DOMANSKI, BERNARD

THE COMPLEXITY OF DECISION PROBLEMS IN GROUP THEORY

City University of New York PH.D. 1980

PLEASE NOTE:

In all cases this material has been filmed in the best possible way from the available copy. Problems encountered with this document have been identified here with a check mark ✓.

1. Glossy photographs _____
2. Colored illustrations _____
3. Photographs with dark background _____
4. Illustrations are poor copy _____
5. Print shows through as there is text on both sides of page _____
6. Indistinct, broken or small print on several pages ✓ throughout _____
7. Tightly bound copy with print lost in spine _____
8. Computer printout pages with indistinct print _____
9. Page(s) _____ lacking when material received, and not available from school or author _____
10. Page(s) _____ seem to be missing in numbering only as text follows _____
11. Poor carbon copy _____
12. Not original copy, several pages with blurred type _____
13. Appendix pages are poor copy _____
14. Original copy with light type _____
15. Curling and wrinkled pages ✓
16. Other _____

University Microfilms International

300 N. ZEEB RD., ANN ARBOR, MI 48106 (313) 761-4700

THE COMPLEXITY OF DECISION PROBLEMS IN GROUP THEORY

by

BERNARD DOMANSKI

A dissertation submitted to the Graduate Faculty in Engineering in partial fulfillment of the requirements for the degree of Doctor of Philosophy, The City University of New York.

1980

This manuscript has been read and accepted for the Graduate Faculty in Engineering in satisfaction of the dissertation requirement for the degree of Doctor of Philosophy.

1/25/80
Date

Michael Anshel
Chairman of Examining Committee

1/25/80
Date

F E Nan
Executive Officer

Professor Michael Anshel, Chairman
Professor Frank Beckman
Professor Harvey Cohn
Doctor William Gewirtz
Doctor Leon Landovitz
Professor Henry Levinson
Professor Jacob Rootenberg

Supervisory Committee

Abstract

THE COMPLEXITY OF DECISION PROBLEMS IN GROUP THEORY

by

Bernard Domanski

Advisor: Professor Michael Anshel

The computational complexity of a number of decision problems for free groups and the groups of Dehn's algorithm are analyzed. Linear time and logspace bounds are obtained using a multitape Turing Machine as the model of computation. The underlying tool used to provide some of these bounds is a linear time pattern matching algorithm.

Acknowledgements

I would like to take this opportunity to thank the Computer Science faculty of the City College of New York for both the undergraduate and graduate education they have given me. I owe a deep debt of gratitude to my graduate advisor, Professor Michael Anshel, whose direction and support have left an indelible impression on me. I would like to acknowledge the support given to me by both the Graduate School and Bell Telephone Laboratories during the period of my doctoral studies. Finally, I am greatly indebted to my wife Shelly and son Robbie, for the support and encouragement they have provided throughout all my academic activities.

Preface

The computational complexity of a number of group theoretic decision problems are analyzed. In particular, linear upper time bounds are obtained for:

- the Conjugacy Problem for Free Groups, (c.f. Chapter 2)
- the Power Problem for Free Groups, (c.f. Chapter 3)
- the Power Conjugacy Problem for Free Groups, (c.f. Chapter 4)
- the Word Problem for the Groups of Dehn's Algorithm (the DA groups) (c.f. Chapter 6)
- the Conjugacy Problem for the Groups of Dehn's Algorithm (c.f. Chapter 7)

A space bound of logspace is obtained for the Word Problem for the groups of Dehn's Algorithm (c.f. Chapter 5).

The model of computation used is a multitape Turing Machine as defined by Stockmeyer (1974) (c.f. 1.9).

What underlies the solution to some of these problems is the use of a linear time pattern matching algorithm, originated by Knuth, Morris, & Pratt (1977), and

implemented on a multitape Turing Machine by Fischer & Paterson (1974) (c.f. sections 2.4 -2.10).

Major Results

Outlined below is an overview of the key definitions, lemmas and theorems in this dissertation.

1. Def: The Conjugacy Problem for Free Groups (c.f. 1.6) is deciding in a finite number of steps whether two words U and V define conjugate elements (c.f. 1.4).

2. Theorem: Two words U and V are conjugate elements in the free group iff the cyclic reduction (c.f. 1.5) of U is a cyclic permutation (c.f. 1.6) of the cyclic reduction of V.

3. Lemma: Given two words U and V of equal length where U and V are cyclically reduced. Then U^2 = PVQ iff U and V are cyclic permutations (c.f. 2.3).

4. Algorithm to solve the Conjugacy Problem for Free Groups:

    1. Cyclic Reduction of U and V (c.f. 2.1, 2.2).

2. Apply the Fischer-Paterson pattern matching Turing Machine to the string U^2, using V as the pattern (c.f. 2.3 - 2.10) in linear time. This makes use of lemma 3 in using theorem 2.

5. Def: The Power-Problem for Free Groups is deciding in a finite number of steps whether a given word V is equivalent to a power of U; i.e. is $V=U^k$ for some finite value of k? (c.f. Chapter 3)

6. Algorithm to solve the Power-Problem for Free Groups:

1. Free Reduction of U and V. (c.f. 3.3 - 3.5)

2. Insure that no trivial relators can occur when creating U^k.

3. Apply the Fischer-Paterson Turing Machine to V using U as the pattern. If the length of U is u, then test whether the k^{th} occurrence of U occurs at position ku, for k=1,2,... (length of V)/u. (c.f. 3.6 - 3.8)

7. Def: The Power-Conjugacy Problem for Free Groups is deciding in a finite number of steps whether a given word V is equivalent to a conjugate of a power of U; i.e. is $V = W^{-1}U^{k}W$ for some word W and some finite value of k. (c.f. Chapter 4)

8. Algorithm to solve the Power-Conjugacy Problem for Free Groups:

   1. Cyclic Reduction of U and V. (c.f. 4.2)

   2. Determine k, where the length of V is equal to the (length of U)^{k}. (c.f. 4.3)

   3. Apply the Fischer-Paterson Turing Machine to the string $U^{k}U^{k}$ using V as the pattern. This makes use of lemma 3 in using theorem 2. (c.f. 4.4 - 4.6)

9. Def: The Word Problem for the Groups of Dehn's Algorithm (DA) (c.f. 1.8) is deciding in a finite number of steps whether a given word W defines the identity element. (c.f. Chapters 5, 6)

10. Def: The complexity bound of logspace is defined as deciding a problem involving an input string of length 2^n in space n, where space is defined as the number of worktape cells visited on a multitape Turing Machine (c.f. 1.9).

