1, σ_2 are labels of u_1, u_2 respectively, then $fic(\sigma_1) = fic(\sigma_2)$ (moreover $fic(\sigma_1)$ is a leaf label). ■

7. TRANSFORMING DERIVATION TREES

In this section we will consider the effect of *const* on the derivation trees of a given Λ -free cf grammar. As a matter of fact we will demonstrate that *const* can be considered as the transformation of *cb* trees into *ln* trees.

In order not to complicate the (already involved) notation and terminology, in this section we will restrict ourselves to the case of *cb* grammars; that is we will consider the transformation *const'* rather than *const*. It is rather easy (although notationally tedious) to extend the considerations of this section to the case of arbitrary Λ -free cf grammars (hence to the case of *const* transformation). At the end of this section we comment briefly on the general case.

Before we state and prove the main result of this section we need an additional terminology. For a node labelled tree T and its node u we define (i) *the structure of u in T* to be the full subgraph of T spanned on the node u , its parent and all its children, (ii) *the lower structure of u in T* to be the full subgraph of T spanned on the node u and all its children.

Let G be an arbitrary complete binary cf grammar and let $\bar{H} = \text{const}'(H)$.

Note that a derivation tree in \bar{H} is obtained by starting at its root and expanding successively its nodes by productions from \bar{H} . In order to get derivation trees in \bar{H} to be *ln* trees we have to assign labels l , m and r to edges leading from the node being expanded. This rule is given, somewhat informally, as follows.

(1) if a production $[Y, B] \rightarrow \alpha[\alpha, C][A, B]$ from group (i.1) of Definition 5.2 is used, then the labelling is as follows:

Figure 7.1.

(2) if a production $[Y, B] \rightarrow a[A, B]$ from group (i.2) of Definition 5.2 is used, then the labelling is as follows:

Figure 7.2.

(3) if a production $[Y, B] \rightarrow a[a, C]$ from group (i.3) of Definition 5.2 is used, then the labelling is as follows:

Figure 7.3.

(4) if a production $[Y, B] \rightarrow a$ from group (i.4) of Definition 5.2 is used, then the labelling is as follows:

Figure 7.4.

(5) if a production $\$ \rightarrow a[a, S]$ from group (ii) of Definition 5.2 is used, then the labelling is as follows:

Figure 7.5.

Theorem 7.1. Let $w \in L(G)$, where $|w| \geq 2$ and let T be a derivation tree corresponding to a derivation of w in G . Then $\delta(T^{(2)})$ is a derivation tree of w in \bar{H} and $yield(\delta(T^{(2)})) = w$.

Proof.

Clearly T is a *cb* tree node labelled by Σ (with leafs labelled by Δ).

(i) $yield(\delta(T^{(2)})) = w$ follows from Lemma 6.4 and from the equality $lab_T(u) = lab_{T^{(2)}}(u)$ for each leaf u of T .

(ii) To prove that $\delta(T^{(2)})$ is a derivation tree in \bar{H} we proceed as follows.

(ii.1) For each inside node u in $V_{T^{(2)}} \setminus \text{root}(V_{T^{(2)}})$ we will consider its (lower) structure in T , then its (lower) structure in $T^{(2)}$ and then the lower structure of $\mu(u)$ in $\delta(T^{(2)})$. We will conclude that the node $\mu(u)$ in $\delta(T^{(2)})$ is expanded using a production from \bar{P} . The following cases exhaust all possibilities.

(ii.1.1) Let u be a left inside node of T with the following structure in T :

Figure 7.6.

where s is a leaf, u_2 is an inside node, \bar{v} is the top node of the complete left path on which u lies; it may be that $\bar{v} = v$ and it may be that $p = u_2$. The structure of u is depicted by using the rectangular frame.

Then the structure of u in $T^{(2)}$ is:

Figure 7.7.

Then the lower structure of $\mu(u)$ in $\delta(T^{(2)})$ is

Figure 7.8.

Hence $\mu^{T^{(2)}}(u)$ is expanded using production $[Y, B] \rightarrow a[a, C][A, B]$.

Since the production $A \rightarrow YC$ (used to expand u in T) is in P , the definition of \bar{H} (see (i.1) of Definition 5.2) implies that the production $[Y, B] \rightarrow a[a, C][A, B]$ is in \bar{P} ; consequently $\mu(u)$ is expanded using a production from \bar{P} .

(ii.1.2.) Let u be a left inside node of T with the following structure in T :

Figure 7.9.

where u_2 is a leaf, \bar{v} is the top node of the complete left path on which u lies; it may be that $\bar{v} = v$. The structure of u is depicted by using the rectangular frame.

Then the structure of u in $T^{(2)}$ is:

Figure 7.10.

Then the lower structure of $\mu(u)$ in $\delta(T^{(2)})$ is:

Figure 7.11.

Hence $\mu(u)$ is expanded using production $[Y, B] \rightarrow a[A, B]$.

Since the production $A \rightarrow Ya$ (used to expand u in T) is in P , the definition of \bar{H} (see (i.2) of Definition 5.2) implies that the production $[Y, B] \rightarrow a[A, B]$ is in \bar{P} ; consequently $\mu(u)$ is expanded using a production from \bar{P} .

(ii.1.3.) Let u be a right inside node of T with the following lower structure in T :

Figure 7.12.

where s is a leaf, u_2 is an inside node and it may be that $p = u_2$. The lower structure of u is depicted using the rectangular frame.

Then the lower structure of u in $T^{(2)}$ is:

Figure 7.13.

Then the lower structure of $\mu(u)$ in $\delta(T^{(2)})$ is:

Figure 7.14.

Hence $\mu(u)$ is expanded using production $[Y, B] \rightarrow a[a, C]$.

Since the production $B \rightarrow YC$ (used to expand u in T) is in P , the definition of \bar{H} (see (i.3) of Definition 5.2) implies that the production $[Y, B] \rightarrow a[a, C]$ is in \bar{P} ; consequently $\mu(u)$ is expanded using a production from \bar{P} .

(ii.1.4.) Let u be a right inside node of T with the following lower structure in T :

Figure 7.15.

where u_2 is a leaf.

Then the lower structure of u in $T^{(2)}$ is:

Figure 7.16.

Then the lower structure of $\mu(u)$ in $\delta(T^{(2)})$ is:

Figure 7.17.

Hence $\mu(u)$ is expanded using production $[Y, B] \rightarrow a$.

Since the production $B \rightarrow Ya$ (used to expand u in T) is in P , the definition of \bar{H} (see (i.4) of Definition 5.2) implies that the production $[Y, B] \rightarrow a$ is in \bar{P} ; consequently $\mu(u)$ is expanded using a production from \bar{P} .

(i.2) To complete the proof that $\delta(T^{(2)})$ is a derivation tree in H one has to show that the production used to expand the root of $\delta(T^{(2)})$ is in \bar{P} (and then use Lemma 6.2). This however follows immediately from the definition of \bar{P} .

The theorem follows now from (i) and (ii). ■

We will demonstrate now that the "converse" of Theorem 7.1 also holds, i.e., given a derivation tree in \bar{H} one can directly transform it into a derivation tree (of the same word) in G .

Let Γ_1, Γ_2 be finite alphabets and let $\$ \notin \Gamma_1 \cup \Gamma_2$ (recall that $\$$ is a reserved root symbol). Let $\Gamma = \Gamma_1 \cup \Gamma_2 \cup \{\$\}$.

Definition 7.1. Let U be a *nlln* tree such that $lab_U(root(U)) = \$$, $\Gamma_U = \Gamma^{(2)} \cup \Gamma_2 \cup \{\$\}$, leaves of U are labelled by elements of Γ_2 , inside nodes of U are labelled by elements of $\Gamma^{(2)}$ and moreover the following conditions are satisfied.

- (i) If u_1, u_2 are nodes of U that lie on the same right path, then $sec(lab_U(u_1)) = sec(lab_U(u_2))$.
- (ii) If u_1 is a middle node of U and u_2 is the left sibling of u_1 , then $fic(lab_U(u_1)) = fic(lab_U(u_2))$.

Then

- (1) The *single labelling function induced by U* , denoted $lab_U^{(1)}$, is the function from V_U into Γ , defined by :

$$lab_U^{(1)}(root(U)) = lab_U(root(U)) = \$,$$

$$\text{for each leaf } u, lab_U^{(1)}(u) = lab_U(u),$$

for each bottom node u of a complete inside right path,

$$lab_U^{(1)}(u) = sec(lab_U(u)),$$

$$\text{for every other node } u \text{ of } U, lab_U^{(1)}(u) = fic(lab_U(u_2)),$$

where u_2 is the right child of u .

- (2) The *single version of U* , denoted $U^{(1)}$, is the *nldb* tree such that

$$un(U^{(1)}) = un(U), \Gamma_{U^{(1)}} = \Gamma \text{ and } lab_{U^{(1)}} = lab_U^{(1)} \quad \blacksquare$$

For a *nldb* tree U as above, $sing_U$ will denote the transformation leading from U to $U^{(1)}$.

Using Lemma 6.5 one easily proves the following result.

Lemma 7.1. Let U be a *nl* n tree satisfying the assumptions of the statement of Definition 7.1. Let $T = \delta^{-1}(U^{(1)})$. Then $\delta(T^{(2)}) = U$. ■

Quite analogously to the proof of Theorem 7.1 (and using Lemma 7.1) one can prove the following result.

Theorem 7.2. Let $w \in L(\bar{H})$, where $|w| \geq 2$ and let T be a derivation tree corresponding to a derivation of w in \bar{H} . Then $\delta^{-1}(T^{(1)})$ is a derivation tree of w in G and $yield(\delta^{-1}(T^{(1)})) = w$. ■

As we have indicated already, in this section we have discussed the *const* transformation as the transformation of derivation trees. To avoid the burdening of our (already involved) notation, this discussion was carried on for the case when a given grammar is a *cb* grammar - in this way we have been dealing with a transformation of *cb* trees into *ln* trees.

The reader should be able to extend these considerations to the case of arbitrary cf grammars (the proof of Theorem 4.1 gives basic intuitions of the general situation). We would like now to discuss briefly, and rather intuitively, few points pertinent to such a general situation.

First of all the range of our tree transformations will be the class of trees more general than left needle trees. A *generalized left needle* tree (a *gln tree* for short) is a tree such that each inside node u of it has precisely one leaf among its children and moreover this "leaf child" is the leftmost among all the children of u .

The transformation δ for a tree T will be defined as follows. As before we identify the nodes of T with positive integers so that by using the equality $\mu(i) = i$, when we depict $\delta(T)$ the function μ is also shown.

(0) For a one node tree , $\delta(T)$ is of the form ,

(1) For a tree T of the form

Figure 7.18.

where the maximal left path is depicted in the bold face, $\delta(T)$ is of the form

Figure 7.19.

where for the node $i(1 \leq i \leq n)$ the root of δ is identified with i (and gets the label of i) while the roots of $\delta(T_{i2}), \dots, \delta(T_{im_i})$ become children of i .

Example 7.1. If T is of the form

Figure 7.20.

then $\delta(T)$ will be of the form

Figure 7.21.

Note that nodes 10 and 11 are new "root" nodes; they are not in the range of μ . ■

8. DISCUSSION

In this paper we have defined a transformation (*const*) which given a Λ -free cf grammar G yields a cf grammar $H = \text{const}(G)$ in *GNF*. H is such that $\text{size}(H) \leq \text{size}(G) \cdot 64 \cdot (\#\Sigma)^2 \cdot \#\Delta$, moreover $\text{maxr}(H) \leq \text{maxr}(G) + 1$. Thus in particular, if the initial grammar is binary (e.g. in Chomsky normal form or in complete binary normal form) then the resulting grammar is in 2-*GNF*.

The transformation *const* that we have introduced has another important feature. It can be viewed as the transformation of derivation trees. That is given a derivation tree T of a word w in G one can transform it directly (without having any other information about either G or H) into a derivation tree of a word w in H . In this sense $\text{size}(G)$ (or $\text{norm}(G)$) may be considered to be irrelevant: given a derivation tree of w in G one constructs a derivation tree of w in H without looking either at H or G at all! This transformation of labelled trees (in a setting more general than the family of derivation trees of cf grammars only) was defined and studied. It has also turned out that using the "dual" of this transformation one can apply it to an arbitrary derivation tree U of a word w in H and obtain a derivation tree T of w in G ; again, in this process neither H nor G is looked up; U contains enough information to construct T .

