On the Complexity of Grammar and Language Problems

By

Amiram Yehudai

B.Sc. (Technion, Israel Institute of Technology, Haifa) 1973
M.S. (University of California) 1974

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Engineering

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA, BERKELEY

Approved:

Michael A Harrison
Susan L Graham
Stuart E Dreyfus

Committee in Charge

ON THE COMPLEXITY OF GRAMMAR AND LANGUAGE PROBLEMS

Amiram Yehudai

Ph.D. Computer Science

Michael A. Harrison

M.A. Harrison
Chairman of Committee

Abstract

Complexity Theory has recently become an important area of research in Computer Science. In recent years, new efficient algorithms were developed to solve various problems. Other problems have been shown to be inherently difficult or altogether unsolvable.

In this thesis, we take a variety of problems relevant to Formal Language Theory and analyze their complexity. In some cases, the main question is whether or not a certain problem is solvable. When problems are known to be solvable, we try to find efficient algorithms for them or otherwise evaluate their computational complexity. We also discuss the complexity of deterministic context free languages as measured by the number of accepting configurations required by a deterministic pushdown automaton accepting the given language.

First we discuss grammatical transformations. Analyzing some well known algorithms, we discover that some of these algorithms are very inefficient (e.g. the algorithm used in the literature to eliminate null rules takes exponential time in the worst case). In some cases, however, we are able to devise new more efficient algorithms (e.g. a linear time null rule elimination algorithm).

Then the "equivalence problem" is considered. Is there an algorithm which decides whether or not two given languages are equal? In the general context free case, this problem is unsolvable. We suggest a general strategy for solving the equivalence problem if some conditions are satisfied. Then we use this strategy to produce a decision procedure for the equivalence of two deterministic context free languages, one of which is simple. The general strategy is based on grammars and is a generalization of an algorithm to decide the equivalence of two simple languages due to Korenjak and Hopcroft.

Finally we deal with the complexity of deterministic context free languages. We propose as a new measure of complexity, the number of accepting configurations required by a deterministic pushdown automaton accepting the given language. Using this measure we obtain an infinite hierarchy of the deterministic context free languages which is related to some well known language families. As a consequence we get "relative closure" results for some subfamilies of the deterministic languages.

Acknowledgments

I would like to express my gratitude to my thesis advisor, Professor Michael A. Harrison. He provided stimulation, collaborated on much of the material and suggested many improvements in the manuscript. I also wish to thank the other members of the thesis committee, Professors Susan L. Graham and Stuart E. Dreyfus. Special gratitude is due my fellow graduate students in the Computer Science Division, for the encouragement and helpful comments they provided. I want to thank Ruth Suzuki for an excellent typing job.

Last but not least I want to thank my wife Jaffa for her love and understanding and for the material support she provided.

The financial support of the National Science Foundation under Grant MCS74-07636-A01 is gratefully appreciated.

ON THE COMPLEXITY OF GRAMMAR AND LANGUAGE PROBLEMS

Amiram Yehudai

Table of Contents

CHAPTER 1

INTRODUCTION AND PRELIMINARIES

Section 1.1 - Introduction and Summary

Complexity Theory has recently become one of the most important areas of research in Computer Science. In recent years new efficient algorithms were developed to solve various problems. Other problems were shown to be inherently difficult or altogether unsolvable.

In this thesis we take a variety of problems relevant to Formal Language Theory and analyze their complexity. In some cases the main question is whether or not a certain problem is solvable. When problems are known to be solvable we try to find efficient algorithms for them or otherwise evaluate their computational complexity. We also discuss the complexity of deterministic context free languages as measured by the number of accepting configurations required by a deterministic pushdown automaton accepting the given language.

The thesis consists of four chapters. The present Chapter 1 is an introduction and review of some preliminary concepts.

In Chapter 2 we consider grammatical transformations. These are algorithms that take one context free grammar and produce another grammar generating the same language subject to certain additional conditions or constraints. We discuss various such transformations, analyzing some well known algorithms. We discover that some of these algorithms are very inefficient (e.g. the algorithm used in the literature to eliminate Λ-rules takes exponential time in the worst case). In some cases, however, we are able to devise new more efficient algorithms (e.g. a linear time Λ-rule eliminating algorithm).

Chapter 3 is devoted to the "equivalence problem": Is there an algorithm which decides whether two given languages are equal? In the most general case this problem is unsolvable. We suggest a general strategy for solving the equivalence problem if some conditions are met. Then we use this strategy to produce a decision procedure for the equivalence of two deterministic context free languages, one of which is simple. The general strategy is based on grammars and is a generalization of an algorithm to decide the equivalence of two simple languages (cf. Korenjak and Hopcroft [1966]).

Chapter 4 deals with the complexity of deterministic context free languages. We propose as a new measure of complexity the number of accepting configurations required by a deterministic pushdown automaton accepting the given language. Using this measure we obtain an infinite hierarchy of the deterministic context free languages which is related to the language families Δ_0 and Δ_1 as discussed by Harrison and Havel [1973], and LR(0), as described in Geller and Harrison [1977]. As a consequence we get a "relative closure" result for the Δ_1 and LR(0) families as follows. If L_1, L_2 are languages in Δ_1(LR(0)) then if $L_1 \cap L_2$ is a deterministic context free language it must also be in Δ_1(LR(0)).

Section 1.2 - Preliminaries

Let X be a set. A partition of X is a collection $\pi = \{X_1, X_2, \ldots\}$ of nonempty subsets $X_i \subseteq X$ such that $X = \bigcup_i X_i$ and the subsets are mutually disjoint. Subsets X_i are called blocks of partition π. When X and Y are sets then any set $\rho \subseteq X \times Y$ is a relation (between X and Y). Let $\rho \subseteq X \times Y$ and $\sigma \subseteq Y \times Z$. We define

$$\rho\sigma = \{(x,z) \in X \times Z \mid x\rho y\sigma z \text{ for some } y \in Y\} ,$$

and, if $X = Y$,

$$\rho^0 = \{(x,x) \mid x \in X\} \quad \text{(the diagonal)} ,$$
$$\rho^{n+1} = \rho^n\rho , \quad n \geq 0 ,$$
$$\rho^* = \bigcup_{n \geq 0} \rho^n \quad \text{(the reflexive and transitive closure of } \rho) ,$$
$$\rho^+ = \rho^*\rho \quad \text{(the transitive closure of } \rho) .$$

Let X and Y be sets of words. Let $XY = \{xy \mid x \in X, y \in Y\}$ where xy is the concatenation of x and y. Define $X^0 = \{\Lambda\}$ where Λ is the null word. For each $i \geq 0$, define $X^{i+1} = X^iX$ and $X^* = \bigcup_{i \geq 0} X^i$. Let $X^+ = X^*X$ and $\emptyset$ denote the empty set. Let x be a word over some alphabet Σ. Then there exists some $n \geq 0$ such that $x = a_1 \cdots a_n$, with $a_i \in \Sigma$ for $1 \leq i \leq n$. Then we define $|x| = n$. Note that if X is a set then $|X|$ designates the cardinality of the set. No confusion should arise from using the same notation for these two concepts, however.

We need the usual concepts of grammars and languages.

Definition 1.2.1. A context-free grammar (hereafter a grammar) G is a 4-tuple

$$G = (V,\Sigma,P,S)$$

where V and Σ are two alphabets, $\Sigma \subseteq V$ (letters in Σ and in $N = V - \Sigma$ are called terminals and nonterminals respectively), $S \in N$ and P is a finite relation, $P \subseteq N \times V^*$ (the set of productions).

As usual, we write $A \to \alpha$ is in P instead of $(A,\alpha) \in P$.

Certain conventions are adopted in usage of symbols. Capital letters near the beginning of the alphabet are used for elements of V or N. Lower case elements like a, b, c for elements of Σ or $\Sigma_\Lambda = \Sigma \cup \{\Lambda\}$. One uses $\alpha,\beta,\gamma,\ldots$ for elements of V^* and $u,v,w,\ldots$ for elements of Σ^*. We use V_Λ to denote the set $V \cup \{\Lambda\}$.

Definition 1.2.2. Let $G = (V,\Sigma,P,S)$ be a grammar. We define a relation $\Rightarrow \subseteq V^* \times V^*$ as follows. For any α, $\beta \in V^*$, $\alpha \Rightarrow \beta$ (read α directly derives β) if and only if $\alpha = \alpha_1 A \alpha_2$, $\beta = \alpha_1 \gamma \alpha_2$ and $A \to \gamma$ is in P for some $A \in N$ and α_1, α_2, $\gamma \in V^*$. In particular, if $\alpha_1 \in \Sigma^*$ or $\alpha_2 \in \Sigma^*$ we sometimes write $\alpha \underset{L}{\Rightarrow} \beta$ or $\alpha \underset{R}{\Rightarrow} \beta$ respectively (α directly derives β leftmost or rightmost). $\overset{*}{\Rightarrow}$, $\overset{*}{\underset{L}{\Rightarrow}}$ and $\overset{*}{\underset{R}{\Rightarrow}}$ are the reflexive transitive closure of $\Rightarrow$, $\underset{L}{\Rightarrow}$ and $\underset{R}{\Rightarrow}$ respectively. $\alpha \overset{*}{\Rightarrow} \beta$ is read α derives β. When the grammar G involved must be specified we write $\underset{G}{\Rightarrow}$, $\overset{*}{\underset{G}{\Rightarrow}}$, etc.

The language generated by G is the language

$$L(G) = \{w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w\} .$$

Two grammars are called equivalent if and only if they generate the same language.

The concept of a derivation is needed. Let $G = (V,\Sigma,P,S)$ be a grammar and suppose, for some $n \geq 0$,

$$\alpha_0 \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \cdots \Rightarrow \alpha_n .$$

Then this sequence is called a <u>derivation</u> of α_n from α_0, or simply a derivation of α_n if $\alpha_0 = S$. If for each i, $0 \leq i < n$, $\alpha_i \underset{L}{\Rightarrow} \alpha_{i+1}$ ($\alpha_i \underset{R}{\Rightarrow} \alpha_{i+1}$) then the derivation is leftmost (respectively rightmost). We may write $\alpha_0 \overset{n}{\Rightarrow} \alpha_n$ to indicate the length of the derivation (i.e. number of steps).

A grammar G is said to be <u>unambiguous</u> if each $x \in L(G)$ has exactly one leftmost derivation.

We say that a language $L \subseteq \Sigma^*$ is <u>prefix-free</u> if $u \in L$ and $uv \in L$ implies $v = \Lambda$.

We will also need to use the theory of strict deterministic grammars and languages, cf. Harrison and Havel [1973,1974].

<u>Definition 1.2.3</u>. Let $G = (V,\Sigma,P,S)$ be a context-free grammar and let π be a partition of the set V of terminal and nonterminal letters of G. Such a partition π is called <u>strict</u> if and only if

(i) $\Sigma \in \pi$

and (ii) For any $A, A' \in N$ and $\alpha, \beta, \beta' \in V^*$ if $A \to \alpha\beta$, $A' \to \alpha\beta'$ are in P and $A \equiv A' \pmod{\pi}$ then

either (a) both $\beta, \beta' \neq \Lambda$ and ${}^{(1)}\beta \equiv {}^{(1)}\beta' \pmod{\pi}$

or (b) $\beta = \beta' = \Lambda$ and $A = A'$.

<u>Definition 1.2.4</u>. A context-free grammar $G = (V,\Sigma,P,S)$ is called <u>strict</u> <u>deterministic</u> if and only if there exists a strict

partition π of V. A language L is called a strict deterministic language if and only if $L = L(G)$ for some strict deterministic grammar G.

Now let us introduce "simple" grammars, cf. Korenjak and Hopcroft [1966].

Definition 1.2.5. A context-free grammar $G = (V,\Sigma,P,S)$ is said to be a simple grammar if for all $A \in N$, $a \in \Sigma$, and $\alpha, \beta \in V^*$,

$$A \to a\alpha \text{ and } A \to a\beta \text{ in } P$$

imply $\alpha = \beta$. A simple language is a language generated by a simple grammar.

It will also be necessary to have the terminology to deal with deterministic pushdown automata, cf. Aho and Ullman [1972] and Harrison and Havel [1973].

Definition 1.2.6. A deterministic pushdown automata (abbreviated DPDA) is a 7-tuple

$$M = (Q,\Sigma,\Gamma,\delta,q_0,Z_0,F)$$

where Q is a finite nonempty set, Σ and Γ are two alphabets, $q_0 \in Q$, $Z_0 \in \Gamma$, $F \subseteq Q$ and δ is a partial function

$$\delta\colon Q \times \Sigma_\Lambda \times \Gamma \xrightarrow[p]{} Q \times \Gamma^*$$

with the property that for any $q \in Q$ and $Z \in \Gamma$, $\delta(q,\Lambda,Z) \neq \emptyset$ implies $\delta(q,a,Z) = \emptyset$ for all $a \in \Sigma$. If for all $q \in Q$ and $Z \in \Gamma$, $\delta(q,\Lambda,Z) = \emptyset$ then this is said to be a real time DPDA.

Next we must describe how a DPDA moves.

Definition 1.2.7. Let $M = (Q,\Sigma,\Gamma,\delta,q_0,Z_0,F)$ be a DPDA and let $\mathcal{Q} = Q \times \Gamma^*$. An element of $\mathcal{Q}$ is called a configuration.

For each $q, q' \in Q$, $a \in \Sigma_\Lambda$, $\alpha, \beta \in \Gamma^*$ and $Z \in \Gamma$ we write $(q,\alpha Z) \xrightarrow[M]{a} (q',\alpha\beta)$ if and only if $\delta(q,a,Z) = (q',\beta)$; read "M moves from configuration $(q,\alpha Z)$ to configuration $(q',\alpha\beta)$ while reading a". M may be omitted whenever understood. We extend this by writing

(i) for all $c \in \mathcal{Q}$, $c \xrightarrow{\Lambda} c$

and (ii) if $c \xrightarrow{x_1} c_1$ and $c_1 \xrightarrow{x_2} c_2$ then $c \xrightarrow{x_1x_2} c_2$.

Sometimes we use the word derivation when referring to a move like $c \xrightarrow{x} c'$.

(q_0,Z_0) is the initial configuration. A configuration (q,γ) is said to be reachable if there exists some $x \in \Sigma^*$ such that $(q_0,Z_0) \xrightarrow{x} (q,\gamma)$.

We now endow a DPDA with an ability to define, or accept, certain languages over its input alphabet.

Definition 1.2.8. Let $M = (Q,\Sigma,\Gamma,\delta,q_0,Z_0,F)$. For a given $K \subseteq \Gamma^*$ define the language $T(M,K) \subseteq \Sigma^*$ as follows:

$$T(M,K) = \{w \in \Sigma^* \mid (q_0,Z_0) \xrightarrow{w} (q,\alpha) \text{ for some } q \in F \text{ and } \alpha \in K\} .$$

A configuration of the form (q,α) where $q \in F$ and $\alpha \in K$ is called an accepting configuration.

Two configurations c and c' are said to be equivalent if

$$\{w \mid c \xrightarrow{w} (q,\alpha) \text{ for some } q \in F \text{ and } \alpha \in K\}$$
$$= \{w \mid c' \xrightarrow{w} (q,\alpha) \text{ for some } q \in F \text{ and } \alpha \in K\} .$$

In particular let

$$T_0(M) = T(M,\Gamma^*) \ ,$$
$$T_1(M) = T(M,\Gamma) \ ,$$
$$T_2(M) = T(M,\Lambda) \ .$$

$\Delta_i = \{T_i(M) | M \text{ is a DPDA}\}$. By Harrison and Havel [1973] Δ_2 is the family of strict deterministic languages, while Δ_0 is the collection of deterministic languages, cf. Aho and Ullman [1973] and Harrison and Havel [1973]. Δ_1 has only been briefly studied in Harrison and Havel [1973]. Δ_2 is a particularly important family because, among other reasons, each $L \in \Delta_0$ can be mapped into Δ_2 by "endmarking", i.e. $L \mapsto L\$$.

We need the basic definitions from finite automata theory.

Definition 1.2.9. A finite automaton is a 5-tuple $A = (Q,\Sigma,\delta,q_0,F)$ where

(i) Q is a finite non-empty set of states,

(ii) Σ is a finite non-empty set of inputs,

(iii) δ is a function from $Q \times \Sigma$ into Q called the direct transition function,

(iv) $q_0 \in Q$ is the initial state,

(v) $F \subseteq Q$ is the set of final states.

Next, it is necessary to extend δ to allow A to accept sequences of inputs.

Definition 1.2.10. Let $A = (Q,\Sigma,\delta,q_0,F)$ be a finite automaton. For each $(q,a,x) \in Q \times \Sigma \times \Sigma^*$, define $\delta(q,\Lambda) = q$ and $\delta(q,ax) = \delta(\delta(q,a),x)$.

The notation $rp(y) = \delta(q_0,y)$, the "response" to y, is sometimes used and denotes the state A reaches after reading y. Finally, A accepts a string $x \in \Sigma^*$ if A, starting in q_0 at the left end of x, goes through some computation and reads all of x, and stops in some final state. The set of all accepted strings is denoted as $T(A)$. More compactly, $T(A) = \{x \in \Sigma^* | \delta(q_0,x) \in F\}$.

Definition 1.2.11. A set $L \subseteq \Sigma^*$ is called regular if there is some finite automaton A such that $L = T(A)$.

Next we present some concepts regarding relations.

Definition 1.2.12. Let R be a relation on X (i.e. $R \subseteq X \times X$). Then R is an equivalence relation if R is

(i) reflexive: for every $a \in X$, $(a,a) \in R$.

(ii) symmetric: for every $a, b \in X$, $(a,b) \in R$ implies that $(b,a) \in R$.

(iii) transitive: for every $a, b, c \in X$, if $(a,b) \in R$ and $(b,c) \in R$ then $(a,c) \in R$.

If R is a relation we will sometimes write aRb instead of $(a,b) \in R$.

Fact. If R is an equivalence relation on a set X, then there is a partition Π induced on X by R. The members of this partition are called equivalence classes. Conversely, if Π is a partition on X then Π induces an equivalence relation $R = X/\Pi$ on X. We will denote by $[x]_R$ the equivalence class (with respect to R) containing the element $x \in X$.

Definition 1.2.13. $rk(R)$ the rank of equivalence relation R is the number of equivalence classes induced by R.

Definition 1.2.14. Let E_1 and E_2 be equivalence relations on X. We say E_1 refines E_2 if $E_1 \subseteq E_2$ (i.e. xE_1y implies that xE_2y).

Fact. Let E_1 and E_2 be equivalence relations on X. If $E_1 \subseteq E_2$ then $rk(E_1) \geq rk(E_2)$.

Definition 1.2.15. Let R be an equivalence relation on Σ^*. R is a right congruence relation if xRy implies that for any word $z \in \Sigma^*$, $(xz)R(yz)$, i.e.

$$(x,y) \in R \Rightarrow (\forall z \in \Sigma^*)\ (xz,yz) \in R\ .$$

CHAPTER 2

COMPLEXITY OF GRAMMATICAL TRANSFORMATIONS

Section 2.1 - Introduction

In this chapter we analyze the time complexity of various grammatical transformations.

Grammatical transformations are algorithms that take a grammar and produce an equivalent grammar (i.e. one that generates the same language) in a certain restricted form. Many algorithms of this type may be found in the textbooks on this subject (Hopcroft and Ullman [1969], Harrison [1978]) and in the literature.

The purpose of this work is to analyze well known algorithms for their complexity and, if they are not efficient, either find better algorithms or show that the problems are inherently difficult.

Some of the algorithms found in the literature are given as constructive proofs for "Normal Form" theorems. They are intended to show that any language can be generated by a grammar of a certain form. As such, these algorithms are meant to be clear rather than efficient (and often there is a tradeoff between these two properties).

Other algorithms may have stood the test of time because they perform efficiently in most cases. Such algorithms may be very inefficient in the worst case, but these inefficiencies may only surface on specially designed grammars.

We would like to present our algorithm in a readable way without hiding the main complexity issues. We choose to write algorithms in Pidgin ALGOL (cf. Aho, Hopcroft and Ullman [1974]). This representation enables us to specify as much or as little of the actual implementation of the algorithm as we find necessary. We then analyze the time complexity assuming the algorithm is executed on a reasonable model of a computer. As hinted above, we are interested in worst case complexity, and as is customary we consider its asymptotic behavior.

To describe our results in terms of "order of magnitude", the following notation is convenient.

Definition 2.1.1. Let

$g(n) = O(f(n))$ if and only if there exist positive constants c, N_0 such that $g(n) \leq cf(n)$ for all $n \geq N_0$

$= \Omega(f(n))$ if and only if there exist positive constants C, N_0 such that $g(n) \geq cf(n)$ for all $n \geq N_0$

$= \Theta(f(n))$ if and only if there exist positive constants c, c' and N_0 such that $cf(n) \leq g(n) \leq c'f(n)$ for all $n \geq N_0$.

One can read $O(f(n))$ to be "order at most $f(n)$", $\Omega(f(n))$ as "order at least $f(n)$", $\Theta(f(n))$ as "order exactly $f(n)$".

Note that when dealing with O or Ω, $=$ is not reflexive. However using Θ leaves $=$ both reflexive and transitive. Θ distributes over the various arithmetic operators.

Since the complexity is computed as a function of the input size, and since the input to the algorithms is (an encoding of) a grammar we need to discuss the size of such an encoding. A reasonable encoding consists mostly of a list of the productions in the grammar. (The size of any additional information such as the list of nonterminals and terminals, as well as delimiters signifying end of production, etc. will be smaller.) There are two principal ways to measure the size of the encoding (i.e. of the production list of a grammar).

Definition 2.1.2. Let $G = (V,\Sigma,P,S)$ be a context free grammar. Define

$$|G| = \sum_{\substack{A \to \alpha \\ \text{in } P}} |A\alpha|$$

and

$$\|G\| = |G| \cdot \log_2 |V| \ .$$

$|G|$ is simply the number of symbols involved in productions. $\|G\|$ is a more realistic measure because it takes into account the number of bits needed to encode each symbol in V (assuming a fixed alphabet). Unless otherwise specified n will denote the size of the input using either measure.

Whenever $\|G\|$ is assumed as the size measure, we need to estimate $|G|$ and $|V|$, in order to compute the complexity. The following lemma establishes some relationship between these quantities.

Lemma 2.1.1. For any context free grammar $G = (V,\Sigma,P,S)$, if $L(G) \neq \emptyset$, $L(G) \neq \{\Lambda\}$, and if every letter in V occurs in at least one production, then

(i) $2 \leq |V| \leq |G|$

(ii) $|V| \leq \frac{2n}{\log n}$ where $n = \|G\| = |G| \log |V|$. (All logs are to the base 2.)

Proof. Since $S \in N$, $|N| \geq 1$. Since $\Sigma \neq \emptyset$, we have $|V| = |N| + |\Sigma| \geq 1 + 1 = 2$. The upper bound of (i) follows from the assumption that each symbol of V appears in at least one production.

From (i), we have

$$n = |G|\log|V| \geq |V|\log|V| \geq 2\log 2 .$$

Consider the function

$$f(x) = 2\sqrt{x} - \log x .$$

If $f(x)$ is differentiated

$$f'(x) = \frac{1}{\sqrt{x}} - \frac{c}{x} = \frac{\sqrt{x} - c}{x}$$

where
$$c = \frac{1}{\log 2} = 1.44269\ldots .$$

Thus $f'(x) > 0$ if $x > c^2 = 2.0813\ldots$. The minimum of the function is at c^2 and is $f(c^2) = 1.8278\ldots$. The function f is monotonically increasing for all $x > c^2$. Since our concern is only at positive integer values, the above argument shows that

$$f(n) > f(3)$$

for all integers $n > 3$. It is also true that

$$f(3) = 1.8791\ldots > 1.8284 = f(2) .$$

Hence

$$f(n) \geq f(2) = 2\sqrt{2} - 1 > 0$$

or for all $n \geq 2$

$$2\sqrt{n} - \log n > 0 .$$

It follows that, for all $n \geq 2$

$$2\sqrt{n} > \log n \ .$$

Taking logs

$$\begin{aligned} \log(2\sqrt{n}) &> \log\log n \\ \log 2 + \frac{1}{2}\log n &> \log\log n \\ \log 2 + \log n - \log\log n &> \frac{1}{2}\log n \\ \log\left(\frac{2n}{\log n}\right) &> \frac{1}{2}\log n \ . \end{aligned}$$

Multiplying by $\frac{2n}{\log n}$ (which is always positive in this range)

$$\frac{2n}{\log n}\log\left(\frac{2n}{\log n}\right) > n \ .$$

But

$$n \geq |V|\log|V|$$

so

$$\frac{2n}{\log n}\log\left(\frac{2n}{\log n}\right) > |V|\log|V| \ .$$

It follows immediately that if $x, y > 1$ and $x\log x > y\log y$ then $x > y$. Taking $x = \frac{2n}{\log n}$ and $y = |V|$, we have shown that

$$\frac{2n}{\log n} > |V| \ . \qquad \square$$

When discussing the complexity of algorithms we will often make two evaluations according to the size measure we use. The use of a specific measure for the size of the input will imply that if the algorithm has a grammar as its output then it is assumed to be written out in the same "format".

An interesting property of amny of the algorithms that perform grammatical transformation is that their time complexity is dominated by the size of the output grammar. In such cases we need only evaluate the size of the output grammar (using either size measure), and show that the computation itself is of the same order of magnitude as that size, in order to obtain the algorithm's complexity.

The rest of the chapter analyzes the complexity of various transformations.

Section 2.2 - Reduction

Reduction is a basic transformation used to achieve a property which is always desirable when dealing with grammars.

Definition 2.2.1. A grammar $G = (V,\Sigma,P,S)$ is said to be reduced if $P = \emptyset$ or for every $A \in N$, $S \overset{*}{\Rightarrow} \alpha A\beta \overset{*}{\Rightarrow} w$ for some $\alpha, \beta \in V^*$, $w \in \Sigma^*$.

It is well known that every language has a reduced grammar.

We shall now describe and analyze an algorithm for reduction which appears in most text books (cf. Aho and Ullman [1972]).

A high level version of the algorithm is

Algorithm 2.2.1.

Input: $G = (V,\Sigma,P,S)$

Output: grammar G' where G' is reduced and $L(G') = L(G)$

```
begin
GEN := $\{A \in N | \exists x \in \Sigma^*, A \overset{*}{\Rightarrow} x\}$;
if $S \in GEN$ then begin
                  $N''$ := GEN;
                  $V''$ := $N'' \cup \Sigma$;
                  $P''$ := $\{A \to \alpha \in P | A\alpha \in V''^*\}$;
                  REACH := $\{A \in N'' | \exists \alpha, \beta \in V''^*, S \overset{*}{\Rightarrow} \alpha A\beta\}$;
                  $N'$ := REACH;
                  $P'$ := $\{A \to \alpha \in P'' | A\alpha \in (N' \cup \Sigma)^*\}$;
                  $\Sigma'$ := $\{a \in \Sigma |$ a appears in some production $A \to \alpha$ in $P'\}$;
                  $V'$ := $N' \cup \Sigma'$
                  $G'$ := $(V',\Sigma',P',S)$
                  end
            else $\tilde{G}'$ := $(\{S\},\emptyset,\emptyset,S)$
end.
```

Example 2.2.1. Let $G = (V,\Sigma,P,S)$ where $\Sigma = \{a,b\}$, $V = \{S,A,B,C,D\} \cup \Sigma$ and P contains $S \to aA|AaB$, $A \to Sb|aC$, $B \to BD$, $C \to a$ and $D \to bC$.

We apply Algorithm 2.2.1.

First GEN is computed: $C \overset{*}{\Rightarrow} a$, $D \overset{*}{\Rightarrow} ba$, $A \overset{*}{\Rightarrow} aa$ and $S \overset{*}{\Rightarrow} aaa$ but for no $x \in \Sigma^*$ does $B \overset{*}{\Rightarrow} x$. So $GEN = \{S,A,C,D\}$. Since $S \in GEN$, we proceed to compute $N'' = \{S,A,C,D\}$, $V'' = N'' \cup \Sigma$ and P'' contains $S \to aA$, $A \to Sb|aC$, $C \to a$ and $D \to bC$. Next REACH is computed: $S \overset{*}{\Rightarrow} S$, $S \overset{*}{\Rightarrow} aA$, $S \overset{*}{\Rightarrow} aaC$ but, for no $\alpha, \beta \in V''^*$ does $S \overset{*}{\Rightarrow} \alpha D\beta$, so $REACH = \{S,A,C\}$. Consequently $N' = \{S,A,C\}$, P' contains $S \to aA$, $A \to Sb|aC$ and $C \to a$, $\Sigma' = \Sigma$ and $V' = N' \cup \Sigma$. Finally $G' = (V',\Sigma',P',S)$.

Algorithm 2.2.1, as presented, does not specify how the sets GEN and REACH are computed. The traditional way to do it is by the so called "nested-sets" construction:

Let $W_0 = \emptyset$ and for $i \geq 0$, $W_{i+1} = W_i \cup \{A \in N | A \to \alpha \in P$ for some $\alpha \in (W_i \cup \Sigma)^*\}$. Also let $U_1 = \{S\}$ and for $i \geq 1$, $U_{i+1} = U_i \cup \{A \in N'' | B \to \alpha A \beta \in P''$ for some $\alpha, \beta \in V''^*$ and $B \in U_i\}$.

The following results are proven in Harrison [1978].

Lemma 2.2.1. Let W_i, U_i be as above, $|N| = n$.

1. (i) For all $i \geq 0$, $W_i \subseteq W_{i+1}$.

 (ii) If, for some i, $W_i = W_{i+1}$, then for all $m > 0$, $W_i = W_{i+m}$.

 (iii) $W_n = W_{n+1}$

 (iv) For all $i \geq 0$, $W_i = \{A \in N | A \overset{*}{\Rightarrow} x$, for some $x \in \Sigma$ in a generation tree of height $\leq i\}$.

 (v) $W_n = GEN$.

2. (i) For all $i \geq 0$, $U_i \subseteq U_{i+1}$.

(ii) If, for some i, $U_i = U_{i+1}$, then for all $m > 0$, $U_i = U_{i+m}$.

(iii) $U_n = U_{n+1}$

(iv) For all $i \geq 0$, $U_i = \{A \in N'' | S \overset{*}{\Rightarrow} \alpha A \beta$ for some $\alpha, \beta \in V''^*$ in a derivation of length $\leq i-1\}$.

(v) U_n = REACH.

Lemma 2.2.1 shows that if we repeat the nested set construction to compute W_n (U_n) we get the desired set GEN (REACH). The next lemma shows that we may need to repeat the computation that many times.

Lemma 2.2.2. There exist grammars for which $W_{n-1} \neq$ GEN and $U_{n-1} \neq$ REACH, where $n = |N|$.

Proof. Let $G = (N \cup \{a\}, \{a\}, P, A_1)$ with $N = \{A_1, A_2, \ldots, A_n\}$ and $P = \{A_i \to A_{i+1} | 1 \leq i < n\} \cup \{A_n \to a\}$.

Clearly $W_0 = \emptyset$, $W_1 = \{A_n\}$, $W_2 = \{A_{n-1}, A_n\}, \ldots,$ $W_{n-1} = \{A_2, A_3, \ldots, A_n\} \subsetneq \{A_1, A_2, \ldots, A_n\} = W_n$ = GEN.

Also (noting that here $N'' = N$, $P'' = P$) we have $U_1 = \{A_1\}$, $U_2 = \{A_1, A_2\}, \ldots, U_{n-1} = \{A_1, A_2, \ldots, A_{n-1}\} \subsetneq \{A_1, A_2, \ldots, A_n\} = U_n$ = REACH. □

We are now ready to compute the time complexity of the algorithm.

Theorem 2.2.1. Let $G = (V, \Sigma, P, S)$ be any grammar. Algorithm 2.2.1 with the nested set construction, produces an equivalent reduced grammar G', in time $O(n^2)$ if $n = |G|$ is used as size measure, or time $O(\frac{n^2}{\log n})$ if $n = \|G\|$ is used.

Proof. The correctness of the algorithm is proved in Harrison [1978].

To compute W_{i+1} from W_i, one needs to scan the entire production set P of G. This takes time $O(n)$. By Lemma 2.2.1, we need to repeat this process at most $|N|$ times (and this upper bound is achievable).

So the time required for the computation of GEN is $O(|N|\cdot n)$. The complexity of the entire algorithm is of the same order of magnitude since REACH is similar in time complexity to GEN and the other steps are of smaller complexity (bounded by the size of the grammar).

To complete the proof for the two size measures we recall that $|N| \leq |V|$ and that by Lemma 2.2.1 $|V| \leq |G|$ and $|V| \leq \frac{2\cdot\|G\|}{\log\|G\|}$. Hence if $|G|$ is the size measure, Algorithm 2.2.1 requires time $O(|N|\cdot|G|) \leq O(|G|\cdot|G|) = O(n^2)$ and using $\|G\|$ as size measure the time required is

$$O(|N|\cdot\|G\|) \leq O(\frac{\|G\|}{\log\|G\|}\|G\|) = O(\frac{n^2}{\log n}) . \qquad \square$$

A question raised by the above analysis is whether or not we can do better. A close examination of the nested set construction shows that while each computation of W_{i+1} involves rescanning the entire grammar, only a small fraction of it is pertinent. Moreover, each production can only yield information about the symbols that appear in it. It appears that if we organize the information provided by the grammar in some meaningful way, scanning the grammar many times will not be required.

We present an algorithm to compute GEN using ideas suggested by Hunt, Szymanski and Ullman [1974]. First we discuss the data structures used by the algorithm in some detail. W and U are both sets. W is used to collect elements known to be in GEN (elements are added but never removed from W). The use of U will become clear later. When the grammar is read, a symbol A is entered in both W and U whenever a production $A \rightarrow x$ where $x \in \Sigma^*$ is encountered (and provided A is not yet in GEN). For each $B \in N$, POS(B) is a multiset (i.e. analogous to a set but elements may appear more than once, cf. Knuth [1969]). Elements of POS(B) are productions in P. In particular when a production $A \rightarrow \alpha$ is read in, it is entered in POS(B) ℓ times if B appears in α ℓ times. This is done for all $B \in N$. The information in POS(B) is later consulted in the process of "updating."

For each production $A \rightarrow \alpha$ in P the integer NONGEN($A \rightarrow \alpha$) denotes the number of occurrences in α of symbols not yet known to generate any terminal string.* This number is constantly updated and if and when it reaches 0, we may conclude that A can generate some terminal string. If that is not already known (i.e. if A is not yet in W) then A is entered in W and in U.

The process of "updating" is as follows. A symbol B is removed from U. B is now known to be in GEN (i.e. to generate some terminal string). Therefore for each $A \rightarrow \alpha$ in P we decrement NONGEN($A \rightarrow \alpha$) by one for each occurrence of B in α. To do this only POS(B) need be inspected (rather than the entire grammar):

*In fact NONGEN($A \rightarrow \alpha$) is defined only for $\alpha \notin \Sigma^*$, but one can assume that the value is 0 for $\alpha \in \Sigma^*$, since this value is never consulted anyway.

for each occurrence of a production $A \to \alpha$ in POS(B), NONGEN($A \to \alpha$) is decremented by one. As mentioned above we add A to W and U whenever, in the course of decrementing NONGEN($A \to \alpha$), it reaches 0 and if A is not in W. Note that U is always a subset of W containing those elements for which "updating" was not yet done. When U becomes empty the algorithm terminates.

GEN appears as a variable in the algorithm. Just before termination it is assigned the value of W.

Next we present the algorithm.

Algorithm 2.2.2.

Input: $G = (V,\Sigma,P,S)$

Output: GEN $= \{A \in N \mid \exists x \in \Sigma^*,\ A \overset{*}{\Rightarrow} x\}$

```
    begin
L1: W := $\emptyset$;
    U := $\emptyset$;
    for all $B \in N$ do POS(B) := $\emptyset$;
L2: for all $A \to \alpha$ in P do
       begin
       if $\alpha \in \Sigma^*$ then begin
                          if $A \notin W$ then begin
                                          W := $W \cup \{A\}$;
                                          U := $U \cup \{A\}$
                                          end
                          end
                    else begin
                          comment $\alpha = \alpha_0 B_1 \cdots \alpha_{k-1} B_k \alpha_k$, $k \geq 1$, $B_i \in N$, $\alpha_i \in \Sigma^*$;
                          NONGEN($A \to \alpha$) := k;
                          for all $1 \leq i \leq n$ do POS($B_i$) := POS($B_i$) $\cup \{A \to \alpha\}$
                          end
       end;
```

```
L3:  while U is not empty do
       begin
       comment $B \in U$;
       $U := U - \{B\}$;
       for all $A \to \alpha$ in POS(B) do
         begin
         NONGEN($A \to \alpha$) := NONGEN($A \to \alpha$) - 1;
         if NONGEN($A \to \alpha$) = 0 and $A \notin W$ then begin
                                                  $W := W \cup \{A\}$;
                                                  $U := U \cup \{A\}$
                                                  end
         end
       end;
L4:  GEN := W
     end.
```

The statement labels L1, L2, L3 and L4 used in the algorithm designate the start of four phases in the algorithm: Initialization (of W, U and POS), reading the grammar (the for loop), "updating" (the while loop), and outputing the result.

The following example illustrates the behavior of the algorithm.

Example 2.2.2. Let $G = (N \cup \{a\}, \{a\}, P, A_1)$, where $N = \{A_1, A_2, \ldots, A_n\}$ and $P = \{A_i \to A_{i+1} \mid 1 \leq i < n\} \cup \{A_n \to a\}$. (This is the grammar used in proving Lemma 2.2.2). Apply Algorithm 2.2.2.

When L2 is reached for the first time $W = \emptyset$, $U = \emptyset$ and POS(B) is empty for all $B \in N$.

After the for loop at L2 is executed once (with the production $A_1 \to A_2$), we obtain NONGEN($A_1 \to A_2$) = 1 and POS(A_2) contains the single element $A_1 \to A_2$ (once). The for loop is then executed with the productions $A_2 \to A_3, \ldots, A_{n-1} \to A_n$ and finally $A_n \to a$.

When L3 is reached for the first time $POS(A_i)$ contains the single element $A_{i-1} \rightarrow A_i$ (once) for all i, $2 \leq i \leq n$. The values of the other variables at that point are summarized in the first column of the following table. Note that afterwards, the values of $POS(B)$ are not changed for any $B \in N$.

The while loop is executed n times. The values of the relevant variables after executing that loop $1,2,\ldots,n$ times are displayed in the following table. Note that the value of $NONGEN(A_n \rightarrow a)$ is given in parenthesis. This variable does not exist in the algorithm, and its value is given for the sake of (theoretical) completeness. As mentioned above, this is true for all values $NONGEN(A \rightarrow \alpha)$ when $\alpha \in \Sigma^*$.

It should be noted here that if the productions in the grammar were ordered differently, then the while loop may have been executed fewer times. However even in this worst ordering the computation is efficient because we only look at the "right points" in the grammar rather than make a full scan every time.

content of some variables	when L3 is reached for the 1st time	when L3 is reached for the 2nd time i.e. after executing while loop once	after executing while loop twice		n-2 times	n-1	n
NONGEN($A_1 \to A_2$)	1	1	1		1	0	0
NONGEN($A_2 \to A_3$)	1	1	1		0	0	0
NONGEN($A_3 \to A_4$)	1	1	1		0	0	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$	$\vdots$
NONGEN($A_{n-3} \to A_{n-2}$)	1	1	1	. . .	0	0	0
NONGEN($A_{n-2} \to A_{n-1}$)	1	1	0		0	0	0
NONGEN($A_{n-1} \to A_n$)	1	0	0		0	0	0
W	$\{A_n\}$	$\{A_n, A_{n-1}\}$	$\{A_n, A_{n-1}, A_{n-2}\}$		$\{A_n, \ldots, A_2\}$	$\{A_n, \ldots, A_1\}$	$\{A_n, \ldots, A_1\}$
U	$\{A_n\}$	$\{A_{n-1}\}$	$\{A_{n-2}\}$		$\{A_2\}$	$\{A_1\}$	$\emptyset$

A completely formal proof of correctness of Algorithm 2.2.2 (using the techniques of Hoare [1969]) is lengthy and rather technical. We will give a less formal argument.

We denote, for any set $M \subseteq N$ and any string $\alpha \in V^*$, $OC(M,\alpha)$ to be the number of occurrences of symbols from M in α. To avoid confusion we will use D and $C \to \beta$ as bound elements from N and P respectively.

The next lemma establishes invariant conditions for the while loop at L3.

Lemma 2.2.3. The following conditions are invariants to the while loop at L3 (i.e. if conditions (1)-(5) hold at L3, and if the while loop is then executed once then conditions (1)-(5) hold after that execution).

(1) For each $D \in N$ and each $C \to \beta$ in P, $OC(\{D\},\beta)$ = the number of times $C \to \beta$ appears in $POS(D)$.

(2) $U \subseteq W$

(3) For each $C \to \beta$ in P $NONGEN(C \to \beta) = OC(N,\beta) - OC(W-U,\beta)$.

(4) $W = \{C \in N \mid \exists \beta \in V^*$ such that $C \to \beta$ is in P and $NONGEN(C \to \beta) = 0\}$

(5) $W \subseteq \{C \in N \mid \exists x \in \Sigma^*, C \overset{*}{\Rightarrow} x\}$

Proof. Assume (1)-(5) hold when the while condition is about to be executed. Also, suppose $U \neq \emptyset$ and $B \in U$. Then the while loop will be executed.

(1) holds after execution of the loop since $POS(D)$ remains unchanged for all $D \in N$.

Execution of the loop removes B from U and then adds zero or more elements onto both W and U. Thus $U \subseteq W$ must remain true. Moreover the only change in $W-U$ is the addition to it of B (before execution of the loop $B \in U$ and $U \subseteq W$, B is removed from U but not from W). For all $C \rightarrow \beta$ in P, execution of the loop decrements $\text{NONGEN}(C \rightarrow \beta)$ by the number of occurrences of $C \rightarrow \beta$ in $\text{POS}(B)$. By (1) that quantity is $\text{OC}(\{B\},\beta)$. So, for all $C \rightarrow \beta$ in P both sides of equation (3) are decremented by the same number so that (3) remains true.

From (3) and the fact that $W-U \subseteq N$ it is clear that $\text{NONGEN}(C \rightarrow \beta) \geq 0$ is satisfied for all $C \rightarrow \beta$ in P. Therefore, since the loop never increments $\text{NONGEN}(C \rightarrow \beta)$ for any $C \rightarrow \beta$ in P, no element may leave the right hand side of equation (4) during execution of the loop. The same is true of W, the left hand side of that equation. An element C may enter the right hand side if $\text{NONGEN}(C \rightarrow \beta)$ is decremented to zero for some $C \rightarrow \beta$ in P so that C was not yet in the set. But whenever that happens, the *if* condition is satisfied and the element is placed in W (and in U). Hence condition (4) is preserved.