11. Algorithm: (Space Complexity for the solution to the Word Problem for the DA Groups)

1. Form the symmetric set of defining relators (c.f. 1.8).

2. Define the tree structure and replacement list for the relators (c.f. 5.1).

3. Let m = the number of generating symbols (including inverses), and let the length of the input be 2^n. Then k, the base used for determining the state symbol alphabet, is defined as:

$$k = 2^{\frac{2^n \log_2 m}{n}}$$

4. Using lexigraphic order, code the input onto the worktape in base k.

5. Whenever a LHS is found, recompute the encoding on the worktape using the new RHS. This may be done recursively.

6. Define the combination list for the state symbol alphabet (c.f. 5.3).

7. Define the Z-list of replacement strings for the state symbol alphabet (c.f. 5.3)

8. Input symbols are encoded in logspace onto worktapes using the combination list and the Z-list. The replacement list is used to replace encoded subwords with their corresponding shorter replacements (c.f. 5.4). The algorithm halts when the input is exhausted.

12. Def: Assume a given DA group has m relators (both trivial and non-trivial). Let the length of the longest Left-Hand Side (LHS) (c.f. 6.1) be p.

13. Algorithm: (Time Complexity for the solution to the Word Problem for the DA Groups)

1. In parallel, apply the Fischer-Paterson Turing Machine to the input word, where one machine exists for each pattern. The patterns are the LHS's of the relators (i.e., m machines in total). (c.f. 6.1)

2. Replace occurrence of LHS's with Right-Hand Sides (RHS) (c.f. 6.2) (and null symbols) thereby shortening the length of the input.

3. Backup p-1 symbols from the first symbol of the RHS and go to 1 (c.f. 6.2).

4. Complexity is of order m*0(p*(length of input)). (c.f. 6.2)

14. Def: The Conjugacy Problem for the DA groups is deciding in a finite number of steps whether two words U and V are conjugate (c.f. Chapter 7).

15. Algorithm for the Conjugacy Problem for the DA Groups:

1. Cyclic Reduction of U and V (c.f. 7.2 - 7.3) in time m*0(p^2* (length of input)).

2. Apply the Fischer-Paterson Turing Machine to U^2 using V as the pattern as in algorithms 4 and 7. (c.f. 7.4)

CONTENTS

LIST OF ILLUSTRATIONS

1. INTRODUCTION

In 1911, Max Dehn formulated what have now become the three fundamental decision problems in group theory, namely the word problem, the conjugacy (or transformation) problem, and the isomorphism problem. (Dehn, 1911, Magnus, Karrass, Solitar, 1966). Today, a great deal of interest has been shown in the area of computational complexity where by computational complexity we mean the amount of time and space required to solve a problem on some mathematical model of computation, (where, time and space are defined relative to a given model) (Aho, Hopcroft, Ullman, 1974, Stockmeyer, 1974).

What is proposed is to tie these two research areas together; that is, to examine the computational complexity of some decision problems in group theory. To do this, a close examination of the problems in group theory is required. Another requirement is to examine the notion of computational complexity in terms of a mathematical model (i.e., a multi-tape Turing machine).

Complexity results are presented for a number of decision problems in group theory. These results are obtained by using techniques from pattern matching (Fischer, Paterson, 1974, Knuth, Morris, Pratt, 1977).

1.1 Fundamentals in Group Theory

In order to properly discuss decision problems in group theory, it seems appropriate to begin with the definition of a group and its properties. From Magnus, Karrass, and Solitar (1966), comes this definition:

Def: A Group $(G,.)$ is a non-empty set G of elements a,b,c, ... together with a binary operation . defined in G for which the following four postulates are satisfied:

1 - If a,b is an ordered pair of elements of G ($a \neq b$ or $a = b$), then there is a uniquely determined element c of G such that $a \cdot b = c$. c is called the Product of a and b. Note also that the dot is usually omitted and so we write $ab = c$.

2 - The operation defined by . in G is associative, i.e., for any elements a,b,c in G we have $(ab)c = a(bc)$.

3 - There exists an element of G, denoted by 1, for which $a \cdot 1 = 1 \cdot a = a$, where a is any element in G. 1 is called the identity or unit element in G.

4 - If a is any element in G, then there exists an element in G, denoted a^{-1} for which

$a \cdot a^{-1} = 1 \cdot a^{-1}$ is called the inverse of a.

In the case that $a \cdot b = b \cdot a$, we say elements a and b commute. If all pairs a and b commute, . is called a commutative operation, and (G,.) is called a commutative or abelian group.

Postulates 1 and 2 allow us to define the product of a sequence of n elements a(1),a(2), ... ,a(n) which is independent of the partitioning of the factors, e.g., (a(1)a(2)) (a(3)a(4)) = ((a(1)a(2))a(3))a(4). In particular, if a(1) = a(2) = ... = a(n) = a, the product a(1)a(2) ... a(n) is denoted by a^n. By using postulates 3 and 4, the definition of a^n can be extended to the case where n is zero or a negative integer. Define $a^0 = 1$ and $a^{(-n)}$ as $(a^{-1})^n$ for positive integers n. It follows then that $a^m \cdot a^n = a^{m+n}$ and $(a^m)^n = a^{mn}$ for all integers m and n.