This situation was discussed in detail for *cb* grammars. In this case we have proved that the following diagram commutes:

Figure 8.1.

where \mathbf{T}_1 is the family of all derivation trees for words in $L(G)$, \mathbf{T}_2 is the family of all trees of the form $T^{(2)}$ where $T \in \mathbf{T}_1$, \mathbf{D}_1 is the family of all derivation trees for words in $L(H)$, \mathbf{D}_2 is the family of all trees of the form $T^{(1)}$ where $T \in \mathbf{D}_1$, *pair* is

the family of all mappings of the form $pair_T$, δ is the family of all mappings of the form δ_T , $sing$ is the family of all mappings of the form $sing_T$ and δ^{-1} is the family of all mappings of the form δ_T^{-1} .

We would like to conclude this paper by discussing possible improvements ("short-cuts") when applying our algorithm (*const*). The aim of this discussion is to suggest rather informally certain short-cuts that are dependent on the form of the grammar rather than on the form of the generated language. These short-cuts will be discussed on the hand of a concrete example.

Let $G = (\Sigma, \Delta, P, E)$ where

$\Sigma = \{E, T, F, a, (,), *, +\}$, $\Delta = \{a, *, +, (,)\}$ and

P consists of the following productions

$$E \rightarrow E + T \mid T,$$

$$T \rightarrow T * F \mid F,$$

$$F \rightarrow (E) \mid a.$$

Hence G is the "classical" grammar for the generation of arithmetic expressions.

If we apply *const* (see Definition 1.1) then we first get G' which has the axiom $[E]$ and the following productions:

$$[E, E] \rightarrow +[+, +][T][E, E],$$

$$[T, E] \rightarrow *[*][F][T, E] \mid [E, E],$$

$$[T, T] \rightarrow *[*][F][T, T],$$

$$[(, E] \rightarrow ([[(, E)]][F, E] \mid a[a, E]([F, E],$$

$$[(, T] \rightarrow ([[(, E)]][F, T] \mid a[a, E]([F, T],$$

$$[(, F] \rightarrow ([[(, E)]][F, F] \mid a[a, E]([F, F],$$

$$[F, E] \rightarrow [T, E],$$

$$[F, T] \rightarrow [T, T],$$

$$[a, E] \rightarrow [F, T],$$

$$\begin{aligned}
[a, F] &\rightarrow [F, F], \\
[E] &\rightarrow ([\langle, E] \mid a[a, E], \\
[T] &\rightarrow ([\langle, T] \mid a[a, T], \\
[F] &\rightarrow ([\langle, F] \mid a[a, F], \\
[b] &\rightarrow b[b, b], \text{ for every } b \in \Delta, \\
[X, X] &\rightarrow \Lambda, \text{ for every } X \in \Sigma.
\end{aligned}$$

Then the grammar H' resulting from $H = \text{const}(G)$ by removing useless nonterminals (and productions for them) looks as follows.

Axiom: $[E]$,

Productions:

$$\begin{aligned}
[E, E] &\rightarrow +[T][E, E] \mid +[T], \\
[T, E] &\rightarrow *[F][T, E] \mid *[F][E, E] \mid *[F], \\
[T, T] &\rightarrow *[F][T, T] \mid *[F], \\
[\langle, E] &\rightarrow ([\langle, E][\rangle])[T, E] \mid ([\langle, E][\rangle])[E, E] \mid ([\langle, E][\rangle]) \mid \\
&\quad a[T, E][\rangle][T, E] \mid a[T, E][\rangle][E, E] \mid a[T, E][\rangle] \mid \\
&\quad a[E, E][\rangle][T, E] \mid a[E, E][\rangle][E, E] \mid a[E, E][\rangle] \mid \\
&\quad a[\rangle][T, E] \mid a[\rangle][E, E] \mid a[\rangle], \\
[\langle, T] &\rightarrow ([\langle, E][\rangle])[T, T] \mid ([\langle, E], [\rangle]) \mid \\
&\quad a[T, E][\rangle][T, T] \mid a[T, E][\rangle] \mid \\
&\quad a[E, E][\rangle][T, T] \mid a[E, E][\rangle] \mid \\
&\quad a[\rangle][T, T], \\
[\langle, F] &\rightarrow ([\langle, E][\rangle]) \mid a[T, E][\rangle] \mid a[E, E][\rangle] \mid a[\rangle], \\
[E] &\rightarrow ([\langle, E] \mid a[T, E] \mid a[E, E] \mid a, \\
[T] &\rightarrow ([\langle, T] \mid a[T, T] \mid a, \\
[F] &\rightarrow ([\langle, F] \mid a, \\
[\rangle] &\rightarrow \rangle.
\end{aligned}$$

H' has the following parameters:

10 nonterminals, 41 productions, $\maxr(H') = 4$ and $\text{size}(H') = 153$.

Now we make the following general observation (the easy proof of the following result is left to the reader).

Lemma 8.1. Let $G = (\Sigma, \Delta, P, S)$ be a Λ -free cf grammar. Let $A \in \Sigma \setminus \Delta$, $Z \subseteq \Sigma^+$ and $W \subseteq \Sigma^+$ be such that

- (i) for each $w \in W$, $first(w)$ is such that $\{x \in \Delta^+ : first(w) \Rightarrow^* x\}$ is a singleton containing a word of length 1; let $ter(first(w))$ be this one letter word,
- (ii) for each $z \in Z$ and each $w \in W$, $A \rightarrow zw$ is in P ; let $P_{A,Z,W}$ denote this set of productions.

Let $R = P \setminus P_{A,Z,W} \cup P_1$, where

$$P_1 = \{A \rightarrow zX : A \rightarrow zw \text{ is in } P_{A,Z,W}, z \in Z \text{ and } w \in W\} \cup \\ \cup \{X \rightarrow (ter(first(w)))w' : A \rightarrow zw \text{ is in } P_{A,Z,W}, z \in Z,$$

$w \in W$ and w' results from w by removing its first letter $\}$,

where X is a new nonterminal symbol not in Σ .

Let $I = (\Sigma \cup \{X\}, \Delta, R, S)$. Then $L(I) = L(G)$. ■

Note that the transformation described above yields a cf grammar I in pseudo *GNF* if the given grammar (G) is in (pseudo) *GNF*; also $maxr(I) \leq maxr(G)$.

This observation can be applied (twice) to G' as follows:

- (1) set $Z = \{([([E], a[a, E]), W = \{()][F, E]\}$ and $A = [([E]$, where the involved productions are:

$$([E] \rightarrow ([([E][])][F, E] \mid a[a, E][])][F, E],$$

- (2) set $Z = \{([([E], a[a, E]), W = \{()][F, T]\}$

and $A = [([T]$, where the involved productions are:

$$([T] \rightarrow ([([E][])][F, T] \mid a[a, E][])][F, T].$$

Applying the construction from the lemma above (X will be a new nonterminal symbol used in (1) above and Y will be a new nonterminal symbol used in (2) above) to G' we get G'' which after removing chain and erasing productions (and removing useless nonterminals) yields the following grammar H'' equivalent to G and in GNF .

Axiom: $[E]$.

Productions:

$$[E, E] \rightarrow +[T][E, E] \mid +[T],$$

$$[T, E] \rightarrow *[F][T, E] \mid *[F][E, E] \mid *[F],$$

$$[T, T] \rightarrow *[F][T, T] \mid *[F],$$

$$[(, E] \rightarrow (([E]X \mid \alpha[T, E]X \mid \alpha[E, E]X \mid \alpha X,$$

$$[(, T] \rightarrow (([E]Y \mid \alpha[T, E]Y \mid \alpha[E, E]Y \mid \alpha Y,$$

$$[(, F] \rightarrow (([E][()] \mid \alpha[T, E][()] \mid \alpha[E, E][()] \mid \alpha[()]),$$

$$X \rightarrow)([T, E] \mid)[E, E] \mid),$$

$$Y \rightarrow)([T, T] \mid),$$

$$[E] \rightarrow ([([E] \mid \alpha[T, E] \mid \alpha[E, E] \mid \alpha,$$

$$[T] \rightarrow ([([T] \mid \alpha[T, T] \mid \alpha,$$

$$[F] \rightarrow ([[F] \mid \alpha,$$

$$[]) \rightarrow).$$

H'' has the following parameters:

12 nonterminals, 34 productions, $\maxr(H'') = 3$ and $\text{size}(H'') = 109$; thus H'' is in 2- GNF .

Now we make a digression and state an easy to prove general result on cf grammars. We need a definition first.

Definition 8.1.

(1) Let $G_1 = (\Sigma_1, \Delta_1, P_1, S_1)$, $G_2 = (\Sigma_2, \Delta_2, P_2, S_2)$ be a cf grammars such that $(\Sigma_1 \setminus \Delta_1) \cap (\Sigma_2 \setminus \Delta_2) = \emptyset$. Then the (G_1, G_2) -composition of G_1 and G_2 is the cf gram-

mar.

$(\Sigma_1 \cup \Sigma_2, \Delta_1 \setminus (\Sigma_2 \setminus \Delta_2) \cup \Delta_2 \setminus (\Sigma_1 \setminus \Delta_1), P_1 \cup P_2, S_1)$;

it is denoted by $comp(G_1, G_2)$.

(2) Let $G_1 = (\Sigma_1, \Delta_1, P_1, S_1)$ and $G_2 = (\Sigma_2, \Delta_2, P_2, S_2)$ be cf grammars such that

(2.1) for each $X \in \Sigma$ there exists a $Y \in \Sigma'$ such that $L(G, X) = L(G', Y)$, and

(2.2) $L(G_1) = L(G_2)$.

Then G_2 is an *extension* of G_1 .

(3) Let $G_1 = (\Sigma_1, \Delta_1, P_1, S_1)$ and $G_2 = (\Sigma_2, \Delta_2, P_2, S_2)$ be cf grammars such that

(3.1) $\Sigma_1 \subseteq \Sigma_2$,

(3.2) $S_2 = S_1$, and

(3.3) for each $X \in \Sigma$, $L(G_1, X) = L(G_2, X)$.

Then G_2 is a *strict extension* of G_1 . ■

It is easily seen that if G_2 is an extension of G_1 then by appropriately renaming nonterminals of G_2 one gets G_2' such that G_2' is a strict extension of G_1 .

Also the following result is easy to prove.

Lemma 8.2. Let G_1, G_2 be cf grammars and let $G = comp(G_1, G_2)$. Let G_1' be the strict extension of G_1 and G_2' be the strict extension of G_2 such that the sets of nonterminal symbols of G_1' and G_2' are disjoint. Then $L(G) = L(comp(G_1', G_2'))$. ■

Note that if G_1' and G_2' from the statement of the above lemma are in pseudo *GNF*, then still $comp(G_1', G_2') = \bar{G}$ does not have to be in pseudo *GNF*. However, the following transformation of \bar{G} yields an equivalent cf grammar \bar{G} which will be in pseudo *GNF* (if G_1' and G_2' were in pseudo *GNF*).

Let $G_1' = (\Sigma_1', \Delta_1', P_1', S_1')$ and $G_2' = (\Sigma_2', \Delta_2', P_2', S_2')$ and let $\bar{G} = (\bar{\Sigma}, \bar{\Delta}, \bar{P}, \bar{S})$. A production from \bar{P} is called *bad* if $\pi = A \rightarrow \alpha$, $\alpha \in \bar{\Sigma}^+$, and $first(\alpha) \notin \bar{\Delta}$; not that in this case if $\pi \in P_1'$, then $first(\alpha) \in \Sigma_2' \setminus \Delta_2'$ and if $\pi \in P_2'$, then $first(\alpha) \in \Sigma_1' \Delta_1'$.