Now suppose (5) holds just before execution of the while loop. Let $C \in N$ be any element that would be placed in W during execution of the loop. Since (4) would hold after execution of the loop it follows that for some production $C \to \beta$ in P, $\mathrm{NONGEN}(C \to \beta)$ would be zero after execution of the loop. From the proof of (3) it follows that for that particular production $\mathrm{NONGEN}(C \to \beta) = \mathrm{OC}(\{B\},\beta)$ just before execution of the loop. By (3) that means $\beta \in ((W-U) \cup \{B\} \cup \Sigma) \subseteq (W \cup \Sigma)^*$. We can write $\beta = \beta_0 B_1 \beta_1 \cdots B_n \beta_n$ for some $n \geq 0$, $B_i \in W$ for all $1 \leq i \leq n$ and $\beta_i \in \Sigma^*$ for all $0 \leq i \leq n$. By (5) (which holds just before execution of the loop), there must exist, for all i, $1 \leq i \leq n$ $x_i \in \Sigma^*$ such that $B_i \overset{*}{\Rightarrow} x_i$ for all i, $1 \leq i \leq n$. Therefore $C \Rightarrow \beta = \beta_0 B_1 \beta_1 \cdots B_n \beta_n \overset{*}{\Rightarrow} \beta_0 x_1 \beta_1 \cdots x_n \beta_n$ but $\beta_0 x_1 \beta_1 \cdots x_n \beta_n \in \Sigma^*$ so that C belongs to the right hand side of equation (5). Since C was an arbitrary element which is added to W during execution of the loop we conclude that (5) is satisfied after execution of the loop.

Lemma 2.2.4. Algorithm 2.2.2 correctly computes GEN in linear time.

Proof. First we consider "partial correctness" (cf. Manna [1974]). We want to show that if the algorithm terminates then $\mathrm{GEN} = \{C \in N \mid \exists x \in \Sigma^*,\ C \overset{*}{\Rightarrow} x\}$. This will follow directly from Lemma 2.2.3 and the next two claims, which deal with the parts of the algorithm before and after the while loop, respectively.

Claim 1. When L3 is reached for the first time (after execution of the initialization and reading phases) conditions (1)-(5) of Lemma 2.2.3 are satisfied.

Proof. It is quite easy to verify that when L3 is reached for the first time (1) is satisfied. Also

(6) For all $C \to \beta$ in P, $\text{NONGEN}(C \to \beta) = \text{OC}(N,\beta)$

(7) $W = \{C \in N \mid \exists x \in \Sigma^*$ so that $C \to x$ is in $P\}$

and (8) $U = W$

Then (2) follows directly from (8), (3) follows from (6) and the fact that $W - U = \emptyset$. From (6) we also obtain that $\text{NONGEN}(C \to \beta) = 0$ if and only if $\beta \in \Sigma^*$, hence using (7) we get $W = \{C \in N \mid \exists x \in \Sigma^*, C \to x$ is in $P\} = \{C \in N \mid$ there exist $C \to \beta$ in P, $\text{NONGEN}(C \to \beta) = 0\}$ and (4) follows. (5) clearly holds since for all $C \in W$, $C \Rightarrow x$ for some $x \in \Sigma^*$.

Claim 2. Suppose (1)-(5) hold at L4, and assume $U = \emptyset$. Then after this line has been executed $\text{GEN} = \{C \in N \mid \exists x \in \Sigma^*, C \overset{*}{\Rightarrow} x\}$.

Proof. For this condition to be satisfied after execution of this line, we must have $W = \{C \in N \mid \exists x \in \Sigma^*, C \overset{*}{\Rightarrow} x\}$ at L4. This will be shown to follow from (1)-(5) and $U = \emptyset$. In particular (3) and $U = \emptyset$ implies that for each $C \to \beta$ in P $\text{NONGEN}(C \to \beta) = \text{OC}(N,\beta) - \text{OC}(W,\beta)$. So that $\text{NONGEN}(C \to \beta) = 0$ if and only if $\beta \in (W \cup \Sigma)^*$. Therefore, using (4), $W = \{C \in N \mid \exists \beta \in (W \cup \Sigma)^*$ such that $C \to \beta$ is in $P\}$.

We now prove that $W = \{C \in N \mid \exists x \in \Sigma^*, C \overset{*}{\Rightarrow} x\}$ by contradiction. Since (5) directly yields containment in one direction we assume, for the sake of contradiction, that $\{C \in N \mid \exists x \in \Sigma^*, C \overset{*}{\Rightarrow} x\} \subsetneq W$, and choose $A \in \{C \in N \mid \exists x \in \Sigma^*, C \overset{*}{\Rightarrow} x\} - W$ such that A has a shortest derivation of a terminal string $A \overset{i}{\Rightarrow} x$ among all elements in this set difference.

Since $i > 0$ we can write $A \Rightarrow B_1B_2\cdots B_m \overset{i-1}{\Rightarrow} x$. Then for each j, $1 \leq j \leq m$, $B_j \overset{*}{\Rightarrow} x_j$ for some $x_j \in \Sigma^*$ in a derivation of length less than i, and $x = x_1\cdots x_m$. By the minimality of i none of the B_j's can be in $\{C \in N | \exists x \in \Sigma^*, C \overset{*}{\Rightarrow} x\} - W$. But all the nonterminals among the B_j's must belong to $\{C \in N | \exists x \in \Sigma^*, C \overset{*}{\Rightarrow} x\}$ and therefore also to W. Then $B_1\cdots B_m \in (W \cup \Sigma)^*$ and $A \in \{C \in N | \exists \beta \in (W \cup \Sigma)^*$ such that $C \rightarrow \beta$ is in $P\} = W$. This contradiction completes the proof of the claim.

We can observe that every element of N can be removed from U at most once (since an element is entered in W and U only when it is not already there, and nothing is ever removed from W). Therefore the while loop is executed at most $|N|$ times. This immediately proves termination and hence total correctness (cf. Manna [1974]).

Before we can compute the time complexity of the algorithm we must specify the implementation of some data objects.

We use an array of bits to implement W so that membership may be checked in constant time.† U is implemented as a stack so choosing an element takes constant time. For each $D \in N$ POS(D) is stored as a list, so that adding an element takes constant time and scanning the entire list requires a constant time per element.

Note that the comment that $\alpha = \alpha_0B_1\cdots B_n\alpha_n$ does not imply extra computation since the various elements may be determined as α is being read in.

The comment $B \in U$ implies the operation "choose an element."

Initialization consists of $|N| + 2$ operations (of assignment to $\emptyset$). For every $A \rightarrow \alpha \in P$, NONGEN($A \rightarrow \alpha$) is once set to a value $n \leq |\alpha|$ and then decremented at most $|\alpha|$ times, and compared to 0

†When uniform cost criterion is used, array indexing takes constant time. Cf. Aho, Hopcroft and Ullman [1974].

that many times. The number of operations involving NONGEN is therefore proportional to the size of G. The same is true for operations on POS, since every position of a nonterminal is recorded once and consulted at most once. As noted above we can have at most $|N|$ operations of each of the following types: adding an element to W, adding an element to U, choosing and removing an element from U. Checking for membership in W can be performed at most $2|P|$ times. In the reading phase, a check may be done for each production and one check per production can occur in the "updating" phase. In fact one can show that only $|P|$ operations are required. Each operation discussed takes a constant amount of time. We conclude that the time required by the algorithm is $O(|G|)$ if we consider reading of a symbol a constant-time operation and $O(\|G\|)$ otherwise. □

An algorithm similar to Algorithm 2.2.2 can be given for other nested constructions. We now present an algorithm to compute REACH.

W and U play the same roles as in Algorithm 2.2.2. For each $B \in N$, $R(B)$ is a set containing all nonterminals A such that $B \to \alpha A \beta$ is in P for some $\alpha, \beta \in V^*$. $R(B)$ can be likened to $POS(B)$ of Algorithm 2.2.2.

Algorithm 2.2.3.

Input: $G = (V,\Sigma,P,S)$

Output: $REACH = \{A \in N \mid \beta_1,\beta_2 \in V^*,\ S \overset{*}{\Rightarrow} \beta_1 A \beta_2\}$

```
     begin
L1:  W := {S};
     U := {S};
     for all $B \in N$ do $R(B) := \emptyset$;
L2:  for all $B \to \alpha \in P$ do $R(B) := R(B) \cup \{A_1,A_2,\ldots,A_n\}$
                         comment $\alpha = \alpha_0 A_1 \alpha_1 \cdots A_n \alpha_n,\ n \geq 0,\ A_i \in N,\ \alpha_i \in \Sigma^*$;
L3:  while U is not empty do
        begin
        comment $B \in U$;
        $U := U - \{B\}$;
        for all $A \in R(B)$ do if $A \notin W$ then begin
                                          $W := W \cup \{A\}$;
                                          $U := U \cup \{A\}$
                                          end
        end;
L4:  REACH := W
     end.
```

Example 2.2.3. Let $G = (V,\Sigma,P,S)$ where $\Sigma = \{a,b\}$, $V = \{S,A,E,C,D\} \cup \Sigma$ and P contains the productions $S \to AEA$, $A \to a$, $E \to bC$, $C \to aA|bE$ and $D \to AC$.

We apply Algorithm 2.2.3. When L2 is reached for the first time $W = U = \{S\}$ and for all $B \in N$, $R(B) = \emptyset$. After executing the for loop (at L2) once with $S \to AEA$ we get $R(S) = \{A,E\}$. Then the for loop is executed with $A \to a,\ldots,D \to AC$. When L3 is finally reached we have $W = U = \{S\}$, $R(S) = \{A,E\}$, $R(A) = \emptyset$, $R(E) = \{C\}$, $R(C) = \{A,E\}$, $R(D) = \{A,C\}$. The values of $R(B)$ are not changed from here on for any $B \in N$.

The *while* loop is now executed with $B = S$. After that we have $W = \{S,A,E\}$ and $U = \{A,E\}$. The loop is executed three more times and finally $W = \{S,A,E,C\}$ and $U = \emptyset$. REACH is the set $\{S,A,E,C\}$ and the computation terminates.

Lemma 2.2.5. Algorithm 2.2.3 correctly computes REACH in linear time.

Proof. Since the proof closely parallels the proofs of Lemma 2.2.3 and Lemma 2.2.4, it will not be given in detail. Instead we present the invariant conditions (analogous to (1)-(5) of Lemma 2.2.3) of the *while* loop:

(1') For each $D \in N$, $R(D) = \{C \in N \mid \exists \beta_1, \beta_2 \in V^*,\ D \to \beta_1 C \beta_2 \text{ is in } P\}$

(2') $U \subseteq W$

(3',4') $W = \{C \in N \mid \text{there exist } D \in W\text{-}U \text{ and } \beta_1, \beta_2 \in V^* \text{ so that } D \to \beta_1 C \beta_2 \text{ is in } P\} \cup \{S\}$

(5') $W \subseteq \{C \in N \mid \exists \beta_1, \beta_2 \in V^*,\ S \overset{*}{\Rightarrow} \beta_1 C \beta_2\}$ □

We now turn back to the original problem, that of reducing a grammar.

Theorem 2.2.2. There is an algorithm that performs reduction of any grammar $G = (V,\Sigma,P,S)$ in time linear in the size of the grammar.

Proof. The algorithm we will use is Algorithm 2.2.1 with Algorithm 2.2.2 computing GEN and Algorithm 2.2.3, applied to $G'' = (V'',\Sigma,P'',S)$, calculating REACH. Since except for GEN and REACH Algorithm 2.2.1 takes linear time, the result follows by Lemma 2.2.4 and Lemma 2.2.5. □

Theorem 2.2.2 shows that reduction is a fairly easy operation. The result implies that we can perform this transformation as fast as one can ever hope to do, and that we need not be content with the algorithms that use nested set construction.

Section 2.3 - Null Rule Elimination

Null rules (i.e. productions like $A \rightarrow \Lambda$) are often undesirable in a grammar either for a theoretical reason (it may be easier to prove properties of grammars with no such rules) or for practical considerations (some parsing techniques may fail to work in the presense of null rules). Transformations to remove null rules are therefore necessary.

Definition 2.3.1. A grammar $G = (V,\Sigma,P,S)$ is said to be Λ-free if $A \rightarrow \Lambda \in P$ implies

(1) $A = S$

and (2) S does not appear in the right hand side of any production.

This definition allows $S \rightarrow \Lambda$ in P to cover the case $\Lambda \in S$. The restriction that if $S \rightarrow \Lambda$ is in P then S does not appear in the right hand side of any production is designed to insure that the production $S \rightarrow \Lambda$ appears only in derivation of Λ.

It is a well known fact that every language has a Λ-free grammar which can be effectively constructed.

The following is the classical algorithm due to Bar-Hillel, Perles and Shamir [1961]. It starts out with a reduced grammar, as do all the algorithms we discuss from here on.

Algorithm 2.3.1.

Input: $G = (V,\Sigma,P,S)$ a reduced grammar

Output: grammar G' such that $L(G') = L(G)$ and G' is Λ-free

```
begin
NULL := $\{A \in N \mid A \overset{*}{\Rightarrow} \Lambda\}$;
N' := N;
P' := $\emptyset$;
for all $A \to \alpha \in P$ do comment $\alpha = \alpha_0 B_1 \cdots B_n \alpha_n$, $n \geq 0$, $B_i \in$ NULL,
                                $\alpha_i \in (V\text{-NULL})^*$;
   for all $(X_1, X_2, \ldots, X_n) \in \{B_1,\Lambda\} \times \{B_2,\Lambda\} \times \cdots \times \{B_n,\Lambda\}$ such that
      $\alpha_0 X_1 \alpha_1 \cdots X_n \alpha_n \neq \Lambda$ do P' := P' $\cup \{A \to \alpha_0 X_1 \alpha_1 \cdots X_n \alpha_n\}$;
if $S \in$ NULL then begin N' := N' $\cup \{S'\}$; P' := P' $\cup \{S' \to S, S' \to \Lambda\}$ end
                 else S' := S;
G' := (N' $\cup \Sigma, \Sigma$, P', S')
end.
```

This algorithm should be followed by reduction, since some nonterminals may become useless.

The computation of NULL needs to be specified (as was the case with the sets GEN and REACH in Algorithm 2.2.1). It turns out, however, that the above algorithm has time complexity which is so large that the way NULL is computed is irrelevant.

Lemma 2.3.1. Algorithm 2.3.1 performs null rule elimination in exponential time.

Proof. The correctness of this algorithm is proved in Harrison [1978].

We will now present a grammar G, for which Algorithm 2.3.1 produces a Λ-free grammar G' which has size exponentially larger than that of G. This will be sufficient to prove the result since the size of the output is clearly a lower bound on the time complexity.

More precisely we will consider an infinite family of grammars $G_1, G_2, \ldots, G_k, \ldots$ where $G_k = (V_k, \Sigma_k, P_k, A)$, $N_k = \{A, B_1, B_2, \ldots, B_k\}$, $\Sigma_k = \{a_1, a_2, \ldots, a_k\}$ and $P_k = \{A \to B_1 B_2 \cdots B_k\} \cup \{B_i \to a_i,\ B_i \to \Lambda \mid 1 \leq i \leq k\}$. (In subsequent discussions the subscript k will be omitted whenever no confusion may arise and we will talk about G, N, Σ, V, P, etc.). We can see that $NULL = N$ since $B_i \Rightarrow \Lambda$ for each i and $A \Rightarrow B_1 B_2 \cdots B_k \overset{k}{\Rightarrow} \Lambda$. The production $A \to B_1 B_2 \cdots B_k$ in P will yield $2^k - 1$ productions in P', namely $A \to \beta$ for every non-null subword β of $B_1 B_2 \cdots B_k$. The result of the transformation (again omitting subscripts) is $G' = (V \cup \{S\}, \Sigma, P', A')$ where $P' = \{A' \to A,\ A' \to \Lambda\} \cup \{A \to X_1 X_2 \cdots X_k \mid X_i \in \{B_i, \Lambda\},\ X_1 X_2 \cdots X_k \neq \Lambda\} \cup \{B_i \to a_i \mid 1 \leq i \leq k\}$.

We can compute the sizes of the grammars involved:

$$
\begin{aligned}
|V| &= 2k + 1 \\
|G| &= k + 1 + (2+1)k = 4k + 1 \\
|V'| &= |V| + 1 = 2k + 2 \\
|G'| &= 2 + 1 + \sum_{j=1}^{k} (j+1)\binom{k}{j} + 2k \\
&= 2k + 3 + \sum_{j=0}^{k} j\binom{k}{j} + \sum_{j=0}^{k} \binom{k}{j} - 1 \\
&= 2k + 2 + k \cdot 2^{k-1} + 2^k \\
|G'| &= (k+2)2^{k-1} + 2k + 2
\end{aligned}
$$

$|G'|$ is exponentially larger than $|G|$, and the same is true for $\|G'\|$ as a function of $\|G\|$. $\square$

The proof indicates a stronger result than the one stated in the lemma.

<u>Corollary</u>. Any algorithm for null rule elimination which produces the same output grammar as Algorithm 2.3.1 takes at least exponential time.

In Graham [1974] an algorithm to eliminate null rules without destroying the (m,n) BRC property is introduced. While the later requirement calls for a more complicated construction than Algorithm 2.3.1, it does resemble it. In particular, when that algorithm is applied to the grammar G in the proof of Lemma 2.3.1 (which is clearly (k,k) BRC), the resulting grammar is essentially G', which is exponentially larger.

Rosenkrantz and Stearns [1970] present a null rule elimination algorithms for LL grammars which guarantees an LL(k+1) grammar as a result if the original grammar is LL(k). This algorithm cannot be used for arbitrary grammars since the construction is shown to produce a finite number of nonterminals only for unambiguous grammars.

The question to ask at this point is: why does this algorithm produce such large grammars, and is there any better way to do it? Clearly, the exponential growth is the result of a "subset construction" reminiscent of the transformation from nondeterministic to deterministic finite automata. The following observation proves useful in the realization that unlike the finite automaton case, null rule elimination may be done without possible exponential explosion. If the length of the right hand sides is bounded by ℓ, then $|G'|$ will only be of size $O(2^{\ell}\cdot|G|)$ since a production $A \rightarrow B_1B_2\cdots B_{\ell}$ will be replaced in Algorithm 2.3.1 by productions of total length $\leq \sum_{j=1}^{\ell} (j+1)\binom{k}{j}$ $= (\ell+2)2^{\ell-1} - 1$. If ℓ is constant this means a linear growth in size.

Since ℓ may be forced to be small by an appropriate transformation (which, as we shall see in section 2.5 is quite efficient) there is a good prospect that Λ-rules can be removed without huge increases in size. This approach will be directly used later (in section 2.5). The next algorithm for null rule elimination uses similar ideas in a somewhat different way.

Every production $A \to \alpha$ in P is written as $A \to \alpha_0 B_1 \cdots B_n \alpha_n$ where the B_j's are the only symbols in the right hand side that are in NULL. The production is then factored using some new nonterminals $A_j(A \to \alpha)$ for all $1 \le j \le n-1$ (or A_j for short). A_j will simulate $\alpha_j B_{j+1} \cdots B_n \alpha_n$. For instance we let $A \to \alpha_0 B_1 A_1$ and $A \to \alpha_0 A_1$ be productions in the new grammar. The second of these two will be used when simulating $A \to \alpha_0 B_1 \cdots B_n \alpha_n$ followed by $B_1 \overset{*}{\Rightarrow} \Lambda$. Special attention is needed for the case where a suffix of the right hand side may derive Λ. If, for example, in the above production $\alpha_1 = \alpha_2 = \cdots = \alpha_n = \Lambda$ then $A \underset{G}{\Rightarrow} \alpha_0 B_1 \cdots B_n \alpha_n \overset{*}{\Rightarrow} \alpha_0 B_1$ need to be simulated by the new grammar. We therefore need $A \to \alpha_0 B_1$ as well. In general we define, for each production, a number m which designates the beginning of such a suffix.

We now present the algorithm. Note that as in Algorithm 2.3.1 some nonterminals may become useless so reduction must follow Algorithm 2.3.2 as well.

Algorithm 2.3.2.

Input: $G = (V,\Sigma,P,S)$ a reduced grammar

Output: grammar G' such that $L(G') = L(G)$ and G' is Λ-free

```
begin
NULL := $\{A \in N | A \overset{*}{\Rightarrow} \Lambda\}$;
N' := N;
P' := $\emptyset$;
L: for all $A \to \alpha$ in P do
    begin
    comment $\alpha = \alpha_0 B_1 \cdots B_n \alpha_n$, $n \geq 0$, $B_j \in$ NULL, and $\alpha_j \in (V\text{-NULL})^*$;
    m := $\min\{\{0 \leq j \leq n | \alpha_j \alpha_{j+1} \cdots \alpha_n = \Lambda\} \cup \{n+1\}\}$;
    if n = 0 then begin
                    if m = 1 then P' := P' $\cup \{A \to \alpha\}$ comment $\alpha \in (V\text{-NULL})^+$;
                    end
              else begin
                    N' := N' $\cup \{A_j(A \to \alpha) | 1 \leq j \leq n-1\}$;
                    comment denote $A_j(A \to \alpha)$ by $A_j$ for short, $A = A_0$;
                    for j = 0 to n-2 do P' := P' $\cup \{A_j \to \alpha_j B_{j+1} A_{j+1}$,
                                                      $A_j \to \alpha_j A_{j+1}\}$;
                    P' := P' $\cup \{A_{n-1} \to \alpha_{n-1} B_n \alpha_n\}$;
                    if $m \geq n$ then P' := P' $\cup \{A_{n-1} \to \alpha_{n-1} \alpha_n\}$
                                comment $\alpha_{n-1}\alpha_n \neq \Lambda$;
                    for j = max{0,m-1} to n-2 do P' := P' $\cup \{A_j \to \alpha_j B_{j+1}\}$;
                    if $1 \leq m \leq n-1$ then P' := P' $\cup \{A_{m-1} \to \alpha_{m-1}\}$
                    end
    end;
if S $\in$ NULL then begin N' := N' $\cup \{S'\}$; P' := P' $\cup \{S' \to S, S' \to \Lambda\}$ end
              else S' := S;
G' := (N' $\cup \Sigma, \Sigma$, P', S')
end.
```

The following example demonstrates the way Algorithm 2.3.2 works.

Example 2.3.1. Let $G = (V,\Sigma,P,S)$ where $\Sigma = \{a,b_1,b_2,b_3\}$, $V = \{A,B_1,B_2,B_3\} \cup \Sigma$ and P contains $A \to B_1aB_2B_3$, $B_1 \to b_1|\Lambda$, $B_2 \to b_2|\Lambda$, $B_3 \to b_3|\Lambda$. Apply Algorithm 2.3.2. The only "interesting" production is $S \to B_1aB_2B_3$. B_1, B_2, $B_3 \in$ NULL so we have $n = 3$, $\alpha_0 = \alpha_2 = \alpha_3 = \Lambda$ and $\alpha_1 = a$. Hence $m = 2$. When the for loop in L is executed with this production we place in P' $A_0 \to B_1A_1|A_1$, $A_1 \to aB_2A_2|aA_2$ and $A_2 \to B_3$ (where $A = A_0$). Then we add $A_1 \to aB_2$ and finally $A_1 \to a$. When the for loop is executed with $B_1 \to b_1$ we get $n = 0$, $m = 1$ so this production is placed in P' (same for $B_2 \to b_2$, $B_3 \to b_3$). When the for loop is executed with $B_1 \to \Lambda$, $n = m = 0$ and nothing is placed in P' (same for $B_2 \to \Lambda$, $B_3 \to \Lambda$).

In the next example we apply Algorithm 2.3.2 to the grammar used in the proof of Lemma 2.3.1. This is to demonstrate the efficiency of the new algorithm.

Example 2.3.2. For any $k > 1$, let $G = (V,\Sigma,P,A)$, $N = \{A,B_1,\ldots,B_k\}$, $\Sigma = \{a_1,\ldots,a_k\}$ and $P = \{A \to B_1B_2\cdots B_k\} \cup \{B_j \to a_j,\ B_j \to \Lambda | 1 \le j \le k\}$. Applying Algorithm 2.3.2 we get $G' = (V',\Sigma,P',A')$, $N' = \{A',\ A = A_0,A_1,\ldots,A_{k-1},B_1,\ldots,B_k\}$. The production $A \to B_1\cdots B_k \in P$ is replaced by a set of new productions (here $n = k$, $m = 0$), and we get $P' = \{A' \to A,\ A' \to \Lambda\} \cup \{B_j \to a_j | 1 \le j \le k\}$ $\cup \{A_j \to B_{j+1}A_{j+1},\ A_j \to A_{j+1},\ A_j \to B_{j+1} | 0 \le j \le k-2\} \cup \{A_{k-1} \to B_k\}$. We get $|V'| = 3k+1$, $|G'| = 9k-2$ so that here both $|G'|$ and $\|G'\|$ are linear in $|G|$ and $\|G\|$ respectively. $\|G\| = (4k+1)\log(2k+1)$, $\|G'\| = (9k-2)\log(3k+1)$.

We now prove the correctness of Algorithm 2.3.2.

Lemma 2.3.2. Algorithm 2.3.2 correctly performs null rule elimination.

Proof. All the algorithm does is scan the grammar G and add productions to P' (and nonterminals to N') as it goes. It is clear that the algorithm terminates. It is also easy to verify that G' is Λ-free. We need to verify that no null rules are placed in P'. The only two places that need to be checked are 1) when $A_{n-1} \to \alpha_{n-1}\alpha_n$ is placed in P' and 2) when $A_{m-1} \to \alpha_{m-1}$ is considered. 1) $A_{n-1} \to \alpha_{n-1}\alpha_n$ is placed in P' only when $m \geq n \geq 1$ so $\alpha_{n-1}\alpha_n \neq \Lambda$. 2) $A_{m-1} \to \alpha_{m-1}$ is placed in P' only if $n \geq 1$ and $1 \leq m \leq n-1$ in which case α_{m-1} is defined and, by minimality of m, $\alpha_{m-1} \neq \Lambda$.

We need to show that $L(G') = L(G)$. First, it is clear that $\Lambda \in L(G')$ if and only if $S' \to \Lambda$ is in P' if and only if $S \in NULL$ if and only if $S \overset{*}{\underset{G}{\Rightarrow}} \Lambda$ if and only if $\Lambda \in L(G)$. Therefore we will now consider nonnull strings in the language.

The following claim is a first step in showing that $L(G) \subseteq L(G')$. It shows that each derivation in G which consists of one production $A \to \alpha$ followed by the derivation of Λ from any number of symbols in α, can be simulated by G'. For example consider all derivations in G' (Example 2.3.1) that corresponds to $A \to B_1aB_2B_3$ followed by some Λ-rules in P.

$$\begin{array}{lllll}
A_0 & \Rightarrow B_1A_1 & \Rightarrow B_1aB_2A_2 & \Rightarrow B_1aB_2B_3 \\
A_0 & \Rightarrow B_1A_1 & \Rightarrow B_1aA_2 & \Rightarrow B_1aB_3 \\
A_0 & \Rightarrow B_1A_1 & \Rightarrow B_1aB_2 & \\
A_0 & \Rightarrow B_1A_1 & \Rightarrow B_1a & \\
A_0 & \Rightarrow A_1 & \Rightarrow aB_2A_2 & \Rightarrow aB_2B_3
\end{array}$$

$$A_0 \Rightarrow A_1 \Rightarrow aA_2 \Rightarrow aB_3$$
$$A_0 \Rightarrow A_1 \Rightarrow aB_2$$
$$A_0 \Rightarrow A_1 \Rightarrow a$$

We now state the claim formally.

Claim 1. Let $A \to \alpha$ be in P, $\alpha = \alpha_0 B_1 \alpha_1 \cdots B_n \alpha_n$, $n \geq 1$, $B_j \in \text{NULL}$, $\alpha_j \in (V\text{-NULL})^*$ and let $A_j = A_j(A\to\alpha)$, $1 \leq j \leq n-1$ be the new symbols introduced into N'. Let $A = A_0$. For each $0 \leq i \leq n-1$ and for any set of indices $i < j_1 < j_2 < \cdots < j_t \leq n$, $0 \leq t \leq n-i$ define X_j, $i < j \leq n$ is defined to be

$$X_j = \begin{cases} B_j \ , & j \in \{j_1, j_2, \ldots, j_t\} \\ \Lambda \ , & \text{otherwise} \ , \end{cases}$$

and denote $\delta_i = \alpha_i X_{i+1} \alpha_{i+1} \cdots X_n \alpha_n$ then $A_i \overset{*}{\underset{G'}{\Rightarrow}} \delta_i$ provided $\delta_i \neq \Lambda$.

Proof of Claim 1. An induction on $n-i$.

Basis: $n - i = 1$. Then $i = n-1$, $0 \leq t \leq 1$ and there are only two possible ways to choose the index set.

Case 1: $t = 0$, we have the empty set and therefore $X_n = \Lambda$ and $\delta_{n-1} = \alpha_{n-1}\alpha_n$. If $\alpha_{n-1}\alpha_n \neq \Lambda$ then $m \geq n$ and the algorithm places $A_{n-1} \to \alpha_{n-1}\alpha_n$ in P'.

Case 2: $t = 1$, $j_1 = n$ and $X_n = B_n$, $\delta_{n-1} = \alpha_{n-1}B_n\alpha_n$. Clearly $A_{n-1} \to \alpha_{n-1}B_n\alpha_n$ is in P'.

Induction step: Assume the claim holds whenever $n-i < r$ for some r, $1 < r \leq n$. Now let $n - i = r$, $0 \leq i \leq n-2$.

Case 1: $\alpha_{i+1}X_{i+2}\cdots X_n\alpha_n = \Lambda$. Then $\delta_i = \alpha_i X_{i+1}$. It also follows that $\alpha_{i+1}\alpha_{i+2}\cdots\alpha_n = \Lambda$ so by definition $m \leq i+1$. We must consider the following subcases:

<u>Subcase a</u>: $\alpha_i X_{i+1} = \Lambda$. Then the claim is vacuous since $\delta_i = \Lambda$.

<u>Subcase b</u>: $X_{i+1} = B_{i+1}$. In this case $A_i \rightarrow \alpha_i B_{i+1}$ is in P' since $\max\{0,m-1\} \leq i \leq n-2$. Hence $A_i \underset{G_i}{\Rightarrow} \alpha_i B_{i+1} = \delta_i$.

<u>Subcase c</u>: $X_{i+1} = \Lambda$, $\alpha_i \neq \Lambda$. Then $m = i+1$ so that $1 \leq m = i+1 \leq n-1$ and we have $A_{m-1} \rightarrow \alpha_{m-1}$ in P'. Hence, $A_i \underset{G'}{\Rightarrow} \alpha_i = \delta_i$.

<u>Case 2</u>: $\alpha_{i+1}X_{i+2}\cdots X_n\alpha_n \neq \Lambda$. Then, by the induction hypothesis $A_{i+1} \overset{*}{\underset{G'}{\Rightarrow}} \alpha_{i+1}X_{i+2}\cdots X_n\alpha_n$. It remains to show, then, that $A_i \rightarrow \alpha_i X_i A_{i+1}$ is in P. Since $0 \leq i \leq n-2$, this is indeed true for the two subcases $X_i = B_i$ and $X_i = \Lambda$.

This completes the proof of Claim 1.

Claim 1 directly yields the following.

<u>Claim 2</u>. For all $A \rightarrow \alpha$ in P such that $\alpha = \alpha_0 B_1 \alpha_1 \cdots B_n \alpha_n$, $n \geq 0$, $B_j \in \text{NULL}$, $\alpha_j \in (\text{V-NULL})^*$ and any choice of $X_j \in \{B_j, \Lambda\}$, $1 \leq j \leq n$, $A \overset{*}{\underset{G'}{\Rightarrow}} \alpha_0 X_1 \cdots X_n \alpha_n$ unless $\alpha_0 X_1 \alpha_1 \cdots X_n \alpha_n = \Lambda$.

<u>Proof of Claim 2</u>. For $n \geq 1$ this is a direct application of the claim with $i = 0$ (recall $A_0 = A$). For $n = 0$ there is one choice namely α_0, and if $\alpha = \alpha_0 \neq \Lambda$ then $A \rightarrow \alpha$ is in P' (the clause $n=0$, $m=1$ in the algorithm).

Claim 2 dealt with a restricted form of derivations in G. We now generalize this result, by considering derivations $\gamma \overset{*}{\underset{G}{\Rightarrow}} \beta$ where γ is either of length one or contains no symbol in NULL.

<u>Claim 3</u>. Let $\gamma \in (\text{V-NULL})^* \cup \text{NULL}$, $\beta \in V^*$ where $\beta = \Lambda$ only if $\gamma = \Lambda$. If $\gamma \overset{*}{\underset{G}{\Rightarrow}} \beta$ then $\gamma \overset{*}{\underset{G'}{\Rightarrow}} \beta$.

Proof of Claim 3. The argument is a double induction, involving the length h of the derivation $\gamma \overset{*}{\underset{G}{\Rightarrow}} \beta$, and $\ell = |\gamma|$ over the well ordered set consisting of pairs (h,ℓ) with ordering described below.

Basis: Let $h = 0$ and $\ell \geq 0$. The claim holds trivially.

Inductive step: Assume, for some h_0 and ℓ_0 such that $h_0 > 0$, $\ell_0 \geq 0$, that the claim holds for all $(h,\ell) < (h_0,\ell_0)$ (i.e. for all h and ℓ such that either $h < h_0$ or $h = h_0$ and $\ell < \ell_0$).

Now let $\gamma \in (V\text{-NULL})^* \cup \text{NULL}$, $|\gamma| = \ell_0$ and $\beta \in V^*$ and suppose $\gamma \overset{h_0}{\underset{G}{\Rightarrow}} \beta$. If $\ell_0 = 0$ then clearly $\gamma = \beta = \Lambda$ and $\gamma \overset{\ell_0}{\underset{G'}{\Rightarrow}} \beta$. So assume $\ell_0 > 0$, $\gamma \neq \Lambda$ and $\beta \neq \Lambda$. We need to consider two cases.

Case 1: $\ell_0 = 1$. So $\gamma = A \in N$ since $\gamma \in \Sigma$ is not possible $(h_0 > 0)$. Factor the derivation as follows. $A \underset{G}{\Rightarrow} \alpha \overset{*}{\underset{G}{\Rightarrow}} \beta$ where $\alpha \neq \Lambda$, $A \rightarrow \alpha$ in P, $\alpha = \alpha_0 B_1 \cdots B_n \alpha_n$, $n \geq 0$, $B_j \in \text{NULL}$, $\alpha_j \in (V\text{-NULL})^*$. We can factor β accordingly: $\beta = \beta_0 \gamma_1 \beta_1 \cdots \gamma_n \beta_n$ such that $\alpha_j \overset{*}{\underset{G}{\Rightarrow}} \beta_j$ for all $0 \leq j \leq n$ and $B_j \overset{*}{\underset{G}{\Rightarrow}} \gamma_j$ for all $1 \leq j \leq n$. Let

$$X_j = \begin{cases} B_j & \text{if } \gamma_j \neq \Lambda \\ \Lambda & \text{if } \gamma_j = \Lambda \end{cases} \quad \text{for } 1 \leq j \leq n .$$

For $\gamma_j = \Lambda$ it is obvious that $X_j \overset{*}{\underset{G'}{\Rightarrow}} \gamma_j$. The rest of the derivations $\alpha_j \overset{*}{\underset{G}{\Rightarrow}} \beta_j$, $0 \leq j \leq n$ and $X_j = B_j \overset{*}{\underset{G}{\Rightarrow}} \gamma_j$, $1 \leq j \leq n$, $\gamma_j \neq \Lambda$ are all of length less than h_0, so we can apply the inductive hypothesis and conclude that $\alpha_j \overset{*}{\underset{G'}{\Rightarrow}} \beta_j$ and $X_j \overset{*}{\underset{G'}{\Rightarrow}} \gamma_j$. All these combine to $\alpha_0 X_1 \alpha_1 \cdots X_n \alpha_n \overset{*}{\underset{G'}{\Rightarrow}} \beta_0 \gamma_1 \cdots \beta_n \gamma_n = \beta$.

By Claim 2, and the fact that $\alpha_0 X_1 \cdots X_n \alpha_n \neq \Lambda$ (since $\beta \neq \Lambda$) $A \overset{*}{\underset{G'}{\Rightarrow}} \alpha_0 X_1 \alpha_1 \cdots X_n \alpha_n$ so that $A \overset{*}{\underset{G'}{\Rightarrow}} \beta$.

Case 2: $\ell_0 > 1$. We can write $\gamma = A\gamma'$ and $\beta = \alpha\beta'$ for some $A \in V\text{-NULL}$, $\gamma' \in (V\text{-NULL})^*$, and $\alpha, \beta' \in V^*$ such that $A \overset{h_1}{\underset{G}{\Rightarrow}} \alpha$ and

$\gamma' \overset{h_2}{\underset{G}{\Rightarrow}} \beta'$. Since $h_1 + h_2 = h_0$ and $h_1, h_2 \geq 0$ it must be the case that $h_1 \leq h_0$ and $|A| = 1 < \ell_0$ and also $h_2 \leq h_0$ and $|\gamma'| = \ell_0 - 1 < \ell_0$. So the induction hypothesis may be applied to obtain $A \overset{*}{\underset{G'}{\Rightarrow}} \alpha$ and $\gamma' \overset{*}{\underset{G'}{\Rightarrow}} \beta'$. Hence $\gamma = A\gamma' \overset{*}{\underset{G'}{\Rightarrow}} \alpha\beta' = \beta$.

This completes the proof of Claim 3.

The next claim dwells on the disjoint sets of productions placed in P' for every production in P. It shows that once we start using productions of one such set we must continue until G' has simulated one production in G (plus derivation of null substrings). This claim will go a long way in preparing for the proof that $L(G') \subseteq L(G)$.

Claim 4. Let $C \in N' - \{S'\}$, $\beta \in V^+$ such that $C \overset{+}{\underset{G'}{\Rightarrow}} \beta$. Then

either (1) $C \underset{G'}{\Rightarrow} \alpha \overset{*}{\underset{G'}{\Rightarrow}} \beta$ for some $\alpha \in V^+$ where $C \to \alpha$ is in P

or (2) $C = A_i(A \to \alpha)$ for some $A \to \alpha$ in P, $\alpha = \alpha_0 B_1 \alpha_1 \cdots B_n \alpha_n$, $n \geq 1$ where $B_j \in \text{NULL}$, $\alpha_j \in (V\text{-NULL})^*$, $0 \leq j \leq n-1$ and there exist $X_{i+1}, X_{i+2}, \ldots, X_n$ where $X_j \in \{B_j, \Lambda\}$ and $\alpha_i X_{i+1} \alpha_{i+1} \cdots X_n \alpha_n \neq \Lambda$ such that the derivation can be factored $C \overset{+}{\underset{G'}{\Rightarrow}} \alpha_i X_{i+1} \cdots X_n \alpha_n \overset{*}{\underset{G'}{\Rightarrow}} \beta$.

Proof of Claim 4. Consider $C \to \gamma$ in P', the production used in the first step of $C \overset{+}{\underset{G'}{\Rightarrow}} \beta$. If $C \to \gamma$ is in P then let $\gamma = \alpha \neq \Lambda$ and (1) must hold. Suppose $C \to \gamma$ is not in P. Then it was placed in P' when the main for loop was executed for some $A \to \alpha$ in P. Then there must exist some i, $0 \leq i \leq n-1$ so that $C = A_i(A \to \alpha)$. We proceed by an induction on $n-i$.

Basis: $n-i = 1$ then $i = n-1$. The only rules with A_{n-1} which may be in P' are $A_{n-1} \to \alpha_{n-1} B_n \alpha_n$ and $A_{n-1} \to \alpha_{n-1}\alpha_n$ (if $\alpha_{n-1}\alpha_n \neq \Lambda$).

One of these must therefore be used in the first step of the derivation $A_{n-1} \overset{+}{\underset{G'}{\Rightarrow}} \beta$ and if we choose $X_n = B_n$ or $X_n = \Lambda$ depending on the production, we get $A_{n-1} \underset{G'}{\Rightarrow} \alpha_{n-1}X_n\alpha_n \overset{*}{\underset{G'}{\Rightarrow}} \beta$.

<u>Induction step</u>: Assume the result true if $n-i < r$ where $1 < r \leq n$. Now let $n-i = r$, $0 \leq i \leq n-2$. The first step in the derivation $A_i \overset{+}{\underset{G'}{\Rightarrow}} \beta$ must employ one of the following productions:

<u>Case 1</u>: $A_i \rightarrow \alpha_i B_{i+1} A_{i+1}$ or $A_i \rightarrow \alpha_i A_{i+1}$ from P'. Then choose $X_{i+1} = B_{i+1}$ or $X_{i+1} = \Lambda$ respectively. $A_i \underset{G'}{\Rightarrow} \alpha_i X_{i+1} A_{i+1} \overset{+}{\underset{G'}{\Rightarrow}} \beta$ (the last step must be of length one or more since $\beta \in V^+$, $A_{i+1} \notin V$). Then we can write $\beta = \beta_1\beta_2$ such that $\alpha_i X_{i+1} \overset{*}{\underset{G'}{\Rightarrow}} \beta_1$ and $A_{i+1} \overset{+}{\underset{G'}{\Rightarrow}} \beta_2$. We apply the inductive hypothesis to this last derivation to conclude that $A_{i+1} \overset{+}{\underset{G'}{\Rightarrow}} \alpha_{i+1}X_{i+1}\cdots X_n\alpha_n \overset{*}{\underset{G'}{\Rightarrow}} \beta_2$. When this is used in the original derivation we obtain

$$A_i \underset{G'}{\Rightarrow} \alpha_i X_{i+1} A_{i+1} \overset{+}{\underset{G'}{\Rightarrow}} \alpha_i X_{i+1} \alpha_{i+1} \cdots X_n \alpha_n \overset{*}{\underset{G'}{\Rightarrow}} \beta_1 \beta_2 = \beta .$$

<u>Case 2</u>: $A_i \rightarrow \alpha_i B_{i+1}$ or $A_i \rightarrow \alpha_i$ in P' in which case $i \geq m-1$ or $i = m-1$ respectively. In any event, we conclude by the definition of m that $\alpha_{i+1}\alpha_{i+2}\cdots\alpha_n = \Lambda$ and if we choose $X_{i+1} = B_{i+1}$ or $X_{i+1} = \Lambda$ respectively, and in addition let $X_{i+2} = X_{i+3} = \cdots = X_n = \Lambda$ we get that $A_i \rightarrow \alpha_i X_{i+1} \alpha_{i+1} X_{i+2} \cdots X_n \alpha_n$ is this first production in the derivation so that indeed $A_i \underset{G'}{\Rightarrow} \alpha_i X_{i+1} \cdots X_n \alpha_n \overset{*}{\underset{G'}{\Rightarrow}} \beta$.