1.2 Length and Inverses

Let a,b,c,... be distinct symbols and form the new symbols a^{-1}, b^{-1}, c^{-1}, ... A word W in the symbols a,b,c is a finite sequence $f_1 f_2 f_3 \ldots f_{n-1} f_n$ where each of the f_v (v = 1,2, ...n) is one of the symbols a,b,c, ... , a^{-1}, b^{-1}, c^{-1}, ...; the length of W (|W|) is the integer n. For notational convenience, the empty word of length zero is introduced and it is

denoted by 1. The symbols in W are written W(a,b,c,...).

It is customary to write the sequence of W without commas, i.e., $f_1 f_2 \ldots f_n$. It is also customary to abbreviate a block of n consecutive symbols a by a^n, and to abbreviate a block of n consecutive symbols a^{-1} by a^{-n}; e.g., the word $a^3 b^2 b^{-1} a^{-2} c^{-1}$ is the same as the word $aaabbb^{-1}a^{-1}a^{-1}c^{-1}$, but is different from the word $a^3 b a^{-2} c^{-1}$. (For brevity, a is often written rather than a^1, b rather than b^1, etc.).

Thus aa^{-1} is a word of length two, $a^2 b a^{-1} b^{-2} b$ is a word in a and b of length seven. $b^2 c^{-2}$ is a word in a,b and c of length four and a word in b and c of length four; 1 is a word in a,b and c of length 0.

The <u>inverse</u> W^{-1} of a word W, where W is $f_1 f_2 \ldots f_n$ is the word $f_n^{-1} f_{n-1}^{-1} \ldots f_1^{-1}$ where if f_v (v=1,2, ... n) is a (or a^{-1}), then $f^{-1}(v)$ is a^{-1} (or a), and similarly if f(v) is b or b^{-1}, c or c^{-1}, etc. For example,

$$(aaba^{-1}b^{-1}b^{-1})^{-1} = (bbab^{-1}a^{-1}a^{-1}),$$

Note that the inverse of the empty word is itself:

$$(aa^{-1})^{-1} = aa^{-1},$$

$$1^{-1} = 1.$$

Note too, that $|W| = |W^{-1}|$ and $(W^{-1})^{-1} = W$, $(WU)^{-1} = U^{-1}W^{-1}$, and $|WU| = |W| + |U|$. For words W and 1 (the empty word), we have $W\,1 = 1\,W = W$.

1.3 Generators And Relators

The notion of a generator and a relator are now defined. If every element of G defined by some word in $a,a^{-1},b,b^{-1},c,c^{-1},\ldots$ then $a,a^{-1},b,b^{-1},c,c^{-1},\ldots$ are called a set of generating symbols or generators for G.

A word $P(a,b,c,\ldots)$ which defines the identity element in G is called a relator. The equation $P(a,b,c,\ldots) = Q(a,b,c,\ldots)$ is called a relation if the word PQ^{-1} is a relator, or equivalently, if P and Q define the same element of G.

In every group G, the empty word and the words aa^{-1}, $a^{-1}a$, bb^{-1}, $b^{-1}b$, cc^{-1}, $c^{-1}c$, etc. are always relators; they are called the trivial relators.

Suppose P,Q,R,... are relators of G. The word W is said to be derivable from P,Q,R,... if the following operations, applied a finite number of times, change W into the empty word:

- Insertion of one of the relators P, P^{-1}, Q, Q^{-1}, R, R^{-1}, etc. or one of the trivial relators between any two consecutive symbols of

W, or before W, or after W.

- Deletion of one of the words P, P^{-1}, Q, Q^{-1}, R, R^{-1}, etc. or one of the trivial relators, if it forms a block of consecutive symbols in W.

It should be clear that if the word W is derivable from the relators P,Q,R,..., then W itself is a relator because the operations defined above, when applied to a word, do not change the element of the group defined by the word. Since the empty word is reached, W must define the identity element of G. For example, let $a^2 = 1$ be a relator for a given group element a. Then the word $W = aaa^{-1}aa^{-1}a$ is derivable by the following steps: $W = aa\ 1\ a^{-1}a$ by the trivial relation, $W = aa\ 1\ 1$ by the trivial relation, and $W = 1$ by the given relation.

If every relator is derivable from relators P, Q, R, ..., then the set P, Q, R, ... is called a set of defining relators for the group G on the generators a, b, c, ... If P, Q, R, ... is a set of defining relators for the group G on the relators a, b, c, ..., the notation

$$\langle a,b,c,\ldots\ ;\ P,Q,R\ldots\rangle$$

is called a presentation of G. A presentation of G is

finitely generated (finitely related) if the number of generators (relators) in it is finite. If a presentation is finitely generated and finitely related, the presentation is said to be finite, or the group is finitely presented. Note that presentations are often given using relations instead of relators; and at other times both relations and relators are used. For example,

$$< a,b;\ a^2 = 1,\ b^2 = 1,\ ab = ba >$$

$$< a,b;\ a^2,\ b^2,\ ab = ba >$$

$$< a,b;\ a^2,\ b^2,\ aba^{-1}b^{-1} >$$

are all interpreted as the last given presentation.

Theorem: There is a unique group with the presentation $< a,b,c,\ldots\ ;\ P,\ Q,\ R,\ \ldots >$, given a set of distinct symbols a,b,c,... and a set (possibly empty) of words P,Q,R,... in a,b,c,... (Magnus, Karrass, Solitar, 1966).

1.4 The Fundamental Decision Problems in Presentations of Groups

Let G be a group defined by means of a given presentation. The three fundamental decision problems in group theory as defined by Dehn are defined below:

I. For an arbitrary word W in the generators, decide in a finite number of steps whether W defines the identity element of G or not. This is known as the word problem for the presentation defining G.

II. For two arbitrary words W_1 and W_2 in the generators, decide in a finite number of steps whether W_1 and W_2 define conjugate elements of G, where W_1 and W_2 are called conjugate elements if there exists a word W_3 such that $W_3^{-1} W_1 W_3 = W_2$. This problem of deciding conjugacy is called the transformation or conjugacy problem for the presentation defining G.

III. Given two presentations of groups G and G', decide in a finite number of steps whether the groups defined by G and G' are isomorphic. This is known as the isomorphism problem for the presentation defining G.