Let $\bar{\bar{G}}$ result from \bar{G} by the following construction:

- (i) if $\pi = A \rightarrow \alpha$ is a bad production from \bar{P} such that $\pi \in P_1'$ then remove π from \bar{G} and introduce instead all productions of the form $A \rightarrow \beta\bar{\alpha}$ where $\alpha = (first(\alpha))\bar{\alpha}$ and β is such that $first(\alpha) \rightarrow \beta$ is in P_2' ,
- (ii) if $\pi = A \rightarrow \alpha$ is a bad production from \bar{P} such that $\pi \in P_2'$ then remove π from \bar{G} and introduce instead all productions of the form $A \rightarrow \beta\bar{\alpha}$ where $\alpha = (first(\alpha))\bar{\alpha}$ and β is such that $first(\alpha) \rightarrow \beta$ is in P_1' .

It is clear that $\bar{\bar{G}}$ is in pseudo *GNF* whenever G_1' and G_2' are in pseudo *GNF*.

Now let us go back to our original grammar G .

Clearly $G = comp(G_1, G_2)$ where

$$G_1 = (\Sigma_1, \Delta_1, P_1, E), G_2 = (\Sigma_2, \Delta_2, P_2, F),$$

$$\Sigma_1 = \{E, T, F, +, *\}, \Delta_1 = \{+, *\}, P_1 = \{E \rightarrow E+T, E \rightarrow T, T \rightarrow T*F, T \rightarrow F\},$$

$$\Sigma_2 = \{E, F, (,), \alpha\}, \Delta_2 = \{(,), \alpha\}, P_2 = \{F \rightarrow (E), F \rightarrow \alpha\}.$$

For G_2 we get immediately an equivalent of grammar $\hat{G}_2 = (\hat{\Sigma}_2, \hat{\Delta}_2, \hat{P}_2, F)$ in *GNF*, where

$$\hat{\Sigma}_2 = \{E, F, [], (,), \alpha\}, \hat{\Delta}_2 = \{(,), \alpha\},$$

$$\hat{P}_2 = \{F \rightarrow (E[]), F \rightarrow \alpha, [] \rightarrow \}\}.$$

Clearly \hat{G}_2 is a strict extension of G_2 .

Applying *const* to G_1 (and removing useless nonterminals) we get the following of grammar H_1' which is in *GNF* and equivalent to G_1 .

Axiom: $[E]$.

Productions:

$$\begin{aligned}
[E, E] &\rightarrow +[T][E, E] \mid +[T], \\
[T, E] &\rightarrow *[F][T, E] \mid *[F][E, E] \mid *[F], \\
[T, T] &\rightarrow *[F][T, T] \mid *[F], \\
[E] &\rightarrow F[T, E] \mid F[E, E] \mid F, \\
[T] &\rightarrow F[T, T] \mid F, \\
[F] &\rightarrow F.
\end{aligned}$$

Clearly H_1' is an extension of G_1 , where E in G_1 corresponds to $[E]$ in H_1' and T in G_1 corresponds to $[T]$ in H_1' . Now if in H_1' we replace $[E]$ by E and $[T]$ by T then we get H_1'' such that H_1'' is a strict extension of H_1' .

Now we let $\bar{G} = comp(H_1'', \hat{G}_2)$ and applying comments following Lemma 8.2 we get the following grammar \bar{G} in GNF and equivalent to G .

Axiom: $[E]$

Productions:

$$\begin{aligned}
[E, E] &\rightarrow +T[E, E] \mid +T, \\
[T, E,] &\rightarrow *F[T, E] \mid *F[E, E] \mid *F, \\
[T, T] &\rightarrow *F[T, T] \mid *F, \\
E &\rightarrow (E[])[T, E] \mid \alpha[T, E] \mid \\
&\quad (E[])[E, E] \mid \alpha[E, E] \mid \\
&\quad (E[]) \mid \alpha, \\
T &\rightarrow ([E][]) [T, T] \mid ([E][]) \mid \\
&\quad \alpha[T, T] \mid \alpha, \\
F &\rightarrow ([E][]) \mid \alpha, \\
[] &\rightarrow).
\end{aligned}$$

\bar{G} has the following parameters:

7 nonterminals, 20 productions, $maxr(\bar{G}) = 4$ and $size(\bar{G}) = 69$.

Notice that we can again apply the construction of Lemma 8.1 to $\bar{\bar{G}}$ obtaining the following grammar $\bar{\bar{H}}$ that is in *GNF* and equivalent to G .

Axiom: $[E]$

Productions:

$[E, E] \rightarrow T[E, E] \mid +T,$

$[T, E] \rightarrow *F[T, E] \mid *F[E, E] \mid *F,$

$[T, T] \rightarrow *F[T, T] \mid *F,$

$E \rightarrow ([E]X \mid \alpha[T, E] \mid \alpha[E, E] \mid \alpha,$

$T \rightarrow ([E]Y \mid \alpha[T, T] \mid \alpha,$

$F \rightarrow ([E][()]) \mid A,$

$()] \rightarrow),$

$X \rightarrow) [T, E] \mid) [E, E] \mid),$

$Y \rightarrow) [T, T] \mid).$

$\bar{\bar{H}}$ has the following parameters:

9 nonterminals, 22 productions, $\maxr(\bar{\bar{H}}) = 3$ and $\text{size}(\bar{\bar{H}}) = 67$; thus $\bar{\bar{H}}$ is in 2-*GNF*.

Thus we have demonstrated that various short-cuts lead to a considerable simplification of $\text{const}(G)$. The above discussion was rather informal, however it points out that techniques of Lemma 8.1 and Lemma 8.2 can lead to a "smaller" grammar. The use of these short cuts in the general methodology of getting "small" grammars in *GNF* deserves further study. This topic is a subject of our current research and we hope to be able to report on it in a future paper.

REFERENCES

- [AU] Aho, A.V. and Ullman, J.D., *The theory of parsing, translation and compiling*, Volumes 1 and 2, Prentice Hall, Englewood Cliffs, 1972.

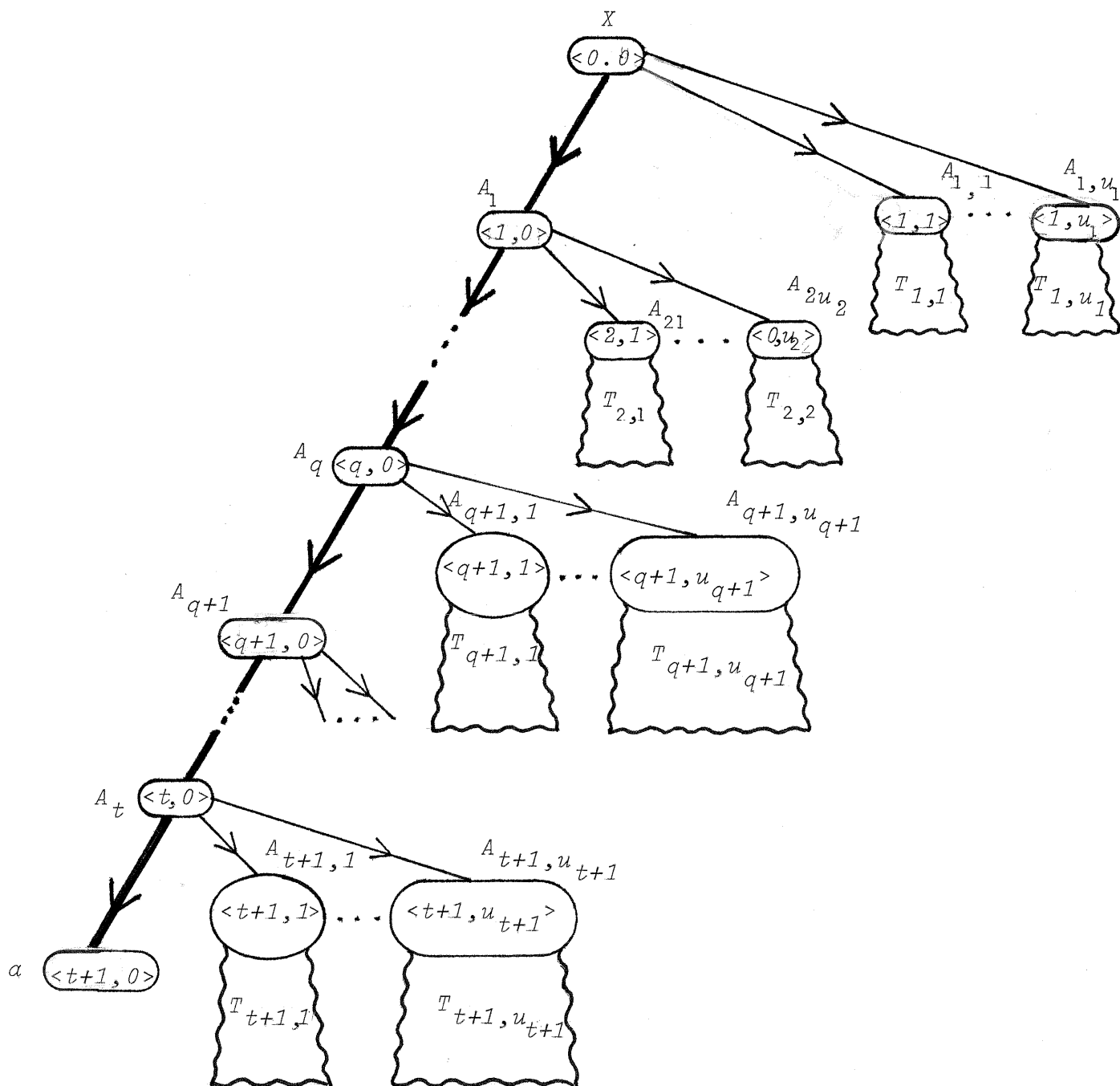


Figure 4.1

T_1 :