The proof of Claim 4 is now complete.

The following is a direct consequence of Claim 4.

Claim 5. Let $A \in N$, $\beta \in V^{+}$ and suppose $A \overset{+}{\underset{G'}{\Rightarrow}} \beta$. Then there exists $\alpha \in V^{+}$, $A \to \alpha$ is in P such that $\alpha = \alpha_0 B_1 \alpha_1 \cdots B_n \alpha_n$, $n \geq 0$, $B_j \in \text{NULL}$ and $\alpha_j \in (V\text{-NULL})^{*}$ and there exist $X_1, X_2, \ldots, X_n$ where $X_j \in \{B_j, \Lambda\}$ and $\alpha_1 X_1 \alpha_2 \cdots X_n \alpha_n \neq \Lambda$ such that the derivation can be factored $A \overset{+}{\underset{G'}{\Rightarrow}} \alpha_1 X_1 \cdots X_n \alpha_n \overset{+}{\underset{G'}{\Rightarrow}} \beta$.

Proof of Claim 5. Apply Claim 4. If case (1) holds then $A \to \alpha$ is in P and in P' so $\alpha \in (V\text{-NULL})^{+}$. Let $n = 0$ then $A \underset{G'}{\Rightarrow} \alpha \overset{+}{\underset{G'}{\Rightarrow}} \beta$ is the desired factorization. If (2) holds the result follows trivially.

Claim 5 will be used in proving the next result.

Claim 6. Let $\gamma \in V^{*}$, $\beta \in V^{*}$. If $\gamma \overset{*}{\underset{G'}{\Rightarrow}} \beta$ then $\gamma \overset{*}{\underset{G}{\Rightarrow}} \beta$.

Proof of Claim 6. An induction on h, the length of the derivation $\gamma \overset{*}{\underset{G'}{\Rightarrow}} \beta$.

Basis: $h = 0$ trivial.

Induction step: Assume the claim true for every $\gamma \in V^{*}$, $\beta \in V^{*}$, $\gamma \overset{h}{\underset{G'}{\Rightarrow}} \beta$ and $h < h_0$ where $h_0 > 0$.

Now let $\gamma \overset{h_0}{\underset{G'}{\Rightarrow}} \beta$, $\gamma \in V^{*}$, $\beta \in V^{*}$. Since $h_0 > 0$ we can write $\gamma = \gamma_1 A \gamma_2$, $\beta = \beta_1 \beta' \beta_2$ so that $A \in N$, γ_1, γ_2, β_1, $\beta_2 \in V^{*}$, $\beta' \in V^{+}$ and $\gamma_1 \overset{h_1}{\underset{G'}{\Rightarrow}} \beta_1$, $\gamma_2 \overset{h_2}{\underset{G'}{\Rightarrow}} \beta_2$, $A \overset{+}{\underset{G'}{\Rightarrow}} \beta'$. Using Claim 5 for the last derivation we get that for some $\alpha \in V^{+}$, $A \to \alpha$ in P, $\alpha = \alpha_0 B_1 \alpha_1 \cdots B_n \alpha_n$ there exist $X_1, X_2, \ldots, X_n$ where $X_j \in \{B_j, \Lambda\}$ for all j, $1 \leq j \leq n$ and $\alpha_1 X_1 \cdots X_n \alpha_n \neq \Lambda$ such that for some $h' > 0$, $A \overset{h'}{\underset{G'}{\Rightarrow}} \alpha_1 X_1 \cdots X_n \alpha_n \overset{h_3}{\underset{G'}{\Rightarrow}} \beta'$. Since $h' + h_1 + h_2 + h_3 = h_0$ it follows that $h_1 < h_0$, $h_2 < h_0$, and $h_3 < h_0$. Also $\alpha_1 X_1 \cdots X_n \alpha_n \in V^{*}$ so we can apply the induction hypothesis to $\gamma_1 \overset{h_1}{\underset{G'}{\Rightarrow}} \beta_1$, $\gamma_2 \overset{h_2}{\underset{G'}{\Rightarrow}} \beta_2$ and $\alpha_1 X_1 \cdots X_n \alpha_n \overset{h_3}{\underset{G'}{\Rightarrow}} \beta'$ and conclude that $\gamma_1 \overset{*}{\underset{G}{\Rightarrow}} \beta_1$, $\gamma_2 \overset{*}{\underset{G}{\Rightarrow}} \beta_2$ and $\alpha_1 X_1 \cdots X_n \alpha_n \overset{*}{\underset{G}{\Rightarrow}} \beta'$. Clearly, for all j, $1 \leq j \leq n$, $B_j \overset{*}{\underset{G}{\Rightarrow}} X_j$ (either by $B_j = X_j$ or $B_j \overset{*}{\underset{G}{\Rightarrow}} \Lambda$ as $B_j \in \text{NULL}$) therefore we can combine all the derivations and get $\gamma = \gamma_1 A \gamma_2 \overset{*}{\underset{G}{\Rightarrow}}$ $\gamma_1 \alpha_1 B_1 \cdots B_n \alpha_n \gamma_2 \overset{*}{\underset{G}{\Rightarrow}} \gamma_1 \alpha_1 X_1 \cdots X_n \alpha_n \gamma_n \overset{*}{\underset{G}{\Rightarrow}} \beta_1 \beta' \beta_2 = \beta$.

It is now a matter of combining the claims to complete the proof: We have already seen that $\Lambda \in L(G')$ if and only if $\Lambda \in L(G)$. Claims 3 and 6 applied to S and $w \in \Sigma^+$, and the knowledge that $S' \overset{*}{\underset{G'}{\Rightarrow}} S$ must start every derivation in G' of a non-null string, combine to prove that $w \in L(G')$ if and only if $w \in L(G)$. This completes the proof of the lemma. □

Lemma 2.3.3. The grammar $G' = (N' \cup \Sigma, \Sigma, P', S')$ obtained from $G = (N \cup \Sigma, \Sigma, P, S)$ by Algorithm 2.3.2 has size

$$|G'| = O(|G|)$$
$$|N'| = O(|G|)$$
$$\|G'\| = O(\|G\| \log \|G\|) .$$

Proof. For each $A \to \alpha$ in P a number of new nonterminals may be placed in N'. In particular, if $\alpha = \alpha_0 B_1 \alpha_1 \cdots B_n \alpha_n$, $B_j \in \text{NULL}$, $\alpha_j \in (V\text{-NULL})^*$ then $n-1$ new nonterminals are added.
But $|\alpha| \leq n$ (equality may be achieved when $\alpha_0 \alpha_1 \cdots \alpha_n = \Lambda$). So $|N'| \leq |N| + \sum_{A \to \alpha \text{ in } P} (|\alpha| - 1) \leq |N| + |G| \leq 2 \cdot |G|$.
Also $|V'| = |N'| + |\Sigma| \leq 2 \cdot |G| + |G|$, $|V'| = O(|G|)$.

To analyze $|G'|$ we examine the productions placed in P' for every $A \to \alpha$ in P. If $\alpha = \alpha_0 B_1 \cdots B_n \alpha_n$ then we can count the number of occurrences of α_j, B_j and A_j in these productions. Each may appear at most a constant number of times (at most 4) so the size of the relevant productions is bounded by $c'\cdot\left(\sum_{j=0}^{n}|\alpha_j| + \sum_{j=1}^{n}|B_j| + \sum_{j=1}^{n-1}|A_j|\right)$ $\leq c'(|\alpha|+n-1) \leq c'(|\alpha|+|\alpha|) \leq c''|A\alpha|$ where c' and c'' are constants. $|G'| \leq \left(\sum_{A\to\alpha\in P} c''|A\cdot\alpha|\right) + |S'\ S| + |S'\ \Lambda| \leq c|G|$.

Finally

$$\|G'\| = |G'|\log|V'| = \mathcal{O}(|G|\log|G|)$$

and since $|G| \leq \|G\|$

$$\|G'\| = \mathcal{O}(\|G\|\log\|G\|)\ . \qquad \square$$

Note that using the measure $|G|$ the output grammar is linear in the size of the input grammar, but many new nonterminals are introduced which may require more space for encoding each symbol.

This change is reflected in the fact that using $\|G\|$ as a measure the output grammar is of size $O(n \log n)$. In Example 2.3.2 above, even $\|G\|$ grew linearly, but this is not always the case.

The following example shows that the additional log in the bound for $\|G'\|$ in Lemma 2.3.3 is not a result of crude estimations.

Example 2.3.3. Choose any function f over the integers such that $f(k) \geq k$ for any $k \geq 1$. Let $G = (V,\Sigma,P,S)$, $N = \{S,A_1,\ldots,A_k\}$, $\Sigma = \{a_1,\ldots,a_k\}$, $P = \{S \to A_1^{f(k)}A_2^{f(k)}\ldots A_k^{f(k)}\} \cup \{A_j \to a_j,\ A_j \to \Lambda \mid 1 \leq j \leq k\}$. Applying Algorithm 2.3.2 yields $G' = (V',\Sigma,P',S')$ where $N' = \{S',\ S=S_0,S_1,\ldots,S_{k\cdot f(k)-1},A_1,\ldots,A_k\}$ and $P' = \{S' \to S,\ S' \to \Lambda\}$ $\cup \{A_j \to a_j \mid 1 \leq j \leq k\} \cup \{S_j \to A_{j'}S_{j+1},\ S_j \to S_{j+1},\ S_j \to A_{j'} \mid 0 \leq j \leq kf(k)-2,$ $j' = \left\lfloor \frac{j}{f(k)} \right\rfloor + 1\} \cup \{S_{kf(k)-1} \to A_k\}$.

Computing sizes we get $|V| = 2k+1$, $|G| = kf(k)+3k+1$ and $|V'| = k\cdot f(k)+2k+1$, $|G'| = 7k\cdot f(k)+2k-2$ so that there exist c_1, c_2 for all $k \geq 2$

$\|G\| = (k \cdot f(k)+3k+1)\log(2k+1) \leq c_1 \cdot k \cdot f(k) \cdot \log k$, $\|G\| \geq f(k)$ and $\|G'\| = (7k \cdot f(k)+2k-2)\log(kf(k)+2k+1) \geq c_2 kf(k)\log f(k)$.

To see how the gap between $\|G\|$ and $\|G'\|$ may grow define a family of functions as follows: for all $k \geq 1$, $f_0(k) = k$ and if $m \geq 0$, $f_{m+1}(k) = 2^{f_m(k)}$. So $f_m(k) = 2^{2^{\cdot^{\cdot^{2^k}}}}$ with m 2's. Then $\log f_m(k) = f_{m-1}(k)$ for all $m \geq 1$ and $\underbrace{\log\log\cdots\log}_{m+1} f_m(k) =$ $= \log \cdot f_0(k) = \log k$.

Now let $f = f_m$ for some $m \geq 1$ in the definition of G. We get that

$$f_m(k) \leq \|G\| \leq c_1 kf_m(k)\log k$$

$$\log\|G\| \leq \log(c_1 kf_m(k)\log k) \leq c_1' f_{m-1}(k)$$

and $$\|G'\| \geq c_2 kf_m(k)f_{m-1}(k)$$

for some constants c_1, c_2, c_1'. So

$$\|G'\| \geq c_2 kf_m f_{m-1}(k) = \frac{c_2}{c_1 c_1'} \cdot \frac{c_1 kf_m(k)\log k \cdot c_1' f_{m-1}(k)}{\log k}$$

$$\geq \frac{c_2}{c_1 c_1'} \cdot \frac{\|G\|\log\|G\|}{\underbrace{\log\log\cdots\log}_{m+1} f_m(k)} \geq c \frac{\|G\|\log\|G\|}{\underbrace{\log\cdots\log}_{m+1}\|G\|} .$$

And the size $\|G'\|$ of the output grammar is more than $\dfrac{cn\log n}{\underbrace{\log\log\cdots\log}_{m+1} n}$ where $\|G\| = n$.

Since m may be chosen at will the upper bound on $\|G'\|$ may be arbitrarily close to $cn\log n$.

So far we have not discussed the computation of NULL. The algorithms in the literature use the nested set construction:

$$W_0 = \emptyset$$

$$\text{for} \quad i \geq 0 \qquad W_{i+1} = W_i \cup \{A \in N | A \to \alpha \in P \text{ for some } \alpha \in W_i^*\}$$

and we can let NULL = $W_{|N|}$.

A result similar to Lemma 2.2.1 can be proved, and the complexity of this construction shown to be $O(\frac{\|G\|^2}{\log\|G\|})$.

As in the case of GEN and REACH we can do better. In fact we can exploit the similarity between NULL and GEN and use Algorithm 2.2.2 as a subroutine. (A similar idea was suggested by Tim Winkler [1977], and in Hunt, Rosenkrantz and Szymanski [1976]).

Algorithm 2.3.3.

Input: $G = (V,\Sigma,P,S)$

Output: $\text{NULL} = \{A \in N | A \overset{*}{\Rightarrow} \Lambda\}$

```
begin
$\bar{G} := (V,\emptyset,P,S)$;
apply Algorithm 2.2.2 to $\bar{G}$, obtaining $\overline{\text{GEN}}$;
NULL := $\overline{\text{GEN}}$
end.
```

Example 2.3.4. Let $G = (V,\Sigma,P,S)$, $\Sigma = \{a,b\}$, $V = \{S,A,B,C\} \cup \Sigma$ and P contains $S \to aAB$, $A \to BC$, $B \to a|\Lambda$, $C \to BB|b$. $\bar{G}$ has the same productions but has nonterminal set $\bar{N} = V$ and $\bar{\Sigma} = \emptyset$. Computing $\overline{\text{GEN}}$ we obtain $\overline{\text{GEN}} = \{B,C,A\}$ so NULL = $\{B,C,A\}$.

Lemma 2.3.4. Algorithm 2.3.3 computes NULL in linear time.

Proof. By the correctness of Algorithm 2.2.2, Algorithm 2.3.3 computes

$$\text{NULL} = \overline{\text{GEN}} = \{A \in V | \exists x \in \emptyset^*, A \underset{\bar{G}}{\overset{*}{\Rightarrow}} x\} = \{A \in V | A \underset{G}{\overset{*}{\Rightarrow}} \Lambda\} = \{A \in N | A \underset{G}{\overset{*}{\Rightarrow}} \Lambda\}$$

which is the desired set.

Algorithm 2.3.3 operates in linear time since Algorithm 2.2.2 does, by Lemma 2.2.4. Note that $|\bar{G}| = |G|$ and $\|\bar{G}\| = \|G\|$. □

We can now present the main result of this section, which considers the combination of Algorithms 2.3.2 and 2.3.3.

Theorem 2.3.1. There is an algorithm that performs null rules elimination on any grammar $G = (V,\Sigma,P,S)$ in time $O(n \log n)$ $(O(n))$ if the size measure is $\|G\|$ $(|G|)$ respectively.

Proof. Follows from Lemmas 2.3.2, 2.3.3 and 2.3.4 and the observation that the time spent by Algorithm 2.3.2 (besides computing NULL) is dominated by the size of the output grammar. □

A polynomial time algorithm for eliminating null rules has been independently obtained by Hunt, Rosenkrantz and Szymanski [1976]. Their algorithm runs in times $O(n^2 \log n)$ or $O(n^2)$ depending on the size measure.

Section 2.4 - Eliminating Chains

It is sometimes an advantage to have grammars with no rules of the form $A \to B$, $A, B \in N$. Rules of this form can result in very long derivations for short strings. In grammars for programming languages such rules often exist with no semantic significance.

Definition 2.4.1. A grammar $G = (V,\Sigma,P,S)$ is said to be chain-free if $P \cap N \times N = \emptyset$.

We present a slightly modified version of the "classical" algorithm for the elimination of chain rules. Here, for each $B \in N$, CHAIN(B) represents the set of all nonterminals A generating B (rather than generated by B). Also, we remove the requirement that the original grammar be Λ-free.

Algorithm 2.4.1.

Input: $G = (V,\Sigma,P,S)$ a reduced grammar

Output: grammar G' such that $L(G') = L(G)$ and G' is chain free

```
begin
for all $B \in N$ do CHAIN(B) := $\{A \in N \mid A \overset{*}{\Rightarrow} B\}$;
P' := $\emptyset$;
for all $B \to \alpha \in P$ do
   if $\alpha \notin N$ then for all $A \in$ CHAIN(B) do P' := P' $\cup \{A \to \alpha\}$;
G := $(V,\Sigma,P',S)$
end.
```

Before we discuss the computation of the sets CHAIN(B), we state the correctness of this algorithm.

Lemma 2.4.1. Algorithm 2.4.1 correctly computes a grammar G' such that $L(G') = L(G)$ and G' is chain free. Moreover if G is Λ-free then so is G'.

Proof. That G' is chain-free is easily seen by inspection. It is also easy to verify that $L(G') = L(G)$, cf. Harrison [1978].

If G is Λ-free then the only null rule which may be present in P is $S \to \Lambda$. Since S never appears in the right hand side of a production $\mathrm{CHAIN}(S) = \{S\}$, and as a result P' may only include one null rule, namely $S \to \Lambda$. □

To examine the size of the grammars produced by the algorithm we first consider an example.

Example 2.4.1. Let $G = (V,\Sigma,P,A_1)$ where $N = \{A_1,A_2,\ldots,A_k\}$, $\Sigma = \{a_1,a_2,\ldots,a_k\}$ and $P = \{A_i \to A_{i+1}, A_i \to a_iA_{i+1} \mid 1 \le i < k\} \cup \{A_k \to a_k\}$. G is Λ-free. Apply Algorithm 2.4.1 to G. We get, for $1 \le i \le k$, $\mathrm{CHAIN}(A_i) = \{A_j \mid 1 \le j \le i\}$, and therefore $P' = \{A_j \to a_iA_{i+1} \mid 1 \le j \le i < k\} \cup \{A_j \to a_k \mid 1 \le j \le k\}$, and $G' = (V,\Sigma,P',A_1)$.

Computing the sizes of these grammars we get $|V| = 2k$, $|G| = 5k-3$ whereas

$$|G'| = \sum_{i=1}^{k-1} 3\cdot i + 2\cdot k = \frac{3k^2+k}{2}\ .$$

We have $|V| = \theta(k)$, $|G| = \theta(k)$, $|G'| = \theta(k^2)$. So $|G'| = \theta(|G|^2)$. Also $\|G\| = |G|\log|V| = \theta(k \log k)$ and $\|G'\| = |G'|\log|V| = \theta(k^2 \log k)$. Note that $\log\|G\| = \theta(\log(k \log k)) = \theta(\log k + \log\log k) = \theta(\log k)$. Hence

$$\|G'\| = \theta(k^2 \log k) = \theta(\frac{(k \log k)^2}{\log k}) = \theta(\frac{\|G\|^2}{\log\|G\|})\ .$$

We can make the following general observation with respect to size:

Lemma 2.4.2. Algorithm 2.4.1, when applied to a grammar $G = (V,\Sigma,P,S)$ produces a grammar G' such that $|G'| = O(|G|^2)$, and $\|G'\| = O(\frac{\|G\|^2}{\log\|G\|})$. These bounds may be achieved.

Proof. For each $B \to \alpha \in P$, P' may include as many as $|\mathrm{CHAIN}(B)|$ productions of the same size. Since $\mathrm{CHAIN}(B) \subseteq N$ we have $|G'| \leq |N|\cdot|G|$ and by Lemma 2.1.1

$$|G'| \leq |G|^2 .$$

Since the alphabet for the output grammar is the same as that of the input grammar we get

$$\begin{aligned}\|G'\| &= |G'|\cdot\log|V| \leq |G|\cdot|V|\cdot\log|V| \\ &= \|G\|\cdot|V| \quad .\end{aligned}$$

But by Lemma 2.11 $|V| \leq \frac{2\|G\|}{\log\|G\|}$ so $\|G'\| = O(\frac{\|G\|^2}{\log\|G\|})$.

That the bounds are reached is evident in Example 2.4.1. □

Assume that the input grammar is Λ-free. This simplifies the computation of $\mathrm{CHAIN}(B)$ for $B \in N$, since it implies that the derivation $A \overset{*}{\Rightarrow} B$ must include only chain rules. In fact we are interested in the reflexive transitive closure R^* of the relation $R \subseteq N \times N$ where $(B,A) \in R$ if and only if $A \to B$ is in P.

Generally speaking we need to compute this reflexive transitive closure in as little time as a function of $|G|$ (or $\|G\|$) as possible. Note that $|G|$ is a good measure of the number of elements in R, i.e. the number of nonzero elements in a matrix representation of the relation R, whereas $|N|$ is the size of this matrix. This implies that we will choose techniques that work well for sparse relations (graphs, matrices): When $|G| = O(|N|)$ the general methods will require time $O(|G|^3)$ or $O(|G|^{2.81})$, whereas sparse representation will enable computation in time $O(|G|^2)$, cf. Aho, Hopcroft and Ullman [1974] for a discussion of various transitive closure algorithms.

One may hope to show that chain rule elimination is at least as hard as computing the closure. For this it is necessary to show that an algorithm for chain rule elimination can be used to compute the closure of any relation (or a directed graph, a boolean matrix). This does not seem to work because of the different data representation in the two problems. If we let a grammar represent some relation by its chain rules, and apply a chain eliminating algorithm, we will need to compute the language generated by the output grammar in order to get the transitive closure.

We shall address ourselves to the issue of a lower bound on the size of a chain free grammar (and thus a lower bound on the time to perform chain elimination) in the next section when we will be able to make some strong assumptions on the form of the output grammar.

Section 2.5 - Chomsky Normal Form and Related Forms

In this section we discuss a number of canonical forms which insist on "short" right hand sides.

We start with the following definition (from Hunt, Szymansky and Ullman [1975]).

Definition 2.5.1. A grammar $G = (V,\Sigma,P,S)$ is said to be in 2-Normal-Form (2NF) if $P \subseteq N \times V_{\Lambda}^{2}$.

The only restriction for 2NF grammars is that the right hand sides be of size at most 2. The next form, however, is slightly more restrictive.

Definition 2.5.2. A grammar $G = (V,\Sigma,P,S)$ is said to be in Chomsky-Normal-Form (CNF) if

(1) $P \subseteq N \times (N^2 \cup \Sigma) \cup \{S \rightarrow \Lambda\}$

and (2) if $S \rightarrow \Lambda$ is in P then S does not appear in the right hand side of any production.

So that CNF restricts the productions to three types: $A \rightarrow BC$, $A \rightarrow a$ or $S \rightarrow \Lambda$ where A, B, $C \in N$ and $a \in \Sigma$. We would like an additional normal form which will be intermediate in nature.

Definition 2.5.3. A grammar $G = (V,\Sigma,P,S)$ is said to be in Strict 2-Normal Form (S2NF) if $P \subseteq N \times (N^2 \cup V \cup \{\Lambda\})$.

Harrison [1978] defines two other related forms which we do not discuss here: canonical two form and binary standard form.

Fact. Every CNF grammar is in S2NF, every S2NF grammar is in 2NF.

The following is a simple algorithm to convert any grammar to 2NF. This is done by factoring "long" right hand sides and introducing new nonterminals where necessary.

Algorithm 2.5.1.

Input: $G = (V,\Sigma,P,S)$ a reduced grammar

Output: a grammar G' in 2NF such that $L(G') = L(G)$

```
begin
N' := N;
P' := ∅;
for all $A \to \alpha \in P$ do
   begin
   if $|\alpha| \leq 2$ then $P' := P' \cup \{A \to \alpha\}$
              else begin
                   comment $\alpha = X_1 X_2 \cdots X_r$, $r \geq 3$, $X_i \in V$;
                   for i := 1 to r-2 do
                      begin
                      $N' := N' \cup \{C_i(A \to \alpha)\}$;
                      comment abbreviate $C_i$, let $C_0 = A$;
                      $P' := P' \cup \{C_{i-1} \to X_i C_i\}$
                      end;
                   $P' := P' \cup \{C_{r-2} \to X_{r-1} X_r\}$
                   end
   end;
$G' := (N' \cup \Sigma, \Sigma, P', S)$
end.
```

Lemma 2.5.1. Algorithm 2.5.1, when applied to $G = (V,\Sigma,P,S)$ correctly produces an equivalent grammar G' in 2NF. Moreover if G is Λ-free (chain free) then so is G'.

Proof. It is easy to see that P' does not contain any production with a right hand side longer than 2.

That $L(G') = L(G)$ can be seen as a result of the following claims.

Claims. For all $A \in N$, $\alpha, \beta \in V^*$,

1) if $A \rightarrow \alpha \in P$ then $A \overset{*}{\underset{G'}{\Rightarrow}} \alpha$

2) if $A \overset{*}{\underset{G}{\Rightarrow}} \beta$ then $A \overset{*}{\underset{G'}{\Rightarrow}} \beta$

3) Let $A \rightarrow \alpha$ be in P, $\alpha = X_1X_2 \cdots X_r$ for some $X_1, X_2, \ldots, X_r \in V$ and let $C_i = C_i(A\rightarrow\alpha)$ be in N' for some i, $0 \leq i \leq r-2$. If $C_i \overset{+}{\underset{G'}{\Rightarrow}} \beta$ then this derivation can be factored $C_i \overset{+}{\underset{G'}{\Rightarrow}} X_{i+1} \cdots X_r \overset{*}{\underset{G'}{\Rightarrow}} \beta$.

4) if $A \overset{*}{\underset{G'}{\Rightarrow}} \beta$ then $A \overset{*}{\underset{G}{\Rightarrow}} \beta$

The proof of the claims is similar to, though much simpler than, the proof of the claims in Lemma 2.3.2:

Claim 1 can be proved by induction on $|\alpha|$, Claim 2 by induction on the length of the derivation, Claim 3 by induction on $r - i = |\alpha| - i$ (with basis $r-i = 2$), and Claim 4 by induction on the length of the derivation.

Finally, productions with right hand side 0 or 1 are included in P' only if they are in P so that the algorithm does preserve Λ-freeness and chain-freeness. □

Lemma 2.5.2. Algorithm 2.5.1 yields a grammar G' whose size depends upon that of G as follows:

$$|G'| \leq 3|G|$$

$$|N'| \leq |N| + |G|$$

$$\|G'\| = O(\|G\| \log \|G\|)$$

The time complexity of the algorithm is dominated by the size of the output.

Proof. For every $A \to \alpha \in P$, where $|\alpha| = r$ (and the production contributes $r+1$ to $|G|$), either $A \to \alpha \in P'$ (if $r \leq 2$) or else we get $r-1$ productions ($C_i \to X_{i+1}C_{i+1}$, $0 \leq i < r-2$, $C_{r-2} \to X_{r-1}X_r$) in P'. In this latter case we also add $r-2$ new nonterminals to $N' - N$.

Hence

$$|G'| = \sum_{\substack{A\to\alpha\in P\\ r=|\alpha|\leq 2}} |A\alpha| + \sum_{\substack{A\to\alpha\in P\\ r=|\alpha|>2}} 3(|A\alpha| - 2)$$

so that
$$|G'| \leq \sum_{A\to\alpha\in P} 3|A\alpha| = 3|G|$$

also
$$|N'| = |N| + \sum_{\substack{A\to\alpha\in P\\ |\alpha|>2}} (|A\alpha| - 3) \leq |N| + |G|$$

we have $|G'| = O(|G|)$ and $|V'| = O(|G|)$ so the bound for $\|G'\|$ is obtained in the same way as was obtained in Lemma 2.3.3.

The statement about time complexity is obvious as there is virtually no computation done. □

The difference between S2NF and 2NF is only in productions with right hand sides of length 2, where S2NF insists that the right hand side be in N^2.

The following is a trivial algorithm to transform a 2NF grammar to S2NF:

Algorithm 2.5.2.

Input: $G = (V,\Sigma,P,S)$, a grammar in 2NF

Output: G', a S2NF grammar such that $L(G') = L(G)$

```
begin
N' := N;
P' := ∅;
for all $A \to \alpha \in P$ do
    if $|\alpha| < 2$ then $P' := P' \cup \{A \to \alpha\}$
              else begin
                   comment $\alpha = X_1X_2,\ X_1, X_2 \in V$;
                   for i = 1,2 do if $X_i \in \Sigma$ then begin
                                                          $N' := N' \cup \{\bar{X}_i\}$;
                                                          $P' := P' \cup \{\bar{X}_i \to X_i\}$;
                                                          $B_i := \bar{X}_i$
                                                          end
                                                     else $B_i := X_i$;
                   $P' := P' \cup \{A \to B_1B_2\}$
                   end;
G' := $(N' \cup \Sigma, \Sigma, P', S)$
end.
```

Lemma 2.5.3. Algorithm 2.5.2 applied to a 2NF grammar G, produces G', a grammar in S2NF so that $L(G) = L(G')$ and the sizes are

$$|G'| \leq |G| + 2|\Sigma| \leq 3|G|$$

$$|V'| \leq |V| + |\Sigma| \leq 2|V|$$

$$\|G'\| \leq 6\|G\|$$

Moreover, the algorithm preserves Λ-freeness and chain-freeness.

Proof. It is easy to verify that G' is in S2NF, and is equivalent to G. For each production in P we place one having the same size in P', and beyond that we may add productions of type $\bar{a} \to a$ for each $a \in \Sigma$, adding as we do $\bar{a}$ to N'. This yields the first two bounds. For the third we note that

$$\|G'\| = |G'|\log|V'| \leq 3|G|\log(2|V|) \leq 6|G|\log|V| = 6\|G\|$$

where the fact that $|V| \geq 2$ was used. The preservation of Λ-freeness and chain-freeness is evident just as in Lemma 2.5.1. □

To transform a S2NF grammar to one in CNF we may first eliminate null rules, and then eliminate chains. We can recall our remark from section 2.3 that the classical algorithm (2.3.1) for null rule elimination works efficiently when the size of the right hand sides is bounded, and conclude that here we can use it (instead of Algorithm 2.3.2 which is usually more efficient) to do the job (provided NULL is computed by Algorithm 2.3.3).

We have established one algorithm (which is composed of several elementary algorithms) to transform any reduced grammar to CNF, namely:

Algorithm 2.5.3.

Input: $G = (V,\Sigma,P,S)$ a reduced grammar

Output: G', a CNF grammar such that $L(G') = L(G)$

```
begin
apply algorithm 2.5.1 to G, obtaining $G_1$ in 2NF
apply algorithm 2.5.2 to $G_1$, obtaining $G_2$ in S2NF
apply algorithm 2.3.2 (or 2.3.1) to $G_2$, obtaining $G_3$, a $\Lambda$-free
     grammar in S2NF
apply algorithm 2.4.1 to $G_3$, obtaining G' in CNF
end.
```

Another way to obtain the same result is

Algorithm 2.5.4.

Input: $G = (V,\Sigma,P,S)$ a reduced grammar

Output: G', a CNF grammar such that $L(G') = L(G)$

```
begin
apply algorithm 2.3.2 to G, obtaining $G_1$, $\Lambda$-free
apply algorithm 2.4.1 to $G_1$, obtaining $G_2$, $\Lambda$-free and chain free
apply algorithm 2.5.1 to $G_2$, obtaining $G_3$, $\Lambda$-free, chain free
    and in 2NF
apply algorithm 2.5.2 to $G_3$, obtaining G' in CNF
end.
```

The only remark we need to make in order to clarify the behavior of this algorithm is that by definition any S2NF grammar which is Λ-free and chain free is in CNF.

Theorem 2.5.1. There exists an algorithm that takes any reduced grammar $G = (V,\Sigma,P,S)$ and outputs an equivalent CNF grammar $G' = (V',\Sigma,P',S')$ such that

$$|G'| = O(|G|^2)$$
$$|V'| = O(|G|)$$
$$\|G'\| = O\left(\frac{\|G\|^2}{\log\|G\|}\right) .$$

The time needed to do this is dominated by the size of the output.

Proof. If we use Algorithm 2.5.3 correctness follows from Lemma 2.5.1, Lemma 2.5.3, Lemma 2.3.2 and Lemma 2.4.1. As for the bounds we obtain directly the following:

$|G_1| = O(|G|)$ and $|V_1| = O(|G|)$ from Lemma 2.5.2

$|G_2| = O(|G_1|)$ and $|V_2| = O(|V_1|)$ from Lemma 2.5.3

$|G_3| = O(|G_2|)$ and $|V_3| = O(|G_2|)$ from Lemma 2.3.3

$|G'| = O(|G_3|^2)$ and $|V'| = |V_3|$ from Lemma 2.4.2

so that
$$|G'| = O(|G|^2)$$
$$|V'| = O(|G|)$$

and it follows that $\|G\| = O(\frac{\|G\|^2}{\log\|G\|})$. The statement about the time complexity also follows similarly. □

A similar result can be obtained using Algorithm 2.5.4 (there we get $|V'| = O(|G|^2)$, which does not change $\|G'\|$).

Corollary. There exists an algorithm that takes any reduced grammar $G = (V,\Sigma,P,S)$ and outputs an equivalent (Λ-free) S2NF grammar $G'' = (V'',\Sigma,P'',S)$ so that

$$|G''| = O(|G|)$$
$$|V''| = O(|G|)$$
$$\|G''\| = O(\|G\|\log\|G\|) .$$

Proof. Use an initial part of Algorithm 2.5.3 and take G'' to be G_2 (G_3). □

We see that the increase in size in Theorem 2.5.1 is mainly due to the elimination of chain rules.

Example 2.5.1. We return to the grammar from Example 2.3.1: for $k > 1$ let $G = (V,\Sigma,P,A)$, $N = \{A,B_1,\ldots,B_k\}$, $\Sigma = \{a_1,\ldots,a_k\}$, $P = \{A \to B_1B_2\cdots B_k\} \cup \{B_i \to a_i,\ B_i \to \Lambda \mid 1 \leq i \leq k\}$.

We apply Algorithm 2.5.3 step by step: $G_1 = (V_1,\Sigma,P_1,A_0)$, $N_1 = \{A_0,A_1,\ldots,A_{k-2},B_1,\ldots,B_k\}$, $P_1 = \{A_i \to B_{i+1}A_{i+1} \mid 0 \le i < k-2\} \cup \{A_{k-2} \to B_{k-1}B_k\} \cup \{B_i \to a_i,\ B_i \to \Lambda \mid 1 \le i \le k\}$.

$G_2 = G_1$ (since all rules with right hand side of length 2 are in $N_1 \times N_1^2$).

We now use Algorithm 2.3.1 for null rule elimination:

$G_3 = (V_3,\Sigma,P_3,A')$, $N_3 = \{A',A_0,\ldots,A_{k-2},B_1,\ldots,B_k\}$,
$P_3 = \{A' \to A_0,\ A' \to \Lambda\} \cup \{A_i \to B_{i+1}A_{i+1} \mid 0 \le i < k-2\} \cup \{A_{k-2} \to B_{k-1}B_k\}$
$\cup \{A_i \to B_{i+1} \mid 0 \le i \le k-2\} \cup \{A_i \to A_{i+1} \mid 0 \le i < k-2\} \cup \{A_{k-2} \to B_k\} \cup \{B_i \to a_i \mid 1 \le i \le k\}$. (Note that NULL $= N_1$).

For convenience let $A' = A_{-1}$, compute $\mathrm{CHAIN}(A_i) = \{A_j \mid -1 \le j \le i\}$ for $-1 \le i \le k-2$, $\mathrm{CHAIN}(B_i) = \{B_i\} \cup \{A_j \mid -1 \le j < \min(i,k-1)\}$ for $1 \le i \le k$.

So that $G' = (V',\Sigma,P',A_{-1})$, $V' = V_3$ and $P' = \{A_{-1} \to \Lambda\}$
$\cup \{A_j \to B_{i+1}A_{i+1} \mid -1 \le j \le i < k-2,\ i \ge 0\} \cup \{A_j \to B_{k-1}B_k \mid -1 \le j \le k-2\}$
$\cup \{A_j \to a_i \mid -1 \le j < \min(i,k-1),\ 1 \le i \le k\} \cup \{B_i \to a_i \mid 1 \le i \le k\}$.

The sizes of the grammars are

$$|G| = 4k+1\ , \qquad |V| = 2k+1$$
$$|G_1| = 6k-3\ , \qquad |V_1| = 3k-1$$
$$|G_3| = 9k-4\ , \qquad |V_3| = 3k$$
$$|G'| = 1 + \sum_{i=0}^{k-3} 3(i+2) + 3k + \sum_{i=1}^{k-1} 2(i+1) + 2k + 2k$$
$$|G'| = \frac{5}{2}k^2 + \frac{13}{2}k - 4 \qquad |V'| = 3k$$

The following lemma sets the stage for an attempt to get a lower bound on the time complexity of chain elimination.

CNF grammars are more restricted than chain-free and Λ-free grammars. Nevertheless any lower bound on the size of a CNF grammar for a certain language yields a lower bound for the size of any chain-free, Λ-free grammar for that language since these two sizes are linearly related.

More precisely

Lemma 2.5.4. Let L be any language and let $n > 0$. If for any grammar G in CNF such that $L(G) = L$, $|G| \geq n$ then for any chain-free, Λ-free grammar G' such that $L(G) = L$, $|G| \geq \frac{1}{9}n$.

Proof. Let L be any language, $n > 0$ and all CNF grammars G for L have $|G| \geq n$. Assume, for the sake of contradiction that G' is a chain-free, Λ-free grammar, $L(G') = L$ and $|G'| < \frac{1}{9}n$. Apply Algorithm 2.5.1 followed by Algorithm 2.5.2. We obtain a grammar G'' in S2NF which is chain-free and Λ-free and hence in CNF. $L(G'') = L$ and, by Lemma 2.5.2 and Lemma 2.5.3, $|G''| \leq 3 \cdot 3 \cdot |G'| < 3 \cdot 3 \cdot \frac{1}{9} \cdot n$ so that $|G''| < n$ which contradicts the hypothesis of the lemma. □

We shall now present a conjecture about a certain family of languages.

Conjecture. For $k \geq 1$, let $L_k = \{a_i b_j | 1 \leq i \leq j \leq k\}$. There exists a constant c such that for every $k \geq 1$ and every CNF grammar G with $L(G) = L_k$, $|G| \geq c \cdot k \log k$.

Discussion. We now present three (families of) grammars for $L = L_k$ (we drop the subscript where possible). The first is not chain free. $G = (V, \Sigma, P, S)$, $\Sigma = \{a_i, b_i | 1 \leq i \leq k\}$,

$N = \{S\} \cup \{A_i, B_i \mid 1 \le i \le k\}$, $P = \{S \to A_i B_i,\ A_i \to a_i,\ B_i \to b_i \mid 1 \le i \le k\} \cup$ $\cup \{B_i \to B_{i+1} \mid 1 \le i < k\}$. It is easy to verify that $L(G) = L$, $|N| = 2k+1$ and $|G| = 9k-2$. Note that the longest chain is $B_1 \Rightarrow B_2 \Rightarrow \cdots \Rightarrow B_k$, so the sets $\mathrm{CHAIN}(B_i)$ of Section 2.4 are large, e.g. $\mathrm{CHAIN}(B_k) =$ $= \{B_1, \dots, B_k\}$, $|\mathrm{CHAIN}(B_k)| = k = \theta(|G|)$. It is not surprising then that an application of Algorithm 2.4.1 to eliminate chains yields grammar $G' = (V, \Sigma, P', S)$, $P' = \{S \to A_i B_i,\ A_i \to a_i \mid 1 \le i \le k\} \cup \{B_i \to b_j \mid 1 \le i \le j \le k\}$ which is CNF and whose size is $|G'| = k^2 + 6k = \theta(k^2)$.

There is a CNF grammar for L that is asymptotically smaller than G'. The idea is that if we are allowed chains we can make them shorter by arranging them as a balanced binary tree, thus limiting the length of a chain to $\log|N| \le \log|G|$ so that when we eliminate the chains the size of the grammar becomes $\theta(k \log k)$.

We define a family of grammars recursively. First we let $N_1'' = \{C_1, D_1\}$, $\Sigma_1 = \{a_1, b_1\}$, $P_1^1 = \{S_1'' \to C_1 D_1\}$ and $P_1^2 = \{C_1 \to a_1,$ $D_1 \to b_1\}$. We define $G_1'' = (\{S_1''\} \cup N_1'' \cup \Sigma_1, \Sigma_1, P_1^1 \cup P_1^2, S_1'')$. Note that N_1'' denotes all the nonterminals except S_1'' and that we grouped the productions into those with a pair of nonterminals in the right hand side $(S_1'' \to C_1 D_1)$, and those with a single terminal $(C_1 \to a_1,\ D_1 \to b_1)$.

Suppose G_k'' is defined for some $k \ge 1$. We make the following assumptions (which are easily verified for $k = 1$): $G_k'' = (V_k'', \Sigma_k, P_k^1 \cup P_k^2, S_k'')$ where $V_k'' = \{S_k''\} \cup N_k'' \cup \Sigma_k$, $\Sigma_k = \{a_i, b_i \mid 1 \le i \le k\}$, $P_k^1 \subseteq \{S_k''\} \times N_k''$, $P_k^2 \subseteq N_k'' \times \Sigma_k$ and $L(G_k'') = L_k = \{a_i b_j \mid 1 \le i \le j \le k\}$.

Define a set of new symbols $\bar{N}_k'' = \{\bar{A} \mid A \in N_k''\}$. Also let $\bar{S}_k''$ be a new symbol, let $\bar{\Sigma}_k = \{a_i, b_i \mid k+1 \le i \le 2k\}$ and $\bar{V}_k'' = \{\bar{S}_k''\} \cup \bar{N}_k'' \cup \bar{\Sigma}_k$. Define an injection $h\colon V_k'' \to \bar{V}_k''$ as follows. For all A in $\{S_k''\} \cup N_k''$ let $h(A) = \bar{A}$. For all i, $1 \le i \le k$, $h(a_i) = a_{i+k}$ and $h(b_i) = b_{i+k}$.

Extend h in the usual way to be a homomorphism from $(V''_k)^*$ into $(\bar{V}''_k)^*$ and let $\bar{G}''_k = (\bar{V}''_k, \bar{\Sigma}_k, \bar{P}^1_k \cup \bar{P}^2_k, \bar{S}''_k)$ where $\bar{P}^1_k = h(P^1_k)$ and $\bar{P}^2_k = h(P^2_k)$. Then $\bar{L}_k = L(\bar{G}''_k) = \{a_i b_j \mid k+1 \leq i \leq j \leq 2k\}$.