The word problem (I) has been solved for many classes of presentations of some specialized form (Magnus, Karrass, Solitar 1966, Cardoza, 1975, Greendlinger, 1960a, presentations in which there are no non-trivial defining relators (this is shown later)). However, there are finite presentations where the word

problem cannot be solved. That is, there exists a word W over the generators for which one can not decide in a finite number of steps whether W defines the identity element of G (Boone, 1955, Novikov, 1955). Thus, there is no algorithm for solving the word problem which will work for every presentation.

The conjugacy problem is more difficult to solve than the word problem. Note that by choosing W_1 to be the empty word, the solution to the conjugacy problem is the solution to the word problem! Therefore, the classes of groups for which the conjugacy problem has been solved must include those classes for which the word problem has been solved. In fact, even though the word problem has been solved for presentations with one defining relator (Magnus, 1932), the conjugacy problem is unsolved. Recent work in this area has been done by Anshel and Stebe (1974).

The isomorphism problem is the most difficult of the three problems of Dehn. It has been shown that even if the presentation is obviously the identity, e.g., $< a ; a >$, the isomorphism problem is unsolvable (Rabin, 1958). Thus, it is usually customary to restrict the presentations of G and G' to some special class. For example, if the presentations for G and G' have no non-trivial defining relators, then the isomorphism problem can be solved (Magnus, Karrass and

Solitar, 1966).

1.5 The Word Problem For Free Groups

At this point, the word problem is examined for a special class of groups, called free groups.

The free group, F_n on the n free generators x_1, x_2, ... x_n, is the group with generators x_1, x_2, ... x_n and no defining relators (other than the trivial relators). That is, $F_n = \langle x_1, x_2, \ldots x_n \rangle$. The word and conjugacy problems for F_n are both solved by using the concepts of freely reduced and cyclically reduced words.

A freely reduced word $x_1, x_2, \ldots x_n$ is a word where the symbols x_i^j, x_i^{-j} do not occur consecutively for i=1,...n and j=-1,+1. Thus the words $x_1 x_2^2 x_3$ and $x_1 x_2 x_3 x_2^{-1} x_1^{-1}$ are freely reduced, while $x_1 x_2^2 x_2^{-2} x_3$ is not.

Two words are called freely equal if each is derivable from the other in the free group. This is denoted as $W_1 \doteq W_2$. There, $W_1 \doteq W_2$ if W_1 can be transformed into W_2 by finitely many insertions or deletions of the trivial relators. For example, $x_1 x_2 x_3^2 x_3^{-1} x_2^{-1} x_2^2 \doteq x_1 x_2 x_3 x_2$. Every word in $x_1, x_2, \ldots x_n$ is freely equal to some freely reduced word, because trivial relators can be deleted

until no more remain (Magnus, Karrass, Solitar, 1966).

Thus, to decide if a given word W defines the identity in F_n, freely reduce W. If the result is not the empty word, then W does not define the identity, and conversely.

A cyclically reduced word in $x_1, x_2, \ldots x_n$ is a freely reduced word which does not begin with x_i^j and end with x_i^{-j}. For example, $x_1 x_2^2 x_3$ and $x_1^{-1} x_2^2 x_3 x_1^{-1}$ are cyclically reduced, while $x_1 x_2 x_3 x_2^{-1} x_1^{-1}$ is not.

1.6 The Conjugacy Problem for Free Groups

Introduced now is another theorem (Magnus, Karrass and Solitar) which is used to solve the conjugacy problem for free groups. First shown is a process Z for cyclically reducing a word. Simply stated, Z first freely reduces the word, and then cancels first and last symbols, if appropriate. For example,

$$\begin{aligned} Z(x_1 x_2 x_3 x_3^{-2} x_2^{-1} x_1^{-1}) &= x_1 x_2 x_3^{-1} x_2^{-1} x_1^{-1} \\ &= x_2 x_3^{-1} x_2^{-1} \\ &= x_3^{-1}. \end{aligned}$$

We define Z inductively as follows:

$$Z(1) = 1$$

$$Z(x_i^j) = x_i^j \qquad (i=1,2,\ldots n,\ j=-1,+1)$$

$$Z(x_i^j \; U \; x_p^q) = x_i^j \; U \; x_p^q \text{ if } i \neq p \text{ or } j \neq -q$$

$$= U \text{ if } i=p \text{ and } j = -q$$

where $i,p=1,2,\ldots n$; $j,q=-1,+1$

U is defined a cyclic permutation of $V = x_1 \; x_2 \; \ldots \; x_n$ if $U = V$ or $U = x_k \; \ldots \; x_n \; x_1 \; x_2 \; \ldots \; x_{k-1}$ for $n>k>1$.

Theorem: If F_n is the free group on the generators $x_1, x_2, \ldots x_n$ then U and V are conjugate elements of F_n iff the cyclic reduction Z(U) is a cyclic permutation of the cyclic reduction of Z(V).

Proof: First suppose that Z(V) is a cyclic permutation of Z(U). That is

$$Z(U) = x_{i_1}^{j_1} \; \ldots \; x_{i_s}^{j_s} \; x_{i_{s+1}}^{j_{s+1}} \; \ldots \; x_{i_p}^{j_p}$$

and -

$$Z(V) = x_{i_{s+1}}^{j_{s+1}} \; \ldots \; x_{i_p}^{j_p} \; x_{i_1}^{j_1} \; \ldots \; x_{i_s}^{j_s}$$

where $i_k = 1, 2, \ldots n$ for all k; $j_k = +1,-1$ for all k. Then $Z(U) = k \; Z(V) \; k^{-1}$

$$\text{where } k = x_{i_1}^{j_1} \; \ldots \; x_{i_s}^{j_s}.$$

Since Z(U) is a conjugate of U, and similarly for Z(V), define conjugate elements of F_n; i.e., $U = T \; V \; T^{-1}$. We

show that Z(U) and Z(V) are cyclic permutations. Let $T = x_i^j$, $i = 1,2,\ldots n$, $j = +1, -1$. Consider

$$Z(x_i^j \text{ V } x_i^{-j}) = Z(x_i^j x_{i_1}^{j_1} \ldots x_{i_r}^{j_r} x_r^{-j}):$$

Case I. No cancellation takes place. That is, $i \neq i_1$ or $j \neq -j_1$ and $i \neq i_r$ or $j \neq -j_r$. Thus $Z(x_i^j \text{ V } x_i^{-j}) = Z(x_i^j x_{i_1}^{j_1} \ldots x_{i_r}^{j_r} x_i^{-j}) = x_{i_1}^{j_1} \ldots x_{i_r}^{j_r} = Z(V)$.