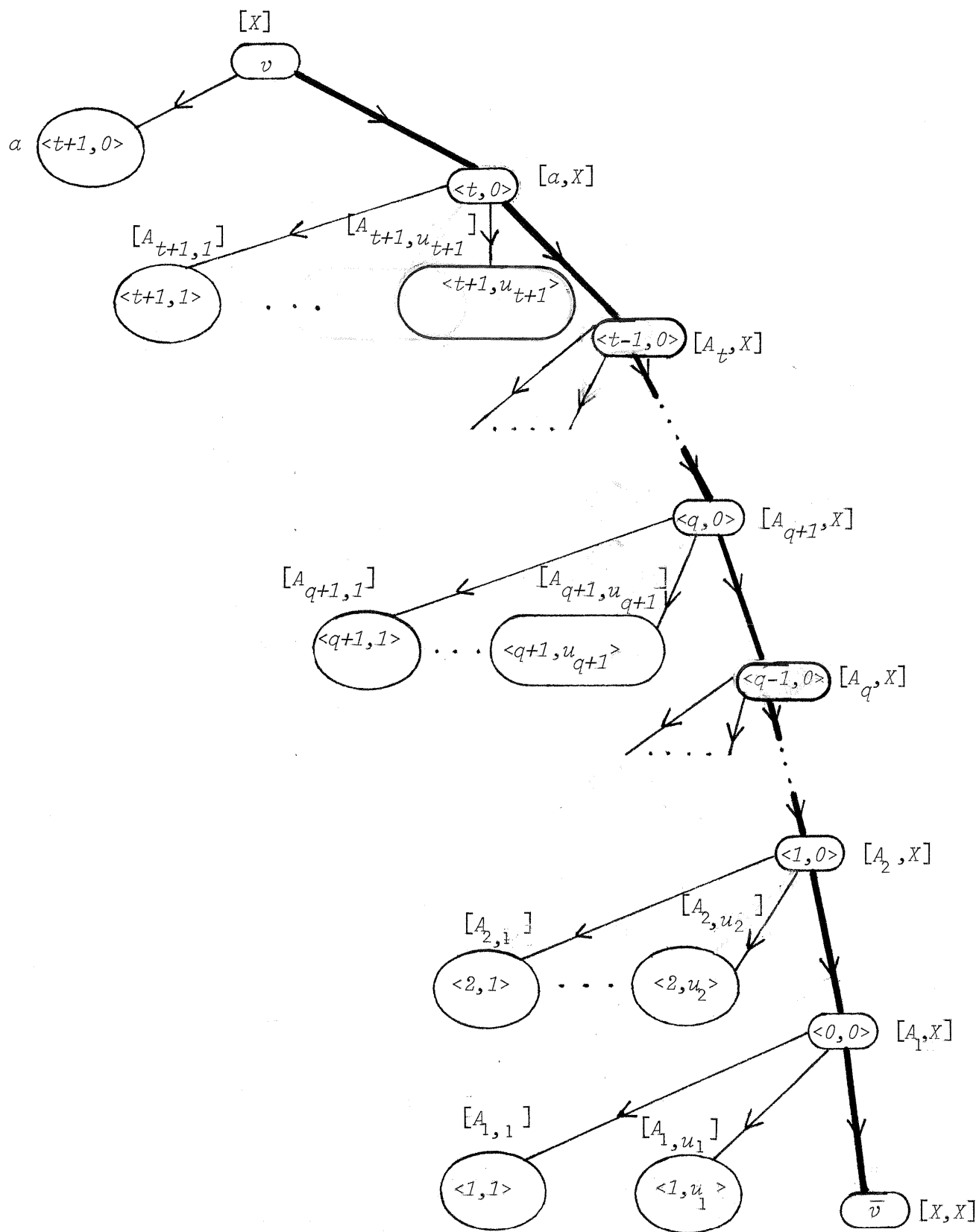
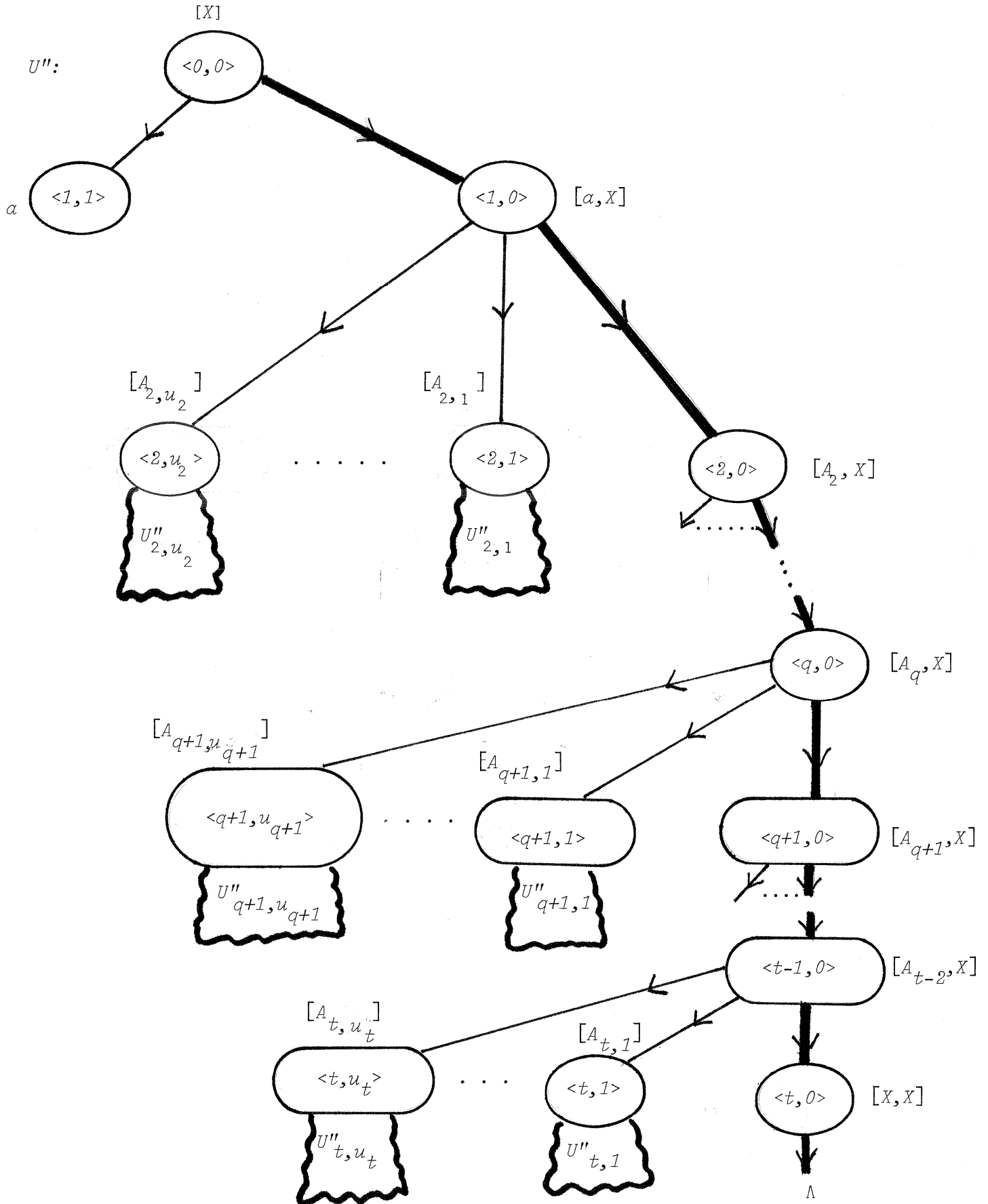


Figure 4.2

25 0x

Figure 4.3



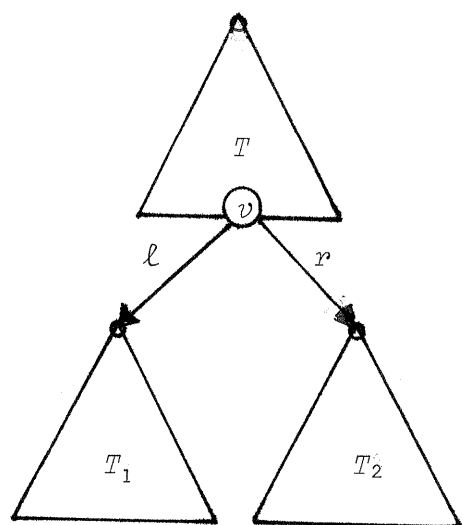


Figure 6.1

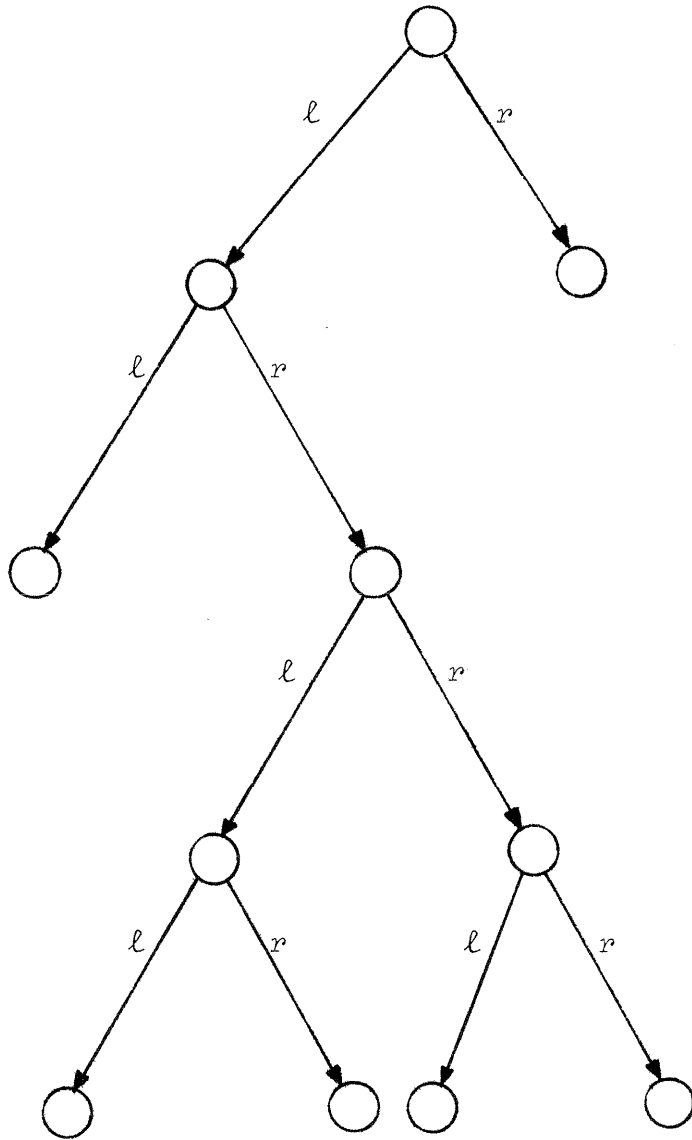


Figure 6.2

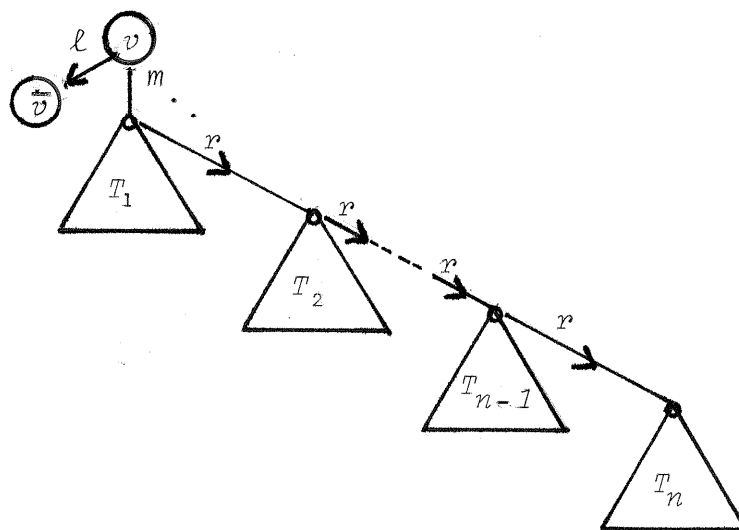


Figure 6.3

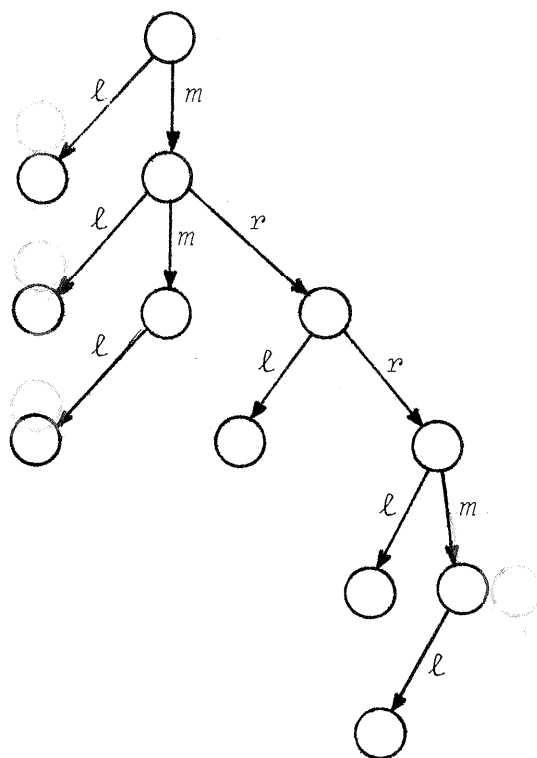
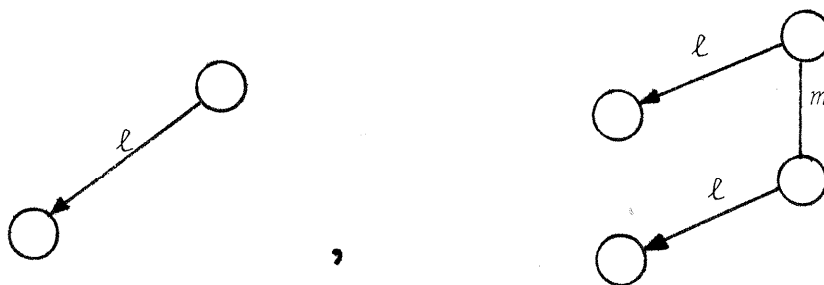