We now use G''_k and $\bar{G}''_k$ to define G''_{2k}. We let $N''_{2k} = N''_k \cup \bar{N}''_k \cup \{C_{2k}, C_{2D}\}$, $P^1_{2k} = \{S''_{2k} \to C_{2k}D_{2k}\} \cup \{S''_{2k} \to EF \mid S''_k \to EF$ is in P^1_k or $\bar{S}''_k \to EF$ is in $\bar{P}^1_k\}$, $P^2_{2k} = \{C_{2k} \to a_i \mid 1 \leq i \leq k\} \cup \{D_{2k} \to b_j \mid k+1 \leq j \leq 2k\} \cup$ $\cup\, P^2_k \cup \bar{P}^2_k$. We let $V''_{2k} = \{S''_{2k}\} \cup N''_{2k} \cup \Sigma_{2k}$ and $G''_{2k} = (V''_{2k}, \Sigma_{2k}, P^1_{2k} \cup P^2_{2k}, S''_{2k})$.

We would like to show that the inductive assumptions on properties of G''_k extend to G''_{2k}. The only property that does not follow trivially is that $L(G''_{2k}) = L_{2k}$. Before we prove it, we characterize L_{2k}:

$$\begin{aligned} L_{2k} &= \{a_i b_j \mid 1 \leq i \leq j \leq 2k\} \\ &= \{a_i b_j \mid 1 \leq i \leq j \leq k\} \cup \{a_i b_j \mid k+1 \leq i \leq j \leq 2k\} \\ &\quad \cup \{a_i b_j \mid 1 \leq i \leq k \text{ and } k+1 \leq j \leq 2k\} \\ &= L_k \cup \bar{L}_k \cup \{a_i \mid 1 \leq i \leq k\} \cdot \{b_j \mid k+1 \leq j \leq 2k\} \end{aligned}$$

w is in $L(G''_{2k})$ if and only if $S''_{2k} \underset{G''_{2k}}{\Rightarrow} EF \overset{2}{\underset{G''_{2k}}{\Rightarrow}} ef$ for some $E, F \in N''_{2k}$, $e, f \in \Sigma_{2k}$ so that $S''_{2k} \to EF$ is in P^1_{2k}, $E \to e$ and $F \to f$ in P^2_{2k} and $w = ef$. This is true if and only if one of the following three cases hold.

<u>Case 1</u>. $EF = C_{2k}D_{2k}$, $e = a_i$ for some $1 \leq i \leq k$ and $f = b_j$ for some $k+1 \leq j \leq k$. So $w \subseteq \{a_i \mid 1 \leq i \leq k\} \cdot \{b_i \mid k+1 \leq j \leq 2k\}$.

<u>Case 2</u>. $S''_k \to EF$ is in P^1_k. In this case there is a corresponding derivation $S'_k \underset{G''_k}{\Rightarrow} EF \overset{2}{\underset{G''_k}{\Rightarrow}} ef = w$ so that $w \in L_k$.

<u>Case 3</u>. $\bar{S}''_k \to EF$ is in $\bar{P}^1_k$. Following Case 2, $w \in \bar{L}_k$.

So we have shown that $w \in L(G''_{2k})$ if and only if $w \in \{a_i \mid 1 \leq i \leq k\} \cdot \{b_j \mid k+1 \leq j \leq 2k\}$ or $w \in L_k$ or $w \in \bar{L}_k$. Hence $L(G''_{2k}) = L_{2k}$.

To compute the sizes note that[†]

$$\begin{aligned}
|N''_1| &= 2\,, & |N''_{2k}| &= |N''_k| + |\bar{N}''_k| + 2 &&= 2\cdot|N''_k| + 2\,, && k \geq 1\\
|P^1_1| &= 1\,, & |P^1_{2k}| &= 1 + |P^1_k| + |\bar{P}^1_k| &&= 2\cdot|P^1_k| + 1\,, && k \geq 1\\
|P^2_1| &= 2\,, & |P^2_{2k}| &= k + k + |P^2_k| + |\bar{P}^2_k| &&= 2\cdot|P^2_k| + 2k\,, && k \geq 1\,.
\end{aligned}$$

The solutions to these recurrence relations are $|N''_k| = 4k-2$, $|P^1_k| = 2k-1$ and $|P^2_k| = k\log k + 2k$ where these hold for each k which is a power of 2. Thus, since $|V''_k| = |\{S''_k\} \cup N''_k \cup \Sigma_k|$,

$$|V''_k| = 1 + 4k - 2 + 2k = 6k - 1$$

and
$$|G''_k| = 3|P^1_k| + 2|P^2_k| = 2k\log k + 10k - 3\,.$$

The claim of the conjecture which we were unable to prove is that this grammar is asymptotically the smallest. It is true that for any $k \geq 2$ we can get a smaller grammar, but the optimizations are local and do not seem to lower the size beyond $\Theta(k \log k)$.

From Lemma 2.5.4 it follows that

Theorem 2.5.2. If the conjecture is true then Λ-free preserving chain rule elimination requires time $\Omega(|G|\log|G|)$.

Proof. By Lemma 2.5.4 the conjecture yields that every chain-free Λ-free grammar that generates L_k is of size at least $c'\cdot k\log k$ for some constant c'. But the grammar (family) G from the above discussion is Λ-free, generates L_k and is of size $\Theta(k)$. If we apply

[†] $|P|$ is the cardinality of the set P.

any chain eliminating algorithm that preserves Λ-freeness to G, the output grammar must be of size at least $c' k \log k$. So the algorithm requires time $\theta(|G| \log |G|)$. $\square$

The reason we had to restrict ourselves to Λ-freeness preserving algorithms is that it is easy to eliminate chain rules by introducing Λ-rules, in a way which is not desirable. We just add on a new symbol T to the nonterminal set, replace each chain rule $A \to B$ by $A \to BT$ and place $T \to \Lambda$ in the production set.

Section 2.6 - Uniquely Invertible Grammars

The property we wish to achieve in this section is helpful in parsing. It makes the reduction phase (cf. Gray and Harrison [1973]) in bottom up parsing trivial.

Definition 2.6.1. A grammar $G = (V,\Sigma,P,S)$ is said to be *uniquely invertible* if for each $A, B \in N$, $\alpha \in V^*$, $A \rightarrow \alpha$, $B \rightarrow \alpha$ in P imply $A = B$.

In Gray and Harrison [1972] it is shown that this property is not compatible with both Λ-freeness and chainfreeness: The language $a^* \cup \{b\}$ does not have any grammar which is Λ-free, chain free and uniquely invertible.

Two problems may now be discussed: obtaining a grammar which is uniquely invertible and Λ-free, or one which is uniquely invertible and chain free.

The second problem is trivial and will not be discussed here (cf. Gray and Harrison [1972]).

For the first problem we discuss two algorithms by Gray and Harrison [1972] and by Graham [1971].

The first algorithm is reproduced here with little change. We do not make any effort to clarify the details of the implementation, since it will be shown that the algorithm is inefficient no matter how it is implemented.

Algorithm 2.6.1.

Input: $G = (V,\Sigma,P,S)$ a Λ-free, chain free grammar

Output: G', a uniquely invertible Λ-free grammar, $L(G') = L(G)$

```
begin
N' := $\{U \subseteq N \mid U \neq \emptyset\} \cup \{S'\}$; comment S', a new symbol;
P' := $\emptyset$;
if $S \to \Lambda$ is in P then P' := $P' \cup \{S' \to \Lambda\}$;
for all $U \in N'$ such that $S \in U$ do P' := $P' \cup \{S' \to U\}$;
for all $B \to x_0B_1x_1 \cdots B_nx_n \in P$ do
   begin
   comment $B_i \in N$, $x_i \in \Sigma^*$;
   for all $A_1, A_2, \ldots, A_n \in N' - \{S'\}$ do
      begin
      A := $\{C \mid C \to x_0C_1x_1 \cdots C_nx_n \in P,\ C_i \in A_i\}$;
      P' := $P' \cup \{A \to x_0A_1x_1 \cdots A_nx_n\}$
      end
   end
end.
```

The proof that Algorithm 2.6.1 is correct can be found in Gray and Harrison [1972] (Theorem 1.5).

This construction is reminiscent of the subset construction to transform nondeterministic finite automata to deterministic finite automata. In many cases, however, the grammar G' shrinks substantially when reduced. We present an example that shows this algorithm is exponential in nature even when reduction is performed.

Example 2.6.1. Let k be a positive even integer. Let $\Sigma = \{a,b,c\}$ and $N = \{A_1,\ldots,A_k\}$. Let $\sigma_a = (1,2,\ldots,k)$ and $\sigma_b = (1,2)$ be two permutations written in cyclic notation. These two permutations generate the symmetric group on k-letters. Define P by

$$A_i \to aA_{\sigma_a(i)} | bA_{\sigma_b(i)} \quad \text{for each } i, \ 1 \le i \le k$$
$$A_i \to c \quad \text{for each odd } i, \ 1 \le i < k$$

For $G = (V,\Sigma,P,A_1)$, we have

$$|V| = k+3 \quad \text{and} \quad |G| = 7k$$

Now the variables of G' will be sets of variables such as $\{A_1,A_3,\ldots\}$. To simplify the notation we will write $\{1,3,\ldots\}$ instead. If the computation of G' is carried out and G' is reduced, we get $N' = \{B \subseteq N \big| |B| = k/2\} \cup \{S'\}$ and $P' = \{S' \to B|$ $A_1 \in B \subseteq N, |B| = k/2\} \cup \{B \to aB', B \to bB'' | B \subseteq N, |B| = k/2, B' = \sigma_a(B),$ $B'' = \sigma_b(B)\} \cup \{\{1,3,\ldots,k-1\} \to c\}$. Clearly

$$|V'| = \binom{k}{k/2} + 4 > 2^{k/2}$$
$$|G'| = 2 + 6\binom{k}{k/2} + 2\cdot\frac{1}{2}\cdot\binom{k}{k/2} > 7\cdot 2^{k/2}$$

so that for some constants $c_1, c_2 > 0$

$$|G'| > c_1 2^{c_2|G|} .$$

Example 2.6.1 clearly indicates that Algorithm 2.6.1 takes exponential time. In fact a much stronger result follows. It says that even if the non-reduced version of the output grammar is never produced, the algorithm is still exponential. More precisely

Theorem 2.6.1. Any algorithm that outputs the same grammar as Algorithm 2.6.1 followed by reduction does takes exponential time.

The next algorithm is adapted from Graham [1971]. Before presenting it we explain the ideas behind it using an example.

Example 2.6.2. Consider a grammar with productions $S \to AB|AS$, $A \to AB|a$, $B \to AB|b$. There are three productions with AB as their right hand side. We would like to retain one of them, say $B \to AB$, and add chain rules $A \to B$ and $S \to A$ to be able to simulate the other productions. However these chains would introduce undesired derivations like $S \Rightarrow A \Rightarrow B \Rightarrow b$. So we have to make distinct copies of the nonterminals for each production. We get two such copies for each of S, A and B. We then place the productions $B_1 \to A_1B_1|A_1B_2|A_2B_1|A_2B_2$ instead of $B \to AB$, and $B_2 \to b$ instead of $B \to b$. We also need a new start symbol S' with productions $S' \to S_1|S_2$. Note that since these are chain productions with (copies of) S as the right hand side, we may not use S_1 and S_2 as the right hand side of any other chain rule we introduce. The new grammar has the following productions: $S' \to S_1|S_2$, $S_1 \to A_1$, $S_2 \to A_1S_1|A_1S_2|A_2S_1|A_2S_2$, $A_1 \to B_1$, $A_2 \to a$, $B_1 \to A_1B_1|A_1B_2|A_2B_1|A_2B_2$ and $B_2 \to b$.

We now describe the variables in the algorithm. We use the variable RHS to store all the strings that appear as Right Hand Sides of productions. For each $\alpha \in$ RHS, LHS(α) is a set and LAST(LHS(α)) is an element of LHS(α). It designates the last element to be removed from LHS(α). LHS(α) is used to store all $A_i \in N'$ such that $A \to \alpha$ is in P and A_i is a copy of $A \in N$ (A is the Left Hand Side of the production). If a copy S_i of S appears in LHS(α), then LAST(LHS(α)) = S_i.

For each $A \in N$, $NRHS(A)$ is an integer which is used to count the Number of Right Hand Sides α such that $A \to \alpha$ is in P. This count is used in determining what index needs to be assigned to a certain copy of A.

We now present the algorithm.

Algorithm 2.6.2.

Input: $G = (V,\Sigma,P,S)$, a chain free grammar

Output: G', a uniquely invertible grammar such that $L(G') = L(G)$

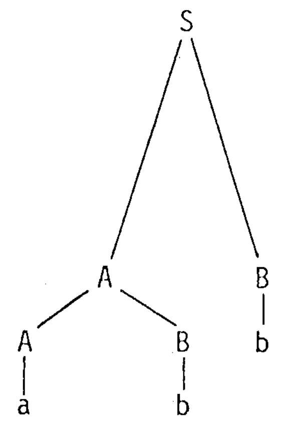
```
     begin
L1:  RHS := $\{\alpha \mid$ there exists $A \in N$ such that $A \to \alpha$ is in $P\}$;
     for all $\alpha \in RHS$ do $LHS(\alpha) := \emptyset$;
     for all $A \in N$ do $NRHS(A) := 0$;
L2:  for all $A \to \alpha \in P$ do
        begin
        $NRHS(A) := NRHS(A)+1$;
        $LHS(\alpha) := LHS(\alpha) \cup \{A_{NRHS(A)}\}$;
        if $A = S$ then $LAST(LHS(\alpha)) := S_{NRHS(S)}$;
        end;
     $N' := \{A_j \mid 1 \le j \le NRHS(A),\ A \in N\} \cup \{S'\}$;
     $P' := \{S' \to S_j \mid 1 \le j \le NRHS(S)\}$;
L3:  for all $\alpha \in RHS$ do
        begin
        comment $\alpha = x_0 B^1 \cdots x_{m-1} B^m x_m$, $m \ge 0$, $x_i \in \Sigma^*$, $B^i \in N$;
        remove C from $LHS(\alpha)$;
        $P' := P' \cup \{C \to \alpha_0 B^1_{j_1} \cdots B^m_{j_m} \alpha_m \mid 1 \le j_i \le NRHS(B^i),\ 1 \le i \le m\}$;
        while $LHS(\alpha)$ is not empty do
           begin
           remove D from $LHS(\alpha)$;
           $P' := P' \cup \{D \to C\}$;
           $C := D$
           end
        end;
     $G' := (N' \cup \Sigma, \Sigma, P', S')$;
     end.
```

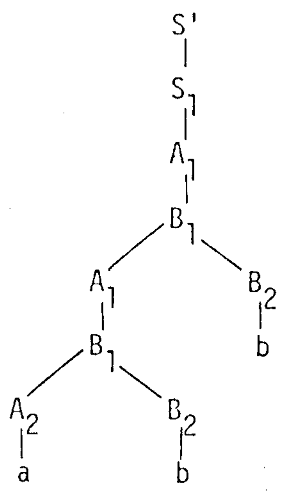
Example 2.6.3. We use the grammar in Example 2.6.2 to observe the way the algorithms works.

When L2 is first reached we have RHS = $\{AB, AS, a, b\}$, $LHS(\alpha) = \emptyset$ for all α in RHS and $NRHS(T) = 0$ for all $T \in N$.

After the do loop in L2 is executed once with $S \to AB$, we get $LHS(AB) = \{S_1\}$ and $NRHS(S) = 1$. The for loop in L2 is then executed for the other productions and when L3 is reached we have $LHS(AB) = \{S_1, A_1, B_1\}$, $LHS(AS) = \{S_2\}$, $LHS(a) = \{A_2\}$, $LHS(b) = \{B_2\}$, $LAST(LHS(AB)) = S_1$ and $LAST(LHS(AS)) = S_2$. $NRHS(S) = NRHS(A) = NRHS(B) = 2$. $N' = \{S_1, S_2, A_1, A_2, B_1, B_2, S'\}$ and $P' = \{S' \to S_1,\ S' \to S_2\}$. The for loop in L3 is now executed with $AB \in RHS$. We first remove B_1 from $LHS(AB)$ (we could have picked A_1 but not S_1 which is the last). We place $B_1 \to A_{j_1}B_{j_2}$ in P' for all $1 \leq j_1 \leq 2$ and $1 \leq j_2 \leq 2$. Then we remove A_1 from $LHS(AB)$ and place $A_1 \to B_1$ in P' and finally $S_1 \to A_1$ is placed in P'. The for loop in L3 is executed with the rest of the elements of RHS, yielding the grammar as in Example 2.6.2.

We illustrate the structure of derivations in the new grammar by placing side by side the derivation tree of abb in G and G'.





Notice that in the G' derivation tree a chain appears between S_1 and the production that has A_1B_2 in its right hand side. This corresponds to the production $S \to AB$ used in the G derivation tree.

The algorithm we presented does not preserve Λ-freeness.[†] This is easy to fix. If we have a Λ-free grammar with $S \to \Lambda$ in P, we can remove this production temporarily, apply the algorithm and finally add $S' \to \Lambda$ back to P'.

Since we have changed at least the appearance of the algorithm quite substantially we now present a proof of its correctness.

Lemma 2.6.1. Algorithm 2.6.2 when applied to a chain free grammar $G = (V,\Sigma,P,S)$ produces a uniquely invertible grammar G' such that $L(G') = L(G)$.

Proof. First, it is clear that the algorithm terminates.

We need to show that G' is uniquely invertible and that $L(G') = L(G)$. Before we proceed we introduce some notation. For all $A \in N$ we call each A_j where $1 \leq j \leq NRHS(A)$ a copy of A, and $\{A_j | 1 \leq j \leq NRHS(A)\}$ is the set of all copies of A. This is extended to strings in V^* as follows. If $\alpha \in V^*$ and $\alpha = x_0B^1x_1 \cdots B^mx_m$ for some $x_0,x_1,\ldots,x_m \in \Sigma^*$ and $B^1,B^2,\ldots,B^m \in N$ then $x_0B^1_{j_1}x_1 \cdots x_{m-1}B^m_{j_m}x_m$ is a copy of α if $B^i_{j_i}$ is a copy of B^i for all i, $1 \leq i \leq m$. We use copies(α) to denote the set of all copies of α.

The following claim characterizes the effects of the first part of the algorithm, up to L3.

[†]The algorithm does not preserve chainfreeness either, as can be expected.

Claim 1. The following hold the first time L3 is reached:

(1) RHS = $\{\alpha \mid \text{there exists } A \in N \text{ so that } A \to \alpha \text{ is in } P\}$

(2) For all $A \in N$, $NRHS(A) = |\{\alpha \mid A \to \alpha \in P\}|$

(3) For all $\alpha \in RHS$, $LHS(\alpha)$ contains exactly one copy of A for each $A \in N$ such that $A \to \alpha$ is in P.

(4) $\{LHS(\alpha) \mid \alpha \in RHS\}$ is a partition of $N'-\{S'\}$ (i.e. each element A_j of $N'-\{S'\}$ appears in exactly one set $LHS(\alpha)$).

(5) For all $\alpha \in RHS$ if $S_j \in LHS(\alpha)$ for some j, then $LAST(LHS(\alpha)) = S_j$.

Proof of Claim 1. (1) is trivial. For each $A \in N$, $NRHS(A)$ is set to 0, then incremented by one for each $A \to \alpha$ in P so (2) holds. A_j is placed in $LHS(\alpha)$ when the for loop in L2 is executed for $A \to \alpha$ in P. So (3) follows. Since the index used is $j = NRHS(A)$ and $NRHS(A)$ is incremented just prior to placing $A_{NRHS(A)}$ in $LHS(\alpha)$, that particular copy of A appears in no other set. Then (4) holds. (5) is trivial.

After label L3 RHS is not changed and for all $A \in N$, $NRHS(A)$ is not changed. For all $\alpha \in RHS$, $LHS(\alpha)$ is changed only when the for loop in L3 is executed with $\alpha \in RHS$ and then elements are removed from $LHS(\alpha)$ one by one. We therefore find it convenient to use the names RHS, $NRHS(A)$ and $LHS(\alpha)$ to denote the values of these variables when L3 is reached for the first time.

We can now examine the productions in P' (at termination time). There are essentially two types of productions; chain rules and others. The chains are either $S' \to S_j$ or $D \to C$ where $D, C \in LHS(\alpha)$ for some α, and where $C \neq \text{LAST}(\text{LHS}(\alpha))$. It follows from that and (4) and (5) of Claim 1 that no right hand side in this group appears in more than one production.

The other rules are of the form $A_j \to \alpha'$ where $A \to \alpha \in P$, $\alpha' \in \text{copies}(\alpha)$, $1 \leq j \leq \text{NRHS}(A)$ and A_j is the first element in $\text{LHS}(\alpha)$. Since G is chain free, no unique invertibility conflicts with the chain rules in P' can occur. Conflicts within this type of production are impossible because for $\alpha, \beta \in \text{RHS}$ if $\alpha \neq \beta$ then $\text{copies}(\alpha) \cap \text{copies}(\beta) = = \emptyset$, and because for each $\alpha \in \text{RHS}$ and $\alpha' \in \text{copies}(\alpha)$ only one production in α' as the right hand side will be included in P'.

This completes the proof that G' is uniquely invertible.

Next we will show that $L(G') = L(G)$.

We start by showing that for every production $A \to \alpha$ in P every copy of α may be derived in G' by some copy of A.

<u>Claim 2</u>. If $A \to \alpha$ is in P then for all $\alpha' \in \text{copies}(\alpha)$ there exists some $A_j \in \text{copies}(A)$ such that $A_j \overset{+}{\underset{G'}{\Rightarrow}} \alpha'$.

(We prove something stronger, that the same A_j derives all these copies of α).

<u>Proof of Claim 2</u>. If $A \to \alpha$ is in P then, for some $1 \leq j \leq \text{NRHS}(A)$, $A_j \in \text{LHS}(\alpha)$. Suppose A_j is the i^{th} element removed from $\text{LHS}(\alpha)$, $i \geq 1$. We prove by induction on i, that $A_j \overset{+}{\underset{G'}{\Rightarrow}} \alpha'$ for all $\alpha' \in \text{copies}(\alpha)$. If $i = 1$ then A_j is the first element to be removed from $\text{LHS}(\alpha)$, and therefore $A_j \to \alpha'$ is placed in P' for each $\alpha' \in \text{copies}(\alpha)$. Assume $A_j \overset{+}{\underset{G'}{\Rightarrow}} \alpha'$ if $i < i_0$, $i_0 > 1$. If A_j is the i_0^{th} element removed then $A_j \to C$ is placed in P' where C is the $(i = i_0 - 1)^{th}$

element removed from LHS(α), and by the inductive hypothesis $A \underset{G'}{\Rightarrow} C \overset{+}{\underset{G'}{\Rightarrow}} \alpha'$ for all $\alpha' \in$ copies(α). This completes the proof of the claim.

We now generalize this claim to derivations:

Claim 3. For all $A \in N$, $\alpha \in V^*$, if $A \overset{*}{\underset{G}{\Rightarrow}} \alpha$ then for each $\alpha' \in$ copies(α) there exists $A_j \in$ copies(A) such that $A_j \overset{*}{\underset{G'}{\Rightarrow}} \alpha'$.

Proof of Claim 3. By induction on the length i of derivation $A \overset{*}{\underset{G}{\Rightarrow}} \alpha$.

Basis: $i = 0$. Then $\alpha = A$, copies(α) = $\{A_{j'} | 1 \leq j' \leq \text{NRHS}(A)\}$. For each $\alpha' = A_{j'} \in$ copies(α) take $j = j'$. Clearly $A_{j'} \overset{*}{\underset{G'}{\Rightarrow}} A_{j'}$.

Induction step: Assume the result holds for all derivations of length $i < i_0$, $i_0 > 0$. Now suppose $i = i_0$, $A \overset{i}{\underset{G}{\Rightarrow}} \alpha$. We can rewrite this as $A \underset{G}{\Rightarrow} \beta \overset{i-1}{\underset{G}{\Rightarrow}} \alpha$, where $A \to \beta$ is in P. Let $\beta = \beta_0 B^1 \beta_1 \cdots B^m \beta_m$ where $B^\ell \in N$, $\beta_\ell \in \Sigma^*$. We can write $\alpha = \beta_0 \alpha_1 \beta_1 \cdots \alpha_m \beta_m$, and for $1 \leq \ell \leq m$ $B^\ell \overset{*}{\underset{G}{\Rightarrow}} \alpha_\ell$, with a derivation of length less than i. For any $\alpha' \in$ copies(α) we can write $\alpha' = \beta_0 \alpha'_1 \beta_1 \cdots \alpha'_m \beta_m$ with $\alpha'_\ell \in$ copies(α_ℓ) for all ℓ, $1 \leq \ell \leq m$ (recall that $\beta_\ell \in \Sigma^*$ so that copies(β_ℓ) = $\{\beta_\ell\}$). We can now apply the induction hypothesis m times to obtain elements $B^\ell_{j_\ell} \in$ copies(B^ℓ) for all ℓ, $1 \leq \ell \leq m$ such that $B^\ell_{j_\ell} \overset{*}{\underset{G'}{\Rightarrow}} \alpha'_\ell$.

Now we consider $\beta' = \beta_0 B^1_{j_1} \cdots B^m_{j_m} \beta_m \in$ copies(β). By Claim 2, there exists $A_j \in$ copies(A) such that $A_j \overset{+}{\underset{G'}{\Rightarrow}} \beta'$. Combining the various derivations we get $A_j \overset{+}{\underset{G'}{\Rightarrow}} \beta' = \beta_0 B^1_{j_1} \cdots B^m_{j_m} \beta_m \overset{+}{\underset{G'}{\Rightarrow}} \beta_0 \alpha'_1 \cdots \alpha'_m \beta_m = \alpha$ which completes the induction and the proof of the claim.

We can now show that $L(G) \subseteq L(G')$: For all $w \in \Sigma^*$ if $w \in L(G)$ then $S \overset{+}{\underset{G}{\Rightarrow}} w$ and $w \in$ copies(w), so by Claim 3 $S_j \overset{+}{\underset{G'}{\Rightarrow}} w$ for some j, $1 \leq j \leq$ NRHS(S). But $S' \to S_j$ is in P' so that $w \in L(G')$.

The following notation will help us prove that $L(G') \subseteq L(G)$.

For all $\alpha' \in V'^{*}$ define $\text{copies}^{-1}(\alpha')$ to be the element $\alpha \in V^{*}$ such that $\alpha' \in \text{copies}(\alpha)$. As was pointed out $\text{copies}^{-1}(\alpha')$ is always uniquely defined (and is simply the string α' with subscripts omitted).

For $A_j \in N'-\{S'\}$, α', β_1', $\beta_2' \in V'^{*}$ we say that $\beta_1' A_j \beta_2' \Rrightarrow \beta_1' \alpha' \beta_2'$ if $A_j \overset{+}{\underset{G'}{\Rightarrow}} B_\ell \underset{G'}{\Rightarrow} \alpha'$ for some $B_\ell \in N'$ where $B_\ell \rightarrow \alpha'$ in P' is the only non-chain rule used in this derivation. $\overset{*}{\Rrightarrow}$ is the reflexive transitive closure of $\Rrightarrow$.

For example, the derivation tree in Example 2.6.3 can be written as $S' \underset{G'}{\Rightarrow} S_1 \Rrightarrow A_1 B_2 \Rrightarrow A_2 B_2 B_2 \Rrightarrow aB_2 B_2 \Rrightarrow abB_2 \Rrightarrow abb$.

The next claim shows the connection between a special type of derivation in G' (namely $\overset{*}{\Rrightarrow}$) and derivations in G'.

Claim 4. If, for some $A_j \in N'$, $\alpha' \in V'^{*}$, $A_j \overset{*}{\Rrightarrow} \alpha'$ in G' then $A \overset{*}{\underset{G}{\Rightarrow}} \text{copies}^{-1}(\alpha')$.

Proof of Claim 4. By induction on the "length" i of the "derivation" $A_j \overset{*}{\Rrightarrow} \alpha'$.

Basis: $i = 0$. Then $\alpha' = A_j$, $\text{copies}^{-1}(\alpha') = A$ and $A \overset{*}{\underset{G}{\Rightarrow}} A$.

Induction step: Assume the claim holds whenever $i < i_0$, $i_0 > 0$. Let $A_j \overset{i}{\Rrightarrow} \alpha'$ for $i = i_0$. Factor it (using the definition of $\Rrightarrow$ $A_j \overset{i-1}{\Rrightarrow} \beta_1' B_\ell \beta_2' \Rrightarrow \beta_1' \gamma' \beta_2' = \alpha'$. By the induction hypothesis $A \overset{*}{\underset{G}{\Rightarrow}} \text{copies}^{-1}(\beta_1' B_\ell \beta_2')$, or if we denote $\beta_1 = \text{copies}^{-1}(\beta_1')$ and $\beta_2 = \text{copies}^{-1}(\beta_2')$, $A \overset{*}{\underset{G}{\Rightarrow}} \beta_1 B \beta_2$. Now, by the definition of $\Rrightarrow$ we must have some $C_m \in N'$ such that $B_\ell \overset{*}{\underset{G}{\Rightarrow}} C_m$ using only chain rules and $C_m \rightarrow \gamma'$ in P' is a non-chain rule. By inspection of the algorithm it is clear that for some $\gamma \in \text{RHS}$, B_ℓ, $C_m \in \text{LHS}(\gamma)$ and moreover $\gamma' \in \text{copies}(\gamma)$. It follows that $B \rightarrow \gamma$, $C \rightarrow \gamma$ are in P and $\gamma = \text{copies}^{-1}(\gamma')$. We can combine our derivations in G to obtain $A \overset{*}{\underset{G}{\Rightarrow}} \beta_1 B \beta_2 \Rightarrow \beta_1 \gamma \beta_2 = \text{copies}^{-1}(\beta_1' \gamma' \beta_2') = \text{copies}^{-1}(\alpha')$, and complete the proof of the claim.

We now prove some sufficient conditions that enable us to express derivations in G' in terms of $\Rrightarrow$.

Claim 5. If for some $A_j \in N'$, $\alpha' \in V'$, $A_j \overset{*}{\underset{G'}{\Rightarrow}} \alpha'$ such that no symbol in α' is generated in the last level of the derivation tree by a chain rule, then $A_j \overset{*}{\Rrightarrow} \alpha'$.

Proof of Claim 5. By induction on the length i of the derivation $A_j \overset{*}{\underset{G'}{\Rightarrow}} \alpha'$.

Basis: $i = 0$, trivial.

Induction step: Assume the claim holds whenever $i < i_0$, $i_0 > 0$. Let $A_j \overset{i}{\underset{G'}{\Rightarrow}} \alpha'$, $i = i_0$. Factor it as $A_j \overset{i-1}{\underset{G'}{\Rightarrow}} \beta_1' B_\ell \beta_2' \Rightarrow \beta_1' \gamma' \beta_2' = \alpha'$ where $B_\ell \rightarrow \gamma' \in P'$ is not a chain rule. Now reorder the derivation as follows. $A_j \overset{*}{\underset{G'}{\Rightarrow}} \beta_1' C_m \beta_2' \overset{*}{\underset{G'}{\Rightarrow}} \beta_1' B_\ell \beta_2' \Rightarrow \beta_1' \gamma' \beta_2$ where the rule, in $A_j \overset{*}{\underset{G'}{\Rightarrow}} \beta_1' C_m \beta_2'$, that generates C_m is not a chain rule and $C_m \overset{*}{\underset{G'}{\Rightarrow}} B_\ell$ uses only chain rules. By the claim's hypothesis no symbol in β_1' and β_2' is generated, in the last step, by a chain rule, so we can use the inductive hypothesis for the derivation $A_j \overset{*}{\underset{G'}{\Rightarrow}} \beta_1' C_m \beta_2'$ (whose length is at most $i-1$) and conclude that $A_j \overset{*}{\Rrightarrow} \beta_1' C_m \beta_2'$. By definition of $\Rrightarrow$ we have $\beta_1' C_m \beta_2' \Rrightarrow \beta_1' \gamma' \beta_2'$ so $A_j \overset{*}{\Rrightarrow} \beta_1' C_m \beta_2' \Rrightarrow \beta_1' \gamma' \beta_2' = \alpha'$ and Claim 5 has been established.

Now let $w \in L(G')$. Then, for some $1 \leq j \leq \text{NRHS}(S)$, $S' \underset{G'}{\Rightarrow} S_j \overset{*}{\underset{G'}{\Rightarrow}} w$. $S_j \overset{*}{\underset{G'}{\Rightarrow}} w$ satisfies the condition of Claim 5 (since no element of Σ^* can be generated by a chain rule) so $S_j \overset{*}{\Rrightarrow} w$. Using

Claim 4 we get $S \overset{*}{\underset{G}{\Rightarrow}} \text{copies}^{-1}(w) = w$ so $w \in L(G)$. This shows that $L(G') \subseteq L(G)$ and combined with what we already know, completes the proof that $L(G) = L(G')$ and the proof of the lemma. $\square$

Next we discuss the complexity of Algorithm 2.6.2, using the following example.

Example 2.6.4. Let $G = (V,\Sigma,P,S)$, $N = \{S,B^1,B^2,\ldots,B^k\}$, $\Sigma = \{a_1,\ldots,a_k,b\}$, and $P = \{S \to B^1B^2\cdots B^k\} \cup \{B^i \to a_i,\ B^i \to b \mid 1 \le i \le k\}$. Apply Algorithm 2.6.2. When L3 is reached for the first time $RHS = \{B^1B^2\cdots B^k, b\} \cup \{a_i \mid 1 \le i \le k\}$, $LHS(B^1B^2\cdots B^k) = \{S_1\}$. For all i, $1 \le i \le k$, $LHS(a_i) = \{B_1^i\}$, $LHS(b) = \{B_2^i \mid 1 \le i \le k\}$ and $N' = \{S',S_1\} \cup \{B_1^i, B_2^i \mid 1 \le i \le k\}$. When the algorithm terminates we get $P' = \{S' \to S_1\} \cup \{S_1 \to B_{j_1}^1 B_{j_2}^2 \cdots B_{j_k}^k \mid 1 \le j_i \le 2 \text{ for all } i,\ 1 \le i \le k\} \cup$
$\cup \{B_1^i \to a_i \mid 1 \le i \le k\} \cup \{B_2^k \to b\} \cup \{B_2^i \to B_2^{i+1} \mid 1 \le i < k\}$, and $G' = (N' \cup \Sigma, \Sigma, P', S')$. G is reduced.

The sizes of the grammars are

$$|V| = 1 + k + k + 1 = 2k + 2$$

$$|G| = k + 1 + k(2+2) = 5k + 1$$

$$|V'| = 2 + 2k + k + 1 = 3k + 3$$

$$|G'| = 2 + (k+1)2^k + 2k + 2 + 2(k-1) = (k+1)2^k + 4k + 2$$

so $|G'| > c_1|G|2^{c_2|G|}$ for some positive constants c_1 and c_2.

Since the size of G' is exponential in the size of the input we get the following result.

Theorem 2.6.2. Algorithm 2.6.2 takes exponential time.

Examination of Example 2.6.4 can convince us that the grammar G' is large mainly due to the fact that there is a production with a long right hand side. In fact, we can use the same approach employed in Section 2.3 to obtain a more efficient algorithm.

Lemma 2.6.3. Algorithm 2.6.2, when applied to a CNF grammar G with no Λ-rules produces a uniquely invertible grammar G' so that $|G'| = O(|G|^2)$ and $\|G'\| = O(\|G\|^2/\log\|G\|)$.

Proof. We estimate the size of the output grammar G'. Let $G = (V,\Sigma,P,S)$. For all $A, B \in N$, $a \in \Sigma$ define $\ell(A,B) =$ $= |\{C|C \to AB \in P\}| = |LHS(AB)|$ and $\ell(a) = |\{C|C \to a \in P\}| = |LHS(a)|$, $r(A) = |\{\alpha|A \to \alpha \in P\}| = NRHS(A)$. Clearly

$$|P| = \sum_{A,B \in N} \ell(A,B) + \sum_{a \in \Sigma} \ell(a) = \sum_{A \in N} r(A)$$

and

$$|G| = 3 \sum_{A,B \in N} \ell(A,B) + 2 \sum_{a \in \Sigma} \ell(a)$$

so that $2|P| \le |G| \le 3|P|$. In the construction of G', we see that $|N'| = \sum_{A \in N} r(A) + 1 = |P| + 1$.

Productions in P' are of several types. First there are $r(S)$ $S' \to S_j$ productions. Then there are productions generated from $LHS(\alpha)$ for $\alpha \in N^2$ and for $\alpha \in \Sigma$, which are either $C \to \alpha_0 B^1_{j_1} \cdots B^n_{j_n} \alpha_n$ type or chain-rules. We get

$$\begin{aligned}|G'| &= 2r(S) + \sum_{\substack{A,B \in N\\ \ell(A,B) \ge 1}} 3r(A)r(B) + 2(\ell(A,B) - 1) + \sum_{\substack{a \in \Sigma\\ \ell(a) \ge 1}} 2\cdot 1 + 2(\ell(a)-1)\\ &= 2r(S) + 3 \sum_{\substack{A,B \in N\\ \ell(A,B) \ge 1}} r(A)r(B) + 2 \sum_{\substack{A,B \in N\\ \ell(A,B) \ge 1}} (\ell(A,B)-1) + 2 \sum_{\substack{a \in \Sigma\\ \ell(a) \ge 1}} \ell(a)\end{aligned}$$

$$|G'| \leq 2r(S) + 3 \cdot \sum_{A,B \in N} r(A)r(B) + 2 \cdot \sum_{\substack{A,B \in N \\ \ell(A,B) \geq 1}} \ell(A,B) + 2 \cdot \sum_{\substack{a \in \Sigma \\ \ell(a) \geq 1}} \ell(a)$$

$$= 2r(S) + 3|P|^2 + 2|P| \leq 3|P|^2 + 4|P| .$$

We conclude that $|V'| = |N'| + |\Sigma| \leq c_1|G|$ and $|G'| \leq c_2|G|^2$ for some constants $c_1, c_2 > 0$.

G is a CNF grammar with no Λ-rules so $P \subseteq N \times N^2 \cup N \times \Sigma$ hence $|G| \leq 3 \cdot |N|^3 + 2 \cdot |N| \cdot |\Sigma| \leq 5|V|^3$ and therefore $|V| \geq (\frac{|G|}{5})^{1/3}$. It follows that $\|G\| = |G|\log|V| \geq |G|\log((\frac{|G|}{5})^{1/3}) \geq c_3|G|\log|G|$ for some constant $c_3 > 0$. Also, since $\|G\| = |G|\log|V| \leq |G|^2$ we have $\log\|G\| \leq 2 \log |G|$. We use these estimates to compute $\|G'\|$.

$$\|G'\| = |G'|\log|V'| \leq c_2|G|^2 \cdot \log(c_1|G|) \leq c_4|G|^2\log|G|$$

for some constant $c_4 > 0$. So

$$\|G'\| \leq \frac{2c_4}{c_3^2} \frac{(c_3|G|\log|G|)^2}{2 \log|G|} \leq \frac{2c_4}{c_3^2} \frac{\|G\|^2}{\log\|G\|}$$

hence $\qquad |G'| = O(|G|^2)$

and $\qquad \|G'\| = O(\frac{\|G\|^2}{\log\|G\|})$. □

The next example shows that these bounds are achievable.

Example 2.6.5. For $k \geq 1$, $G = (V,\Sigma,P,A^0)$, $N = \{A^0,A^1,\ldots,A^k\}$, $\Sigma = \{a_1,\ldots,a_k\}$, $P = \{A^i \to A^1A^1 | 0 \leq 1 \leq k\} \cup \{A^1 \to A^jA^j | 2 \leq j \leq k\} \cup$ $\cup \{A^i \to a_i | 1 \leq i \leq k\}$. The computation of the algorithm yields

$$\mathrm{RHS} = \{A^iA^i, a_i \mid 1 \le i \le k\}$$

$$\mathrm{LHS}(A^1A^1) = \{A_1^0, A_1^1, \ldots, A_1^k\} \quad (A_1^0 \text{ last in the list})$$

$$\mathrm{LHS}(A^iA^i) = \{A_i^1\} \quad \text{for } 2 \le i \le k$$

$$\mathrm{LHS}(a_1) = \{A_{k+1}^1\}$$

$$\mathrm{LHS}(a_i) = \{A_2^i\} \quad \text{for } 2 \le i \le k$$

so that $N' = \{S', A_1^0\} \cup \{A_i^1 \mid 1 \le i \le k+1\} \cup \{A_1^j, A_2^j \mid 2 \le j \le k\}$,
$P' = \{S' \to A_1^0\} \cup \{A_1^k \to A_{i_1}^1 A_{i_2}^1 \mid 1 \le i_1, i_2 \le k+1\} \cup \{A_1^j \to A_1^{j+1} \mid 0 \le j < k\} \cup$
$\cup \{A_i^1 \to A_{j_1}^i A_{j_2}^i \mid 1 \le j_1, j_2 \le 2,\ 2 \le i \le k\} \cup \{A_{k+1}^1 \to a_1\} \cup \{A_2^i \to a_i \mid 2 \le i \le k\}$,
$G' = (N' \cup \Sigma, \Sigma, P', S')$.

The sizes of the grammars involved are

$$|V| = 2k+1\ , \quad |G| = 3(k+1) + 3(k-1) + 2k = 8k$$

$$|V'| = 2 + k + 1 + 2(k-1) + k = 4k+1$$

$$|G'| = 2 + 3(k+1)^2 + 2k + 3\cdot 2^2(k-1) + 2 + 2(k-1)$$

$$= 3k^2 + 22k - 7\ .$$

We now obtain the main results of this section.

Theorem 2.6.3. Any CNF grammar G can be transformed to a uniquely invertible Λ-free grammar in time $O(n^2)$ if $|G|$ is the size measure ($O(\frac{n^2}{\log n})$ if $\|G\|$ is the size measure).

Proof. For CNF grammars with no Λ-rules the theorem follows from Lemma 2.6.3 and the fact that the computation of Algorithm 2.6.2 is dominated by the size of the output grammar. If the grammar has a rule $S \to \Lambda$ we use the technique mentioned following Algorithm 2.6.2 which does not change the complexity. □

Theorem 2.6.4. Any grammar G can be transformed to a uniquely invertible Λ-free grammar in time $O(n^4)$, using $|G|$ as a size measure.

Proof. An immediate consequence of Theorem 2.5.1 and Theorem 2.6.3.

$\square$

CHAPTER 3
THE EQUIVALENCE PROBLEM

Section 3.1 - Introduction

The equivalence problem for a language family is that of deciding whether or not two given languages in that family (described by means of grammars generating them or machines accepting them) are equal. In a similar way one can talk about the equivalence problem for one family of languages versus another one. In this case the given languages are known to be in the two families respectively.

The equivalence problem is known to be undecidable for context free languages, cf. Hopcroft and Ullman [1969]. The equivalence problem for deterministic context free languages has been open for a long time. Korenjak and Hopcroft [1966], Rosenkrantz and Stearns [1970], Valiant [1973] and Taniguchi and Kasami [1976] have shown that the problem is decidable for various subfamilies of the deterministic languages.