Case II. Cancellation takes place at both ends; that is $i = i_1 = i_r$ and $j = -j_1 = -j_r$. Thus here $Z(x_i^j \text{ V } x_i^{-j}) = Z(x_{i_2}^{j_2} \ldots x_{i_{r-1}}^{j_{r-1}})$. Note that $Z(V) = Z(x_{i_1}^{j_1} x_{i_2}^{j_2} \ldots x_{i_{r-1}}^{j_{r-1}} x_{i_1}^{-j_1})$ (by the premise that cancellation takes place at both ends), which is equal to $Z(x_{i_2}^{j_2} \ldots x_{i_{r-1}}^{j_{r-1}})$. So again $Z(V) = Z(x_i^j \text{ V } x_i^{-j})$.

Case III. Cancellation takes place only at the left end. That is $i = i_1$ and $j = -j_1$, but $i \neq i_r$ or $j \neq -j_r$. $Z(x_i^j \text{ V } x_i^{-j}) = x_{i_2}^{j_2} \ldots x_{i_r}^{j_r} x_{i_1}^{j_1}$, while $Z(V) = x_{i_1}^{j_1} \ldots x_{i_r}^{j_r}$. Here $Z(x_i^j \text{ V } x_i^{-j})$ is a cyclic permutation of Z(V).

Case IV. Cancellation takes place only at the right end. That is, $i = i_r$ and $j = j_r$ but $i \neq i_1$ or $j \neq -j_1$. Thus $Z(x_i^j \text{ V } x_i^{-j}) = Z(x_{i_r}^{j_r} x_{i_1}^{j_1} \ldots x_{i_{r-1}}^{j_{r-1}}) =$

$x_{i_r}^{j_r} x_{i_1}^{j_1} \ldots x_{i_{r-1}}^{j_{r-1}}$, and this is a cyclic permutation of $Z(V) = Z(x_{i_1}^{j_1} \ldots x_{i_r}^{j_r}) = x_{i_1}^{j_1} \ldots x_{i_r}^{j_r}$. Thus, in all four cases, $Z(x_i^j \; V \; x_i^{-j})$ is a cyclic permutation of $Z(V)$. This shows that $Z(V)$ is a cyclic permutation of $Z(T \; V \; T^{-1})$ whenever T has a length of one.

Assume now the inductive hypothesis that $Z(K \; V \; K^{-1})$ is a cyclic permutation of $Z(V)$. Then, by the above four cases, $Z(x_i^j \; K \; V \; K^{-1} \; x_i^{-j})$ is a cyclic permutation of $Z(K \; V \; K^{-1})$ and so $Z(x_i^j \; K \; V \; K^{-1} \; x_i^{-j})$ is a cyclic permutation of $Z(V)$. Q.E.D.

1.7 Related Problems

The Power-Problem for Free Groups is deciding in a finite number of steps whether a given word V is equivalent to a power of a given word U; i.e., is $V = U^k$ for some finite value of k.

The Power-Conjugacy Problem for Free Groups is deciding in a finite number of steps whether a given word V is equivalent to the conjugate of a power of U; i.e., is $V = W^{-1} U^k W$ for some word W and some finite value of k. These problems are examined in later chapters.

1.8 The Groups of Dehn's Algorithm

In free groups, the solution to the word problem is obtained by a relatively simple algorithm which can be described as a monotonic reduction process: given W, a non-empty word in the generators of G which defines the identity element, one can replace a subword if W by a shorter word and thus derive a new word, W', of shorter length than W which still defines the identity element. Furthermore, the allowable replacements for the subword come from a finite list.

Dehn, in 1912, discovered a class of groups (other than the free groups) whose word problem is also solved by a monotonic reduction process (see Magnus, Karrass, Solitar, 1966). Dehn's solution, however, was a geometric one. Greendlinger (1960), discovered an algebraic solution to the problem. We introduce here the concepts that define what we call the groups solvable by Dehn's algorithm, i.e., the DA Groups.

From Magnus, Karrass, Solitar (1966), we get the following definitions:

Let G be a group on the generators a_1, a_2, . . . and let R_u (u=1,2,3,...m) be a finite set of defining relators for G in the generators. The relators R_u have the following properties:

- The R_u are cyclically reduced and non-empty.
- For every relator R_e in R_u, the set of defining relators R_u also contains R_e^{-1} and all the cyclic permutations of R_e and R_e^{-1}.

We call such a set of defining relators, $\{R_u\}$, a symmetric or symmetrized set. Every finite set of relators can be replaced by a symmetric set (involving the same generators) as follows: add the missing inverses to the set of relators, cyclically reduce all the relators thus far derived, and then add all the cyclic permutations. Thus, symmetric sets are closed under the operations of inverses and cyclic permutations.

If $\{R_u\}$ is a symmetric set of defining relators for G, where G is a DA-group, we can solve the word problem for G by the following monotonic reduction process:

1. Find a syllable (subword) V of W such that V is identical with a syllable of one of the relators $R_w \in \{R_u\}$, such that $|V| > \frac{1}{2}|R_w|$.
2. Replace V with the inverse of the remaining segment of the relator R_w, thus reducing the length of the word W.

3. Continue this process until either

   a. the word has been reduced to the identity element (length 0) in which case W=1, or

   b. No subword V of W matches a subword of a relator R_W such that $|V|>\frac{1}{2}|R_W|$. In this case, $W\neq 1$.