Figure 6.4

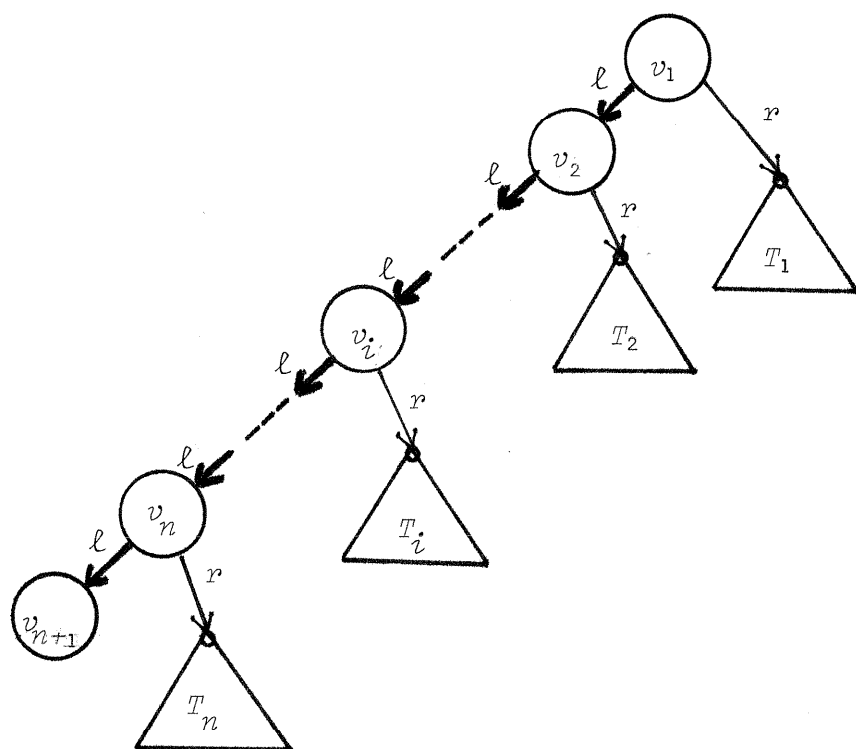


Figure 6.5

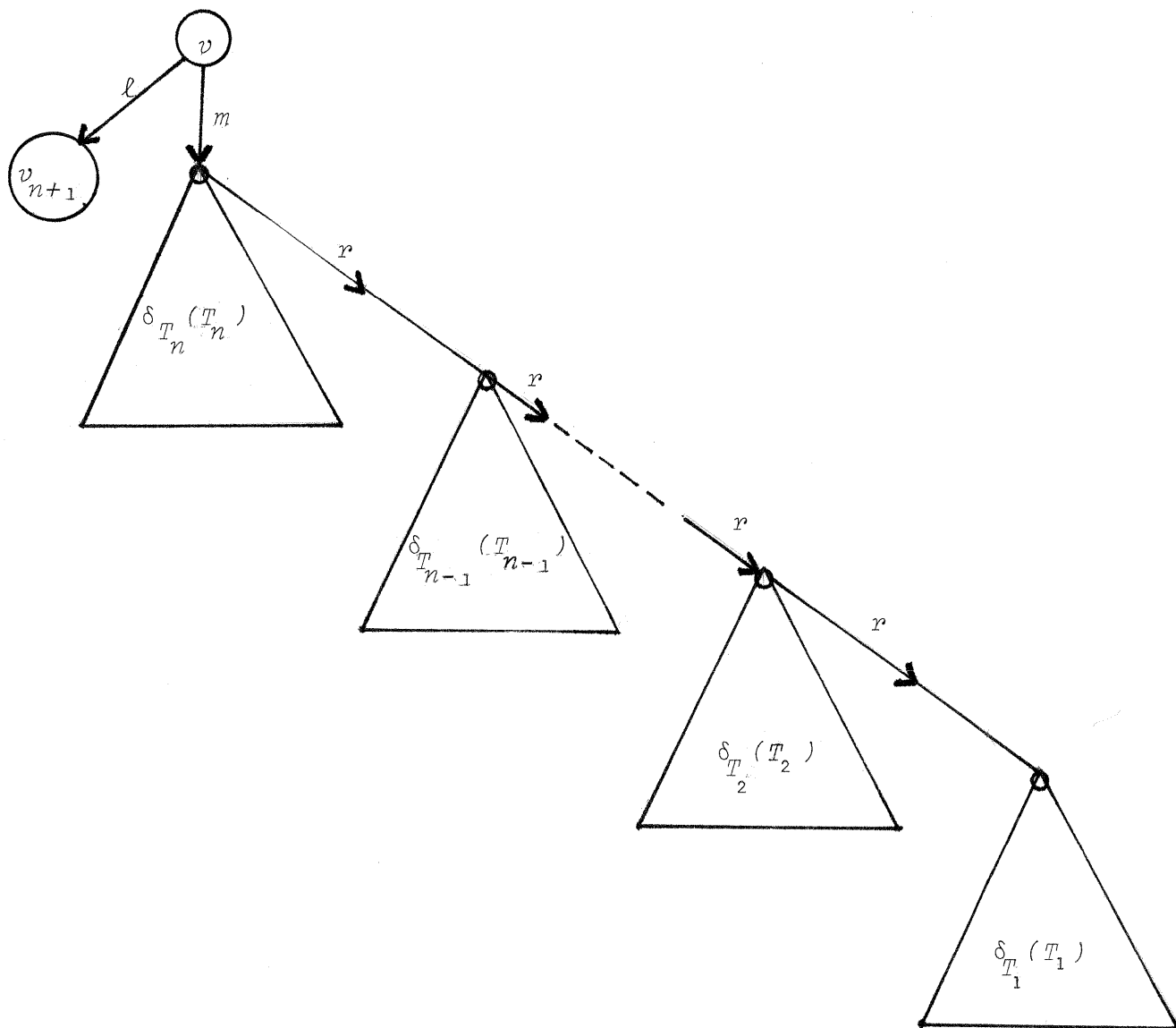


Figure 6.6

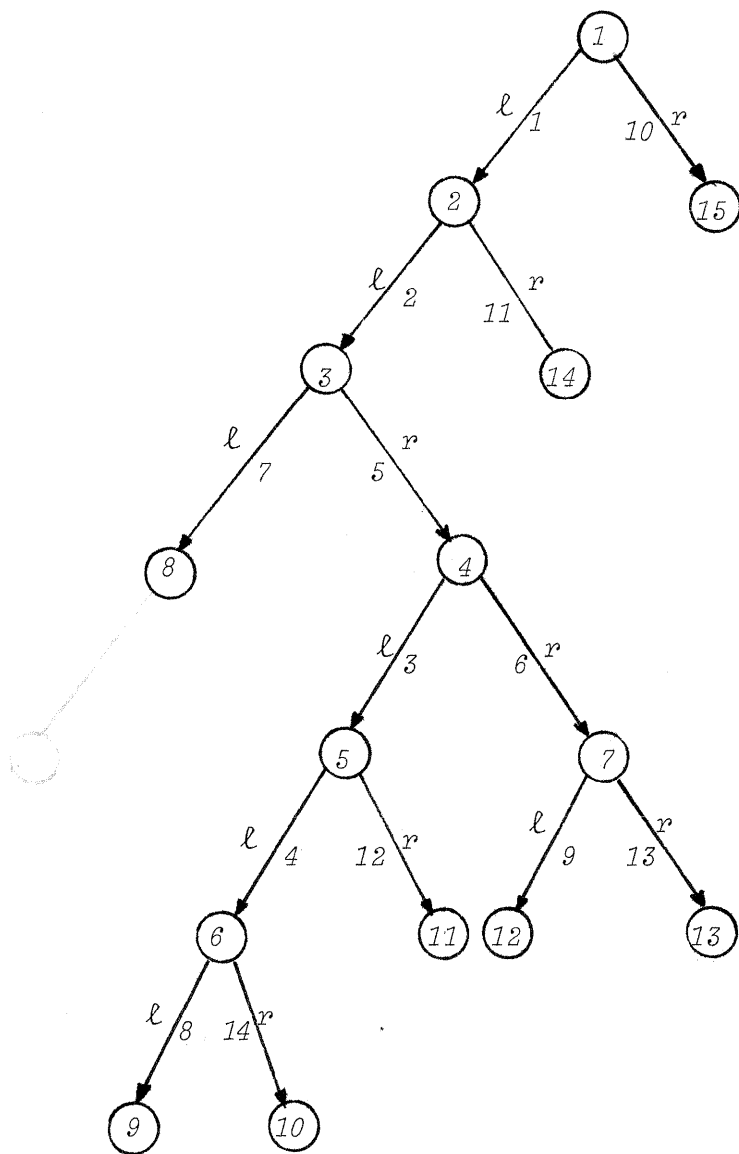


Figure 6.7

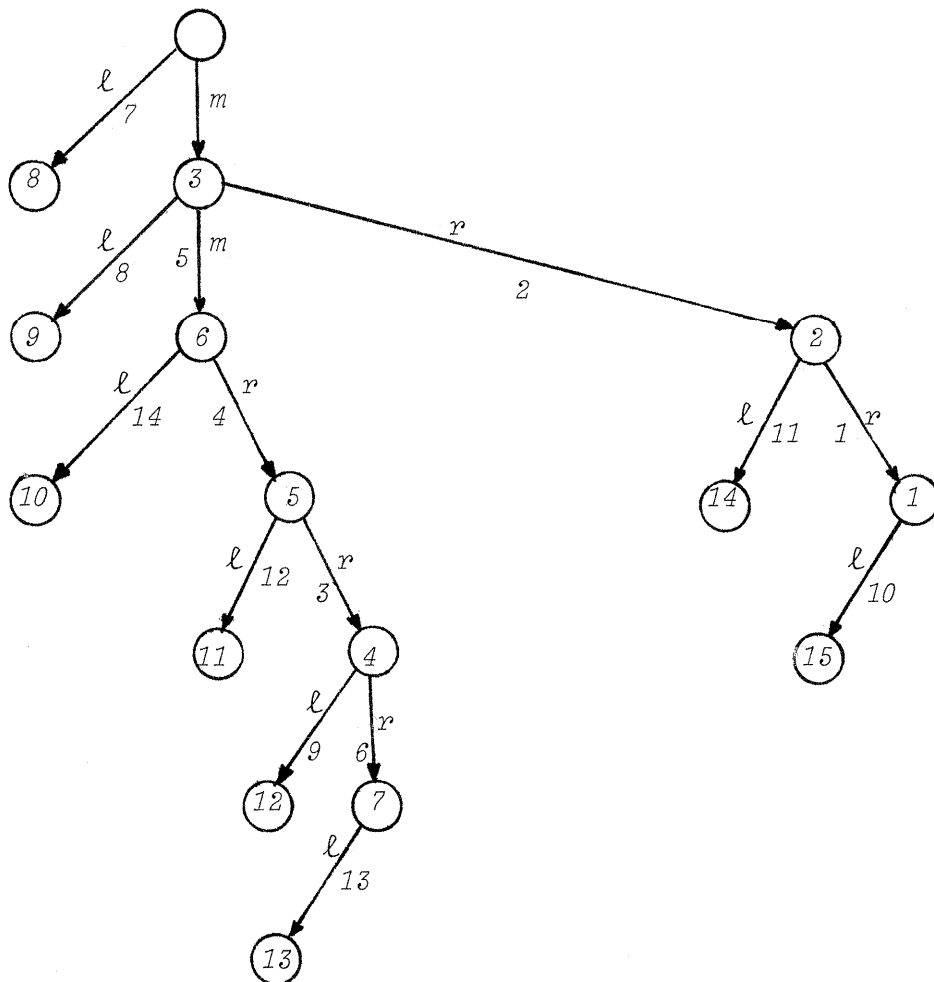


Figure 6.8

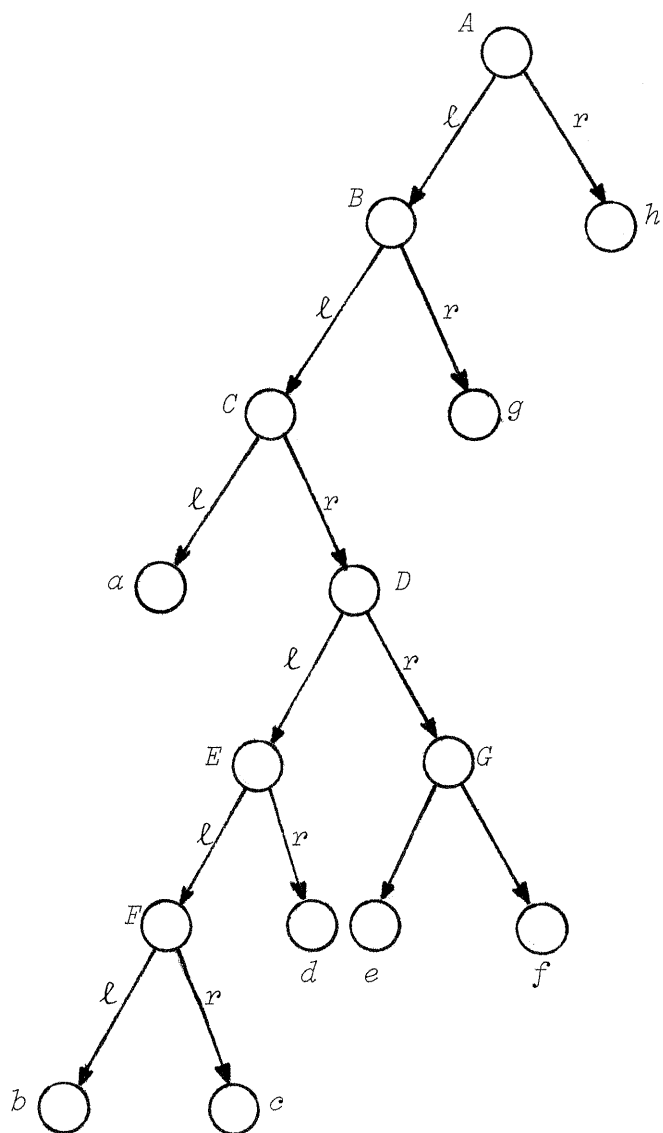


Figure 6.9

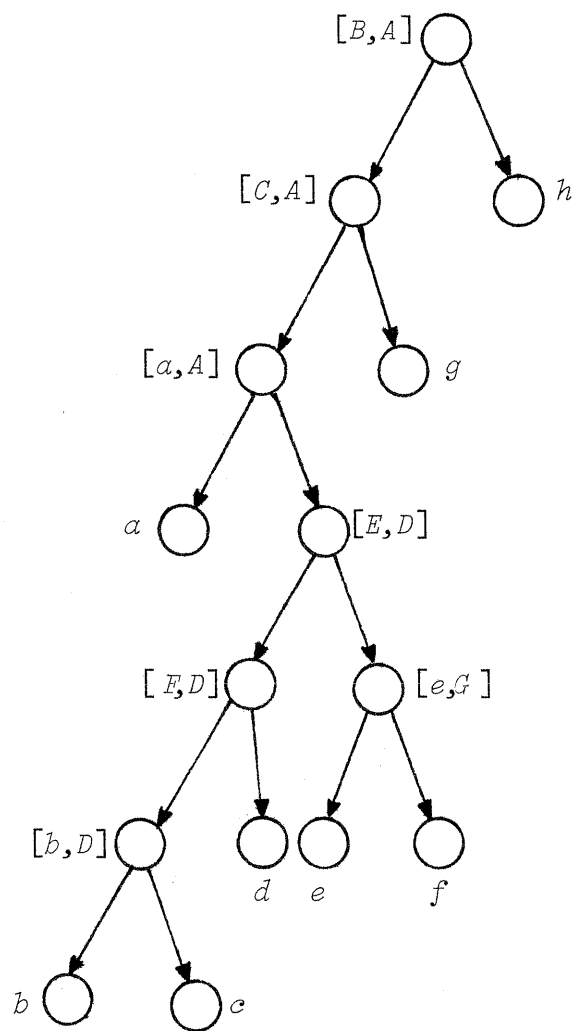


Figure 6.10

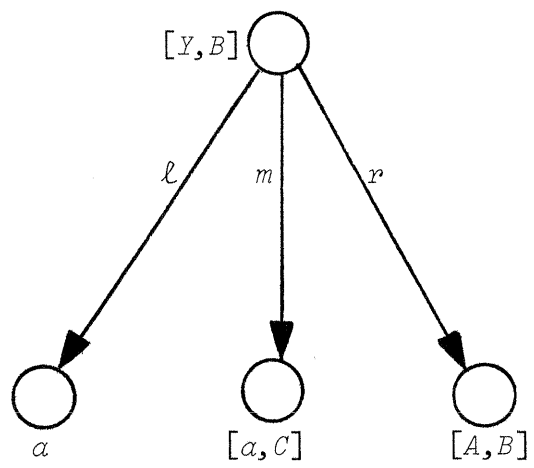