The positive decidability results for subfamilies of the deterministic languages are established by providing decision algorithms. These procedures are of two general types. One type of algorithm considers two DPDA's M_1 and M_2. A PDA M is constructed to simulate M_1 and M_2 concurrently. M exploits a certain property of the language family that enables two equivalent machines to be simulated using just one pushdown store. For example if M_1 and M_2 are known to accept two LL languages (cf. Rosenkrantz and Stearns [1970]) then the height of the stack of M_1 and M_2 can differ by no more than a constant if they accept the same language. If the simulation cannot be carried out all the way (because the desired property is no longer satisfied) then M accepts. Otherwise M accepts if and only if exactly one of M_1, M_2 accepts. Hence M accepts the empty set if and only if M_1 and M_2 are equivalent, and the decidability of this instance of the equivalence problem follows from the solvability of the emptiness problem for PDA's.

The second type of algorithm considers grammars rather than DPDA's. Such is the algorithm, due to Korenjak and Hopcroft [1966], that decides the equivalence of the languages generated by two simple grammars in standard 2 form, cf. Greibach [1965]. One of the reasons that the simple languages are interesting is that they form the smallest known family with undecidable inclusion ($L_1 \subseteq L_2$) problem, cf. Friedman [1976]. We illustrate the algorithm by an example on an intuitive and informal level.

Example 3.1.1. Let G_1 be a simple grammar with productions $S_1 \to aAB$, $A \to aAB|b$, $B \to b$ and G_2, a simple grammar with productions $S_2 \to aC$, $C \to aCD|bD$, $D \to b$. For each nonterminal string α, we use $L(\alpha)$ to denote the language generated from α (using productions of both grammars).

$L(G_1) = L(G_2)$ if and only if $L(S_1) = L(S_2)$. Since the only S_1 production is $S_1 \to aAB$ it is clear that $L(S_1) = \{w \in \Sigma^* | S_1 \overset{*}{\underset{L}{\Rightarrow}} w\} =$ $= \{w \in \Sigma^* | S_1 \Rightarrow aAB \overset{*}{\underset{L}{\Rightarrow}} w\} = a\{u \in \Sigma^* | AB \overset{*}{\Rightarrow} u\} = aL(AB)$. In a similar way $L(S_2) = aL(C)$ so $L(S_1) = L(S_2)$ if and only if $L(AB) = L(C)$.

A similar analysis may be made on AB and on C. Here A, for instance, has more than one production. Hence $L(AB) = L(A)L(B) =$ $= \{w \in \Sigma^* | A \overset{*}{\underset{L}{\Rightarrow}} w\}L(B) = \{w \in \Sigma^* | A \underset{L}{\Rightarrow} aAB \overset{*}{\underset{L}{\Rightarrow}} w \text{ or } A \underset{L}{\Rightarrow} b \overset{*}{\Rightarrow} w\}L(B) = aL(AB)L(B) \cup bL(B)$

$= aL(ABB) \cup bL(B)$. Clearly these two components are disjoint and since $L(C) = aL(CD) \cup bL(D)$ is obtained in a similar way, we get that $L(AB) = L(C)$ if and only if $L(ABB) = L(CD)$ and $L(B) = L(D)$.

This "reduction" of one equivalence problem to other instances of that problem is called an "A-transformation."

We can apply this transformation again, and display the result in the form of a tree (Figure 3.1.1). Note that since $L(B) = L(D) = \{b\}$, the nodes labeled B and D become a leaf in the tree indicating, in this case, that this instance of the equivalence problem has been answered in the affirmative. We mark this node with a check.

We recall that these grammars are in GNF and simple so that for each $c \in \Sigma$ and $T \in N_1 \cup N_2$, there is at most one production $T \to c\alpha$ with $\alpha \in N^*$. Consequently, applications of the A-transformation to the pair $T_1\beta_1, T_2\beta_2$ will yield pairs like $\alpha_1\beta_1, \alpha_2\beta_2$ where $T_1 \to c\alpha_1$ and $T_2 \to c\alpha_2$ are productions in the grammars.

In our example, the process of applying the A-transformation never terminates. Pairs AB^{n+1}, CD^n are generated for each $n \geq 0$. We need a new argument to create "reductions", in other words we need a new transformation. Consider ABB and CD. b is a shortest terminal string derived from A (the first symbol in ABB). CD must derive some string that starts with b, unless $L(ABB) \neq L(CD)$. In fact, for the prefix C of CD, $C \overset{*}{\underset{L}{\Rightarrow}} bD$. Hence if $L(ABB) = L(CD)$ then $bL(BB) = bL(DD)$ and therefore $L(BB) = L(DD)$. We can also write $L(ABB) = L(A)L(BB) = L(A)L(DD)$ but $L(ABB) = L(CD)$ so that $L(ADD) = L(CD)$ and because the languages involved are prefix free, $L(AD) = L(C)$. Conversely, if $L(BB) = L(DD)$ and $L(AD) = L(C)$ then

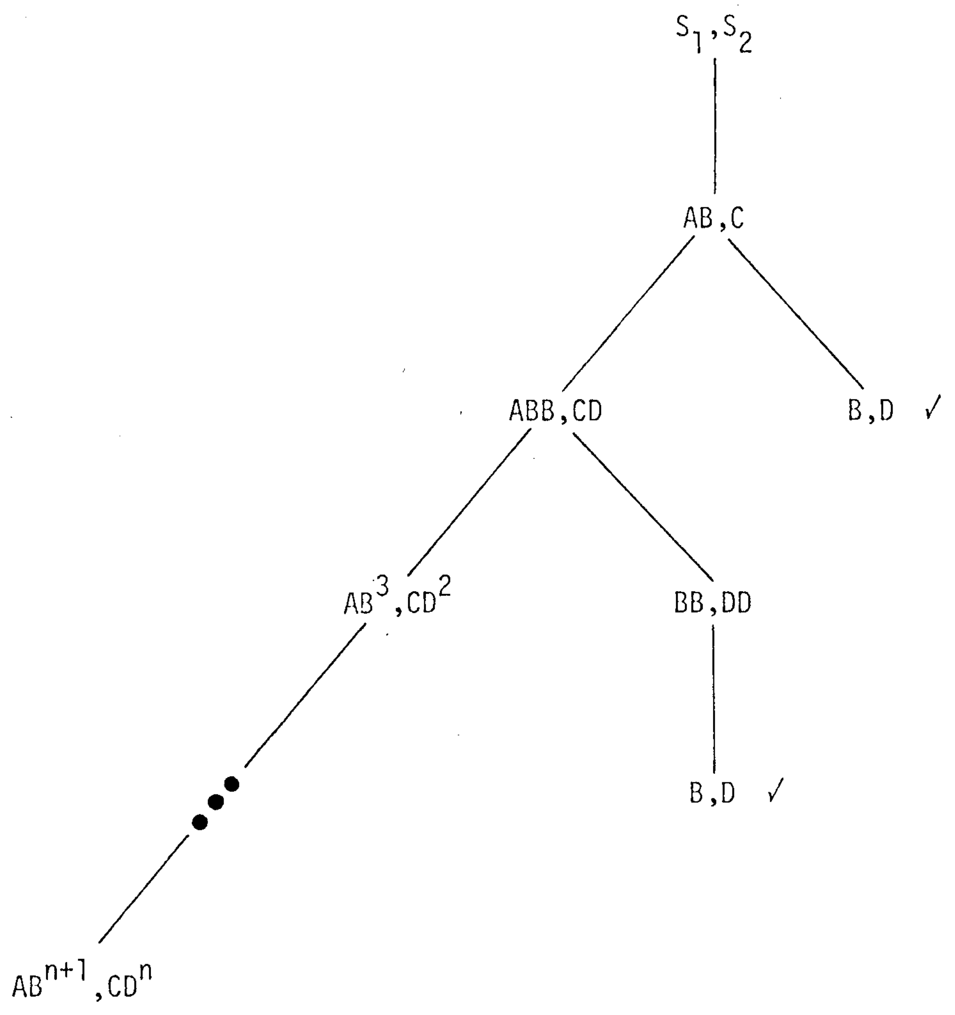


Figure 3.1.1

$L(ABB) = L(A)L(BB) = L(A)L(DD) = L(AD)L(D) = L(C)L(D) = L(CD)$. We have shown that $L(ABB) = L(CD)$ if and only if $L(BB) = L(DD)$ and $L(AD) = L(C)$ (although we have left out many unproven facts). This kind of reduction is called a "B-transformation".

Figure 3.1.2 illustrates a series of applications of A- and B-transformation (the latter are marked with a B). All the leaves in the tree, except one, are check marked as conditions known to be true. The only leaf not check marked is labeled AD,C. However, a node with the same label already appears in the tree as an ancestor of this leaf. This indicates a loop in the derivation and if $L(AD) \neq L(C)$ then the leaf labeled AD,C cannot, if continued, display the shortest conflicting terminal string of the internal node labeled AD,C. We therefore conclude that $L(G_1) = L(G_2)$.

Example 3.1.1 presented some of the ideas behind the algorithm due to Korenjak and Hopcroft [1966] without proof. (A complete proof of a more general case appears elsewhere in this chapter.) This algorithm led to the earliest decidability result of the equivalence problem for a language family that includes nonregular languages. Rosenkrantz and Stearns [1970] showed that the equivalence problem is decidable for the LL languages (which properly contains the simple languages). The algorithm they used was of the machine simulation type (as described earlier). So are the algorithms in some more recent results: decidability of equivalence of nonsingular languages (due to Valiant [1973]), deterministic finite turn languages (Valiant [1973,1974]), deterministic one counter languages (Valiant [1973] and Valiant and Paterson [1975]) and nonsingular versus deterministic

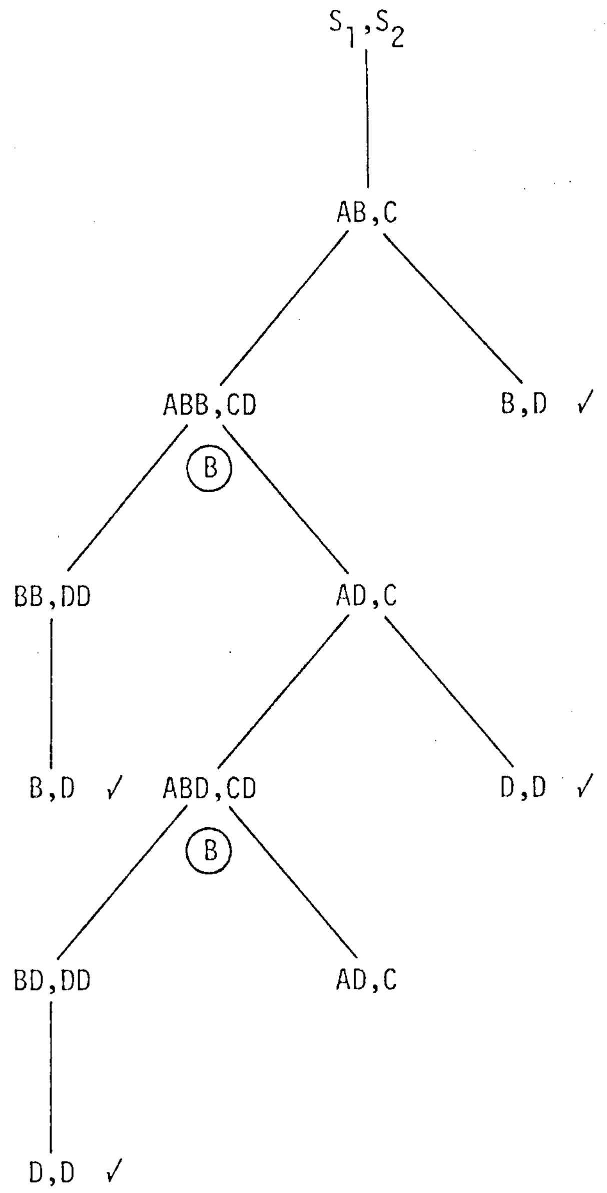


Figure 3.1.2

languages (Taniguchi and Kasami [1976]).

It seems worthwhile to investigate the grammar oriented decision procedure further. First, this algorithm is indicative of the structure of the languages in question whereas the machine simulation algorithm essentially looks at the symmetric difference of the two languages. Also, it may be possible to obtain decidability results for families for which it is unknown whether the equivalence problem is decidable.

The purpose of this chapter is to generalize the ideas in Korenjak and Hopcroft [1966] as much as possible. In section 3.2 we define, in general terms, what a transformation is, what properties it should possess to be useful in the decision procedure. We define strategies which indicate which transformations should be used and when. We discuss the ways in which a tree similar to the one in Example 3.1.1 is devised, and how it is kept finite. A metatheorem about decidability of equivalence is then proved.

Section 3.3 introduces some properties of strict deterministic grammars and real time strict deterministic grammars, cf. Harrison and Havel [1972,1973], which are needed later.

In section 3.4 a variety of transformations, mostly generalizations of those found in Korenjak and Hopcroft [1966] are introduced and some of their properties proved.

Section 3.5 proves, using the metatheorem from section 3.2, that the equivalence problem is decidable for simple language versus deterministic languages. This result was mentioned without proof in Harrison and Havel [1972].

Finally, in section 3.6 we discuss possible extensions of our results. It is hoped that the general presentation of section 3.2 can be tuned to prove decidability of equivalence for various other families of languages.

Section 3.2 - The General Framework for Deciding Equivalence of Grammars

In this section we develop the mechanism for deciding whether two grammars are equivalent. This is done in the most general way possible: Our only assumption will be that the grammars in question are in Greibach Normal Form (GNF) and that no null rules be present. Later in this chapter we will use these tools for deciding equivalence for some particular classes of grammars, but these tools may be used for other classes as well.

Throughout the rest of the chapter we will assume the existence of two GNF grammars with no null rules; let $G_i = (V_i,\Sigma,P_i,S_i)$, $i = 1,2$. We will assume $N_1 \cap N_2 = \emptyset$ where $N_i = V_i - \Sigma$, $i = 1,2$. Let $r+1$ be the maximal length of a right hand side of any production in $P_1 \cup P_2$, then G_1 and G_2 are in standard r-form (cf. Greibach [1965]), $r \geq 1$. We will denote $N = N_1 \cup N_2$, $P = P_1 \cup P_2$. Further, we use $F(Q)$ to denote all finite subsets of a set Q. In particular denote $M = F(N^*)$. Also $M_i = F(N_i)$, $\bar{M} = F(N^+)$, $\bar{M}_i = F(N_i^+)$, $i = 1,2$. Derivations like $\alpha \overset{*}{\Rightarrow} \beta$, $\alpha, \beta \in V^*$ where $V = N \cup \Sigma$, will be understood to use productions from P (i.e. they may include productions from both grammars).

Since we will use finite subsets of N^* quite extensively, we set aside the letters $\eta, \mu, \nu, \zeta, \xi$ to denote them. $\alpha, \beta, \gamma, \delta$ will still be used to denote elements of N^*. We sometimes use an element of N^* to denote the set including only that element.

For every finite subset of N^* we denote the size of the shortest and longest elements as follows. If $\mu \in M$, $\mu - \{\Lambda\} \neq \emptyset$ then $m_\ell(\mu) = \min\{|\beta| \,|\, \beta \in \mu,\ \beta \neq \Lambda\}$ and $m_u(\mu) = \max\{|\beta| \,|\, \beta \in M,\ \beta \neq \Lambda\}$.

Note that Λ is disregarded here. Whenever $m_\ell(\mu) = m_u(\mu)$ is understood we denote $m(\mu) = m_\ell(\mu) = m_u(\mu)$.

Some additional definitions will be needed:

Definition 3.2.1. For each $\alpha \in N^*$ and $\mu \in M$

$$L(\alpha) = \{x \in \Sigma^* \mid \alpha \overset{*}{\underset{L}{\Rightarrow}} x\}$$

and

$$L(\mu) = \{x \in L(\beta) \mid \beta \in \mu\} \quad .$$

Note that productions from both grammars may be used.

Definition 3.2.2. For each $A \in N$, $a \in \Sigma$, $R(A,a) = \{\gamma \mid A \to a\gamma \in P\}$. Note that $R(A,a) \subseteq N^*$. Extend this notion for $\alpha \in N^*$, $x \in \Sigma^*$, $R^*(\alpha,x) = \{\gamma \in N^* \mid \alpha \overset{*}{\underset{L}{\Rightarrow}} x\gamma\}$.

Note that the length of the derivations involved is $|x|$, because our grammars are in GNF and are Λ-free.

We are now ready to define a notion that is central to our discussion.

Definition 3.2.3. For $\mu, \nu \in M$ we say that μ is equivalent to ν, written $\mu \equiv \nu$, if and only if $L(\mu) = L(\nu)$.

It is not too hard to see that $\equiv$ is a congruence relation.

Lemma 3.2.1. Let G_i, $i = 1,2$ be as above. The relation $\equiv$ is a congruence relation on M, i.e. it is an equivalence relation and furthermore for η, μ, ν, ξ in M if $\eta \equiv \mu$ and $\nu \equiv \xi$ then $\eta\nu \equiv \mu\xi$.

Proof. The proof that $\equiv$ is an equivalence relation is immediate.

Suppose $\eta \equiv \mu$ and $\nu \equiv \xi$ for some $\eta, \mu, \nu, \xi \in M$. For each $x \in \Sigma^*$

$$x \in L(\eta\nu)$$

if and only if there exist $y, z \in \Sigma^*$ such that

$$x = yz\ , \quad y \in L(\eta) \text{ and } z \in L(\nu)\ .$$

This holds if and only if

$$x = yz\ , \quad y \in L(\mu) \text{ and } z \in L(\xi)$$

which, in turn, holds if and only if

$$x \in L(\mu\xi)\ .$$

Thus $\eta\nu \equiv \mu\xi$. □

The following constants exist for every grammar (and, in our case, every pair of grammars).

Definition 3.2.4. For each $A \in N$ let $\ell(A) = \min\{|x| \,\big|\, A \overset{*}{\underset{L}{\Rightarrow}} x,$ $x \in \Sigma^*\}$ and let $\ell = \max\{\ell(A) | A \in N\}$.

Lemma 3.2.2. Let G_i, $i = 1,2$ be as above.

(1) If $\alpha \in N^+$ and $|\alpha| = m$ then there exists $x \in \Sigma^*$ such that $\alpha \overset{*}{\underset{L}{\Rightarrow}} x$ and $|x| \leq m\ell$. Moreover, for all $y \in \Sigma^*$ if $|y| < m$ then $\alpha \overset{*}{\not\Rightarrow} y$.

(2) Let $\mu, \nu \in M$. If there exists $\beta \in \nu$ such that for all $\Lambda \neq \alpha \in \mu$, $|\alpha| > \ell|\beta|$ then $\mu \not\equiv \nu$.

(3) Let $x \in \Sigma^*$, $|x| \leq \ell$, $\beta \in N^*$, $|\beta| > \ell$ and suppose $\beta \overset{*}{\underset{L}{\Rightarrow}} x\gamma$ for some $\gamma \in N^*$. We can write $\beta = \beta'\beta''$ with $|\beta'| = \ell$ so that $\beta' \overset{*}{\underset{L}{\Rightarrow}} x\gamma'$, $\gamma = \gamma'\beta''$.

Proof. (1) is a trivial consequence of the fact that the grammars are reduced and Λ-free, and of the definition of ℓ.

If $\beta \in \nu$ then there exists $x \neq \Lambda$, $|x| \leq \ell|\beta|$, $x \in L(\beta) \subseteq L(\nu)$. If $\mu \equiv \nu$ then there must exist $\alpha \in \mu$, $\alpha \neq \Lambda$, $\alpha \overset{*}{\underset{L}{\Rightarrow}} x$, i.e. $|\alpha| \leq |x| \leq \ell|\beta|$. But this contradicts our assumption about all the elements $\alpha \neq \Lambda$ of μ. Hence $\mu \not\equiv \nu$, proving (2).

For (3) factor $\beta \overset{*}{\underset{L}{\Rightarrow}} x\gamma$ as follows: Let $\beta = B_1B_2\cdots B_m$, $B_i \in N$ for all i, $1 \leq i \leq m$ and let $x\gamma = \gamma_1\gamma_2\cdots\gamma_m$ so that $B_i \overset{*}{\underset{L}{\Rightarrow}} \gamma_i$ for all i, $1 \leq i \leq m$. Since the grammars in question are Λ-free, none of the γ_i is empty. Recalling that $x\gamma \in \Sigma^*N^*$ there is a unique j, $1 \leq j \leq m$ such that $\gamma_1,\gamma_2,\ldots,\gamma_{j-1} \in \Sigma^+$, $\gamma_j \in \Sigma^+N^*$ and $\gamma_{j+1},\gamma_{j+2},\ldots,\gamma_m \in N^+$. But in GNF grammars the derivations $B_i \overset{*}{\underset{L}{\Rightarrow}} \gamma_i$ for $j < i \leq m$ must be trivial; that is $B_i = \gamma_i$ (because γ_i does not start with a terminal). Let $\gamma_j = x_j\gamma_j'$ where $x_j \in \Sigma^+$, $\gamma_j' \in N^*$. Then $x = \gamma_1\gamma_2\cdots\gamma_{j-1}x_j$ where each of these j parts has length at least 1. So $j \leq |\gamma_1\gamma_2\cdots\gamma_{j-1}x_j| = |x| \leq \ell$. We now let $\beta' = B_1\cdots B_\ell$, $\beta'' = B_{\ell+1}\cdots B_m$ and $\gamma' = \gamma_j'B_{j+1}\cdots B_\ell$ (or just γ_j if $\ell = j$) and the derivation is as described. $\square$

Corollary. Let $\mu, \nu \in M$. If $m_\ell(\mu) > \ell m_\ell(\nu)$ then $\mu \not\equiv \nu$.

Proof. By definition of m_ℓ, there exists $\beta \in \nu$ with $|\beta| = m_\ell(\nu)$ and for all $\Lambda \neq \alpha \in \mu$, $|\alpha| \geq m_\ell(\mu)$. The result now follows from part (2) of the lemma. $\square$

We need some additional terminology to explore situations where $\mu \not\equiv \nu$.

Definition 3.2.5. Let G_i, $i = 1,2$ be as above, and let $\mu, \nu \in M$. We define the set of witnesses for μ and ν as

$$\bar{W}(\mu,\nu) = \{x \in \Sigma^* \mid x \in L(\mu) \text{ if and only if } x \notin L(\nu)\} \quad .$$

The set of shortest witnesses is

$$W(\mu,\nu) = \{x \in \bar{W}(\mu,\nu) \mid \text{for each } y \in \bar{W}(\mu,\nu),\ |x| \leq |y|\} \ .$$

Thus a witness is a terminal string which distinguishes μ from ν.

We now introduce the building blocks for our decision algorithms.

Definition 3.2.6. A transformation[†] T is a partial function from $M \times M$ into $\{\underset{\sim\sim\sim\sim}{\text{fail}}\} \cup \{U \mid U \subseteq M \times M,\ U \text{ finite and nonempty}\}$. If $T(\mu,\nu) = \underset{\sim\sim\sim\sim}{\text{fail}}$ we say that T failed on (μ,ν). If $T(\mu,\nu)$ is defined and is not $\underset{\sim\sim\sim\sim}{\text{fail}}$ then $T(\mu,\nu) = \{(\mu_1,\nu_1),\ldots,(\mu_m,\nu_m) \mid \mu_i,\nu_i \in M$ for all $1 \leq i \leq m$ where $m \geq 1\}$.

Two important properties of transformation are now introduced which are relevant in determining which transformations are useful in testing for equivalence.

Definition 3.2.7. A transformation T is valid with respect to a set $Q \subseteq M \times M$ if the following conditions hold for each $(\mu,\nu) \in Q$ at which T is defined.

(1) $\mu \equiv \nu$ implies $T(\mu,\nu) = \{(\mu_1,\nu_1),\ldots,(\mu_m,\nu_m)\}$ for some $m \geq 1$ and $\mu_i \equiv \nu_i$ for $1 \leq i \leq m$

and (2) $\mu \not\equiv \nu$ implies that either $T(\mu,\nu) = \underset{\sim\sim\sim\sim}{\text{fail}}$ or there exists $(\mu_i,\nu_i) \in T(\mu,\nu)$ and $\mu_i \not\equiv \nu_i$.

The next property is a little more complicated.

Definition 3.2.8. A transformation T is monotone with respect to a set $Q \subseteq M \times M$ if for each $(\mu,\nu) \in Q$ such that $\mu \not\equiv \nu$ and $T(\mu,\nu) = \{(\mu_1,\nu_1),\ldots,(\mu_m,\nu_m)\}$ and if y is a shortest witness of μ and ν then there exist i, $1 \leq i \leq m$ and some shortest witness x for μ_i and ν_i such that $|x| < |y|$.

Clearly the above definition need not insist that x be a shortest witness.

[†]Not to be confused with grammatical transformations.

An obvious result of the last two definitions is

Lemma 3.2.3. Let T be a transformation which is valid (monotone) with respect to $Q \subseteq M \times M$ then T is also valid (monotone) with respect to any set $Q' \subseteq Q$.

We will usually have a (small) set of transformations which will be repeatedly applied to elements of some set $Q \subseteq M \times M$. The following notion formulates the choice of a specific transformation:

Definition 3.2.9. Let τ be a finite set of transformations and $Q \subseteq M \times M$. A strategy for τ and Q is a computable function S from $M \times M$ to subsets of τ such that for all $(\mu,\nu) \in M \times M$ and $T \in \tau$

(i) If $T \in S(\mu,\nu)$ then $T(\mu,\nu)$ is defined.

(ii) If $(\mu,\nu) \in Q - \{(\emptyset,\emptyset)\}$ then $S(\mu,\nu) \neq \emptyset$.

A strategy specifies which transformation(s) may be applied to a given pair. We want each of the elements of Q (with the exception of $(\emptyset,\emptyset)$ since they are known to be equivalent) to have one or more transformations applicable to it. We allow certain freedom in that $S(\mu,\nu)$ may be a set.

Now we introduce a more global property of transformations and a strategy.

Definition 3.2.10. Let τ be a set of transformations, $Q \subseteq M \times M$ and let S be a strategy for τ and Q. S leaves Q closed under τ if for each $(\mu,\nu) \in Q$ and $T \in S(\mu,\nu)$ such that $T(\mu,\nu) = \{(\mu_1,\nu_1),\ldots,(\mu_m,\nu_m)\}$ then for each $1 \leq i \leq m$, $(\mu_i,\nu_i) \in Q$.

The idea here is that a "good" strategy may only generate pairs that are in Q. Finally we are ready to present the process of using transformations to build a tree.

Definition 3.2.11. Let τ be a set of transformations, $Q \subseteq M \times M$ and S a strategy for τ and Q. A τ, Q, S transformation tree (omit τ, Q, S whenever understood) is a tree with a potentially infinite number of nodes each labelled by an element of $Q \cup \{\text{fail}\}$, where each elementary subtree corresponds to an application of $T \in \tau$ where $T \in S(\mu,\nu)$. If the root of the tree is labeled (μ,ν) we say that it is a transformation tree for μ, ν.

Note that in general not every (μ,ν) may have a transformation tree.

Lemma 3.2.4. Let τ be a set of transformations, $Q \subseteq M \times M$ and S a strategy for τ and Q. If S leaves Q closed under τ then every $(\mu,\nu) \in Q$ has a transformation tree.

Proof. The condition of the lemma guarantees that all the internal nodes of the tree being constructed will be labeled by elements of Q, thus eliminating the only problem that may prevent the existence of a transformation tree. □

We now establish the properties of transformation trees that make them useful in decision procedures.

Lemma 3.2.5. Let τ be a set of valid and monotone transformations with respect to $Q \subseteq M \times M$ and let S be a strategy for τ and Q. Suppose $\mu, \nu \in Q$ and $\mu \not\equiv \nu$ and y is a shortest witness for

μ and ν. For any transformation tree for μ and ν there exists a finite path from the root to a leaf with successive labels

$$(\mu_0,\nu_0),(\mu_1,\nu_1),\ldots,(\mu_t,\nu_t),\underset{\sim\sim\sim\sim}{\text{fail}}$$

where $\mu_0 = \mu$, $\nu_0 = \nu$ and there exist strings $y = y_0, y_1, \ldots, y_t \in \Sigma^*$ such that

(i) $t \leq |y|$

(ii) for each i, $0 \leq i \leq t$, $\mu_i \not\equiv \nu_i$ and y_i is a shortest witness for μ_i and ν_i

and (iii) for each i, $1 \leq i \leq t$, $|y_i| < |y_{i-1}|$.

<u>Proof</u>. The argument is an induction on $|y|$.

<u>Basis</u>: $y = \Lambda$. By the definition of monotonicity, any $T \in \tau$ can only fail on μ and ν. Thus $t = 0 \leq |y|$. The other conditions are trivially satisfied.

<u>Induction Step</u>: Assume the result true whenever $|y| \leq k$ and $k > 0$. Let $|y| = k+1$ where y is a shortest witness for μ, ν. Let $T \in \tau$ be the transformation applied at the root. By validity, either $T(\mu,\nu) = \underset{\sim\sim\sim\sim}{\text{fail}}$ in which case the argument proceeds as in the basis or else $T(\mu,\nu) = \{(\mu_1',\nu_1'),\ldots,(\mu_m',\nu_m')\}$ and for some i, $1 \leq i \leq m$, $\mu_i \not\equiv \nu_i$ with shortest witness y', $|y'| < |y|$ (by monotonicity). Let $(\mu_1,\nu_1) = (\mu_i',\nu_i')$ and $y_1 = y'$. The induction hypothesis applies to (μ_1,ν_1) with witness y_1 and yields the result. $\square$

<u>Corollary</u>. Let τ, Q, S, μ, ν and y be as in Lemma 3.2.5. Then a transformation tree for μ, ν must have a path from the root to a $\underset{\sim\sim\sim\sim}{\text{fail}}$ leaf with no two nodes labeled by the same pair.

Proof. Two different nodes on the same path have shortest witnesses of different lengths. □

Lemma 3.2.5 characterizes transformation trees for inequivalent pairs. The next result deals with equivalent pairs.

Lemma 3.2.6. Let τ be a set of valid and monotone transformations with respect to $Q \subseteq M \times M$ and let S be a strategy for τ and Q. Suppose $\mu, \nu \in Q$, $\mu \equiv \nu$. Then no transformation tree for μ, ν may have a fail leaf.

Proof. The argument is an induction on the level n of a node in a transformation tree for an equivalent pair.

Basis: $n = 0$. The only node in level 0 of a transformation tree for μ, ν is labeled $(\mu,\nu) \neq$ fail.

Induction Step: Suppose that for all $\mu, \nu \in Q$, $\mu \equiv \nu$ and all $n \leq n_0$ where $n_0 > 0$, no node at level n of a transformation tree for μ, ν is labeled by fail. Let $n = n_0 + 1$. Consider a leaf s_n in a tree of height n. This node is on a path $(s_0,\ldots,s_n)$ from the root s_0. Consider the subtree rooted at s_1 which also contains the leaf s_n. Since each $T \in \tau$ is valid, the pair (μ_1,ν_1) which labels s_1 must satisfy $\mu_1 \equiv \nu_1$. The induction hypothesis applies to the subtree so s_n is not labeled fail. □

Because the transformation trees we have been studying can be infinite, it is convenient to define a partial transformation tree as the finite tree which results by taking a transformation tree which has a cross section ξ and taking everything above and including ξ. An example is shown in Fig. 3.2.1.

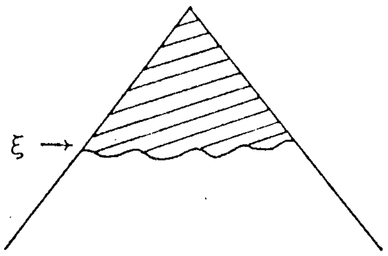


Figure 3.2.1

The shaded area is a partial tree of the full infinite tree.

There are many different ways to form partial trees. Let us establish the convention that no path in the tree is continued if it reaches a node label which has appeared earlier. That includes an earlier appearance as its own ancestor.

Definition 3.2.12. Let τ be a set of transformations, $Q \subseteq M \times M$ and $\mathcal{S}$ a strategy for τ and Q. Let t be a $\tau,Q,\mathcal{S}$-transformation tree. If there exists a cross section ξ of the tree whose nodes are all labeled by elements of Q that appear as labels in nodes above this cross section (or by fail, or $(\emptyset,\emptyset)$), we say that t', the partial tree defined by ξ, is a $\tau,Q,\mathcal{S}$-equivalence tree (omit $\tau,Q,\mathcal{S}$ when understood). t' is said to be the equivalence tree associated with t.

The following result presents a situation where equivalence trees are guaranteed to exist.

Lemma 3.2.7. Let τ be a set of transformations, $Q \subseteq M \times M$ and $\mathcal{S}$ a strategy for τ and Q. If Q is finite then every transformation tree has an equivalence tree associated with it.

Proof. If Q is finite then when the nodes of a transformation tree are generated (say breadth first) we must reach a cross section ξ as described in Definition 3.2.12 after only a finite number of nodes have been created. (Note that the number of non-internal nodes is finite if the number of internal nodes is finite.) $\square$.

The properties of equivalence trees are now explored.

Lemma 3.2.8. Let τ be a set of transformations, Q a finite subset of $M \times M$ and S a strategy for τ and Q. Let t' be an equivalence tree for $(\mu,\nu) \in Q$. Then $\mu \equiv \nu$ if and only if t' does not have any leaf labeled fail.

Proof. Starting from t' we construct a tree t_1' as follows. For each leaf labeled $(\mu',\nu') \neq (\emptyset,\emptyset)$ in the tree such that the internal node labeled (μ',ν') (which must exist) is not an ancestor of that leaf, append to this leaf a copy of the subtree rooted at that internal node. Repeat the process for all such leaves. This process must terminate since it cannot generate paths of length more than $|Q|$. t_1' includes every path from root to leaf of some transformation tree for μ, ν that has no label repeated twice. Moreover t_1' has a fail leaf if and only if t' has a fail leaf.

If $\mu \equiv \nu$ then by Lemma 3.2.6 t_1' contains no fail leaves and t' cannot have any. Conversely, if $\mu \not\equiv \nu$ then t_1' must have a fail leaf by the corollary to Lemma 3.2.5. (Note the condition that no label be repeated along the path.) Therefore t' must have a fail leaf. $\square$

We are now ready to state and prove the main results of this section.

Theorem 3.2.1. Let τ be a finite set of transformations, Q a finite subset of $M \times M$ and S a strategy for τ and Q such that all transformations in τ are valid and monotone with respect to Q, and S leaves Q closed under τ. Then for each $(\mu,\nu) \in Q$ it is decidable whether or not $\mu \equiv \nu$.

Proof. Let (μ,ν) be any member of Q. By Lemma 3.2.4 there exists a transformation tree for (μ,ν) since S leaves Q closed under τ. Furthermore, by Lemma 3.2.7 every such transformation tree t has an equivalence tree t' since Q is finite. By Lemma 3.2.8 $\mu \equiv \nu$ if and only if t' does not have any leaf labeled fail. Since t' has a finite number of nodes and each node's direct descendants are computable, the decidability follows. □

The following is the key result which will be used in subsequent sections.

Theorem 3.2.2. Let G_1, G_2 be two classes of grammars such that every grammar in $G_1 \cup G_2$ is in GNF and has no null rules. Assume that the following is true for every $G_1 \in G_1$, $G_2 \in G_2$ with $G_i = (V_i,\Sigma,P_i,S_i)$, $N_1 \cap N_2 = \emptyset$, $N = N_1 \cup N_2$, $M = F(N^*)$: there exists a finite set τ of transformations, a finite subset Q of $M \times M$ such that $(s_1,s_2) \in Q$, and a strategy for τ and Q such that all transformations in τ are valid and monotone with respect to Q and S leaves Q closed under τ. Then the equivalence problem of G_1, G_2 is decidable.

Proof. This result is a direct corollary of Theorem 3.2.1. □

In order to use Theorem 3.2.2 we will need to specify G_1, G_2, τ, Q and S and prove all the three required properties: validity, monotonicity, and closure of Q under τ.

Section 3.3 - Some Properties of Strict Deterministic and Real Time Strict Grammars

Before we turn to define the specific transformations we will use, we need to specify the grammar domain a little further. While some transformations will be defined for every pair of GNF grammars (in the absence of Λ-rules), others will require a smaller family of grammars in order to have the desired properties. In particular, we will be interested in the family of GNF (without Λ-rules) strict deterministic grammars, cf. Harrison and Havel [1973] and Geller, Harrison and Havel [1976]. Whenever the two grammars $G_i = (V_i,\Sigma,P_i,S_i)$, $i = 1,2$ are specified as such, we will assume the existence of strict partitions π_1, π_2 on V_1, V_2. Recall that $\Sigma \in \pi_1$, $\Sigma \in \pi_2$. Let S be a new symbol not in N. Define $\pi = \pi_1 \cup \pi_2 \cup \{S\}$. In particular if $\pi_i = \{\Sigma,N_{i1},\ldots,N_{im_i}\}$, $i = 1,2$, $m_i \geq 1$ then $\pi = \{\Sigma,N_{11},\ldots,N_{1m_1}, N_{21},\ldots,N_{2m_2},\{S\}\}$. To avoid confusion, the fact that two elements of N are in the same equivalence class of π will be denoted $A \equiv A'$ (mod π), and π will be mentioned. On occasion we will define some new grammars for which π will become a strict partition.

We devote the rest of this section to proving certain properties of the relevant grammar families.

The first part of the following definition is from Harrison and Havel [1973].

Definition 3.3.1. Let α, $\beta \in V^*$ and let A, $B \in V$ such that $A \neq B$. If $\alpha = \gamma A \alpha_1$ and $\beta = \gamma B \beta_1$ then the pair (A,B) is called the distinguishing pair of α and β.

We define the relation $DE \subseteq N^* \times N^*$ as follows:

$DE = \{(\alpha,\beta) \mid$ there exists A,B such that (A,B) is the distinguishing pair of α and β and $A \quad B \pmod{\pi}\}$

(DE stands for Distinguishable by a π-Equivalent pair).

Note that any two strings $\alpha, \beta \in V^*$ have a distinguishing pair unless one of them is a prefix of the other. When it exists, a distinguishing pair is unique.

The definition of DE requires that a distinguishing pair does exist and that this pair appear in the same block of π. So DE is a relation on strings that depends on the grammars in question.

The following facts about DE will be useful:

Lemma 3.3.1. For every $\alpha, \alpha_1, \alpha_2, \beta_1, \beta_2 \in N^*$

(i) $(\beta_1,\beta_2) \in DE$ if and only if $(\alpha\beta_1,\alpha\beta_2) \in DE$

(ii) $(\alpha_1,\alpha_2) \in DE$ implies $(\alpha_1\beta_1,\alpha_2\beta_2) \in DE$

(iii) if $|\alpha_1| = |\alpha_2|$ then

$(\alpha_1\beta_1,\alpha_2\beta_2) \in DE$ implies either $(\alpha_1,\alpha_2) \in DE$
or $\alpha_1 = \alpha_2$ and $(\beta_1,\beta_2) \in DE$

Proof. (i) $(\beta_1,\beta_2) \in DE$ if and only if $\beta_1 = \gamma C\delta_1$, $\beta_2 = \gamma D\delta_2$ and $C \equiv D \pmod{\pi}$. That is true if and only if $\alpha\beta_1 = \gamma' C\delta_1$, $\alpha\beta_2 = \gamma' D\delta_2$ $(\gamma' = \alpha\gamma)$ and $C \equiv D \pmod{\pi}$, which is true if and only if $(\alpha\beta_1,\alpha\beta_2) \in DE$.

(ii) If $(\alpha_1,\alpha_2) \in DE$ then $\alpha_1 = \gamma C\delta_1$, $\alpha_2 = \gamma D\delta_2$ and $C \equiv D \pmod{\pi}$. That implies that $\alpha_1\beta_1 = \gamma C\delta_1'$, $\alpha_2\beta_2 = \gamma D\delta_2'$

$(\delta_1' = \delta_1\beta_1, \delta_2' = \delta_2\beta_2)$ and $C \equiv D \pmod{\pi}$. It implies that $(\alpha_1\beta_1, \alpha_2\beta_2) \in DE$.

(iii) $(\alpha_1\beta_1, \alpha_2\beta_2) \in DE$ implies that $\alpha_1\beta_1 = \gamma C\delta_1$, $\alpha_2\beta_2 = \gamma D\delta_2$, $C \equiv D \pmod{\pi}$. Since $|\alpha_1| = |\alpha_2|$ there are two possible cases depending on the length of γ. If $|\gamma| \geq |\alpha_1| = |\alpha_2|$ then we can write $\gamma = \alpha_1\gamma' = \alpha_2\gamma'$ and $\beta_1 = \gamma' C\delta_1$, $\beta_2 = \gamma' D\delta_2$, $C \equiv D \pmod{\pi}$ which implies $\alpha_1 = \alpha_2$ and $(\beta_1, \beta_2) \in DE$. If $|\gamma| < |\alpha_1| = |\alpha_2|$ then $\alpha_1 = \gamma C\beta_1'$, $\alpha_2 = \gamma D\beta_2'$ (where $\beta_1'\beta_1 = \delta_1$, $\beta_2'\beta_2 = \delta_2$). Since $C \equiv D \pmod{\pi}$ this implies $(\alpha_1, \alpha_2) \in DE$. □

We define properties of finite sets of strings.

Definition 3.3.2. Let $\mu \in M$. μ is said to be a set of associates if for each $\alpha, \beta \in \mu$, $\alpha \neq \beta$, $(\alpha, \beta) \in DE$.

A set of associates is one in which every pair of strings is distinguishable by a π-equivalent pair. The properties of sets of associates stem from properties of sentential forms in strict deterministic grammars, in which the distinguishing pair is π-equivalent.

Definition 3.3.3. Let $\mu \in M$. μ is said to be unambiguous if, for each $x \in \Sigma^*$ there is at most one leftmost derivation from an element of μ to x.

We say μ is prefix free if $L(\mu)$ is prefix free.

The unambiguity of μ depends not only on this set, but on the grammars involved.

As a direct consequence of the definition we get the following result.

Lemma 3.3.2. If μ is a set of associates then so is each subset μ' of μ.

Proof. Trivial. $\square$

The following two lemmas establish some important properties of sets of associates.

Lemma 3.3.3. Let $G_i = (V_i,\Sigma,P_i,S_i)$, $i = 1,2$ be strict deterministic GNF grammars. Let $\mu \in M$ be a set of associates. Then μ is unambiguous and prefix free.