This algorithm describes how W is freely reduced for the DA groups denoted as free R-reduction. Greendlinger (1960) applied this monotonic reduction process to certain classes of presentations of groups solvable by Dehn's algorithm (see Magnus, Karrass, Solitar, 1966 for a discussion of less than 1/K groups).

The Conjugacy Problem is solved in a straightforward manner for these groups. Words U and V are first cyclically reduced. This is done by first free R-reducing them and then cancelling occurrences of relators that begin at the right and end at the left. The remaining word is called cyclic R-reduced. Finally, if U and V are cyclic permutations, then we conclude that they are conjugate elements.

1.9 The Model of Computation

The complexity of an algorithm is defined as the amount of "time" and "space" required by the algorithm that solves the given problem. To define "time" and

"space", a model of computation is needed. Here, a multitape Turing Machine as described by Stockmeyer (1974) is used as the model. Note that the multitape Turing Machine is not the only model of computation that is available (i.e. RAM, RASP, etc.). More information concerning these alternative models of computation is contained in Aho, Hopcroft and Ullman (1974).

For the sake of brevity, a multitape Turing Machine is abbreviated TM. On a TM, the tapes which handle the input/output processes are separated from the tapes which serve as memory. Every TM consists of a finite state control and K+2 tapes (K>0): an input tape, K worktapes, and an output tape. The tape head that reads the input tape is read-only and moves two ways (left and right). The worktape heads are read-write, two way, and the output head write-only, right moving. See diagram 1.

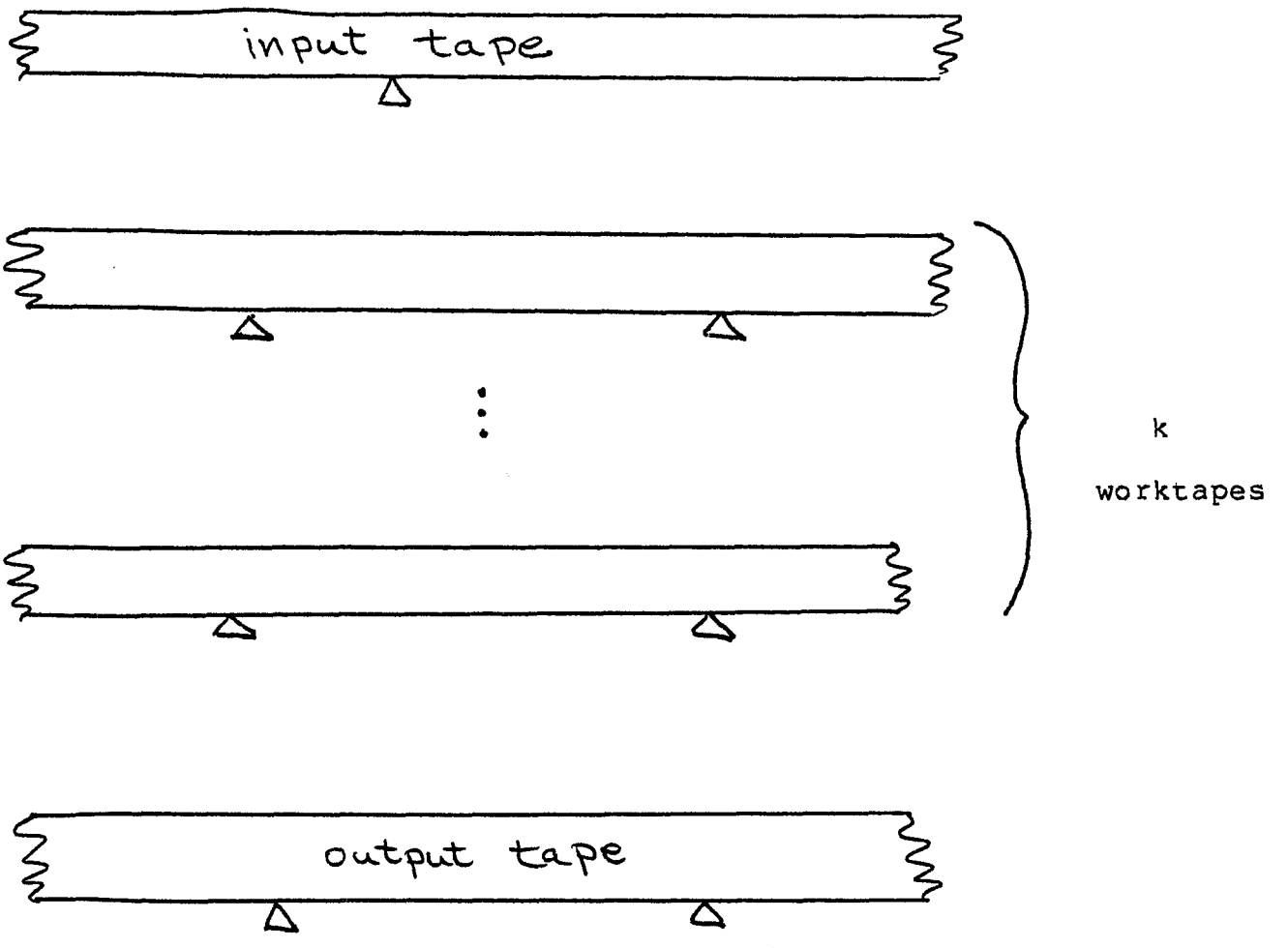


Diagram 1

Depending upon the current state of finite control and the symbol being scanned, the TM may, in <u>one step</u>,

- change state,
- print new symbols where worktape and output heads are located, and
- shift tape heads to allowable adjacent positions.

The time required for a TM to complete its computation is defined as the number of steps executed by the TM. The space used by a TM is defined as the number of different worktape cells visited by the TM during the computation.

Given input of length n, we define

linear time (O(n)): The Turing Machine executes c*n steps in the course of its computation for some constant c>1.

real time Linear time where c=1.

log space (O(log n)): The Turing Machine visits c*(log n) worktape cells during the course of its computation.