Figure 7.1

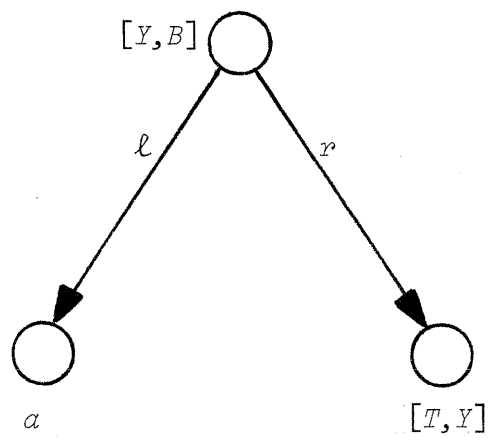


Figure 7.2

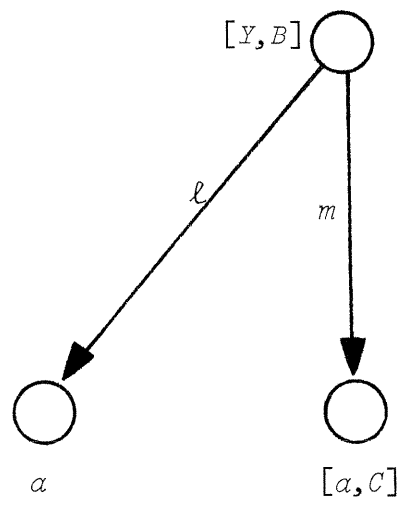


Figure 7.3

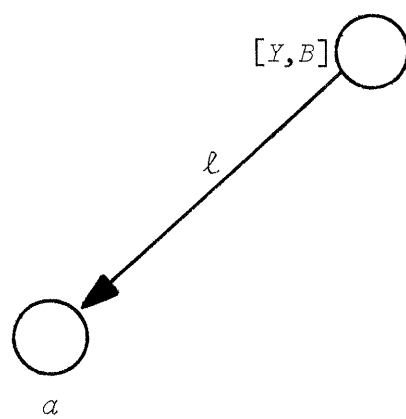


Figure 7.4

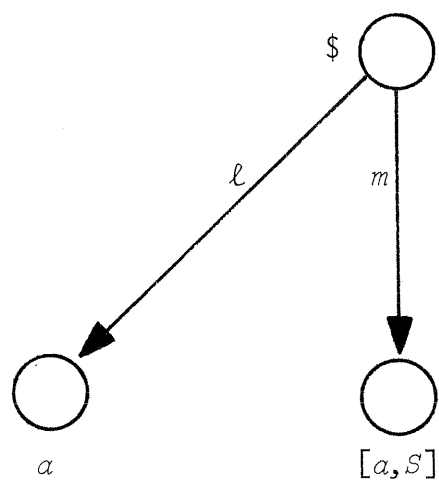


Figure 7.5

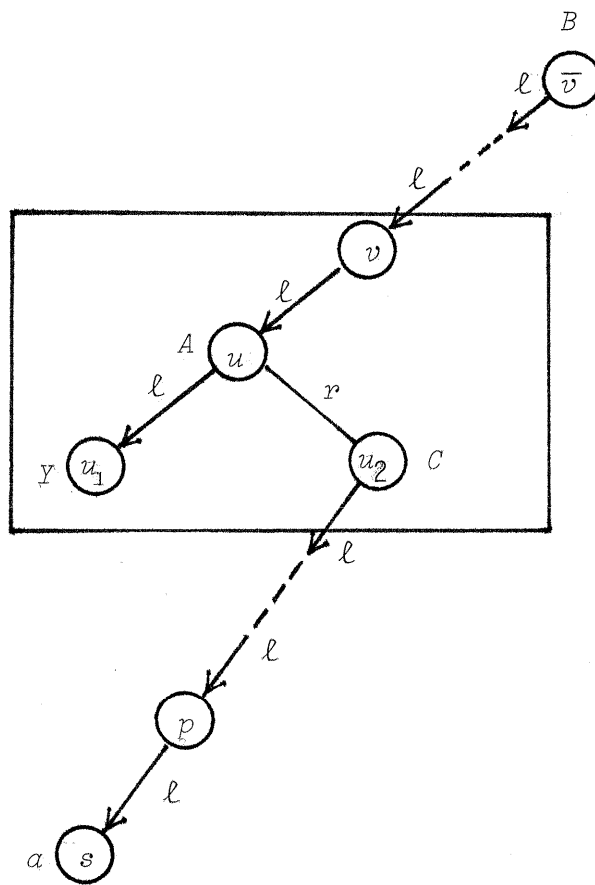


Figure 7.6

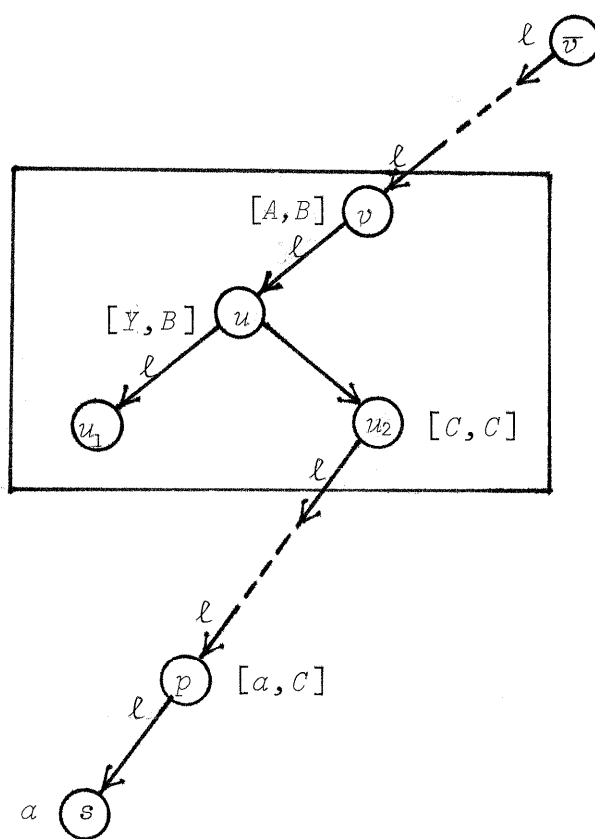


Figure 7.7

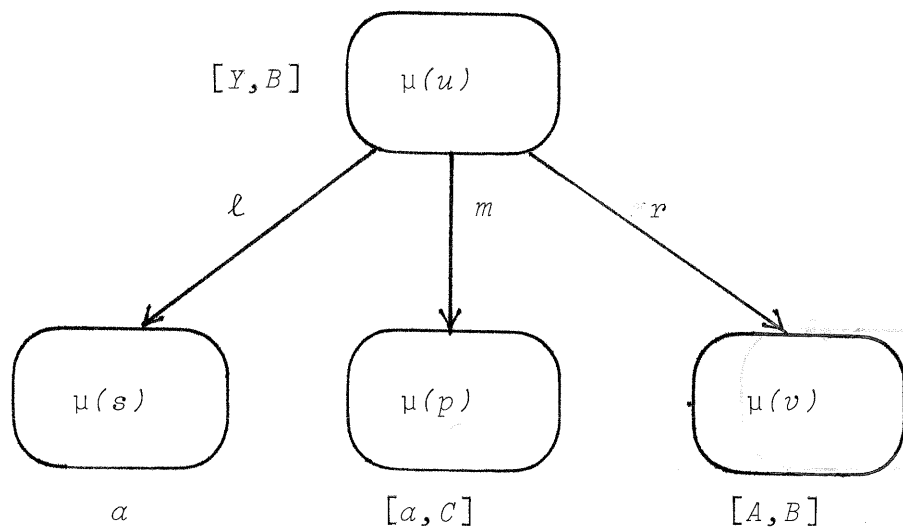


Figure 7.8