Proof. Define a grammar $G = (V \cup \{S\},\Sigma,P \cup \{S \to \alpha | \alpha \in \mu\},S)$. Recall the partition π on $V \cup \{S\}$. We will show that π is strict for this grammar. First $\Sigma \in \pi$. Now let $A, A' \in N \cup \{S\}$, $\alpha, \beta, \beta' \in V^*$ such that $A \to \alpha\beta$, $A \to \alpha\beta' \in P$ and $A \equiv A' \pmod{\pi}$. The following cases are possible.

Case 1. $A \in N_1$. Then, by definition of π, $A' \in N_1$ and $A \equiv A' \pmod{\pi_1}$. Moreover $A \to \alpha\beta$, $A \to \alpha\beta' \in P_1$ so, since π_1 is strict for G_1, either both $\beta, \beta' \neq \Lambda$ and ${}^{(1)}\beta \equiv {}^{(1)}\beta' \pmod{\pi_1}$ and therefore ${}^{(1)}\beta \equiv {}^{(1)}\beta' \pmod{\pi}$, or $\beta = \beta' = \Lambda$ and $A = A'$. In any event the strictness of π is not violated.

Case 2. $A \in N_2$. This case is symmetrical to the previous one.

Case 3. $A = S$. Then, since S is in a singleton block of π, $A' = S$. Then $\alpha\beta, \alpha\beta' \in \mu$. Since μ is a set of associates $(\alpha\beta,\alpha\beta') \in DE$, and the strictness of π is obeyed.

We have shown that G is strict deterministic. Moreover $L(G) = L(\mu)$ since, for all $w \in \Sigma^*$, $S \overset{*}{\underset{L}{\Rightarrow}} w$ if and only if $\alpha \overset{*}{\underset{L}{\Rightarrow}} w$ for some $\alpha \in \mu$.

Theorem 2.2 in Harrison and Havel [1973] now yields the prefix freeness of $L(\mu)$. The unambiguity of μ follows from Corollary 1 to the Left Part Theorem (Theorem 2.1 in Harrison and Havel [1974]). $\square$

Lemma 3.3.4. Let $\mu \in M$ be a set of associates, α a prefix of some element of μ and $x \in \Sigma^*$. If $\alpha \overset{*}{\underset{L}{\Rightarrow}} x$ then, for all $\alpha'\beta' \in \mu$ with $|\alpha'| = |\alpha|$ and for all $y \in \Sigma^*$ if $\alpha'\beta' \overset{*}{\underset{L}{\Rightarrow}} xy$ then $\alpha' = \alpha$.

Proof. Suppose $\alpha\beta \in \mu$ for some $\beta \in N^*$ (which must exist). By Lemma 3.3.1(iii), since $|\alpha'| = |\alpha|$ and $(\alpha'\beta',\alpha\beta) \in DE$ either $(\alpha',\alpha) \in DE$ or $\alpha' = \alpha$. In the second case we are done. In the first case factor the derivation $\alpha'\beta' \overset{*}{\underset{L}{\Rightarrow}} xy$ to $\alpha' \overset{*}{\underset{L}{\Rightarrow}} x'$, $\beta' \overset{*}{\underset{L}{\Rightarrow}} y'$, $xy = x'y'$. Clearly either $x' = x$ or one of them is a proper prefix of the other. If $\alpha' \neq \alpha$ then $\{\alpha',\alpha\}$ is a set of associates and the relationship between x' and x violates either the unambiguity or the prefix freeness of that set. $\square$

Next we present some operations on sets which leave them as sets of associates.

Lemma 3.3.5. Let $G_i = (V_i,\Sigma,P_i,S_i)$, $i = 1,2$ be strict deterministic GNF grammars. Let $\mu,\nu,\nu_1,\ldots,\nu_s \in M$ be sets of associates, $s \geq 1$. Then

(i) For any $x \in \Sigma^*$, $\bigcup_{\alpha \in \mu} R^*(\alpha,x)$ is a set of associates and if $\alpha_1 \neq \alpha_2$, $\alpha_1, \alpha_2 \in \mu$ then $R^*(\alpha_1,x) \cap R^*(\alpha_2,x) = \emptyset$.

(ii) For any $\beta \in N^*$, $\{\gamma | \beta\gamma \in \mu\}$ is a set of associates.

(iii) For $m \geq 0$, $\{\gamma | \gamma\beta \in \mu$ for some $\beta \in N^*$ and $|\gamma| = m\}$ is a set of associates.

(iv) If $\{\mu_1, \ldots, \mu_s\}$ is a partition of μ, $s \geq 1$ then $\bigcup_{i=1}^{s} \mu_i \nu_i$ is a set of associates.

Proof. We will use the construction of G from Lemma 3.3.3 to prove part (i).

But first let $\gamma_1, \gamma_2 \in \eta$, $\gamma_1 \neq \gamma_2$ where $\eta = \bigcup_{\alpha \in \mu} R^*(\alpha, x)$, $x \in \Sigma^*$. Then for some $\alpha_1, \alpha_2 \in \mu$, $\gamma_1 \in R^*(\alpha_1, x)$ and $\gamma_2 \in R^*(\alpha_2, x)$. Consider the following derivations (in G) $S \underset{L}{\Rightarrow} \alpha_1 \overset{*}{\underset{L}{\Rightarrow}} x\gamma_1$, $S \underset{L}{\Rightarrow} \alpha_2 \overset{*}{\underset{L}{\Rightarrow}} x\gamma_2$. Both derivations are of length $|x| + 1$ so we can use Lemma 2.2 in Harrison and Havel [1973] to obtain that $(x\gamma_1, x\gamma_2) \in DE$. Therefore, by Lemma 3.3.1(i), $(\gamma_1, \gamma_2) \in DE$.

Suppose $\alpha_1 \neq \alpha_2$ and $\gamma_1 = \gamma_2$. Then, since the grammars G_1, G_2 are reduced there exists $y \in \Sigma^*$ such that $\gamma_1 = \gamma_2 \overset{*}{\Rightarrow} y$. Then $\alpha_1 \overset{*}{\underset{L}{\Rightarrow}} xy$, $\alpha_2 \overset{*}{\underset{L}{\Rightarrow}} xy$ contradicting the unambiguity of μ.

To prove (ii) let $\eta = \{\gamma | \beta\gamma \in \mu\}$ for some $\beta \in N^*$. Then let $\gamma_1, \gamma_2 \in \eta$, $\gamma_1 \neq \gamma_2$, $\beta\gamma_1, \beta\gamma_2 \in \mu$ so $(\beta\gamma_1, \beta\gamma_2) \in DE$ hence by Lemma 3.3.1(i) $(\gamma_1, \gamma_2) \in DE$.

For (iii) let $\eta_m = \{\gamma | \gamma\beta \in \mu$ for $\beta \in N^*$, $|\gamma| = m\}$ for some $m \geq 0$. Let $\gamma_1, \gamma_2 \in \eta_m$, $\gamma_1 \neq \gamma_2$. For some $\beta_1, \beta_2 \in N^*$, $\gamma_1\beta_1, \gamma_2\beta_2 \in \mu$ so $(\gamma_1\beta_1, \gamma_2\beta_2) \in DE$ and $|\gamma_1| = |\gamma_2|$. By Lemma 3.3.1(iii), $(\gamma_1, \gamma_2) \in DE$ since they are not equal.

In order to prove (iv) denote $\eta = \bigcup_{i=1}^{s} \mu_i \nu_i$ where $\mu_1, \mu_2, \ldots, \mu_s$ are disjoint subsets of μ such that $\bigcup_{i=1}^{s} \mu_i = \mu$ and each ν_i is a set of associates for $1 \leq i \leq s$. Let $\gamma_1, \gamma_2 \in \eta$, $\gamma_1 \neq \gamma_2$. Then

$\gamma_1 = \alpha_1\beta_1$, $\gamma_2 = \alpha_2\beta_2$ and there exist i_1, i_2, $1 \leq i_1, i_2 \leq s$, $\alpha_1 \in \mu_{i_1}$, $\alpha_2 \in \mu_{i_2}$, $\beta_1 \in \nu_{i_1}$ and $\beta_2 \in \nu_{i_2}$. If $\alpha_1 = \alpha_2$ then, by the disjointness of μ_i, we have $i_1 = i_2$ so that $\beta_1, \beta_2 \in \nu_{i_1} = \nu_{i_2}$. But $\beta_1 \neq \beta_2$ (or else $\gamma_1 = \gamma_2$) so $(\beta_1,\beta_2) \in DE$ hence $(\alpha_1\beta_1,\alpha_2\beta_2) \in DE$. Now assume $\alpha_1 \neq \alpha_2$. Recalling $\alpha_1, \alpha_2 \in \mu$ we get $(\alpha_1,\alpha_2) \in DE$ so that $(\alpha_1\beta_2,\alpha_2\beta_1) \in DE$. □

Next we discuss another family of grammars, which will be used later.

Definition 3.3.4. Let $G = (V,\Sigma,P,S)$ be a strict deterministic GNF grammar (with no Λ-rules), with a minimal strict partition π. G is a canonical real time strict deterministic grammar if, for all $A, A' \in N$, $A \equiv A' \pmod{\pi}$, $\gamma, \gamma' \in N^*$, $A \to a\gamma$, $A' \to a\gamma' \in P$ implies $|\gamma| = |\gamma'|$.

This definition is slightly stronger than that of real time strict deterministic given by Harrison and Havel [1972] so we can state the following.

Lemma 3.3.6. If G is canonical real time strict deterministic then it is real time strict deterministic.

Proof. Follows directly from Definition 3.3.4 above and Definition 2.2 in Harrison and Havel [1972]. □

However, the language families associated with the two families are identical.

Lemma 3.3.7. A language is real time strict deterministic if and only if it is generated by a canonical real time strict deterministic grammar.

Proof. That every canonical real time grammar generates a real time strict language follows from Lemma 3.3.6.

The proof of the converse follows the "only if" part of the proof of Theorem 2.2 in Harrison and Havel [1972]. That proof constructs a real time grammar from a real time DPDA. It is stated there that the grammar is Λ-free and is in GNF. An examination of that proof shows that it is also canonical real time. (Rules $A \to a\gamma$, $A' \to a\gamma' \in P$ for $A \equiv A'$ corresponds to the same move in the DPDA, $\delta(q,a,Z) = (p, Z_k \cdots Z_1)$ where $A = \overline{q,Z,q'}$, $A' = \overline{q,Z,q''}$, and $|\gamma| = |\gamma'| = k$). □

The property of canonical real time grammars is extended to derivations by the following two lemmas.

Lemma 3.3.8. Let $G = (V,\Sigma,P,S)$ be a canonical real time grammar. Let $A, A' \in N$, $A \equiv A' \pmod{\pi}$ where π is the minimal strict partition, and suppose $x \in \Sigma^*$, $\gamma, \gamma' \in N^*$ with $A \overset{*}{\underset{L}{\Rightarrow}} x\gamma$, $A' \overset{*}{\underset{L}{\Rightarrow}} x\gamma'$. Then $|\gamma| = |\gamma'|$.

Proof. An induction on $|x|$ (which is equal to the length of each of the derivations).

Basis: $|x| = 0$. Then $\gamma = A$, $\gamma' = A$ and indeed $|\gamma| = |\gamma'|$.

Induction Step: Assume the result holds for all x, $|x| < n$, $n > 0$.

Let $x \in \Sigma^n$. Then we can write $x = x'a$, $a \in \Sigma$, $x' \in \Sigma^{n-1}$. The derivations in question may be factored as follows:

$$A \overset{*}{\underset{L}{\Rightarrow}} x'\gamma_1 = x'B\gamma_2 \Rightarrow x'a\alpha\gamma_2 = x\gamma$$

$$A' \overset{*}{\underset{L}{\Rightarrow}} x'\gamma_1' = x'B'\gamma_2' \Rightarrow x'a\alpha'\gamma_2' = x\gamma'$$

where $B \to a\alpha$, $B' \to a\alpha' \in P$. By the extension of the properties of a strict partition (Lemma 2.2 in Harrison and Havel [1973]), and since $A \overset{n-1}{\underset{L}{\Rightarrow}} x'B\gamma_2$, $A \overset{n-1}{\Rightarrow} x'B'\gamma_2'$, $B \equiv B' \pmod{\pi}$. By the induction hypothesis (applied to these same subderivations) $|\gamma_2| = |B\gamma_2| - 1$ $= |B'\gamma_2'| - 1 = |\gamma_2'|$. Finally, by the definition of canonical real time (for $B \to a\alpha$, $B' \to a\alpha' \in P$, $B \equiv B' \pmod{\pi}$) implies $|\alpha| = |\alpha'|$. It follows that $|\gamma| = |\alpha| + |\gamma_2| = |\alpha'| + |\gamma_2'| = |\gamma'|$. □

<u>Lemma 3.3.9</u>. Let $G = (V,\Sigma,P,S)$ be a canonical real time grammar with a minimal strict partition π. Let $(\alpha,\alpha') \in DE$, $|\alpha| = |\alpha'|$ and suppose, for some $x \in \Sigma^*$, $\gamma, \gamma' \in N^*$, $\alpha \overset{*}{\underset{L}{\Rightarrow}} x\gamma$, $\alpha' \overset{*}{\underset{L}{\Rightarrow}} x\gamma'$. Then $|\gamma| = |\gamma'|$.

<u>Proof</u>. Define $G' = (V \cup \{S',¢\},\Sigma \cup \{¢\},P \cup \{S' \to ¢\alpha, S' \to ¢\alpha'\},S')$ and let $\pi' = \pi \cup \{S'\}$. It is easy to see that π' is a minimal strict partition for G' (as in Lemma 3.3.3), and G' is canonical real time (since the two new productions satisfy the condition). Clearly, $S' \underset{L}{\Rightarrow} ¢\alpha \overset{*}{\underset{L}{\Rightarrow}} ¢x\gamma$ and $S' \underset{L}{\Rightarrow} ¢\alpha' \overset{*}{\underset{L}{\Rightarrow}} ¢x\gamma'$ are derivations in G'. By Lemma 3.3.8 $|\gamma| = |\gamma'|$. □

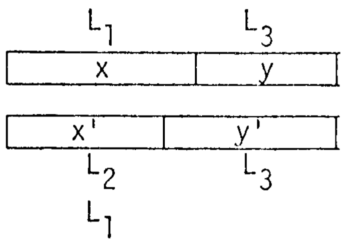
<u>Corollary</u>. Let G and π be as above. Let $\mu = M$ be a set of associates, $m_\ell(\mu) = m_u(\mu)$, $x \in \Sigma^*$, and define $\eta = \bigcup_{\gamma \in \mu} R^*(\gamma,x)$. Then $m_\ell(\eta) = m_u(\eta)$.

<u>Proof</u>. Choose any $\gamma, \gamma' \in \eta$. Then there exists some $\alpha, \alpha' \in \mu$ and $\alpha \overset{*}{\underset{L}{\Rightarrow}} x\gamma$, $\alpha' \overset{*}{\underset{L}{\Rightarrow}} x\gamma'$. But $(\alpha,\alpha') \in DE$ and $\alpha = \alpha'$ (since μ is a set of associates and $m_\ell(\mu) = m_u(\mu)$). So Lemma 3.3.9 can be invoked to show that $|\gamma| = |\gamma'|$. Since this is true for every pair of elements in η, it follows that $m_\ell(\eta) = m_u(\eta)$. □

Finally, we need a certain property of prefix free sets (similar to Lemma 12 in Korenjak and Hopcroft [1966]).

<u>Lemma 3.3.10</u>. Let L_1, L_2, L_3 be nonempty sets of words and suppose L_1 and L_2 are prefix free. If $L_1L_3 = L_2L_3$ then $L_1 = L_2$.

<u>Proof</u>. We proceed by contradiction. Let x be a shortest string such that x is in exactly one of L_1, L_2. Without loss of generality let $x \in L_1$ and $x \notin L_2$. Let y be a shortest word in L_3. Then $xy \in L_1L_3$ and therefore $xy \in L_2L_3$. We can write $xy = x'y'$, $x' \in L_2$ (hence $x' \neq x$) and $y' \in L_3$. By the minimality of $|y|$, $|y'| \geq |y|$ so $|x'| \leq |x|$, but $x' \neq x$ so that in fact $|x'| < |x|$. Hence, by the minimality of $|x|$, $x' \in L_1$. But then x' is a proper prefix of $x \in L_1$, contradicting the hypothesis that L_1 is prefix free.



□

Section 3.4 - Specific Transformation and Their Properties

In this section we will introduce a number of transformations and prove some of their properties. We will try to maintain some balance between the generality of these transformations and the existence of the desirable properties. As a result, some of those properties will be proven later when we specify properties of the grammars. The transformations T_A and T_B are generalizations of the A and B transformations of Korenjak and Hopcroft [1966]. T_ℓ is adapted from the same source.

First we discuss T_A.

Definition 3.4.1. Let $G_i = (V_i,\Sigma,P_i,S_i)$, $i = 1,2$ be two GNF grammars with no null rules. Let $\mu, \nu \in M$, $\mu \neq 0$, $\nu \neq 0$, both $\mu, \nu \in \bar{M}$ (i.e. Λ does not belong to either μ or ν) then we can write

$$\mu = \bigcup_{i=1}^{t} A_i\alpha_i \ , \qquad \nu = \bigcup_{j=1}^{s} B_j\beta_j$$

where A_i, $B_j \in N$, μ_i, $\nu_j \in N^*$. In this case the A-transformation T_A is defined as $T_A(\mu,\nu) = \{(\bigcup_{i=1}^{t} R(A_i,a)\alpha_i, \bigcup_{j=1}^{s} R(B_j,a)\beta_j) \mid a \in \Sigma\}$. $T_A(\mu \cup \{\Lambda\}, \nu \cup \{\Lambda\}) = T_A(\mu,\nu)$. $T_A(\mu,\nu)$ is undefined whenever exactly one of μ, ν includes Λ, or either of μ, ν is empty.

Note that the number of pairs in $T_A(\mu,\nu)$ is exactly $|\Sigma|$ which is independent of μ, ν.

An example will clarify the way T_A works.

Example 3.4.1. Let G_1 have the productions:

$$\begin{aligned} S_1 &\to aAS_1|aA|bAB|cA \\ A &\to aB \\ B &\to b \end{aligned}$$

G_2 has the productions:

$$\begin{aligned} S_2 &\to aC|aS_2D|bDC|b \\ D &\to c \\ C &\to aD \end{aligned}$$

Then

$$T_A(S_1,S_2) = \{(\{AS_1,A\},\{C,S_2D\}),(\{AB\},\{DC,\Lambda\}),(\{A\},\emptyset)\}$$

and

$$T_A(\{AS_1,A\},\{C,S_2D\}) = \{(\{BS_1,B\},\{D,CD,S_2DD\}),(\emptyset,\{DCD,D\}),(\{\emptyset,\emptyset\})\} .$$

Note that the shortest witness for S_1, S_2 is b, while $\{AB\}$, $\{DC,\Lambda\}$ has a shorter witness, Λ.

The properties we desire are quite easy to prove for T_A.

Lemma 3.4.1. For any G_1, G_2 in GNF without null rules T_A is valid and monotone with respect to $M \times M$.

Proof. We assume, without loss of generality, that $\Lambda \notin \mu$, $\Lambda \in \nu$. The argument is subdivided into separate claims.

Claim 1. If $\mu \equiv \nu$ and $T_A(\mu,\nu)$ is defined then $T_A(\mu,\nu) = \{(\mu_1,\nu_1),\ldots,(\mu_m,\nu_m)\}$ for some $m \geq 1$ and $\mu_k \equiv \nu_k$ for each k, $1 \leq k \leq m$.

Proof of Claim 1. Suppose $\mu \equiv \nu$ and $T_A(\mu,\nu)$ is defined. By the definition of T_A, $T_A(\mu,\nu) = \{(\mu_1,\nu_1),\ldots,(\mu_m,\nu_m)\}$ for some $m \geq 1$. In particular we have $\mu = \bigcup_{i=1}^{t} A_i\alpha_i$ and $\nu = \bigcup_{j=1}^{s} B_j\beta_j$ for A_i, $B_j \in N$, α_i, $\beta_j \in M$, and for each k, $1 \leq k \leq m$, $\mu_k = \bigcup_{i=1}^{t} R(A_i,a_k)\alpha_i$, $\nu_k = \bigcup_{i=1}^{s} R(B_j,a_k)\beta_j$ for some $a_k \in \Sigma$.

Now suppose, for some $x \in \Sigma^*$, $x \in L(\mu_k)$. Then, for some $1 \leq i \leq t$ and some $\gamma \in R(A_i,a_k)$, $\gamma\alpha_i \overset{*}{\underset{L}{\Rightarrow}} x$. Therefore, by definition of R, $A_i\alpha_i \underset{L}{\Rightarrow} a_k\gamma\alpha_i \overset{*}{\underset{L}{\Rightarrow}} a_kx$ so that $a_kx \in L(\mu)$. But then $a_kx \in L(\nu)$, so that for some $1 \leq j \leq s$, $B_j\beta_j \overset{*}{\underset{L}{\Rightarrow}} a_kx$. Since no null rules are present it must be the case that $B_j\beta_j \underset{L}{\Rightarrow} a_k\delta\beta_j \overset{*}{\underset{L}{\Rightarrow}} a_kx$ where $B_j \to a_k\delta \in P$ so that $\delta \in R(B_j,a_k)$. We conclude from $\delta\beta_j \overset{*}{\underset{L}{\Rightarrow}} x$ that $x \in L(R(B_j,a_k)\beta_j) \subseteq L(\nu_k)$.

The same argument applied in the other direction completes the proof that $\mu_k \equiv \nu_k$ and establishes Claim 1.

Claim 2. If $\mu \not\equiv \nu$ and $T_A(\mu,\nu) = \{(\mu_1,\nu_1),\ldots,(\mu_m,\nu_m)\}$ for some $m \geq 1$ then for any $y \in \Sigma^+$, $y = ay'$ with $a \in \Sigma$, $y \in \bar{W}(\mu,\nu)$ if and only if there is some k, $1 \leq k \leq m$ such that $y' \in \bar{W}(\mu_k,\nu_k)$.

Proof of Claim 2. Let $\mu = \bigcup_{i=1}^{t} A_i\alpha_i$, $\nu = \bigcup_{j=1}^{s} B_j\beta_j$ for A_i, $B_j \in N$ and α_i, $\beta_j \in N^*$. For $y \neq \Lambda$, $y \in L(\mu)$ if and only if, for some $1 \leq i \leq t$ and $\gamma \in R(A_i,a)$, $A_i\alpha_i \underset{L}{\Rightarrow} a\gamma\alpha_i \overset{*}{\underset{L}{\Rightarrow}} ay' = y$. This holds if and only if, for some i, $y' \in L(R(A_i,a)\alpha_i)$ if and only if $y' \in L(\mu_k)$ for some k such that $a = a_k$.

A similar argument holds for ν: for $y \neq \Lambda$, $y \in L(\nu)$ if and only if $y' \in L(\nu_k)$ where $a = a_k$. Now $y \in \bar{W}(\mu,\nu)$ if and only if $y \neq \Lambda$ and y belongs to exactly one of $L(\mu)$, $L(\nu)$. This holds if, for some k, $a = a_k$ and y' belongs to exactly one of $L(\mu_k)$, $L(\nu_k)$. Finally this is true if for some k, $y' \in \bar{W}(\mu_k,\nu_k)$.

We now prove the Lemma. Claim 1 establishes the first clause of validity (which deals with the case $\mu \equiv \nu$). If $\mu \not\equiv \nu$ and if $T_A(\mu,\nu)$ is defined then any witness of μ, ν is of the form $y = ay'$ (since Λ does not distinguish μ and ν). Then Claim 2 may be used to establish the second clause of validity as well as monotonicity: If $y = ay'$ is a shortest witness for μ, ν then y' is a witness for μ_k, ν_k for some $1 \leq k \leq m$. Hence $\mu_k \not\equiv \nu_k$ and T_A is monotone. □

We would like to show that T_A preserves some properties of the sets. The next lemma shows this for sets of associates.

Lemma 3.4.2. Let G_1, G_2 be two strict deterministic GNF grammars without Λ-rules. Let $T_A(\mu,\nu) = \{(\mu_1,\nu_1),\ldots,(\mu_m,\nu_m)\}$, $m \geq 1$ for some $\mu,\nu,\mu_1,\ldots,\mu_m,\nu_1,\ldots,\nu_m \in M$. If μ (respectively ν) is a set of associates then, for all $1 \leq h \leq m$, μ_h (respectively ν_h) is a set of associates.

Proof. T_A is symmetric in nature, so it is enough to prove the result for μ.

Let $\mu = \bigcup_{i=1}^{t} A_i\alpha_i$ be a set of associates $A_i \in N$, $\alpha_i \in N^*$ for $1 \leq i \leq t$. Consider any μ_h, $1 \leq h \leq m$. $\mu_h = \bigcup_{i=1}^{t} R(A_i,a)\alpha_i$ for some $a \in \Sigma$. We need to show that μ_h is a set of associates,

$$R(A_i,a)\alpha_i = \{\gamma \mid A_i \to a\gamma \in P\}\alpha_i = \{\gamma\alpha_i \mid A_i\alpha_i \overset{*}{\underset{L}{\Rightarrow}} a\gamma\alpha_i\} = R^*(A_i\alpha_i,a) .$$

So that $\mu_h = \bigcup_{\delta \in \mu} R^*(\delta,a)$. By Lemma 3.3.5(i) μ_h is a set of associates. □

Next we deal with the effects of T_A on the length of the strings involved.

Lemma 3.4.3. Let G_1, G_2 be two standard r-form grammars with no Λ-rules. Let $T_A(\mu,\nu) = \{(\mu_1,\nu_1),\ldots,(\mu_m,\nu_m)\}$, $m \geq 1$, for some $\mu,\nu,\mu_1,\ldots,\mu_m,\nu_1,\ldots,\nu_m \in M$. Then for all $1 \leq h \leq m$

$$m_u(\mu_h) \leq m_u(\mu) + r - 1$$

and

$$m_u(\nu_h) \leq m_u(\nu) + r - 1 .$$

Proof. Let $\mu = \bigcup_{i=1}^{t} A_i\alpha_i$. Then for each $1 \leq h \leq m$ there exists some $a \in \Sigma$ such that $\mu_h = \bigcup_{i=1}^{t} R(A_i,a)\alpha_i$. By definition of m_u, $m_u(\mu_h) = |\gamma\alpha_i|$ for some $1 \leq i \leq t$ and some $\gamma \in R(A_i,a)$. However $|\gamma| \leq r$ and $|\alpha_i| = |A_i\alpha_i| - 1 \leq m_u(\mu) - 1$ so $\mu_u(\mu_h) \leq m_u(\mu) + r - 1$. The same argument works for ν. □

T_B may now be presented and its properties examined.

Definition 3.4.2. Let $G_i = (V_i,\Sigma,P_i,S_i)$, $i = 1,2$ be two GNF grammars with no null rules. Let $\mu, \nu \in M$, $\mu \neq \emptyset$, $\nu \neq \emptyset$. If $\Lambda \notin \mu$ and for all $\delta \in \nu$, $|\delta| > \ell$ then we can write

$$\mu = \bigcup_{i=1}^{t} A_i \bigcup_{j=1}^{r_i} \alpha_{ij} \quad \text{where } A_i \in N,\ \alpha_{ij} \in N^+$$

and $A_{i_1} \neq A_{i_2}$ if $i_1 \neq i_2$,

$$\nu = \bigcup_{k=1}^{s} \beta_k'\beta_k'' \quad \text{where } \beta_k' \in N^{\ell},\ \beta_k'' \in N^+ .$$

In this case we define the B-transformation T_B as follows: for all $1 \leq i \leq t$ let x_i be a shortest terminal string derived from A_i. For each $1 \leq i \leq t$, $1 \leq k \leq s$ let $\xi_{i,k} = R^*(\beta_k',x_i) = \{\gamma \in N^* \mid \beta_k' \overset{*}{\underset{L}{\Rightarrow}} x_i\gamma\}$. Then

$$T_B(\mu,\nu) = \{(\bigcup_{j=1}^{r_i}\alpha_{ij}, \bigcup_{k=1}^{s}\xi_{i,k}\beta_k'') \mid 1 \le i \le t\} \cup \{(\bigcup_{i=1}^{t}A_i\xi_{i,k}, \beta_k') \mid 1 \le k \le s\} .$$

We refer to these two sets and their elements as type 1 and type 2 target pairs. Also $T_B(\mu\cup\{\Lambda\},\nu\cup\{\Lambda\}) = T_B(\mu,\nu)$. If μ and ν do not satisfy the above conditions $T_B(\mu,\nu)$ is undefined.

Note that some of the $\xi_{i,k}$ may be empty.

We present an example to illustrate the way T_B works.

Example 3.4.2. Let G_1, G_2 be the following grammars: the productions of G_1 are $S_1 \to aD|aE$, $D \to bE|b|cD$, $E \to bD|cE|c$ and G_2's productions are $S_2 \to aA|aB$, $A \to bBA|c|cA$, $B \to b|bB|cAB$, $\ell = 2$. Let $\mu = \{D,E\}$, $\nu = \{ABA,BAB\}$. Write $\mu = \bigcup_{i=1}^{2}A_i\cdot\Lambda$, $A_1 = D$, $A_2 = E$ and $\nu = \bigcup_{k=1}^{2}\beta_k'\beta_k''$, $\beta_1' = AB$, $\beta_1'' = A$, $\beta_2' = BA$, $\beta_2'' = B$. We let $x_1 = b$, $x_2 = c$ be the shortest terminal strings derived from A_1, A_2. Then $\xi_{1,1} = \{\gamma|\beta_1' \overset{*}{\underset{L}{\Rightarrow}} x_1\gamma\} = \{\gamma|AB \overset{*}{\underset{L}{\Rightarrow}} b\gamma\} = \{BAB\}$, $\xi_{1,2} = \{\gamma|BA \overset{*}{\underset{L}{\Rightarrow}} b\gamma\} = \{A,BA\}$, $\xi_{2,1} = \{B,AB\}$, $\xi_{2,2} = \{ABA\}$. Then

$$T_B(\{D,E\},\{ABA,BAB\}) = \{(\{\Lambda\},\{BABA,AB,BAB\}), (\{\Lambda\},\{BA,ABA,ABAB\}), (\{DBAB,EB,EAB\},\{AB\}),(\{DA,DBA,EABA\},\{BA\})\} .$$

We now turn to prove validity and monotonicity for T_B. While proving these properties was quite simple for T_A, it will be harder in this case, and some restriction on the pairs (μ,ν) will be required.

Lemma 3.4.4. Let $\mu, \nu \in M$ such that $\mu \equiv \nu$ and $T_B(\mu,\nu)$ is defined. Then all type 1 target pairs are equivalent.

Proof. Let $\mu = \bigcup_{i=1}^{t}A_i \bigcup_{j=1}^{r_i}\alpha_{ij}$, $\nu = \bigcup_{k=1}^{s}\beta_k'\beta_k''$ as in the definition of T_B. For a fixed i, let $y \in L(\bigcup_{j=1}^{r_i}\alpha_{ij})$. This holds if and only if $x_i y \in L(A_i \bigcup_{j=1}^{r_i} \alpha_{ij})$. Since $\mu \equiv \nu$ that is true if and only if $x_i y \in L(\beta'_k \beta''_k)$ for some $1 \leq k \leq s$. That is, if and only if there exists $1 \leq k \leq s$ such that $\beta'_k \beta''_k \overset{*}{\underset{L}{\Rightarrow}} x_i y$. Since $|x_i| \leq \ell$, while $|\beta'_k| = \ell$, the above derivation may be factored $\beta'_k \beta''_k \overset{*}{\underset{L}{\Rightarrow}} x_i \gamma \beta''_k \overset{*}{\underset{L}{\Rightarrow}} x_i y$ where $\gamma \in \xi_{i,k}$. So that the derivation exists if and only if, for some k, $1 \leq k \leq s$, and some $\gamma \in \xi_{i,k}$, $y \in L(\gamma \beta''_k)$. This holds if and only if $y \in L(\xi_{i,k} \beta''_k)$ for some k, $1 \leq k \leq s$. Hence $\bigcup_{j=1}^{r_i} \alpha_{ij} \equiv \bigcup_{k=1}^{s} \xi_{i,k} \beta''_k$. □

<u>Lemma 3.4.5</u>. Let $\mu, \nu \in M$ where $\mu = \bigcup_{i=1}^{t} A_i \bigcup_{j=1}^{r_i} \alpha_{ij}$, $\nu = \bigcup_{k=1}^{s} \beta'_k \beta''_k$ as in the definition of T_B. Suppose $\mu \equiv \nu$. Then $\bigcup_{k=1}^{s} \bigcup_{i=1}^{t} A_i \xi_{i,k} \beta''_k \equiv \bigcup_{k=1}^{s} \beta'_k \beta''_k$.

<u>Proof</u>. Let $y \in L(\bigcup_{k=1}^{s} \bigcup_{i=1}^{t} A_i \xi_{i,k} \beta''_k)$. This holds if and only if for some k and i, $1 \leq k \leq s$, $1 \leq i \leq t$, $y_1, y_2 \in \Sigma^*$, $y_1 y_2 = y$, $A_i \overset{*}{\underset{L}{\Rightarrow}} y_1$ and $y_2 \in L(\xi_{i,k} \beta''_k)$. Using Lemma 3.4.4, this is true if and only if for $1 \leq i \leq t$, $y_1, y_2 \in \Sigma^*$, $y_1 y_2 = y$, $A_i \overset{*}{\underset{L}{\Rightarrow}} y_1$ and $y_2 \in L(\bigcup_{j=1}^{r_i} \alpha_{ij})$. That is if and only if for some i, $1 \leq i \leq t$, $y \in L(A_i \bigcup_{j=1}^{r_i} \alpha_{ij})$. This holds if and only if $y \in L(\mu)$ since $\mu = \bigcup_{i=1}^{t} A_i \bigcup_{j=1}^{r_i} \alpha_{ij}$, and since $\mu \equiv \nu$ it follows that $\bigcup_{k=1}^{s} \bigcup_{i=1}^{t} A_i \xi_{i,k} \beta''_k \equiv \nu$ □

The last two lemmas establish the fact that whenever we know that $\bigcup_{k=1}^{s} \bigcup_{i=1}^{t} A_i \xi_{i,k} \beta''_k \equiv \bigcup_{k=1}^{s} \beta'_k \beta''_k$ implies, for all k, $1 \leq k \leq s$, $\bigcup_{i=1}^{t} A_i \xi_{i,k} \equiv \beta'_k$, the first clause of validity holds.

The second clause holds, as the following lemma shows (by proving the contrapositive).

Lemma 3.4.6. Let $\mu, \nu \in M$, $T_B(\mu,\nu)$ is defined. Suppose all the target pairs (types 1 and 2) are equivalent. Then $\mu \equiv \nu$.

Proof. Let $\mu = \bigcup_{i=1}^{t} A_i \bigcup_{j=1}^{r_i} \alpha_{ij}$, $\nu = \bigcup_{k=1}^{s} \beta_k' \beta_k''$ as above. Let $\bigcup_{j=1}^{r_i} \alpha_{ij} \equiv \bigcup_{k=1}^{s} \xi_{i,k}\beta_k''$ for all $1 \leq i \leq t$ and $\bigcup_{i=1}^{t} A_i\xi_{i,k} \equiv \beta_k'$ for all k, $1 \leq k \leq s$. From that last equivalence we can conclude that $\bigcup_{k=1}^{s} \bigcup_{i=1}^{t} A_i\xi_{i,k}\beta_k'' \equiv \bigcup_{k=1}^{s} \beta_k'\beta_k'' = \nu$ (by the fact that $\equiv$ is a congruence relation and properties of unions of sets). We will now consider the left hand side of the last equivalence. We can write $\bigcup_{k=1}^{s} \bigcup_{i=1}^{t} A_i\xi_{i,k}\beta_k'' = \bigcup_{i=1}^{t} A_i(\bigcup_{k=1}^{s} \xi_{i,k}\beta_k'')$ and using the hypothesis about type 2 target pairs $\bigcup_{i=1}^{t} A_i(\bigcup_{k=1}^{s} \xi_{i,k}\beta_k'') \equiv \bigcup_{i=1}^{t} A_i(\bigcup_{j=1}^{r_i} \alpha_{ij}) = \mu$. It follows that $\mu \equiv \nu$. □

Monotonicity will depend on the properties of strict deterministic grammars.

Lemma 3.4.7. Let $G_i = (V_i,\Sigma,P_i,S_i)$, $i = 1,2$ be two strict deterministic GNF grammars without Λ-rules. Let Q_1 be the family of pairs of sets of associates. T_B is monotone with respect to Q_1.

Proof. Let $\mu, \nu \in M$ be sets of associates, $\mu \not\equiv \nu$ and assume that $T_B(\mu,\nu)$ is defined. Further assume $\mu = \bigcup_{i=1}^{t} A_i \bigcup_{j=1}^{r_i} \alpha_{ij}$ and $\nu = \bigcup_{k=1}^{s} \beta_k'\beta_k''$ as above. First we need the following claim.

Claim 1. For all $1 \leq i \leq t$ and $y \in \Sigma^*$, $x_iy \in \bar{W}(\mu,\nu)$ if and only if $y \in \bar{W}(\bigcup_{j=1}^{r_i} \alpha_{ij}, \bigcup_{k=1}^{s} \xi_{i,k}\beta_k'')$.

Proof of Claim 1. $x_iy \in L(\mu)$ if and only if for some j, $1 \leq j \leq r_i$, $A_i\alpha_{ij} \overset{*}{\underset{L}{\Rightarrow}} x_iy$ (from Lemma 3.3.4 and the fact that $A_i \overset{*}{\underset{L}{\Rightarrow}} x_i$). Further, this is true if and only if $y \in L(\bigcup_{j=1}^{r_i} \alpha_{ij})$.

On the other hand, $x_i y \in L(\nu)$ if and only if, for some k, $1 \leq k \leq s$, $\beta'_k \beta''_k \overset{*}{\underset{L}{\Rightarrow}} x_i y$. By definition of $\xi_{i,k}$ this is true if and only if $\beta'_k \beta''_k \overset{*}{\underset{L}{\Rightarrow}} x_i \gamma \beta''_k \overset{*}{\underset{L}{\Rightarrow}} x_i y$, for some $\gamma \in \xi_{i,k}$ and some k, $1 \leq k \leq s$, if and only if $y \in L(\bigcup_{k=1}^{s} \xi_{i,k}\beta''_k)$. Now $x_i y \in \bar{W}(\mu,\nu)$ if and only if exactly one of the following hold: $x_i y \in L(\mu)$, $x_i y \in L(\nu)$. This is true if and only if exactly one of $y \in L(\bigcup_{j=1}^{r_i} \alpha_{ij})$, $y \in L(\bigcup_{k=1}^{s} \xi_{i,k}\beta''_k)$ is true. Finally that is true if and only if $y \in \bar{W}(\bigcup_{j=1}^{r_i} \alpha_{ij}, \bigcup_{k=1}^{s} \xi_{i,k}\beta''_k)$.

Now we examine the set of shortest witnesses for μ, ν, $W(\mu,\nu)$. We distinguish two cases.

Case 1. There exist $x_i y \in W(\mu,\nu)$, for some i, $1 \leq i \leq t$. By Claim 1, $x_i y \in \bar{W}(\mu,\nu)$ implies $y \in \bar{W}(\bigcup_{j=1}^{r_i} \alpha_{ij}, \bigcup_{k=1}^{s} \xi_{i,k}\beta''_k)$ so that one of the target pairs has a shorter witness than $x_i y$.

Case 2. None of the elements of $W(\mu,\nu)$ have x_i as their prefix, for any i, $1 \leq i \leq t$. The following fact will prove helpful.

Claim 2. If Case 2 holds and if m is the length of a shortest witness of μ, ν then for all $1 \leq i \leq t$, $\bigcup_{j=1}^{r_i} \alpha_{ij}$ and $\bigcup_{k=1}^{s} \xi_{i,k}\beta''_k$ agree on all strings of length no more than $m - |x_i|$.

Proof of Claim 2. Let $y \in \bar{W}(\bigcup_{j=1}^{r_i} \alpha_{ij}, \bigcup_{k=1}^{s} \xi_{i,k}\beta''_k)$. Then, by Claim 1, $x_i y \in \bar{W}(\mu,\nu)$. However, by the condition of Case 2, $x_i y$ cannot be a shortest witness for μ, ν so that $|x_i y| > m$. Hence $|y| = |x_i y| - |x_i| > m - |x_i|$, and the claim has been established by proving its contrapositive.

Now let $y \in W(\mu,\nu)$. y is in exactly one of $L(\mu)$, $L(\nu)$ so there are two subcases.

<u>Subcase 2a</u>. $y \in L(\mu)$. Then for some i, $1 \le i \le t$, $y_1, y_2 \in \Sigma^*$, $y = y_1y_2$, $A_i \overset{*}{\underset{L}{\Rightarrow}} y_1$ and $y_2 \in L(\bigcup_{j=1}^{r_i} \alpha_{ij})$. By definition of x_i we have $|y_1| \ge |x_i|$ also $|y| = m$ so that $|y_2| = |y| - |y_1| \le m - |x_i|$. We can therefore use Claim 2 to obtain $y_2 \in L(\bigcup_{k=1}^{s} \xi_{i,k}\beta_k'')$. It follows that $y \in L(A_i\xi_{i,k}\beta_k'')$ for some i, k, $1 \le i \le t$, $1 \le k \le s$. Now we factor $y = y_1'y_2'$, $y_1' \in L(A_i\xi_{i,k})$ and $y_2' \in L(\beta_k'')$. y_1' cannot be in $L(\beta_k')$ for then $y = y_1'y_2' \in L(\beta_k'\beta_k'') \subseteq L(\nu)$ contradicting the fact that y is a witness for μ, ν. We conclude that
$y_1' \in \bar{W}(\bigcup_{i=1}^{t} A_i\xi_{i,k}\beta_k')$ and $|y_1'| < |y|$ (because $y_2' \ne \Lambda$).

<u>Subcase 2b</u>. $y \in L(\nu)$. Then for some k, $1 \le k \le s$, and some $y_1', y_2' \in \Sigma^*$, $y = y_1'y_2'$, $\beta_k' \overset{*}{\underset{L}{\Rightarrow}} y_1'$, $\beta_k'' \overset{*}{\underset{L}{\Rightarrow}} y_2'$. For the sake of contradiction suppose $y_1' \in L(\bigcup_{i=1}^{t} A_i\xi_{i,k})$. Then, for some i, $1 \le i \le t$, $y \in L(A_i\xi_{i,k}\beta_k'')$. Write $y = y_1y_2$, $A_i \overset{*}{\underset{L}{\Rightarrow}} y_1$, $y_2 \in L(\xi_{i,k}\beta_k'')$. But $|y_1| \ge |x_i|$, so $|y_2| = |y| - |y_1| \le m - |x_i|$ yielding (by Claim 2), $y_2 \in L(\bigcup_{j=1}^{r_i} \alpha_{ij})$ so $y = y_1y_2 \in L(A_i \bigcup_{j=1}^{r_i} \alpha_{ij}) \subseteq L(\mu)$. This contradicts the fact that y is a witness for μ, ν and thus establishes that $y_1' \notin L(\bigcup_{i=1}^{t} A_i\xi_{i,k})$ so $y_1' \in \bar{W}(\bigcup_{i=1}^{t} A_i\xi_{i,k}, \beta_k')$ and $|y_1'| \le |y|$.