1.10 A High-Level Notation for Expressing Turing Machine Algorithms

This section introduces some standard high-level language notation which makes the presentation of Turing Machine algorithms simpler. The reader interested in work of this kind should consult (Knuth, 1969).

The contents of a Turing Machine (TM) tape W as denoted as an array of symbols. Thus, the symbol being scanned by tape head H on tape W is denoted W(H). When presenting the contents of a TM tape, Hs implies tape

head H is currently reading the symbol s. For example: W: 12H34. Here, the contents of tape W is 1,2,3, and 4, and W(H)=3.

Moving a tape head H right one position is denoted by H=H+1, and moving left one position by H=H-1.

A DO WHILE loop is used to imply repeating a set of instructions while some condition remains true. For example:

```
I=1;
DO WHILE (I<4);
   I=I+1;
END;
```

I will be incremented a total of 3 times during execution of this loop, leaving I with a final value of 4.

An IF-THEN-ELSE statement is used to denote conditional execution of certain operations. For example:

```
I=2;
IF I<4 THEN I=I+1;
       ELSE I=0;
```

In this example, I would have a final value of 3. Had I been initialized to 10, I would have a final value of 0.

These statements can be combined to define more complicated operations. For example:

```
IF I<4
   THEN DO; I=I+1;
            J=J-1;
        END;
   ELSE DO WHILE (J>10);
            I=I+1;
            J=J-5;
        END;
```

2. THE TIME COMPLEXITY OF THE CONJUGACY PROBLEM FOR FREE GROUPS

A linear upper time bound is shown to exist for the solution to the conjugacy problem for free groups. The algorithm uses a multitape Turing Machine as the model of computation, as well as techniques from pattern matching.

2.1 Cyclic Reduction of Input

The first phase of the algorithm cyclically reduces the input words U and V. This is done in linear time. Assume the input is coded as follows on the input tape: (blanks) V$U& (blanks). The input head initially is positioned under the leftmost symbol of V.

The same algorithm can be applied to cyclically reduce U as well as V. Thus, an algorithm is described which cyclically reduces any one word, W of length m, in time O(m). This algorithm will be modified so that it works on our input string V$U&.

Given the input W$, where $ is a marker symbol, the algorithm requires two worktapes. This algorithm consists of two parts: a) freely reducing the input and b) cancelling first and last symbols wherever possible.

Consider part a, freely reducing W. Assume the input tape and one worktape are configured as follows:

INPUT: $IX_1^p\ X_2^q\ .\ .\ .\ X_n^z\$$

WORKTAPE (W): Hb

Let HX denote that tape head (H) is currently scanning the symbol X. Let b denote a blank symbol.

Step 1 of the algorithm copies the first symbol of the input tape onto the worktape. The input head then moves right one position.

```
W(H) = INPUT (I);
I = I+1;
```

The following procedure outlines the next m-1 steps of the algorithm. Assume the input symbol currently being scanned is X_i^j and the worktape symbol on the worktape currently being scanned is X_k^l. Consider the following:

Case a) i=k and j=-1

We want to cancel these two symbols. Therefore, 1) erase the current symbol being scanned on the worktape, 2) move the worktape head <u>left</u> one position, and 3) advance the input head <u>right</u> one position.

Case b) $i \neq k$ or j=1

This implies adjacent symbols have been

scanned which do not cancel. Hence, 1) advance the worktape head right one position, 2) write X_i^j onto the worktape, and 3) advance the input head right one position.

In each step, the input head is advanced one position to the right, and inverse symbols are erased wherever possible. The algorithm produces W', the freely reduced version of W, on the worktape when the delimiter is scanned, in this case, the $ symbol. Thus, this algorithm operates in <u>real time</u> (m steps).

```
DO WHILE (INPUT(I) ≠ $);
   IF W(H) = - INPUT(I)
      THEN DO;  W(H) = b;
                H = H-1;
                I = I+1;
           END;
      ELSE DO;  H = H+1;
                W(H) = INPUT(I);
                I = I+1;
           END;
END;
```

The action of the algorithm is illustrated below when $W = X_2^2X_1^2X_1^{-1}X_2X_3X_2^{-2}$. The contents of both the input and worktapes are displayed at the conclusion of each step.

(I = Input Tape; W = Work Tape)

step 1 INPUT: $X_2IX_2X_1X_1X_1^{-1}X_2X_3X_2^{-1}X_2^{-1}\$$

W: HX_2

step 2 (case b) INPUT: $X_2X_2IX_1X_1X_1^{-1}X_2X_3X_2^{-1}X_2^{-1}\$$

W: X_2HX_2

step 3 (case b) INPUT: $X_2X_2X_1IX_1X_1^{-1}X_2X_3X_2^{-1}X_2^{-1}\$$

W: $X_2X_2HX_1$

step 4 (case b) INPUT: $X_2X_2X_1X_1IX_1^{-1}X_2X_3X_2^{-1}X_2^{-1}\$$

W: $X_2X_2X_1HX_1$

step 5 (case a) INPUT: $X_2X_2X_1X_1X_1^{-1}IX_2X_3X_2^{-1}X_2^{-1}\$$

W: $X_2X_2HX_1$

step 6 (case b) INPUT: $X_2X_2X_1X_1X_1^{-1}X_2IX_3X_2^{-1}X_2^{-1}\$$

W: $X_2X_2X_1HX_2$

step 7 (case b) INPUT: $X_2X_2X_1X_1X_1^{-1}X_2X_3IX_2^{-1}X_2^{-1}\$$

W: $X_2X_2X_1X_2HX_3$

step 8 (case b) INPUT: $X_2X_2X_1X_1X_1^{-1}X_2X_3X_2^{-1}IX_2^{-1}\$$

W: $X_2X_2X_1X_2X_3HX_2^{-1}$

step 9 (case b) INPUT: $X_2X_2X_1X_1X_1^{-1}X_2X_3X_b^{-1}X_2^{-1}I\$$

W: $X_2X_2X_1X_2X_3X_2^{-1}HX_2^{-1}$

At this point, the delimeter symbol $ is scanned, and the algorithm halts with the freely reduced version of W, W', on the worktape, which has been produced in

real time (9 steps).