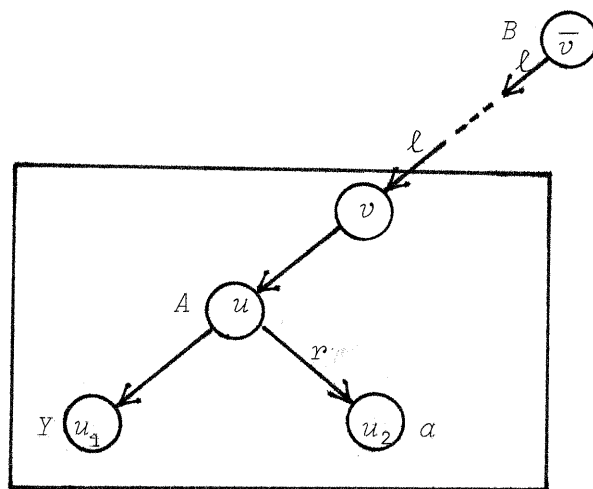


Figure 7.9

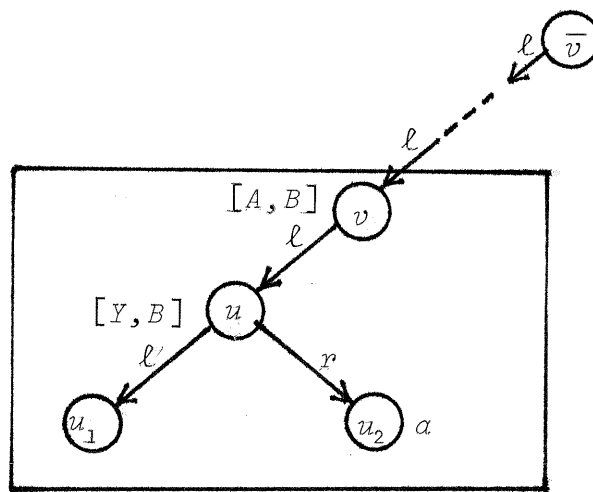


Figure 7.10

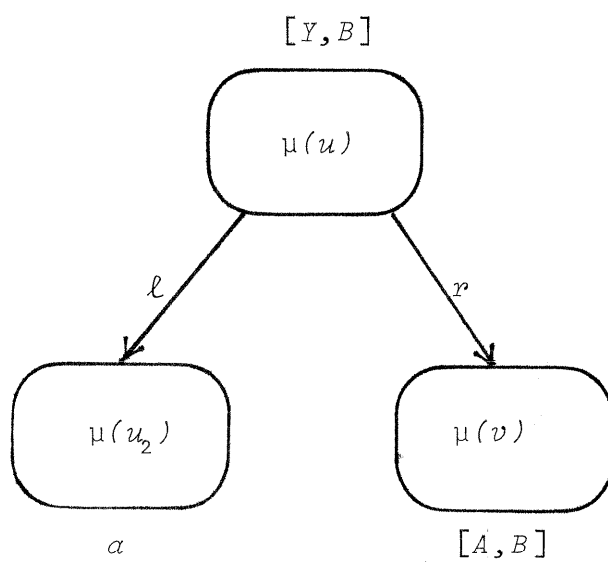


Figure 7.11

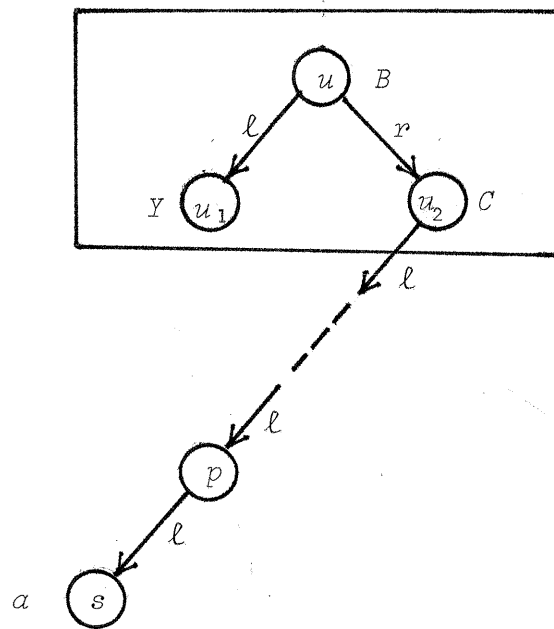


Figure 7.12

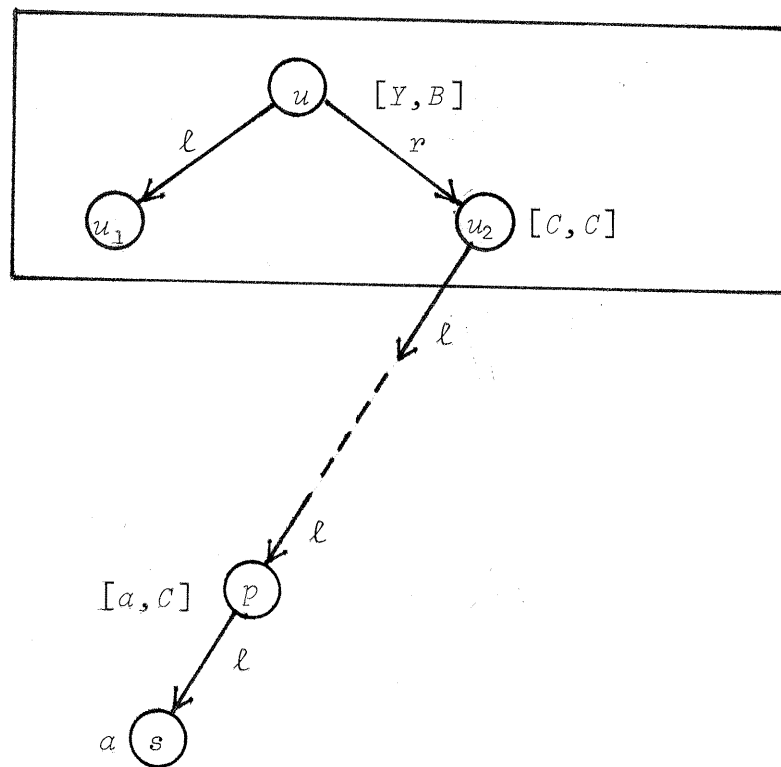


Figure 7.13

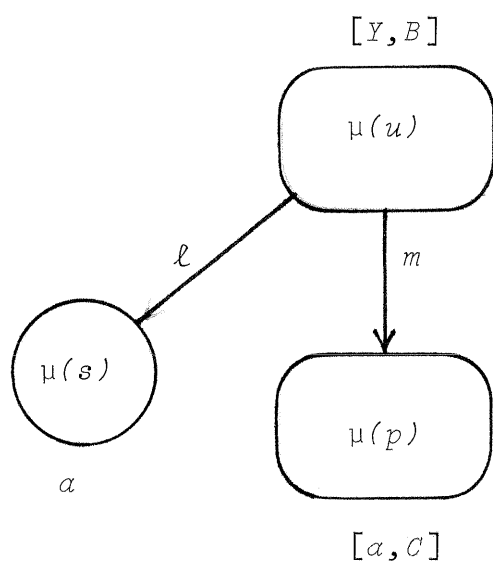


Figure 7.14

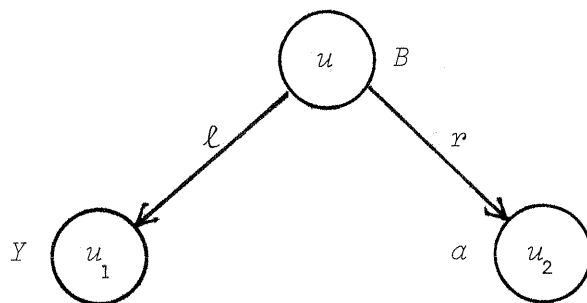


Figure 7.15