We have shown that in any event some target pair has a witness which is shorter than the shortest witness of μ, ν. This completes the proof of monotonicity. □

As in the case of T_A, we show that T_B preserves the property of sets being sets of associates. Lemma 3.3.5 will frequently be used in the course of proving this result.

Lemma 3.4.8. Let G_1, G_2 be two strict deterministic GNF grammars without Λ-rules. Let $T_B(\mu,\nu) = \{(\mu_1,\nu_1),\dots,(\mu_m,\nu_m)\}$, for some $\mu,\nu,\mu_1,\dots,\mu_m,\nu_1,\dots,\nu_m \in M$. If μ and ν are sets of associates then so are μ_h, ν_h for each h, $1 \le h \le m$.

Proof. Let $\mu = \bigcup_{i=1}^{t} A_i \bigcup_{j=1}^{r_i} \alpha_{ij}$, $\nu = \bigcup_{k=1}^{s} \beta_k'\beta_k''$ and $\xi_{i,k} = R^*(\beta_k',x_i)$ as in the definition of T_B.

$$T_B(\mu,\nu) = \{(\bigcup_{j=1}^{r_i}\alpha_{ij}, \bigcup_{k=1}^{s}\xi_{i,k}\beta_k'') \mid 1 \le i \le t\} \cup \{(\bigcup_{i=1}^{t} A_i\xi_{i,k},\beta_k') \mid 1 \le k \le s\} .$$

Since the A_i's are distinct we can write, for any i, $1 \le i \le t$, $\bigcup_{j=1}^{r_i}\alpha_{ij} = \{\gamma \mid A_i\gamma \in \mu\}$ so by Lemma 3.3.5(ii) $\bigcup_{j=1}^{r_i}\alpha_{ij}$ is a set of associates. By (iii) of that lemma the set $\eta = \bigcup_{k=1}^{s}\beta_k' = \{\gamma \mid \gamma\delta \in \nu, |\gamma| = \ell\}$ is a set of associates. Therefore, using (i) of the same lemma we get that for each i, $1 \le i \le t$, the set $\bigcup_{k=1}^{s}\xi_{i,k} = \bigcup_{k=1}^{s} R^*(\beta_k',x_i)$ $= \bigcup_{\gamma\in\eta} R^*(\gamma,x_i)$ is a set of associates. Moreover $R^*(\beta_{k_1}',x_i) \cap R^*(\beta_{k_2}',x_i) = \emptyset$ unless $\beta_{k_1}' = \beta_{k_2}'$. If we rewrite ν as $\nu = \{\beta_k'\zeta_k \mid k \in I\}$ where $I \subseteq \{1,2,\dots,s\}$ is a set of indices of all the distinct β_k' and $\zeta_k = \{\beta_{k'}'' \mid \beta_k' = \beta_{k'}'\}$, we can then write $\bigcup_{k=1}^{s}\xi_{i,k}\beta_k'' = \bigcup_{k\in I}\xi_{i,k}\zeta_k$ where $\bigcup_{k\in I}\xi_{i,k}$ is a set of associates, the $\xi_{i,k}$ are disjoint and ζ_k is a set of associates for each $k \in I$ (since $\zeta_k = \{\gamma \mid \beta_k'\gamma \in \nu\}$). Therefore, by Lemma 3.3.5(iv) $\bigcup_{k=1}^{s}\xi_{i,k}\beta_k''$ is a set of associates for each i, $1 \le i \le t$.

Turning to $\bigcup_{i=1}^{t} A_i\xi_{i,k}$ we note that $A_1,A_2,\dots,A_t$ is a partition on $\bigcup_{i=1}^{t} A_i$ which is a set of associates (by Lemma 3.3.5(iii) applied to μ with $m = 1$. Recall the A_i's are distinct). Furthermore

$\xi_{i,k}$ is a set of associates so Lemma 3.3.5(iv) can be invoked to prove that $\bigcup_{i=1}^{t} A_i\xi_{i,k}$ is a set of associates for each $1 \leq k \leq s$.

Finally $\{\beta_k'\}$ is clearly a set of associates for each $1 \leq k \leq s$. So we succeeded in showing that all the sets in question are sets of associates. □

The effect of T_B on the length of strings is now examined.

Lemma 3.4.9. Let G_1, G_2 be two standard r-form grammars with no Λ-rules. Let $T_B(\mu,\nu) = \{(\mu_1,\nu_1),\dots,(\mu_m,\nu_m)\}$, for some $\mu,\nu,\mu_1,\dots,\mu_m,\nu_1,\dots,\nu_m \in M$. Then for all h, $1 \leq h \leq m$

$$m_u(\mu_h) \leq \max\{m_u(\mu)-1, r\ell+1\}$$
$$m_u(\nu_h) \leq \max\{m_u(\nu)+(r-1)\ell, \ell\} \quad .$$

Proof. Let $\mu = \bigcup_{i=1}^{t} A_i \bigcup_{j=1}^{r_i} \alpha_{ij}$, $\nu = \bigcup_{k=1}^{s} \beta_k'\beta_k''$ as in the definition of T_B. Then

$$T_B(\mu,\nu) = \{(\bigcup_{j=1}^{r_i} \alpha_{ij}, \bigcup_{k=1}^{s} \xi_{i,k}\beta_k'') \mid 1 \leq i \leq t\} \cup \{(\bigcup_{i=1}^{t} A_i\xi_{i,k}, \beta_k') \mid 1 \leq k \leq s\} \ .$$

First consider the length of strings in $\xi_{i,k}$. If $\gamma \in \xi_{i,k}$ then $\beta_k' \overset{*}{\underset{L}{\Rightarrow}} x_i\gamma$, where $|\beta_k'| = \ell$, $|x_i| \leq \ell$, and the length of the derivation is $|x_i|$. By induction on $|x_i|$ we can show that $|\gamma| \leq |\beta_k'| + |x_i|(r-1)$ (since each step in the derivation drops one nonterminal and adds up to $r-1$ nonterminals to the sentential form). Hence $|\gamma| \leq \ell + \ell(r-1) = r\ell$. It follows that $m_u(\xi_{i,k}) \leq r\ell$, for all i, k, $1 \leq i \leq t$, $1 \leq k \leq s$. We can now see that for each i, $1 \leq i \leq t$,

$$m_u(\bigcup_{j=1}^{r_i} \alpha_{ij}) \leq m_u(A_i \bigcup_{j=1}^{r_i} \alpha_{ij}) - 1 \leq m_u(\mu) - 1$$

and

$$m_u(\bigcup_{k=1}^{s} \xi_{i,k}\beta_k'') \leq m_u(\bigcup_{k=1}^{s} \xi_{i,k}) + |\beta_k''| \leq r\ell + |\beta_k'\beta_k''| - |\beta_k'|$$
$$\leq r\ell + m_u(\nu) - \ell = m_u(\nu) + (r-1)\ell$$

and for each k, $1 \leq k \leq s$

$$m_u(\bigcup_{i=1}^{t} A_i\xi_{i,k}) \leq 1 + m_u(\bigcup_{i=1}^{t} \xi_{i,k}) \leq 1 + r\ell$$

and

$$m_u(\beta_k') = \ell .$$

For each h, $1 \leq h \leq m$ either $\mu_h = \bigcup_{j=1}^{r_i} \alpha_{ij}$ for some i, $1 \leq i \leq t$ or $\mu_h = \bigcup_{i=1}^{t} A_i\xi_{i,k}$ for some k, $1 \leq k \leq s$. So either $m_u(\mu_h) \leq m_u(\mu) - 1$ or $m_u(\mu_h) \leq r\ell + 1$. In any event $m_u(\mu_h)$ is bounded by the largest of the two. Applying the same argument to $m_u(\nu_h)$ completes the proof of the lemma. □

Next we introduce three simple transformations whose properties are easily proved.

Definition 3.4.3. Let G_i, $i = 1,2$ be two grammars with no Λ-rules and let $\mu, \nu \in M$.

$T_\emptyset$: If exactly one of μ, ν is the empty set then $T_\emptyset(\mu,\nu)$ = fail. $T_\emptyset(\mu,\nu)$ is undefined otherwise.

T_Λ: If $\mu = \nu = \Lambda$ then $T_\Lambda(\mu,\nu) = (\emptyset,\emptyset)$. If exactly one of μ, ν includes Λ then $T_\Lambda(\mu,\nu)$ = fail. Otherwise $T_\Lambda(\mu,\nu)$ is undefined.

T_ℓ: If $m_\ell(\mu) > \ell m_\ell(\nu)$ or $m_\ell(\nu) > \ell m_\ell(\mu)$ then $T_\ell(\mu,\nu) = \underset{\sim\sim\sim\sim}{\text{fail}}$. Otherwise $T_\ell(\mu,\nu)$ is undefined.

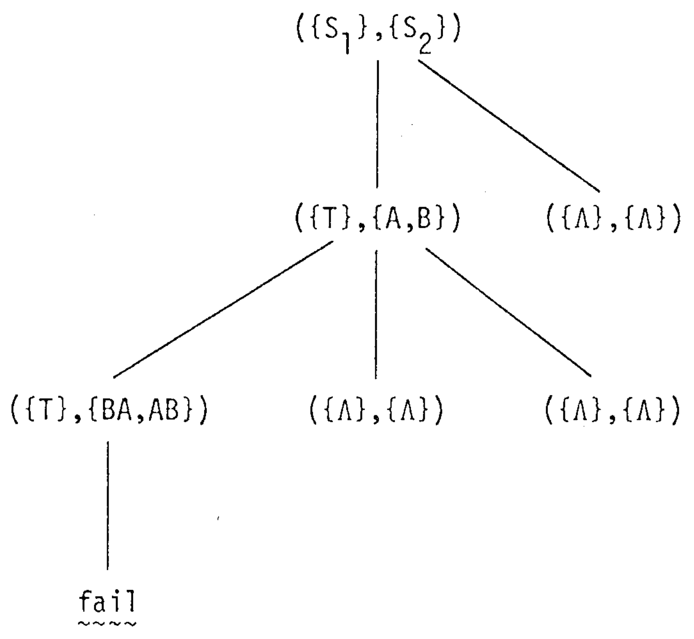
Lemma 3.4.10. $T_\emptyset$, T_Λ, T_ℓ are valid and monotone with respect to $M \times M$.

Proof. $T_\emptyset$ is defined only for a case where $\mu \not\equiv \nu$ (e.g. $\mu = \emptyset$ and $\nu \neq \emptyset$). The first clause of validity is vacuous, the second clause holds since $T_\emptyset(\mu,\nu) = \underset{\sim\sim\sim\sim}{\text{fail}}$ in this case, and monotonicity holds vacuously. If $T_\Lambda(\mu,\nu)$ is defined then if $\mu \equiv \nu$ we have $\mu = \nu = \Lambda$, $T_\Lambda(\Lambda,\Lambda) = (\emptyset,\emptyset)$ and the first clause of validity holds. If $\mu \not\equiv \nu$ (Λ is in one but not the other) then $T_\Lambda(\mu,\nu) = \underset{\sim\sim\sim\sim}{\text{fail}}$ so the second clause holds, and monotonicity holds vacuously. If $T_\ell(\mu,\nu)$ is defined then $m_\ell(\mu) > \ell m_\ell(\nu)$ or $m_\ell(\nu) > \ell m_\ell(\mu)$. By the corollary to Lemma 3.2.1 $\mu \not\equiv \nu$. The proof now parallels the one for $T_\emptyset$. □

An example for T_ℓ may prove helpful.

Example 3.4.3. Let G_1 have the productions $S_1 \to aT|b$, $T \to aT|b|c$. G_2 is the grammar with productions $S_2 \to aA|aB|b$, $A \to b|aBA$, $B \to c|aAB$. We can see that $\ell = 1$. Consider $\mu = \{T\}$, $\nu = \{BA,AB\}$. $m_\ell(\nu) = 2 > 1\cdot 1 = \ell m_\ell(\mu)$ so $T_\ell(\mu,\nu) = \underset{\sim\sim\sim\sim}{\text{fail}}$. Indeed no string of length less than 2 may be derived from ν. T is guaranteed to derive strings of length 1 or less. (In fact $T \overset{*}{\underset{L}{\Rightarrow}} b$.)

We now show how this instance of T_ℓ detects the inequivalence of G_1 and G_2. We start generating a transformation tree for S_1, S_2 by applying T_A whenever possible. We then apply T_ℓ to $\{T\}$, $\{BA,AB\}$.


({S₁},{S₂})
({T},{A,B})
({Λ},{Λ})
({T},{BA,AB})
({Λ},{Λ})
({Λ},{Λ})
fail

Section 3.5 - The Decidability of Equivalence for Deterministic vs. Simple Grammars

Section 3.2 provided the general mechanism for deciding whether two grammars are equivalent. Section 3.4 defined a number of transformations and explores some of their properties in certain generality. We now focus our attention on the families of grammars for which we want to prove that the equivalence problem is decidable. For this section we assume that G_1 is a strict deterministic grammar in standard r-form while G_2 is a simple grammar, cf. Korenjak and Hopcroft [1966], in standard r-form, $r \geq 1$. As before no Λ-rules are present. Recall that a simple grammar is also strict deterministic.

We need to specify the set τ of transformations to be used, the finite subset Q of $M \times M$ which will be the set of labels for our equivalence trees and the strategy that will govern the construction of these trees.

We let $\tau = \{T_A, T_B, T_\Lambda, T_\emptyset, T_\ell\}$.

Define Q as the intersection of the following sets:

$$Q_1 = \{(\mu,\nu) \in M \times M \mid \mu \text{ and } \nu \text{ are sets of associates}\}$$
$$Q_2 = \{(\mu,\nu) \in M \times M \,\big|\, |\nu| \leq 1 \text{ and } \nu \in M_2 = F(N_2^*)\}$$
$$Q_3 = \{(\mu,\nu) \in M \times M \mid m_u(\mu) \leq r\ell + 1 \text{ and } m_u(\nu) \leq r\ell(\ell+1)\} .$$

So each pair (μ,ν) in Q must possess three properties. Both μ and ν must be sets of associates, ν must be a singleton (or empty) and consists of only symbols from the simple grammar and the sizes of strings in μ and ν must be bounded by $r\ell + 1$ and $r\ell(\ell+1)$ respectively. It is this last property that makes Q finite.

The strategy we will employ will force the application of T_Λ, $T_\emptyset$ or T_ℓ whenever they are defined. When none of these transformations is defined, T_A and T_B may be applied (if they are defined) with the only restriction being that T_A may not be used for pairs with long strings.

Formally, denote $\tau' = \{T_\Lambda, T_\emptyset, T_\ell\}$.

Define a strategy S as follows: For all $(\mu,\nu) \in M \times M$

1) for $T \in \tau'$, $T \in S(\mu,\nu)$ if and only if $T(\mu,\nu)$ is defined.

2) $T_A \in S(\mu,\nu)$ if and only if

(a) for all $T \in \tau'$, $T(\mu,\nu)$ is undefined

and (b) $T_A(\mu,\nu)$ is defined

and (c) $m_u(\mu) \leq r(\ell-1)+2$

3) $T_B \in S(\mu,\nu)$ if and only if

(a) for all $T \in \tau'$, $T(\mu,\nu)$ is undefined

and (b) $T_B(\mu,\nu)$ is defined.

Before we discuss the properties of the entities we defined, we have to make sure that S is indeed a strategy. Property (i) from Definition 3.2.9 is easily verifiable. As for property (ii), elements of $Q - \{(\emptyset,\emptyset)\}$ for which neither of T_Λ, $T_\emptyset$ apply are of the form (μ,ν) where $\mu \neq \emptyset$, $\nu \neq \emptyset$ and Λ is in none of them (or in both which is a possibility that may be discarded since $T(\mu \cup \{\Lambda\}, \nu \cup \{\Lambda\}) = T(\mu,\nu)$ for $T \in \{T_\ell, T_A, T_B\}$). We now show that for each such pair at least one of T_ℓ, T_A, T_B applies. If $m(\nu) > \ell m_\ell(\mu)$ then T_ℓ applies. If $\ell < m(\nu) \leq \ell m_\ell(\mu)$ then T_B applies and if $m(\nu) \leq \ell$, then T_A applies.

The only transformation whose validity and monotonicity must be proved is T_B.

Lemma 3.5.1. T_B is valid and monotone with respect to Q.

Proof. Let $(\mu,\nu) \in Q$ such that $\mu \equiv \nu$ and $T_B(\mu,\nu)$ is defined. Then $\mu = \bigcup_{i=1}^{t} A_i \bigcup_{j=1}^{r_i} \alpha_{ij}$ and $\nu = \bigcup_{k=1}^{s} \beta_k' \beta_k''$. Here $s = 1$, so $\nu = \{\beta_1'\beta_1''\}$. By Lemma 3.4.4 all type 1 target pairs are equivalent. By Lemma 3.4.5 $\bigcup_{k=1}^{s} \bigcup_{i=1}^{t} A_i \xi_{i,k} \beta_k'' \equiv \bigcup_{k=1}^{s} \beta_k' \beta_k''$. Rewriting this using $s = 1$ we get

$$\bigcup_{i=1}^{t} A_i \xi_{i,1} \beta_1'' \equiv \beta_1' \beta_1''$$

or

$$\left(\bigcup_{i=1}^{t} A_i \xi_{i,1}\right) \beta_1'' \equiv \beta_1' \beta_1'' \ .$$

$\bigcup_{i=1}^{t} A_i \xi_{i,1}$ is a set of associates (by Lemma 3.4.8) and so is $\{\beta_1'\}$. Therefore $L(\bigcup_{i=1}^{t} A_i \xi_{i,1})$ and $L(\beta_1')$ are prefix free sets (by Lemma 3.3.3). Using Lemma 3.3.10 we conclude that $\bigcup_{i=1}^{t} A_i \xi_{i,1} \equiv \beta_1'$ which is the only type 2 target pair. So T_B is valid.

Lemma 3.4.7 shows that T_B is monotone with respect to Q_1 (the family of pairs of sets of associates). But $Q_1 \subseteq Q = Q_1 \cap Q_2 \cap Q_3$ so using Lemma 3.2.3 we conclude that T_B is monotone with respect to Q.

$\square$

We can now state the properties of all the transformations in question.

Lemma 3.5.2. τ is a set of valid and monotone transformations with respect to Q.

Proof. T_A is valid and monotone with respect to $M \times M$ (Lemma 3.4.1). The same is true for T_Λ, $T_\emptyset$ and T_ℓ (Lemma 3.4.10).

Using Lemma 3.2.3 for T_A, T_Λ, $T_\emptyset$, T_ℓ $(Q \subseteq M \times M)$ and adding the result of Lemma 3.5.1 completes the proof of this lemma. □

Next we need to show the closure property. Again, most of the work has already been done. The only new property needed here is stated in the next lemma.

Lemma 3.5.3. Let G_1, G_2 be as above. Suppose $(\mu,\nu) \in Q_2$. If $T_A(\mu,\nu)$ $(T_B(\mu,\nu))$ is defined and is equal to $\{(\mu_1,\nu_1),\ldots,(\mu_m,\nu_m)\}$, then $(\mu_h,\nu_h) \in Q_2$ for all $1 \le h \le m$.

Proof. Let $(\mu,\nu) \in Q_2$. Then, if T_A (T_B) is defined at this point $\nu = \{\beta\}$, $\beta \in N_2^+$.

First consider T_A. Since $\beta \in N_2^+$, write $\beta = B_1\beta_1$ with $B_1 \in N_2$, $\beta_1 \in N_2^*$. Then, for any $1 \le h \le m$, $\nu_h = R(B_1,a)\beta_1$.
G_2 is simple hence $|R(B_1,a)| \le 1$ (i.e. there is at most one production $B_1 \to a\gamma \in P$), so ν_h is either empty or $\nu_h = \{\gamma\beta_1\}$ where $\gamma\beta_1 \in N_2^*$.

For T_B write $\beta = \beta_1'\beta_1''$, $\beta_1' \in N_2^\ell$, $\beta_1'' \in N_2^*$. Then, for any h, $1 \le h \le m$ two forms are possible for ν_h.

Case 1. $\nu_h = \xi_{i,1}\beta_1''$. Since G_2 is simple $\xi_{i,1} = R^*(\beta_1',x_i)$ is either empty or a singleton hence $|\nu_h| \le 1$, $\nu_h \in M_2$.

Case 2. $\nu_h = \{\beta_k'\}$. Here the result follows trivially. □

We can now put some results together to get the following lemma.

Lemma 3.5.4. S leaves Q closed under τ.

Proof. Let $(\mu,\nu) \in Q$, and suppose some $T \in \tau$ is applied with $T(\mu,\nu) = \{(\mu_1,\nu_1),\ldots,(\mu_m,\nu_m)\}$. Let h be any integer $1 \leq h \leq m$. We would like to show that $(\mu_h,\nu_h) \in Q$. First, since $T(\mu,\nu) \neq$ fail we must have $T \in \{T_A,T_B\}$, or $T = T_\Lambda$, $\mu = \nu = \{\Lambda\}$. The latter case is easily dispatched, so we consider only the former.

$Q \subseteq Q_1$ so we can use Lemma 3.4.2 (for T_A) or Lemma 3.4.8 (for T_B) to show that $(\mu_h,\nu_h) \in Q_1$.

$Q \subseteq Q_2$ so by Lemma 3.5.3 $(\mu_h,\nu_h) \in Q_2$.

Since $T \in \{T_A,T_B\}$, $T_\ell(\mu,\nu)$ is undefined (by definition of S) so that

$$m_u(\nu) = m_\ell(\nu) \leq \ell m_\ell(\mu) \leq \ell m_u(\mu) .$$

Now we need to make a distinction between T_A and T_B.

If $T = T_A$, then, by the definition of S

$$m_u(\mu) \leq r(\ell-1)+2 ,$$

hence

$$m_u(\nu) \leq \ell[r(\ell-1)+2] .$$

Using Lemma 3.4.3, we get

$$m_u(\mu_h) \leq m_u(\mu)+r-1 \leq r(\ell-1)+2+r-1 = r\ell+1$$

and

$$\begin{aligned} m_u(\nu_h) &\leq m_u(\nu)+r-1 \leq \ell[r(\ell-1)+2]+r-1 \\ &= r\ell(\ell+1)-2r\ell+2\ell+r-1 \\ &= r\ell(\ell+1)-(2\ell-1)(r-1) \leq r\ell(\ell+1) \end{aligned}$$

(since $2\ell \geq 1$ and $r \geq 1$).

If $T = T_B$, then (by the fact that $(\mu,\nu) \in Q_2$)

$$m_u(\mu) \leq r\ell+1$$

and hence

$$m_u(\nu) \leq \ell(r\ell+1) .$$

Using Lemma 3.4.9 we get

$$m_u(\mu_h) \leq \max\{r\ell+1-1, r\ell+1\} = r\ell+1$$

and
$$m_u(\nu_h) \leq \max\{\ell(r\ell+1)+(r-1)\ell, \ell\}$$
$$= \max\{r\ell(\ell+1), \ell\} = r\ell(\ell+1)$$

(since $r\ell(\ell+1) \geq \ell$).

In any event $(\mu_h, \nu_h) \in Q_3$. It follows that $(\mu_h, \nu_h) \in Q_1 \cap Q_2 \cap Q_3 = Q$. □

The proof of this lemma presents some facts about the nature of the strategy S (i.e. the way the equivalence tree is constructed). Only for Q_3, namely the bounding of length of strings, did we need to consult the strategy to decide which of T_A and T_B would be used. T_A cannot be applied repeatedly (in some cases) because "long" μ's may be produced, requiring application of T_B.

Also note that the fact that $m_u(\nu) = m_\ell(\nu)$ plays an important role in the proof.

Theorem 3.5.1. For any pair of strict deterministic GNF grammars without Λ-rules G_1 and G_2 such that G_2 is simple it is decidable whether or not $L(G_1) = L(G_2)$.

Proof. Let τ, Q and S be as defined in this section. For each pair (μ,ν) in Q, $m_u(\mu) \leq r\ell+1$ and $m_u(\nu) \leq r\ell(\ell+1)$. Since r and ℓ are constants for any given G_1 and G_2 this implies that strings in μ and ν are bounded in size. Therefore Q is finite.

Clearly $(S_1, S_2) \in Q$.

By Lemma 3.5.2 all the transformations in τ are valid and monotone with respect to Q. By Lemma 3.5.4 S leaves Q closed under τ. We can therefore use Theorem 3.2.2 to obtain our result.

□

Theorem 3.5.2. It is decidable whether two deterministic languages, one of which is simple, are equivalent.

Proof. Let L_1 be a deterministic language (described as deterministic pushdown automaton). Then $L_1\$$ is strict deterministic and by Geller, Harrison and Havel [1976] we can construct a strict deterministic GNF grammar G_1 for $L_1\$$ which will not have any Λ-rules (since $\Lambda \notin L_1\$$). If L_2 is any given simple language then so is $L_2\$$, and a GNF grammar G_2 with no Λ-rules can be constructed for it (from a simple grammar for L_2 or a simple machine). $L_1 = L_2$ if and only if $L(G_1) = L(G_2)$ so using Theorem 3.5.1, we establish the decidability of the equivalence problem. □

Section 3.6 - Possible Extensions

In section 3.5 we have seen that the techniques of Korenjak and Hopcroft [1966] may be extended to decide the equivalence of a (strict) deterministic grammar to a simple grammar. The natural question is: Can we do any better?

If we examine our results and their development we see two problems which may stand in the way of further extension. One is the validity of T_B. We need the condition that $\bigcup_{k=1}^{s} \bigcup_{i=1}^{t} A_i\xi_{i,k}\beta''_k \equiv \bigcup_{k=1}^{s} \beta'_k\beta''_k$ implies, for every k, $1 \leq k \leq s$, $A_i\xi_{i,k} \equiv \beta'_k$. The other stumbling block is the ability to make Q finite. In particular we need to tie $m_u(\nu)$ to $m_\ell(\nu)$, as T_ℓ will guarantee that $m_\ell(\nu)$ will not grow beyond a certain bound. We need to bound the longest string in ν in order to limit the number of such possible sets.

We can overcome the second of these problems for a fairly large family of grammars. We can show that a connection between $m_u(\nu)$ and $m_\ell(\nu)$ exists if G_2 is canonical real time strict.

We replace Q_2 by Q'_2,

$$Q'_2 = \{(\mu,\nu) \in M \times M \mid \nu \in M_2 \text{ and } m_\ell(\nu) = m_u(\nu)\} \quad .$$

Lemma 3.6.1. Let G_1 be a GNF strict deterministic grammar. Let G_2 be a canonical real time strict grammar. For any $(\mu,\nu) \in Q'_2 \cap Q_1$ such that $T_A(\mu,\nu)$ $(T_B(\mu,\nu))$ is defined and is equal to $\{(\mu_1,\nu_1),\ldots,(\mu_m,\nu_m)\}$, $m \geq 1$, $(\mu_h,\nu_h) \in Q'_2$ for all $1 \leq h \leq m$.

Proof. Suppose $T_A(\mu,\nu)$ is defined for $(\mu,\nu) \in Q'_2 \cap Q_1$ and $T_A(\mu,\nu) = \{(\mu_1,\nu_1),\ldots,(\mu_m,\nu_m)\}$, $m \geq 1$. Then write $\nu = \bigcup_{j=1}^{s} B_j\beta_j$ and for each $1 \leq h \leq m$, $\nu_h = \bigcup_{j=1}^{s} R(B_j,a)\beta_j$ for some $a \in \Sigma$. If ν is a set of associates then so is $\bigcup_{j=1}^{s} B_j$ (by Lemma 3.3.5(iii)). Also $m_\ell(\bigcup_{j=1}^{s} B_j) = m_u(\bigcup_{j=1}^{s} B_j) = 1$. So by the corollary to Lemma 3.3.9 it follows that for $\eta = \bigcup_{j=1}^{s} R(B_j, a)$, $m_\ell(\eta) = m_u(\eta)$, that is all the strings in η are of the same length. Since all β_j's are of equal length it follows that the same is true for ν_h: $m_\ell(\nu_h) = m_u(\nu_h)$. It is clear that $\nu_h \in M_2$ so $(\mu_h, \nu_h) \in Q_2'$.

Now consider T_B. Suppose $T_B(\mu,\nu)$ is defined for $(\mu,\nu) \in Q_2' \cap Q_1$, and suppose $T_B(\mu,\nu) = \{(\mu_1,\nu_1),\ldots,(\mu_m,\nu_m)\}$, $m \geq 1$. Let $\mu = \bigcup_{i=1}^{t} A_i \bigcup_{j=1}^{r_i} \alpha_{ij}$, $\nu = \bigcup_{k=1}^{s} \beta_k' \beta_k''$ and $\xi_{i,k} = R^*(\beta_k', x_i)$ as in the definition of T_B. For any h, $1 \leq h \leq m$, either $\nu_h = \bigcup_{k=1}^{s} \xi_{i,k} \beta_k''$ for some $1 \leq i \leq t$ or $\nu_h = \beta_k'$ for some $1 \leq k \leq s$. In the latter case we clearly have $(\mu_h, \nu_h) \in Q_2'$. In the former case we observe the fact that $\bigcup_{k=1}^{s} \beta_k'$ is a set of associates (by Lemma 3.3.5(iii)), and $m_\ell(\bigcup_{k=1}^{s} \beta_k') = m_u(\bigcup_{k=1}^{s} \beta_k') = \ell$ so using the corollary to Lemma 3.3.9 for $\eta = \bigcup_{k=1}^{s} R^*(\beta_k', x_i)$ we get $m_\ell(\eta) = m_u(\eta)$. Since all β_k'''s have equal length (equal to $m_\ell(\nu) - \ell = m_u(\nu) - \ell$) we conclude that $m_\ell(\nu_h) = m_u(\nu_h)$. Clearly $\nu_h \in M_2$ so $(\mu_h, \nu_h) \in Q_2$. $\square$

If we could find a family of languages for which the other condition, namely that $\bigcup_{k=1}^{s} \bigcup_{i=1}^{t} A_i \xi_{i,k} \beta_k'' \equiv \bigcup_{k=1}^{s} \beta_k' \beta_k''$ implies for every k, $1 \leq k \leq s$, $A_i \xi_{i,k} \equiv \beta_k'$ is met, then we would have extended the result further.

Another direction for further research is to look for new transformations, possibly replacing T_B, which are valid and monotone for a certain family of grammars.

CHAPTER 4

A HIERARCHY OF DETERMINISTIC LANGUAGES

Section 4.1 - Introduction

In this chapter we discuss an infinite hierarchy of deterministic languages defined by the number of accepting configurations required by DPDAs accepting them.

We start by introducing, in Section 4.2, a hierarchy of languages using a model similar to a finite state machine. This hierarchy includes languages which are not necessarily context free.

In Section 4.3 we obtain families of deterministic languages as a restriction of the hierarchy presented in Section 4.2. We then characterize these families in terms of the properties of DPDAs accepting the languages. In particular we show that the j-th family in our hierarchy is the family of all languages accepted by a DPDA with no more than j accepting configurations. This result is then used in Section 4.4 to obtain some interesting consequences.

Section 4.2 - U-Automata and the Hierarchy They Induce

We define a model of computation which will play an important role in obtaining a hierarchy of deterministic languages. A similar model was used by Salomaa [1964].

Definition 4.2.1. An Unbounded-automaton (U-automaton for short) is a 5-tuple $A = (Q,\Sigma,\delta,q_0,F)$ where

(i) Q is a countable non-empty set of states,

(ii) Σ is a finite non-empty set of inputs,

(iii) δ is a function from $Q \times \Sigma$ into Q called the direct transition function,

(iv) $q_0 \in Q$ is the initial state,

(v) $F \subseteq Q$ is the set of final states.

Note that this is a generalization of the definition (1.2.9) of a finite automaton. Here Q and F may be infinite sets.

We use the same conventions as in finite automata theory when we discuss the extension of δ to strings in Σ^*, acceptance of a language, etc.

Next we note that despite the deterministic nature of U-automata, they are extremely powerful.

Lemma 4.2.1. Let $L \subseteq \Sigma^*$ be any language. Then there exists an U-automaton A_L such that $L = T(A_L)$.

Proof. Let $A_L = (Q_L,\Sigma,\delta_0,q_0,F)$ where $Q_L = \Sigma^*$, $q_0 = \Lambda$, $F = L$ and δ_0 is defined, for all $x \in \Sigma^*$ and $a \in \Sigma$ as $\delta_0(x,a) = xa$. Clearly then, for each $x, y \in \Sigma^*$, $\delta_0(x,y) = xy$ and in particular $rp(y) = \delta_0(\Lambda,y) = y$ so that $T(A_L) = \{y \mid rp(y) \in F\} = F = L$. $\square$

Note that Lemma 4.2.1 does not even require that the language be recursively enumerable.

Next we discuss two relations on strings. The first is well known from the investigation of regular sets (cf. Hopcroft and Ullman [1969]), and the second was introduced by Geller and Harrison [1977] and is defined here in a broader context.

Definition 4.2.2. Let $L \subseteq \Sigma^*$. We define the right congruence relation induced by L, R_L as follows:

for each $x, y \in \Sigma^*$, $(x,y) \in R_L$ if and only if
for all $z \in \Sigma^*$, $xz \in L$ if and only if $yz \in L$.

The relative right congruence relation induced by L, R_L' is defined as

$$R_L' = R_L \cap L \times L \ .$$

R_L plays an important role in finite automata theory. Nerode's Theorem shows, among other things, that L is regular if and only if $rk(R_L)$ is finite.

The following definitions are adapted from finite automata theory (cf. Hopcroft and Ullman [1969]).

Definition 4.2.3. Let $A = (Q,\Sigma,\delta,q_0,F)$ be a U-automaton. States $q, q' \in Q$ are said to be equivalent (written $q \equiv q'$) if and only if for every $x \in \Sigma^*$, $\delta(q,x) \in F$ if and only if $\delta(q',x) \in F$. If $q, q' \in Q$ are not equivalent then any $x \in \Sigma^*$ such that exactly one of $\delta(q,x)$, $\delta(q',x)$ is in F, is said to distinguish q and q'.

Definition 4.2.4. Let $A = (Q,\Sigma,\delta,q_0,F)$ be a U-automaton. A is reduced if and only if for all $q, q' \in Q$, $q \equiv q'$ implies $q = q'$.

The following result characterizes the connection between R_L and R_L'.

Lemma 4.2.2. Let $L \subseteq \Sigma^*$ and suppose $\bar{L} = \Sigma^* - L$. Then $R_L = R_L' \cup R_{\bar{L}}'$ and $R_L' \cap R_{\bar{L}}' = \emptyset$.

Proof. Suppose $(x,y) \in R_L$. Then, using the definition of R_L with $z = \Lambda$ it follows that either $x, y \in L$ or $x, y \in \bar{L}$. So $R_L = (R_L \cap L\times L) \cup (R_L \cap \bar{L}\times\bar{L})$. Now $R_L \cap L\times L = R_L'$ (by definition). Also $R_L = R_{\bar{L}}$ so $R_L \cap \bar{L}\times\bar{L} = R_{\bar{L}} \cap \bar{L}\times\bar{L} = R_{\bar{L}}'$. It follows that $R_L = R_L' \cup R_{\bar{L}}'$.

The second part of the lemma follows from the fact that $R_L' \subseteq L \times L$, $R_{\bar{L}}' \subseteq \bar{L} \times \bar{L}$ and $L \cap \bar{L} = \emptyset$. $\square$

We can restate the lemma as follows.

Corollary. For all $L \subseteq \Sigma^*$

$$rk(R_L) = rk(R_L') + rk(R_{\bar{L}}') .$$

Next we present a generalization of a result in finite automata theory.

Lemma 4.2.3. For each $L \subseteq \Sigma^*$, $rk(R_L)$ $(rk(R_L'))$ is equal to the smallest number of (final) states in a U-automaton accepting L.

Proof. We present two constructions that establish the result. They are taken from the proof of Nerode's Theorem (cf. Hopcroft and Ullman [1969]) and hence will not be described in much detail.

First let $L \subseteq \Sigma^*$. Then define $A_L = (Q,\Sigma,\delta,q_0,F)$ as follows: $Q = \{[x]_{R_L}\}$, $q_0 = [\Lambda]_{R_L}$, $F = \{[x]_{R_L} | x \in L\} = \{[x]_{R_L'}\}$ and for all $x \in \Sigma^*$, $a \in \Sigma$, $\delta([x]_{R_L},a) = [xa]_{R_L}$. It can be shown that $L = T(A_L)$. It follows that the smallest number of (final) states in a U-automaton accepting L is bounded above by $rk(R_L)$ $(rk(R_L'))$.

Then suppose $A = (Q,\Sigma,\delta,q_0,F)$ is a U-automaton with the smallest number of (final) states such that $L = T(A)$. We can define a relation R as follows: For each $x, y \in \Sigma^*$, $(x,y) \in R$ if and only if $\delta(q_0,x) = \delta(q_0,y)$. It is easy to see that R is a right congruence relation and that $R \subseteq R_L$. Letting $R' = R \cap L \times L$ we get $R' \subseteq R_L'$, and R' is a right congruence relation. Hence $rk(R) \geq$ $\geq rk(R_L)$ and $rk(R') \geq rk(R_L')$. But by definition $rk(R) \leq |Q|$ and $rk(R') \leq |F|$. It follows that the smallest number of (final) states in a U-automaton accepting L is bounded below by $rk(R_L)$ $(rk(R_L'))$.

□

The next lemma establishes a hierarchy of languages.

Lemma 4.2.4. For each i and j, $i > j \geq 1$ there exists a regular language $L_{i,j} \subseteq a^*$ such that $rk(R_{L_{i,j}}) = i$ and $rk(R'_{L_{i,j}}) = j$.

Proof. For each i and j, $i > j \geq 1$ define $L_{i,j} = \{a^{\ell+mi} | 0 \leq \ell < j \text{ and } m \geq 0\}$.

To prove the lemma we will present a U-automaton $A_{i,j}$ with i states, j final states and we will show that $A_{i,j}$ accepts $L_{i,j}$ and is reduced. Let $A_{i,j} = (Q_i,\{a\},\delta_i,0,F_j)$ where $Q_i = \{0,1,\ldots,i-1\}$, $F_j = \{0,1,\ldots,j-1\}$ and for all $0 \leq k < i$, $\delta_i(k,a) = k+1 \bmod i$. It is not hard to see that for every ℓ and m such that $0 \leq \ell < i$

and $m \geq 0$, $\delta(0,a^{\ell+mi}) = \ell$. Hence $T(A_{i,j}) = \{a^t | 0 \leq \delta(0,a^t) < j\}$ $= \{a^{\ell+mi} | 0 \leq \ell < j,\ m \geq 0\} = L_{i,j}$.

To see that $A_{i,j}$ is reduced we will show that for each two states k_1, k_2, $0 \leq k_1, k_2 < i$, there exists a string a^k, $k \geq 0$ that distinguishes k_1 from k_2. Without loss of generality assume $k_1 < k_2$. We consider three cases.

Case 1. $0 \leq k_1 < k_2 < j$. Let $k = j - k_2$. Then $0 = 0+0$ $< k_1 + (j-k_2) = k_1 + k = j + (k_1-k_2) < j+0 = j$ and $k_2 + k = k_2 + j - k_2$ $= j < i$. Thus $\delta_i(k_1,a^k) \in F_j$ but $\delta_i(k_2,a^k) \notin F_j$.

Case 2. $0 \leq k_1 < j \leq k_2 < i$. Let $k = 0$. Then $0 \leq k_1 + k$ $= k_1 < j$ and $j \leq k_2 + k = k_2 < i$. Thus states k_1 and k_2 are distinguished.

Case 3. $j \leq k_1 < k_2 < i$. Let $k = i - k_2$. Then $j = j+0$ $< k_1 + (i-k_2) = k_1 + k = i + (k_1-k_2) < i$ and $k_2 + k = i$, $i \bmod i = 0 < j$. Thus $\delta_i(k_1,a^k) \notin F_j$ while $\delta_i(k_2,a^k) = 0 \in F_j$.

We conclude that for each k_1 and k_2, $0 \leq k_1, k_2 < i$ there exists some $k \geq 0$ such that exactly one of $\delta_i(k_1,a^k)$ and $\delta_i(k_2,a^k)$ is in F_j. So k_1 and k_2 are indeed distinguishable.

This completes the proof of the lemma. $\square$

We will be interested in a restriction of the hierarchy described by Lemma 4.2.4. We define families of languages as follows.

Definition 4.2.5. Let Σ be a finite alphabet. For each $j \geq 1$ define $\mathcal{L}_j = \{L \subseteq \Sigma^* | rk(R_L') \leq j\}$.

The main result of the section can now be stated.

Theorem 4.2.1.

(i) $\emptyset \neq \mathcal{L}_1 \subsetneq \mathcal{L}_2 \subsetneq \cdots$

(ii) For each $j \geq 1$ there exists a regular language L which is in $\mathcal{L}_{j+1}$ but not in $\mathcal{L}_j$.

Proof. (ii) follows from Lemma 4.2.4 using $L = L_{j+2,j+1}$. (Recall that $L_{i,j}$ is defined for $i > j \geq 1$.) (i) is a direct consequence of (ii). □

Section 4.3 - Restriction to Deterministic Languages

Section 4.2 established a hierarchy of languages defined by the number of final states in a U-automaton accepting them. In this section we obtain a hierarchy of deterministic languages defined by the number of accepting configurations in a DPDA accepting them.

First we define families of deterministic languages.

Definition 4.3.1. Let Σ be a finite alphabet. For $j \geq 1$ define $\mathcal{D}_j = \mathcal{L}_j \cap \Delta_0$, where Δ_0 is the family of all deterministic languages.

Next a hierarchy is obtained as in Section 4.2.

Lemma 4.3.1. $\emptyset \neq \mathcal{D}_1 \subsetneq \mathcal{D}_2 \subsetneq \cdots$

Proof. The lemma is a direct result of Theorem 4.2.1, Definition 4.3.1 and the fact that all the regular languages are contained in Δ_0. □

The families $\mathcal{D}_j$ are defined in terms of $\mathrm{rk}(R_L')$. We would like to have a characterization that will more closely reflect the way DPDAs accept these languages.

As a first step we want to show that Δ_1 is exactly the family of languages that are in $\mathcal{D}_j$ for some $j \geq 1$. This is essentially the result of Theorem 4.2 of Geller and Harrison [1977]. We will prove it here using a modified version of Valiant's regularity test.