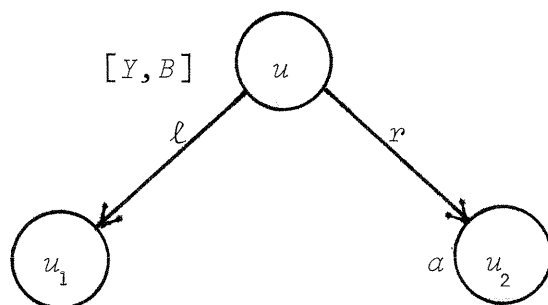


Figure 7.16

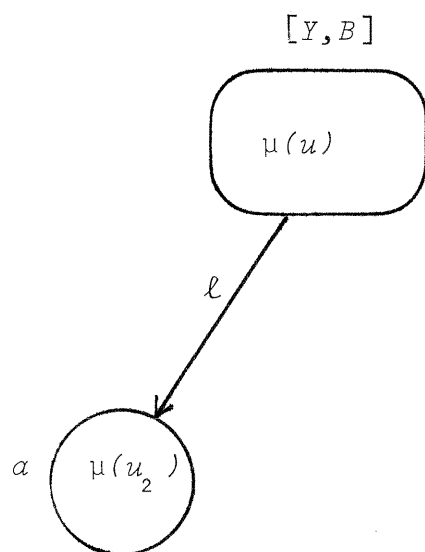


Figure 7.17

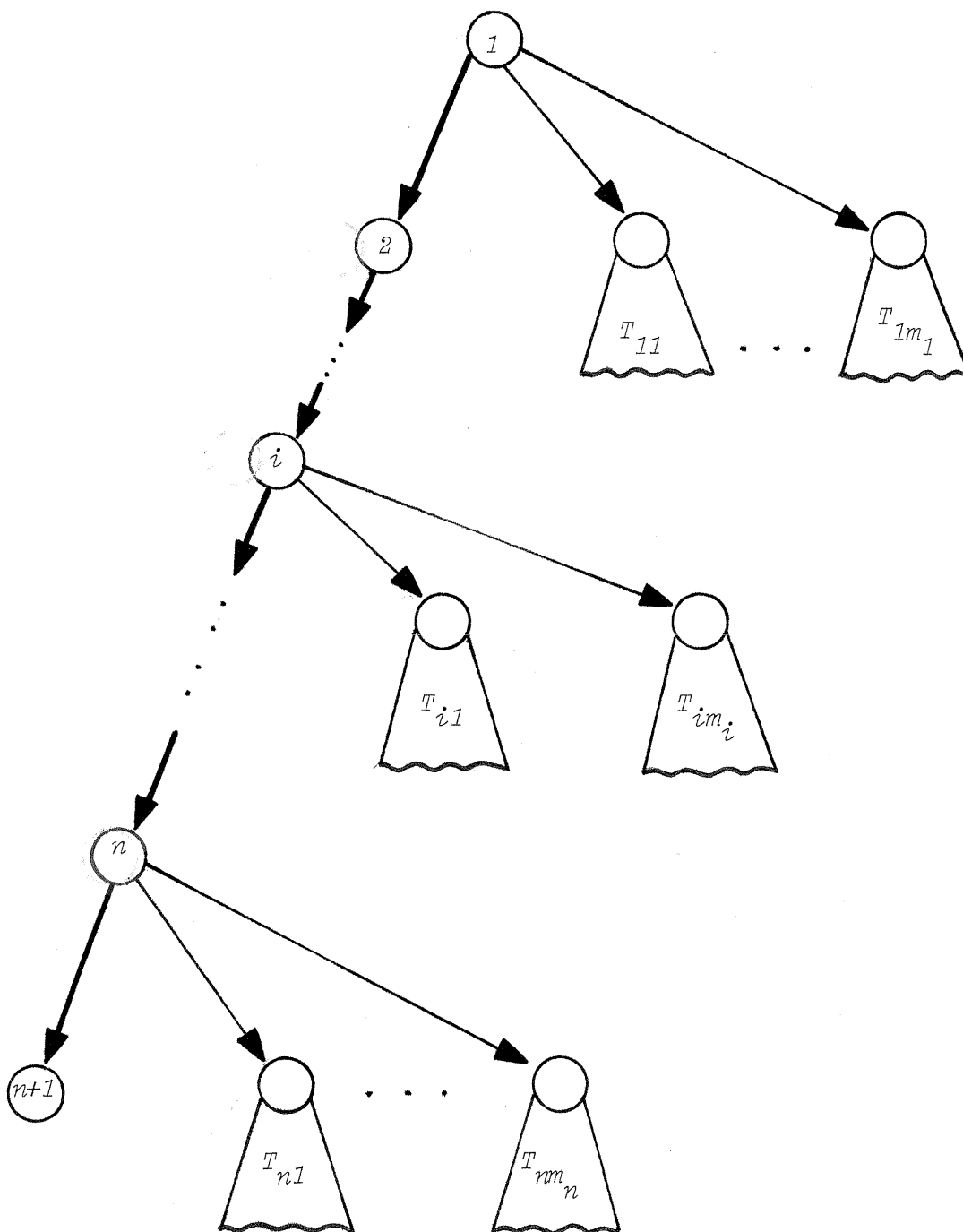
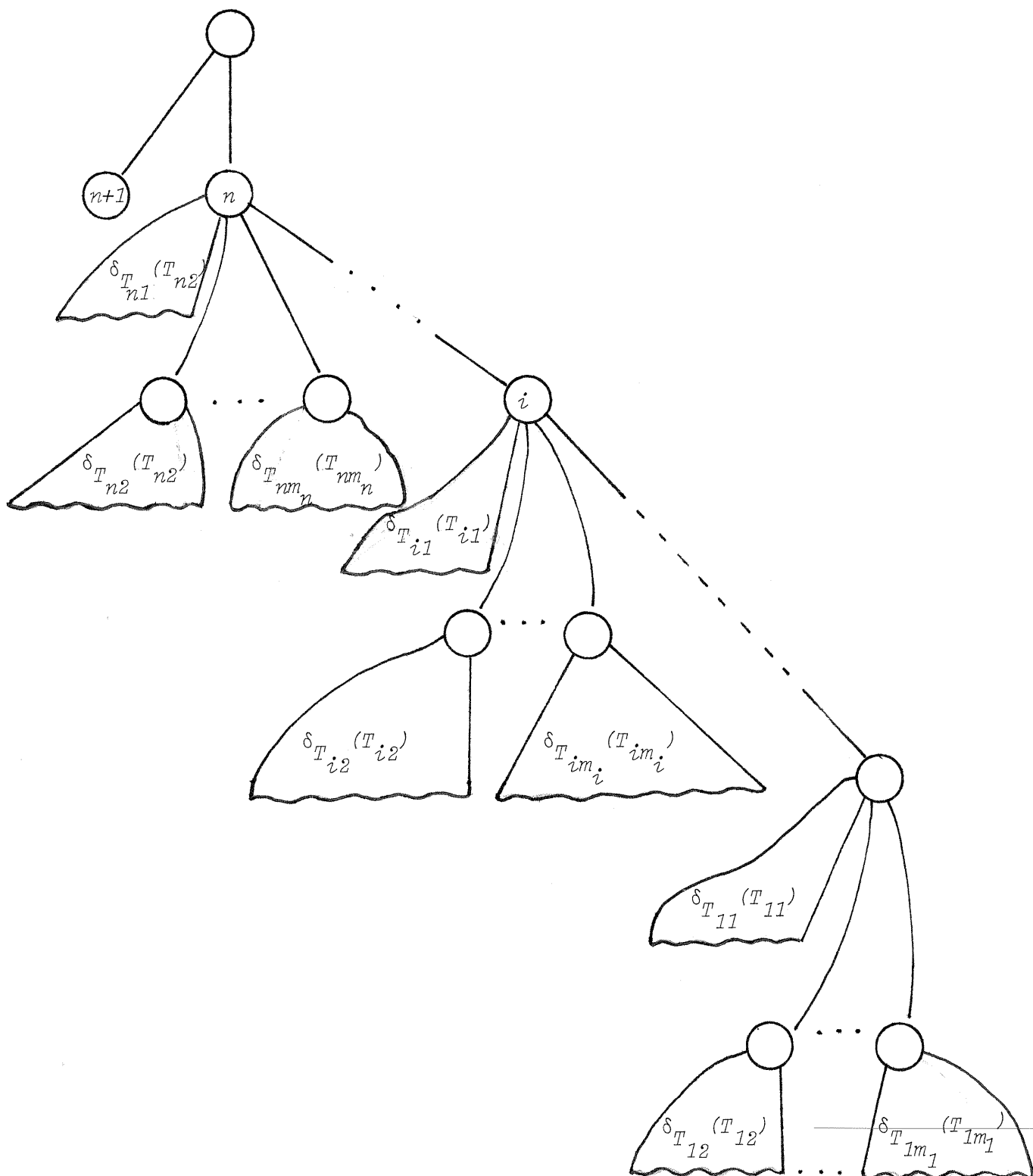


Figure 7.18

Figure 7.19



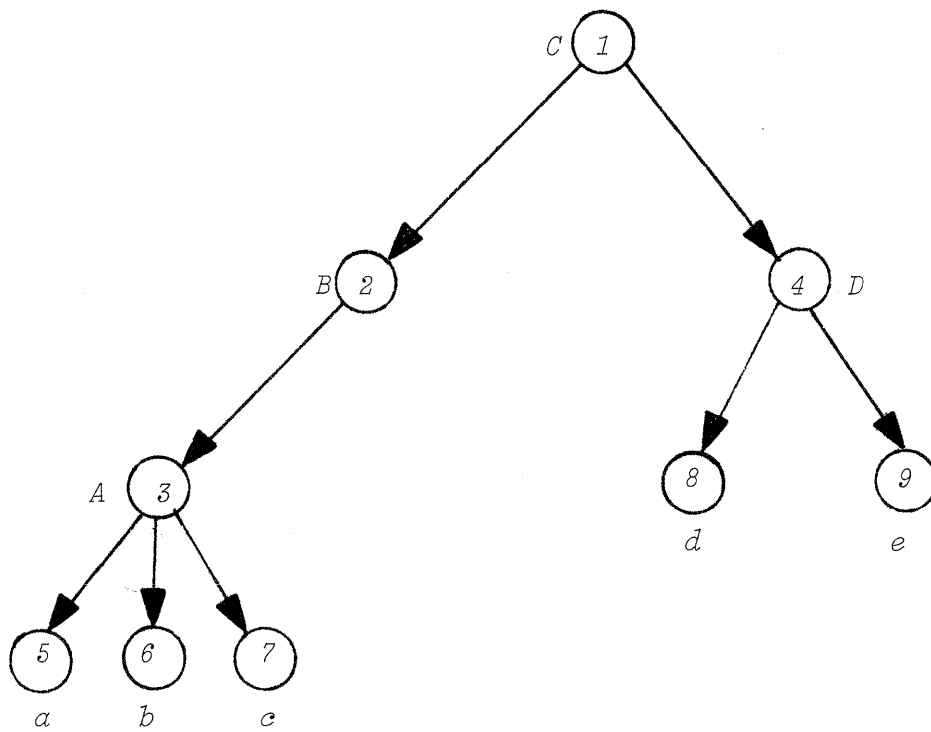


Figure 7.20

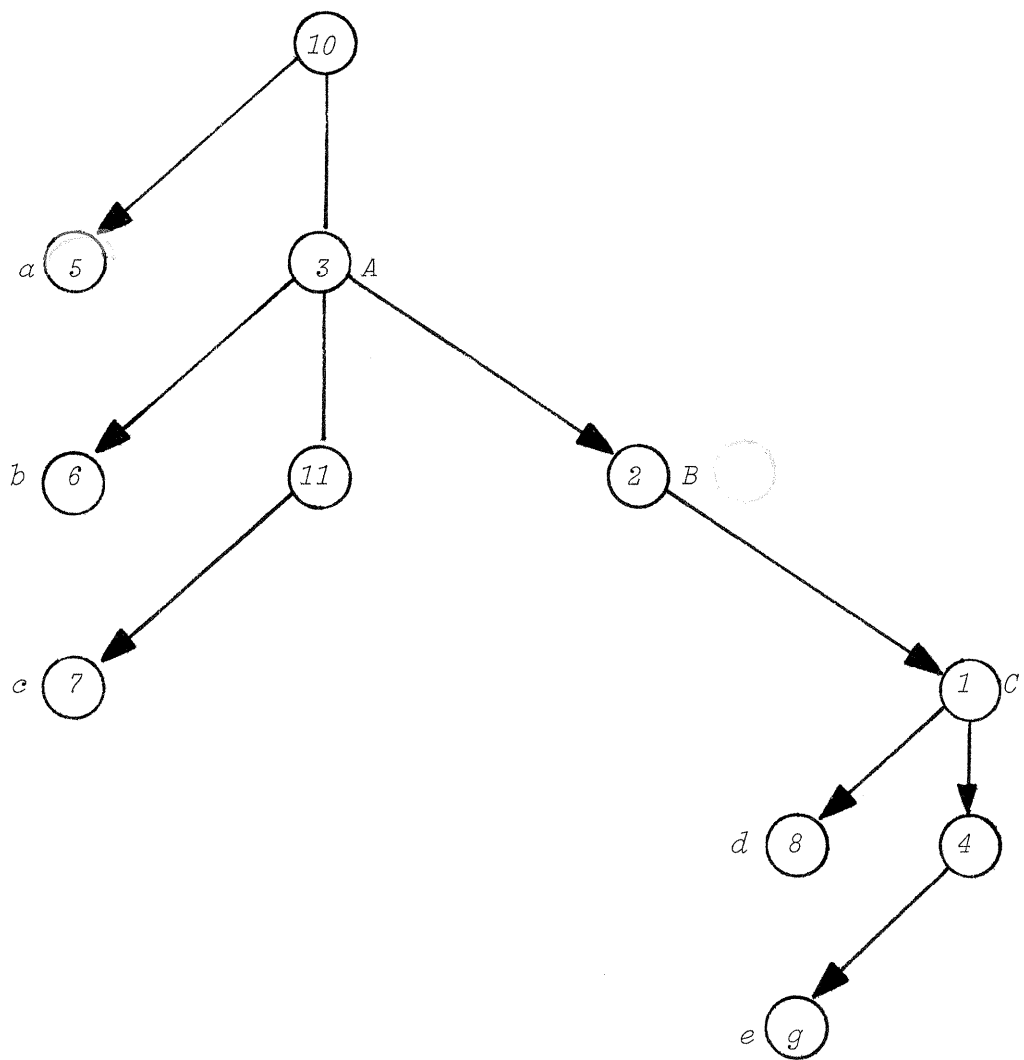


Figure 7.21

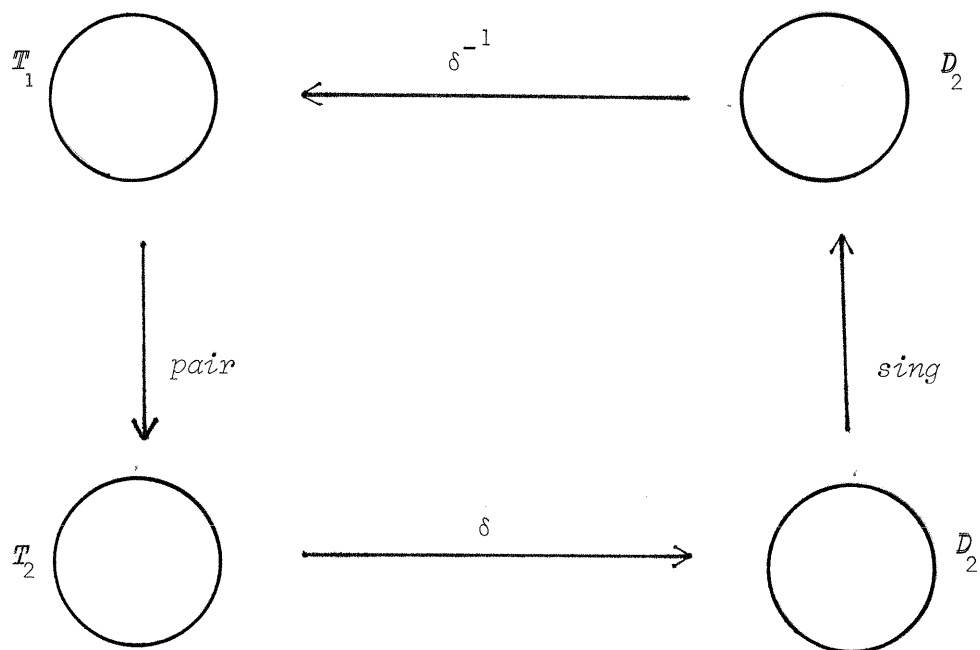


Figure 8.1