Familiarity with Valiant [1975] is required since we use many of the concepts from that paper. We repeat here only the most relevant definitions.

Valiant's choice of symbols is different from ours. Since our presentation closely follows his, we try to transliterate the names in a consistent fashion. The following table provides the correspondence between the two systems of notation.

Valiant's Notation	Our Notation
s	q
w	α
q	$\|Q\|$
α	x
ε	Λ (in Σ^*)
Ω	Λ (in Γ^*)
Υ	t
γ	y

Let $M = (Q,\Sigma,\Gamma,\delta,q_0,Z_0,F)$ be a DPDA. We use c_s to denote the initial configuration (q_0,Z_0). For two configurations c and c' we write $c \equiv c'$ if they are equivalent[†] and $c \not\equiv c'$ otherwise.

If $c = (q,\alpha)$ for some $q \in Q$ and $\alpha \in \Gamma^*$ then $|\alpha|$ is the <u>height</u> of c, denoted $|c|$.

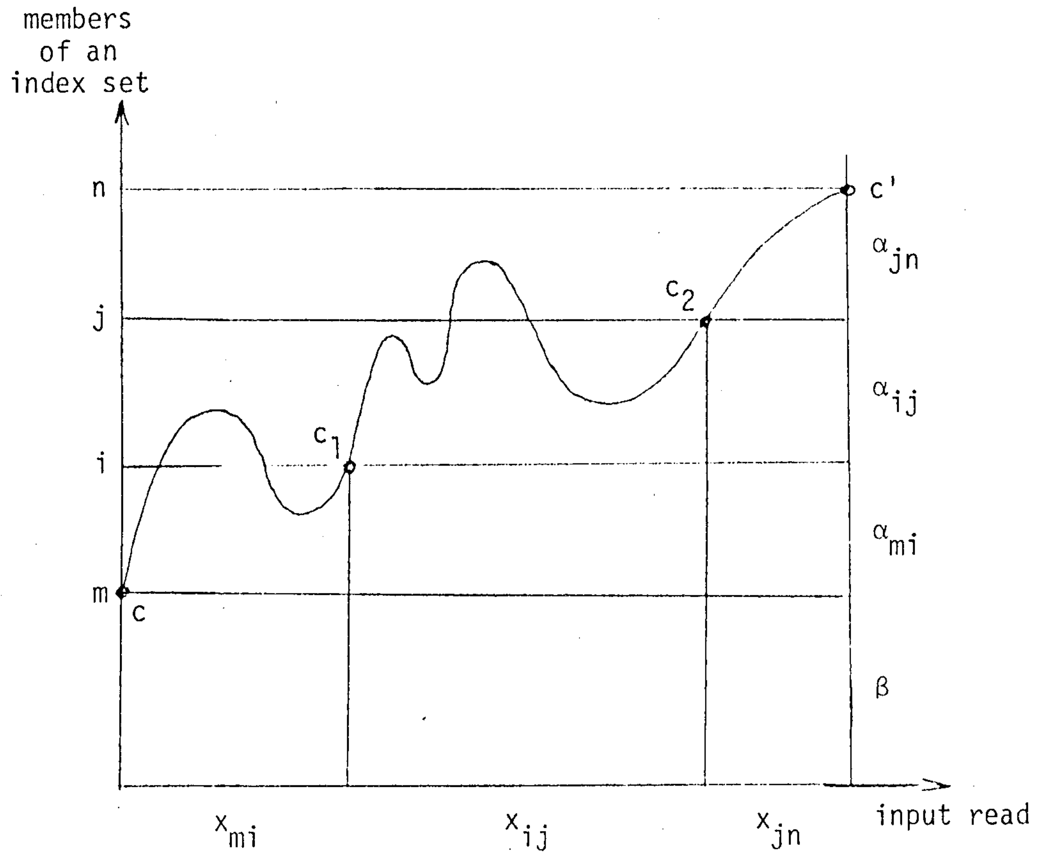
<u>Definition 4.3.2</u>. All the following definitions are assumed to be with respect to some particular derivation $c \xrightarrow{x} c'$. c_1 is a <u>stacking</u> (s-) <u>configuration</u> in the derivation if and only if it is not followed subsequently by any configuration of height $\leq |c_1|$. It is a <u>popping</u> (p-) <u>configuration</u> if and only if it is not preceded by any configuration of height $\leq |c_1|$. A derivation is a <u>stacking derivation</u> ($c \uparrow (x)\ c'$) if and only if c is an s-configuration in it, a <u>popping derivation</u> ($c \downarrow (x)\ c'$) if and only if c' is a p-configuration in it. An <u>index set</u> N is an ordered sequence of nonnegative integers. It

[†]Cf. Definition 1.2.8.

induces <u>segments</u> α_{ij} in a stack α, where α_{ij} is the substring of α from the (i+1)-st letter to the j-th inclusive, and $i,j \in N$. N also induces s- or p-configurations in a derivation, namely those of heights $i+1$, and i respectively, if defined, for each $i \in N$. See Figures 4.3.1 and 4.3.2.

By definition, in any derivation there can be at most one s-configuration, and one p-configuration, of any given height. If $i < j$ then the string x_{ij} will denote that substring of x read in the part of the derivation from the s-configuration of height $i+1$ to the one of height $j+1$. Notice that if these two have the same state and top of stack symbol, then by iterating the substring x_{ij} in the input k times, we obtain configurations like c', but with the stack segment α_{ij} iterated k times in the stack. If $i < j$ then x_{ji} will denote that substring of x read from the p-configuration of height j to the one of height i. Where defined, it will be convenient to say that in the x derivation "the string x_{ji} <u>pops</u> the segment α_{ij}." $(q,\alpha) = c \downarrow (x)\ c'$ is a <u>j-derivation</u> with respect to N if and only if fewer than j segments of α induced by pairs of integers consecutive in N, are popped by nonnull substrings of x.

The segment α' is <u>ℓ-invisible</u> in $(q,\alpha\alpha'\alpha")$ with respect to N if and only if for any q' and any ℓ-derivation $(q,\alpha\alpha'\alpha") \downarrow (x)$ $(q',\alpha\alpha')$, it is the case that $(q',\alpha\alpha') \downarrow (\Lambda)\ (q',\alpha)$. If α' is ℓ-invisible then the configurations $(q,\alpha\alpha'\alpha")$ and $(q,\alpha\alpha")$ may only be distinguished (i.e. the existence of α' on the stack may only be detected) by derivations that are not ℓ-derivations. In other words,



$$c \vdash (x_{mi})\ c_1 \vdash (x_{ij})\ c_2 \vdash (x_{jn})\ c'$$

$c' = (q',\alpha) \qquad \alpha = \beta\alpha_{mi}\alpha_{ij}\alpha_{jn}$

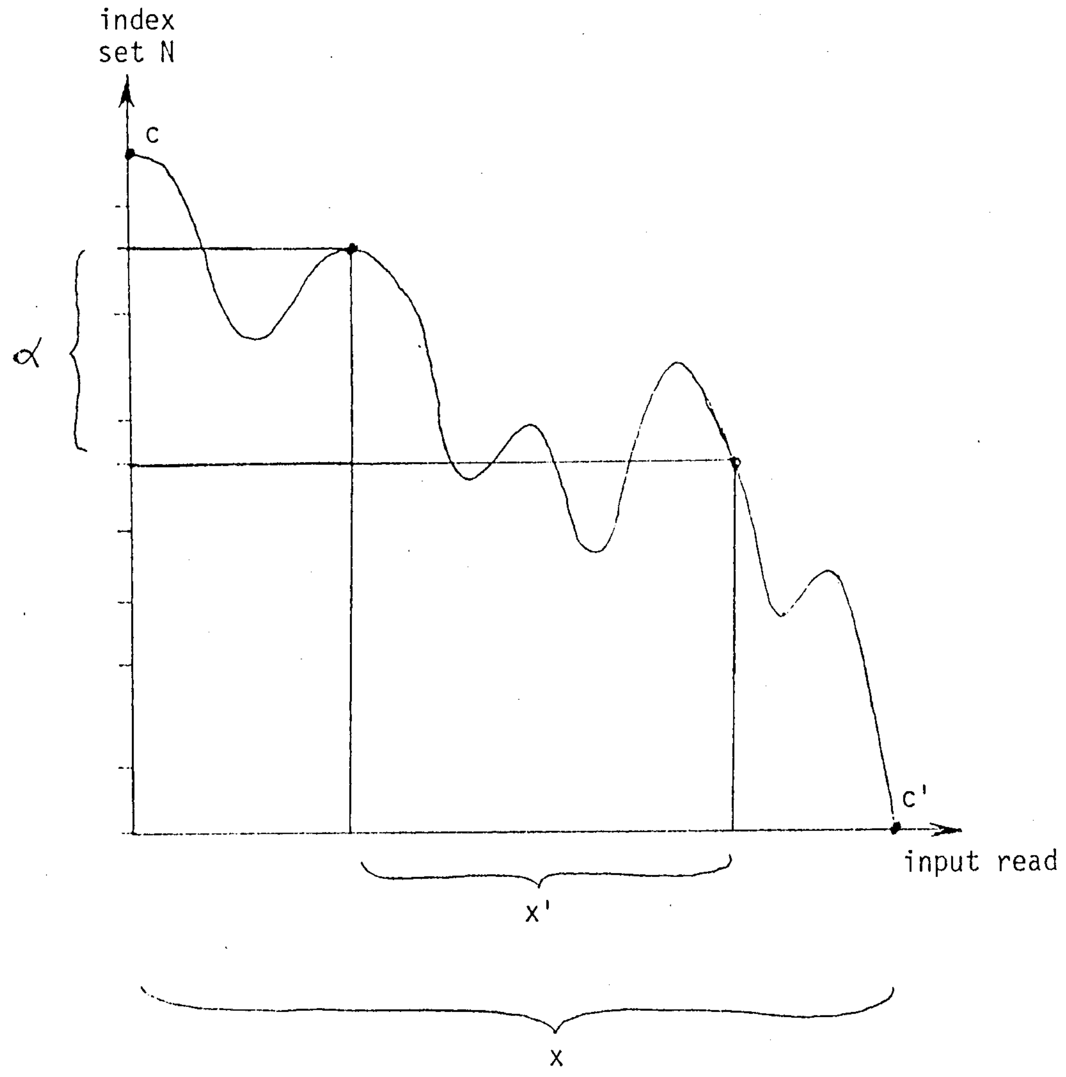
$c_1 = (q_1,\beta\alpha_{mi})$ is an s-configuration

$c_2 = (q_2,\beta\alpha_{mi}\alpha_{ij})$ is an s-configuration

$c = (q,\beta)$

If $q_1 = q_2$ then $c \vdash (x_{mi}x_{ij}^{k}x_{jn})(q',\beta\alpha_{mi}\alpha_{ij}^{k}\alpha_{jn})$

Figure 4.3.1 - Stacking derivations



$c \downarrow (x)\ c'$

x' pops α'

If fewer than j segments of x are nonnull then this is a j-derivation.

Figure 4.3.2 - Popping derivations

only derivations that pop at least ℓ disjoint segments of α'' by nonnull input strings may distinguish $(q,\alpha\alpha'\alpha'')$ and $(q,\alpha\alpha'')$.

α is <u>null-transparent</u> if and only if for all $q, q' \in Q$,

$$(q,\alpha) \downarrow (\Lambda)\ (q',\Lambda) \Rightarrow (q',\alpha) \downarrow (\Lambda)\ (q',\Lambda) .$$

If α is null-transparent then if $(q,\alpha) \downarrow (\Lambda)\ (q',\Lambda)$ then for all $m \geq 1$, $(q,\alpha^m) \downarrow (\Lambda)\ (q',\Lambda)$. Thus if one copy of α in a stack $(\alpha_1\alpha^m)$ is popped by Λ then so are all the rest, and the state that is finally reached will be independent of m.

For a discussion of ℓ-invisible and null transparent segments see Valiant [1975].

Next we prove a lemma that will enable us to obtain the result about Δ_1. The lemma essentially strengthens the main claim of Theorem 4 in Valiant [1975]. Valiant shows that if $c = (q,\alpha)$ is a reachable configuration with a large height then either there exists an equivalent shorter configuration or there is an infinite set of pairwise inequivalent configurations. We include the requirement that all the configurations in that infinite set have the state q.

<u>Lemma 4.3.2</u>. Let M be a DPDA. There exists a number t (depending upon M) such that for each reachable configuration c of M with height greater than t

either (i) there exists a reachable configuration c' with smaller height so that c' is equivalent to c

or (ii) there is an infinite collection of pairwise disjoint reachable configurations all having the same state as c.

Proof. We follow the proof of Theorem 4 of Valiant [1975] very closely.

Suppose $c = (q,\alpha)$, $|\alpha| = n > t$. $c_s \xrightarrow{x} c$. Valiant constructs configuration $c' = (q,\alpha_{0i}\alpha_{jn})$† where $\alpha = \alpha_{0i}\alpha_{ij}\alpha_{jn}$, and α_{ij} is $|Q||Q|!$-invisible in c. Assuming $c \not\equiv c'$, Valiant produces configurations c_1, c_2, c_1' and c_2' such that it is impossible that both $c_1 \equiv c_2$ and $c_1' \equiv c_2'$. Without loss of generality it is assumed that $c_1 \not\equiv c_2$. Valiant also presents the strings x_{0k}, x_{km}, x_{mn}, y_{nm} and y_{mk} in Σ^*†† such that $x = x_{0k}x_{km}x_{mn}$. It is shown that $c \downarrow (y_{nm})\ c_2$ and $c_2 \downarrow (y_{mk})\ c_1$. c_1 and c_2 have the same state, say $q_1 \in Q$. Then $c_1 = (q_1,\alpha_{0k})$ and $c_2 = (q_1,\alpha_{0k}\alpha_{km})$, $c = (q,\alpha_{0k}\alpha_{km}\alpha_{mn})$, so $(q,\alpha_{mn}) \downarrow (y_{nm})\ (q_1,\Lambda)$ and $(q_1,\alpha_{km}) \downarrow (y_{mk})\ (q_1,\Lambda)$. We define, for each $r \geq 0$, $c_0(r) = (q,\alpha_{0k}\alpha_{km}^r\alpha_{mn})$ and $c_1(r) = (q_1,\alpha_{0k}\alpha_{km}^r)$.†††

Then, for each $r \geq 0$, $c_s \xrightarrow{x_{0k}x_{km}^r x_{mn}} c_0(r)$, $c_0(r) \downarrow (y_{nm})\ c_1(r)$ and for all $\ell \geq 0$, $c_1(\ell+r) \downarrow (y_{mk}^{\ell})\ c_1(r)$. Note that $c_1 = c_1(0)$ and $c_2 = c_1(1)$.

Claim 1. For all $r \geq 0$, $c_1(r) \not\equiv c_1(r+1)$.

Proof of Claim 1. Let $r \geq 0$. $c_1(r) \downarrow (y_{mk}^r)\ c_1(0) = c_1$ and $c_1(r) \downarrow (y_{mk}^r)\ c_1(1) = c_2$. Since $c_1 \not\equiv c_2$ we must conclude that $c_1(r) \not\equiv c_1(r+1)$.

Let ℓ_0 be the length of a shortest string distinguishing $c_1(0) = c_1$ and $c_1(1) = c_2$.

---

†In Valiant [1975], $c = (s,\omega)$ and $c' = (s,\omega_{0i}\omega_{jn})$.

††Called α_{0k}, α_{km}, α_{mn}, γ_{nm} and γ_{mk} respectively by Valiant.

†††New configurations not discussed by Valiant.

Claim 2. For all $\ell > \ell_0$, $c_1(1) \not\equiv c_1(\ell+1)$.

Proof of Claim 2. Suppose that for some $\ell > \ell_0$, $c_1(1) \equiv c_1(\ell+1)$. $c_1(1) \downarrow (y_{mk})\ c_1(0)$ and $c_1(\ell+1) \downarrow (y_{mk})\ c_1(\ell)$ so it follows that $c_1(\ell) \equiv c_1(0)$. Let z be any string of length ℓ_0 that distinguishes $c_1(0)$ and $c_1(1)$. z must distinguish $c_1(\ell)$ and $c_1(\ell+1)$ but $|z| = \ell_0 < \min(\ell,\ell+1)$ and by Lemma 2 of Valiant [1975] and the fact that α_{km} is null transparent, z cannot distinguish them. The contradiction proves the claim.

From Claim 2 and the fact that $c_0(r) \xrightarrow{y_{nm}y_{mk}^{r-1}} c_1(1)$ and $c_0(\ell+r) \xrightarrow{y_{nm}y_{mk}^{r-1}} c_1(\ell+1)$ it follows that $c_0(r) \not\equiv c_0(\ell+r)$ for all $r \geq 1$ and $\ell > \ell_0$.

Finally we conclude that $\{c_0(d(\ell_0+1)+1) | d \geq 0\}$ is an infinite family of pairwise disjoint reachable configurations whose state is the same as that of c.

We have shown that if (i) of the lemma does not hold, then (ii) must be true. □

We will need the following result about ℓ-invisible segments. It essentially shows that ℓ-invisible segments depend only on the top portion of the stack and the state.

Lemma 4.3.3. Let $\ell \geq 0$. Suppose $c = (q, \alpha\alpha'\alpha'')$, $c_1 = (q, \alpha_1\alpha'\alpha'')$ and let N and N_1 be two index sets with the property that when N and N_1 are applied to c and c_1 respectively they induce the same segments in $\alpha'\alpha''$.

Then α' is ℓ-invisible in c with respect to N if and only if α' is ℓ-invisible in c_1 with respect to N_1.

Proof. Follows from the definition of ℓ-invisible segments and the fact that $(q, \alpha\alpha'\alpha'') \downarrow (x)\ (q', \alpha\alpha')$ if and only if $(q, \alpha_1\alpha'\alpha'') \downarrow (x)\ (q', \alpha_1\alpha')$. $\square$

Another result we will need relates the relation R_L' and configurations in a DPDA accepting L.

Lemma 4.3.4. Let $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ and $L = T(M)$. Suppose M has j pairwise inequivalent reachable accepting configurations. Then $rk(R_L') \geq j$.

Proof. Let $c_1, c_2, \ldots, c_j$ be pairwise inequivalent reachable accepting configurations. Then there exist $x_1, x_2, \ldots, x_j \in L$ so that $(q_0, Z_0) \xrightarrow{x_i} c_i$ for all i, $1 \leq i \leq j$. Also, for each $1 \leq i_1, i_2 \leq j$, there exists $y_{i_1 i_2} \in \Sigma^*$ such that if $c_{i_1} \xrightarrow{y_{i_1 i_2}} c$ and $c_{i_2} \xrightarrow{y_{i_1 i_2}} c'$ then exactly one of c, c' is an accepting configuration (i.e. $y_{i_1 i_2}$ is a string distinguishing c_{i_1} and c_{i_2}). It follows that for all $1 \leq i_1, i_2 \leq j$, $(x_{i_1}, x_{i_2}) \notin R_L'$ so $rk(R_L') \geq j$. $\square$

We now restate Theorem 4.2 of Geller and Harrison [1977] which characterizes Δ_1 by the rank of the relative right congruence relation.

Theorem 4.3.1. $\Delta_1 = \bigcup_{j\geq 1} \mathcal{D}_j$, i.e. Δ_1 is exactly the family of all deterministic languages L for which $rk(R_L')$ is finite.

Proof. Geller and Harrison [1977] prove that if $L \in \Delta_1$ then R_L' is finite.

For the converse let $L \in \mathcal{D}_j$ for some $j \geq 1$. Then $rk(R_L') = j$ and L is accepted by a DPDA $M = (Q,\Sigma,\Gamma,\delta,q_0,Z_0,F)$. Let Y be the bound from Lemma 4.3.2, and suppose (q,α) is a reachable accepting configuration with $|\alpha| = n > t$. Let c' be the configuration obtained from c by removing a $|Q||Q|!$-invisible segment. Suppose $c \not\equiv c'$. Then, by Lemma 4.3.2, there exists an infinite set of pairwise inequivalent reachable accepting configurations. But by Lemma 4.3.4 this implies that R_L' has infinite rank. The contradiction establishes that $c \equiv c'$.

By Lemma 4.3.3 the computation of the $|Q||Q|!$-invisible segment, needed to obtain c', depends only on the top t symbols of α (and, of course, q). So we can precompute all the relevant $|Q||Q|!$-invisible segments and store them.

We now construct a DPDA M'. M' simulates M, remembering whether or not the stack is of height less than t. Whenever M enters a final state and the stack has more than t symbols M' pops the top t symbols. Using a precomputed table M' eliminates the $qq!$-invisible segment and pushes the resulting string back on the stack. This process is repeated if the stack is still of height greater than t.

In order to keep track of the stack height M' uses a two-track stack. The second track contains a number between 1 and t (the height of the stack up to that point) or a special symbol for stack height greater than t.

M' accepts only when the stack is of height at most t. By Lemma 4.3.2 and this construction M' accepts the same language as M.

Finally, a DPDA M'' that accepts with a single symbol on its stack may be constructed to simulate M'. It encodes t stack symbols of M' by one symbol. $\square$

Theorem 4.3.1 gives a "collective" characterization of all languages that are in $\mathcal{D}_j$ for some $j \geq 1$. We would like to sharpen this result by giving a precise characterization of $\mathcal{D}_j$ for any $j \geq 1$.

First we need a technical lemma about DPDAs.

Lemma 4.3.5. Let $L \in \Delta_1$. Then $L = T_1(M) = T(M,\{Z_f\})$ for some DPDA $M = (Q,\Sigma,\Gamma,\delta,q_0,Z_0,F)$ where $Z_f \in \Gamma$ (i.e. $L = T_1(M)$ and for all $q \in F$ and $w \in \Sigma^*$, $(q_0,Z_0) \xrightarrow{w} (q,Z)$ implies $Z = Z_f$). Moreover all the configurations of the form (q,Z_f) for $q \in F$ are reachable.

Proof. Let $L \in \Delta_1$. Then $L = T_1(M') = T(M',\Gamma)$ for some DPDA $M' = (Q',\Sigma,\Gamma',\delta',q_0',Z_0',F')$. First we construct an equivalent DPDA M'' whose stack alphabet is composed of symbols that are allowed only on the bottom of the stack and symbols that are allowed only elsewhere. Formally let $M'' = (Q',\Sigma,\Gamma' \cup \bar{\Gamma}',\delta'',q_0',\bar{Z}_0',F)$ where $\bar{\Gamma}' = \{\bar{Z} | Z \in \Gamma'\}$ is a set of new symbols, and for all $q \in Q'$, $a \in \Sigma_\Lambda$ and $Z \in \Gamma'$ if $\delta'(q,a,Z) = (p,\gamma)$ for some $p \in Q'$, $\gamma \in \Gamma'^*$ then

$\delta''(q,a,Z) = (p,\gamma)$ and $\delta''(q,a,\bar{Z}) = (p,\bar{\gamma})$ where $\bar{\gamma}$ is defined as

$$\bar{\gamma} = \begin{cases} \Lambda & \text{if } \gamma = \Lambda \\ \bar{Y}\gamma_1 & \text{if } \gamma = Y\gamma_1 \end{cases} \quad \text{with } Y \in \Gamma.$$

It is easy to verify that $(q_0,Z_0') \xrightarrow[M']{w} (q,Z\alpha)$ if and only if $(q_0,\bar{Z}_0') \xrightarrow[M'']{w} (q,\bar{Z}\alpha)$. Hence $L = T_1(M'',\bar{\Gamma}')$ and for each reachable configuration (q,α), $\alpha \in \bar{\Gamma}'\Gamma'^*$. It follows that the stack is of height one if and only if the top of the stack is a symbol in $\bar{\Gamma}'$. This helps M'' "know" when it is in an accepting configuration.

Next we construct another equivalent DPDA $M = (Q,\Sigma,\Gamma,\delta,q_0',\bar{Z}_0',F)$ where $Q = Q' \cup F$, $F = \{(q,\bar{Z}) \in F' \times \bar{\Gamma}' \mid (q,\bar{Z}) \text{ is reachable}\}$,[†] $\Gamma = \Gamma' \cup \bar{\Gamma}' \cup \{Z_f\}$, Z_f is a new symbol, and δ is defined as follows.

(i) $\delta(q,a,Z) = \delta''(q,a,Z)$ for all $a \in \Sigma_\Lambda$ and $q \in Q'$, $Z \in \Gamma' \cup \bar{\Gamma}'$ such that $(q,Z) \notin F$

(ii) $\delta(q,\Lambda,\bar{Z}) = ((q,\bar{Z}),Z_f)$ for all $(q,\bar{Z}) \in F$

(iii) $\delta((q,\bar{Z}),a,Z_f) = \delta''(q,a,\bar{Z})$ for all $(q,\bar{Z}) \in F$ and $a \in \Sigma_\Lambda$.

M simulates M'' using moves of type (i). Whenever M'' enters a (reachable) accepting configuration $(q,\bar{Z})$ M makes a Λ-move (of type (ii)) to the corresponding final state $(q,\bar{Z})$ (which encodes both the state and stack of M'') and places Z_f as the only symbol in the stack. If the input has not been consumed, computation may continue using a type (iii) move to simulate the next step, followed by moves of type (i).

It can be shown that for each $q \in F'$, $\bar{Z} \in \bar{\Gamma}'$ and $w \in \Sigma^*$ $(q_0',\bar{Z}_0') \xrightarrow[M]{w} ((q,\bar{Z}),Z_f)$ if and only if $(q_0',\bar{Z}_0') \xrightarrow[M'']{w} (q,\bar{Z})$, and that M may be in a state $(q,\bar{Z}) \in F$ only when its stack contains Z_f alone. Hence $L = T_1(M) = T(M,\{Z_f\})$.

[†]Note that it is decidable whether or not $(q,\bar{Z})$ is reachable. Simply construct a DPDA identical to M but with only one accepting configuration $(q,\bar{Z})$ and check for emptiness.

By definition of F and the way M simulates M'' it is clear that $((q,\bar{Z}),Z_f)$ is reachable for each $(q,\bar{Z}) \in F$. □

Note that we could construct a DPDA that accepts L with only one final state using k special accepting stack symbols. In this case the symbol on the stack encodes the state and stack of M''.

The next result sharpens that of Theorem 4.3.1. It will not be necessary to use results from Valiant [1975] to prove this next theorem. We will only need Theorem 4.3.1 itself.

Theorem 4.3.2. For each $j \geq 1$, $\mathcal{D}_j$ is the family of languages accepted by a DPDA with at most j reachable accepting configurations.

Proof. Suppose L is accepted by a DPDA $M = (Q,\Sigma,\Gamma,\delta,q_0,Z_0,F)$, i.e. $L = T_0(M)$, and suppose there are exactly $j' \leq j$ accepting configurations that are reachable. Denote them by (q_i,α_i) for $1 \leq i \leq j'$ where $q_i \in F$ and $\alpha_i \in \Gamma^*$ for all i, $1 \leq i \leq j'$.

Define a U-automaton A as follows: $A = (Q_1,\Sigma,\delta_1,q_{01},F_1)$ where $Q_1 = Q \times \Gamma^*$, $q_{01} = (q_0,Z_0)$, $F_1 = \{(q_i,\alpha_i) \mid 1 \leq i \leq j'\}$ and for all $q \in Q$, $\gamma \in \Gamma^*$ and $a \in \Sigma$, $\delta_1((q,\gamma),a) = (q',\gamma')$ if and only if $(q,\gamma) \xrightarrow{a} (q',\gamma')$ for some $q' \in Q$, $\gamma' \in \Gamma^*$ where no move can be made from (q',γ'). A is well defined (i.e. δ_1 is a function) since M is deterministic. It is not hard to show that $T(A) = T_0(M) = L$ and since $|F_1| = j' \leq j$ we can use Lemma 4.2.3 to see that $L \in \mathcal{L}_j$. But L is known to be a deterministic language so $L \in \mathcal{D}_j$.

Conversely suppose $L \in \mathcal{D}_j$. We want to prove that L is accepted by a DPDA with at most j reachable accepting configurations. By

Theorem 4.3.1 $L \in \Delta_1$, so by Lemma 4.3.5 $L = T_1(M) = T(M,\{Z_f\})$ for some DPDA $M = (Q,\Sigma,\Gamma,\delta,q_0,Z_0,F)$ such that $Z_f \in \Gamma$ and (q,Z_f) is reachable for each $q \in F$. Assume that among all DPDAs satisfying these conditions M has a smallest set of final states F. Let $|F| = k$ and suppose $F = \{f_1,f_2,\ldots,f_k\}$.

Define a relation $R' \subseteq L \times L$ as follows: $(x,y) \in R'$ if and only if $(q_0,Z_0) \xrightarrow{x} (f_i,Z_f)$ and $(q_0,Z_0) \xrightarrow{y} (f_i,Z_f)$ for some i, $1 \leq i \leq k$ (i.e. both x and y take M to the same accepting configuration (f_i,Z_f)).

We can show that $R' \subseteq R_L'$. Let $(x,y) \in R'$. Then there exists i, $1 \leq i \leq k$ such that for all $z \in \Sigma^*$, $q \in Q$ and $\gamma \in \Gamma^*$, $(q_0,Z_0) \xrightarrow{x} (f_i,Z_f) \xrightarrow{z} (q,\gamma)$ if and only if $(q_0,Z_0) \xrightarrow{y} (f_i,Z_f) \xrightarrow{z} (q,\gamma)$. It follows that for each $z \in \Sigma^*$, $xz \in L$ if and only if $yz \in L$. Hence $(x,y) \in R_L'$ and indeed $R' \subseteq R_L'$.

We proceed to prove that in fact $R' = R_L'$. Assume, for the sake of contradiction, that $R' \subsetneq R_L'$. Then there exist $x, y \in L$ such that $(x,y) \in R_L'$ but $(x,y) \notin R'$. Then $(q_0,Z_0) \xrightarrow{x} (f_i,Z_f)$ and $(q_0,Z_0) \xrightarrow{y} (f_{i'},Z_f)$ where $i \neq i'$. But on the other hand for all $z \in \Sigma^*$, $xz \in L$ if and only if $yz \in L$. Hence M operates the same way (with respect to acceptance) whether started in (f_i,Z_f) or $(f_{i'},Z_f)$. So these are equivalent accepting configurations and may be merged.

Formally we define a DPDA $M' = (Q,\Sigma,\Gamma,\delta',q_0,Z_0,F')$ where $F' = F - \{q_{i'}\}$ and for all $q \in Q$, $a \in \Sigma_\Lambda$ and $Z \in \Gamma$ if $(q,a,Z) \neq (q_{i'},\Lambda,Z_f)$ then $\delta'(q,a,Z) = \delta(q,a,Z)$, $\delta'(q_{i'},\Lambda,Z_f) = (q_i,Z_f)$. It is easy to verify that $L = T_1(M') = T(M',\{Z_f\})$ for some $Z_f \in \Gamma$ and for all $q \in F'$, (q,Z_f) is reachable. But $|F'| = |F| - 1$, contradicting the minimality of $|F|$.

We have proven that $R' = R_L'$. Hence $k = rk(R') = rk(R_L')$. $L \in \mathcal{D}_J \subseteq \mathcal{L}_j$ so $rk(R_L') \leq j$. Hence M has $k \leq j$ reachable accepting configurations. □

The proof of Theorem 4.3.2 provides yet another result.

Corollary. Let $L \subseteq \Sigma^*$ and $j \geq 1$. $L \in \mathcal{D}_j$ if and only if $L = T(M,\{Z_f\}) = T_1(M)$ for some DPDA $M = (Q,\Sigma,\Gamma,\delta,q_0,Z_0,F)$ for which $Z_f \in \Gamma$ and $|F| \leq j$.

Section 4.4 - Some Consequences of the Hierarchy

In this section we use the results of previous sections to prove certain properties of the families Δ_1 and LR(0).

We show that the family of LR(0) is precisely $\mathcal{D}_1$. We then prove a "conditional" closure result for both Δ_1 and LR(0) under intersection. The intersection of languages in these families remains in that family provided it is deterministic. We show a connection between Δ_1 and regular sets. Finally we discuss a certain decidability question regarding Δ_1.

First, we note the connection between LR(0) languages, as discussed by Geller and Harrison [1977], and our hierarchy.

Theorem 4.4.1. The family of LR(0) languages is exactly $\mathcal{D}_1$.

Proof. Theorem 3.1 of Geller and Harrison [1977] shows that the LR(0) family is exactly the family of all languages L such that $L = T(M,\{Z_f\}) = T(M,\Gamma)$ for some DPDA $M = (Q,\Sigma,\Gamma,\delta,q_0,Z_0,F)$ for which $Z_f \in \Gamma$ and $|F| = 1$. By the Corollary to Theorem 4.3.2 this is precisely $\mathcal{D}_1$. □

The next lemma will lead to a "conditional" closure result on various families of languages.

Lemma 4.4.1. Suppose $L_1, L_2 \subseteq \Sigma^*$. Let $L_1 \in \mathcal{D}_{j_1}$, $L_2 \in \mathcal{D}_{j_2}$ and let $L = L_1 \cap L_2$. If $L \in \Delta_0$ then $L \in \mathcal{D}_j$ where $j = j_1 j_2$.

Proof. Let $L_1 \in \mathcal{D}_{j_1}$, $L_2 \in \mathcal{D}_{j_2}$. Then $L_1 \in \mathcal{L}_{j_1}$ and $L_2 \in \mathcal{L}_{j_2}$. Hence there exist two U-automata $A_i = (Q_i,\Sigma,\delta_i,q_{0i},F_i)$, $i = 1,2$ where $L_i = T(A_i)$. Construct a U-automata $A = (Q,\Sigma,\delta,q_0,F)$ where

$Q = Q_1 \times Q_2$, $q_0 = (q_{01}, q_{02})$, $F = F_1 \times F_2$ and for all $(q_1, q_2) \in Q_1 \times Q_2$, $a \in \Sigma$, $\delta((q_1,q_2),a) = (\delta(q_1,a),\delta(q_2,a))$. A simple induction shows that $\delta((q_{01},q_{02}),w) = (\delta(q_{01},w),\delta(q_{02},w))$ for all $w \in \Sigma^*$. It follows that $w \in T(A)$ if and only if $w \in T(A_1) \cap T(A_2)$. Hence $L = T(A)$, and since A has $|F_1 \times F_2| = |F_1||F_2| = j_1 j_2$ final states $L \in \mathcal{L}_j$ for $j = j_1 j_2$. By the hypothesis of the lemma $L \in \Delta_0$ so we conclude that $L \in \mathcal{D}_j$, $j = j_1 j_2$. □

Lemma 4.4.1 may be used for the families Δ_1 and LR(0).

<u>Theorem 4.4.2</u>. Suppose L_1, L_2 are both in Δ_1 (respectively LR(0)). Then, if $L = L_1 \cap L_2$ is a deterministic language, it must also be in Δ_1 (respectively LR(0)).

<u>Proof</u>. Let $L_1, L_2 \in \Delta_1$, $L = L_1 \cap L_2 \in \Delta_0$. Then, for some $j_1, j_2 \geq 1$, $L_1 \in \mathcal{D}_{j_1}$ and $L_2 \in \mathcal{D}_{j_2}$. Using Lemma 4.4.1 $L \in \mathcal{D}_j$ for $j = j_1 j_2$ so $L \in \Delta_1$.

Let $L_1, L_2 \in$ LR(0), $L = L_1 \cap L_2 \in \Delta_0$. Then $L_1, L_2 \in \mathcal{D}_1$ and hence, by that lemma, $L \in \mathcal{D}_{1\cdot 1} =$ LR(0). □

Note how the use of U-automata simplified the proof of Lemma 4.4.1.

Next we establish an interesting connection between Δ_1 and the regular sets.

<u>Theorem 4.4.3</u>. Let $L \in \Sigma^*$. L and $\bar{L} = \Sigma^* - L$ are both in Δ_1 if and only if L is regular.

Proof. Suppose L is regular. Then $rk(R_L)$ is finite. By the Corollary to Lemma 4.2.2 $rk(R_L) = rk(R_L') + rk(R_L^{\perp})$. So both $rk(R_L')$ and $rk(R_L^{\perp})$ are finite. It is also clear that both L and $\bar{L}$ may be accepted by a DPDA. Hence, by definition of $\mathcal{D}_j$ and Theorem 4.3.1, $L, \bar{L} \in \Delta_1$.

Conversely suppose $L, \bar{L} \in \Delta_1$. Then by Theorem 4.3.1 $rk(R_L')$ and $rk(R_L^{\perp})$ are finite. So $rk(R_L) = rk(R_L') + rk(R_L^{\perp})$ is finite and L is regular. □

If we denote by $\bar{\Delta}_1$ the family of languages L whose complement $\bar{L} = \Sigma^* - L$ is in Δ_1, and if we let Reg denote the regular sets, we can rewrite this last result as follows.

Corollary. $\Delta_1 \cap \bar{\Delta}_1 = \text{Reg}$

An interesting decidability question is whether or not one can determine, for a given language $L \in \Delta_1$, the minimum j such that $L \in \mathcal{D}_j$. The following theorem shows that this is likely to be very hard. In fact even when we know that $L \in \mathcal{D}_2$ it may be hard to decide whether or not $L \in \mathcal{D}_1$.

Theorem 4.4.4. There is an algorithm to decide if a $\mathcal{D}_2$ language is in $\mathcal{D}_1$ if and only if there is an algorithm to decide if two deterministic languages are equal.

Proof. This is a restatement of Corollary 1 to Theorem 5.3 in Geller and Harrison [1977]. □

References

Aho, A.V. and Ullman, J.D. [1972], The Theory of Parsing, Translating and Compiling, Vol. I, Prentice Hall, Englewood Cliffs, New Jersey.

Aho, A.V. and Ullman, J.D. [1973], The Theory of Parsing, Translating and Compiling, Vol. II, Prentice Hall, Englewood Cliffs, New Jersey.

Aho, A.V., Hopcroft, J.E. and Ullman, J.D. [1974], The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Mass.

Bar-Hillel, Y., Perles, M. and Shamir, E. [1961], "On Formal Properties of Simple Phrase Structure Grammars," Zeitschrift fur Phonetik, Sprachwissenschaft, und Kommunikationsforschung, Vol. 14, pp. 143-172.

Friedman, E.P. [1976], "The Inclusion Problem for Simple Languages," Theoretical Computer Science, Vol. 1, pp. 297-316.

Geller, M.M., Harrison, M.A. and Havel, I.M. [1976], "Normal Forms of Deterministic Grammars," Discrete Mathematics, Vol. 16, pp. 313-321.

Geller, M.M. and Harrison, M.A. [1977], "On LR(k) Grammars and Languages," Theoretical Computer Science, Vol. 4.

Ginsburg, S. and Greibach, S.A. [1966], "Deterministic Context-Free Languages," Information and Control, Vol. 9, pp. 602-648.

Graham, S.L. [1971], "Precedence Languages and Bounded Right Context Languages," CS 223, Computer Science Department, Stanford University.

Graham, S.L. [1974], "On Bounded Right Context Languages and Grammars," SIAM Journal of Computing, Vol. 3, pp. 224-254.

Gray, J.N. and Harrison, M.A. [1972], "On the Covering and Reduction Problems for Context-Free Grammars," Journal of the Association for Computing Machinery, Vol. 19, pp. 675-698.

Gray, J.N. and Harrison, M.A. [1973], "Canonical Precedence Schemes," Journal of the Association for Computing Machinery, Vol. 20, pp. 213-234.

Greibach, S.A. [1965], "A New Normal Form Theorem for Context Free Phrase Structure Grammars," Journal of the Association for Computing Machinery, Vol. 12, pp. 42-52.

Harrison, M.A. [1978], Introduction to Formal Language Theory, Addison-Wesley, Reading, Mass. (to appear).

Harrison, M.A. and Havel, I.M. [1972], "Real Time Strict Deterministic Languages," SIAM Journal of Computing, Vol. 1, pp. 333-349.

Harrison, M.A. and Havel, I.M. [1973], "Strict Deterministic Grammars," Journal of Computer and Systems Science, Vol. 7, pp. 237-277.

Harrison, M.A. and Havel, I.M. [1974], "On the Parsing of Deterministic Languages," Journal of the Association for Computing Machinery, Vol. 21, pp. 525-548.

Hoare, C.A.R. [1969], "An Axiomatic Basis for Computer Programming," Communication for the Association for Computing Machinery, Vol. 12, pp. 576-581.

Hopcroft, J.E. and Ullman, J.D. [1969], Formal Languages and Their Relation to Automata, Addison-Wesley, Reading, Mass.

Hunt, H.B. III, Rosenkrantz, D.J. and Szymanski, T.G. [1976], "On the Equivalence, Containment and Covering Problems for the Regular and Context Free Languages," Journal of Computer and System Sciences, Vol. 12, pp. 222-268.

Hunt, H.B. III, Szymanski, T.G. and Ullman, J.D. [1974], "Operations on Sparse Relations and Efficient Algorithms for Grammar Problems," Conference Record of IEEE 15th Annual Symposium on Switching and Automata Theory, New Orleans, Louisiana.

Hunt, H.B. III, Szymanksi, T.G. and Ullman, J.D. [1975], "On the Complexity of LR(k) Testing," Technical Report 180, Princeton University.

Knuth, D.E. [1969], Seminumerical Algorithms, Addison-Wesley, Reading, Mass.

Korenjak, A.J. and Hopcroft, J.E. [1966], "Simple Deterministic Languages," Conference Record of IEEE 7th Annual Symposium on Switching and Automata Theory, Berkeley, California.

Manna, Z. [1974], Mathematical Theory of Computation, McGraw-Hill, New York.

Rosenkrantz, D.J. and Stearns, R.E. [1970], "Properites of Deterministic Top-down Grammars," Information and Control, Vol. 17, pp. 226-255.

Salomaa, A. [1964], "On the Reducibility of Events Represented in Automata," Annales Academiae Scientiarum Fennicae, Series AI 353.

Taniguchi, K. and Kasami, T. [1976], "A Result on the Equivalence Problem for Deterministic Pushdown Automata," Journal of Computer and System Sciences, Vol. 13, pp. 38-50.

Valiant, L.G. [1973], "Decision Procedures for Families of Deterministic Pushdown Automata," University of Warwick Computer Centre, Report No. 7.

Valiant, L.G. [1974], "The Equivalence Problem for Deterministic Finite-turn Pushdown Automata," Information and Control, Vol. 25, pp. 123-133.

Valiant, L.G. [1975], "Regularity and Related Problems for Deterministic Pushdown Automata," Journal of the Association for Computing Machinery, Vol. 22, pp. 1-10.

Valiant, L.G. and Paterson, M.S. [1975], "Deterministic One-counter Automata," Journal of Computer and System Sciences, Vol. 10, pp. 340-350.

Winkler, T. [1977], private communication.