Next W' is cyclically reduced. This is accomplished by using a second worktape whose head is called J. When the $ from the input tape is read, 1) worktape head 1 and the input head are moved right one position, and 2) a $ is written onto both worktapes. A copy - backwards operation is then performed; that is, both heads are moved left and the symbols from worktape 1 are copied onto worktape 2. This procedure is repeated until a blank is scanned. This operation requires at most m+2 steps.

```
H=H+1;
I=I+1;

W1(H)=$;
W2(J)=$;

DO WHILE (W1(H)  ≠  b);

   H=H-1;
   J=J-1;
   W2(J)=W1(H);
END;
```

To illustrate, we continue with the present example. Only the states of the two work tapes (W1 and W2) are shown.

step 1: INPUT: $X_2X_2X_1X_1X_1^{-1}X_2X_3X_2^{-1}X_2^{-1}\$I$

W1: $X_2X_2X_1X_2X_3X_2^{-1}X_2^{-1}H\$$

W2: $J\$$

Copy-backwards

step 2 W1: $X_2X_2X_1X_2X_3X_2^{-1}HX_2^{-1}\$$

W2: $JX_2^{-1}\$$

.
.
.

step 8 W1: $HX_2X_2X_1X_2X_3X_2^{-1}X_2^{-1}\$$

W2: $JX_2X_2X_1X_2X_3X_2^{-1}X_2^{-1}\$$

step 9 (stop) W1: $HbX_2X_2X_1X_2X_3X_2^{-1}X_2^{-1}\$$

W2: $JbX_2X_2X_1X_2X_3X_2^{-1}X_2^{-1}\$$

(Note: b is a blank square)

Next, the worktape heads are positioned at opposite ends of the two copies of W'. This requires that worktape head J moves right one position at a time over W' until the delimiter symbol, \$, is found. Worktape head J is then moved left one position (scanning the rightmost symbol of W') and worktape head 1 is moved right one position (scanning the leftmost symbol of W'). This operation requires at most m+2 steps.

```
DO WHILE (W2(J) ≠ $);
   J=J+1;
END;
J=J-1;
H=H+1;
```

Continuing with the example, the configuration of worktape 2 is displayed:

step 1 W2: $JX_2X_2X_1X_2X_3X_2^{-1}X_2^{-1}\$$

step 2 W2: $X_2JX_2X_1X_2X_3X_2^{-1}X_2^{-1}\$$

.
.
.

step 7 W2: $X_2X_2X_1X_2X_3X_2^{-1}JX_2^{-1}\$$

step 8 W2: $X_2X_2X_1X_2X_3X_2^{-1}X_2^{-1}J\$$

step 9 W1: $HX_2X_2X_1X_2X_3X_2^{-1}X_2^{-1}\$$
W2: $X_2X_2X_1X_2X_3X_2^{-1}JX_2^{-1}\$$

A third worktape is used which contains the final cyclically reduced word W whose head is named K. Recall that the tape heads are positioned at opposite ends of W'. X_i^j from worktape 1 is compared with X_k^1 from worktape 2. Consider the possible outcomes:

Case a) i=k and j=-1

This implies that the two symbols being scanned are inverses and should be cancelled. Here 1) worktape head H is moved <u>right</u> one position, and 2) worktape head J is moved <u>left</u> one position.

```
DO WHILE (W1(H) ≠ $);
   IF W1(H)=-W2(J)
      THEN DO;  H=H+1;
                J=J-1;
           END;
```

Case b) $i \neq k$ or $j=1$

This implies the two symbols being scanned should not be cancelled. In this case 1) the symbol scanned from worktape 1 is written onto worktape 3, 2) worktape heads 1 and 3 are moved <u>right</u> one position and 3) worktape head 2 is moved <u>left</u> one position.

```
      ELSE DO;
           W3(K)=W1(H);
              H=H+1;
              K=K+1;
              J=J-1;
           END;
   END;
```

Case c) $ scanned from worktape 1

This implies each symbol on the worktapes have been scanned, and thus, the algorithm halts. Worktape 3 contains W", the cyclically reduced version of W.

To illustrate, the example is continued, showing the states of the three worktapes at the conclusion of each step.

step 1 (case a)
W1: $X_2HX_2X_1X_2X_3X_2^{-1}X_2^{-1}\$$
W2: $X_2X_2X_1X_1X_3JX_2^{-1}X_2^{-1}\$$
W3: Kb

step 2 (case a)
W1: $X_2X_2HX_1X_2X_3X_2^{-1}X_2^{-1}\$$
W2: $X_2X_2X_1X_2JX_3X_2^{-1}X_2^{-1}\$$
W3: Kb

step 3 (case b)
W1: $X_2X_2X_1HX_2X_3X_2^{-1}X_2^{-1}\$$
W2: $X_2X_2X_1JX_2X_3X_2^{-1}X_2^{-1}\$$
W3: X_1Kb

step 4 (case b)
W1: $X_2X_2X_1X_2HX_3X_2^{-1}X_2^{-1}\$$
W2: $X_2X_2JX_1X_2X_3X_2^{-1}X_2^{-1}\$$
W3: X_1X_2Kb

step 5 (case b)
W1: $X_2X_2X_1X_2X_3HX_2^{-1}X_2^{-1}\$$
W2: $X_2JX_2X_1X_2X_3X_2^{-1}X_2^{-1}\$$
W3: $X_1X_2X_3Kb$

step 6 (case a)
W1: $X_2X_2X_1X_2X_3X_2^{-1}HX_2^{-1}\$$
W2: $JX_2X_2X_1X_2X_3X_2^{-1}X_1^{-1}\$$
W3: $X_1X_2X_3Kb$

step 7 (case a)
W1: $X_2X_2X_1X_2X_3X_2^{-1}X_2^{-1}H\$$
W2: $JbX_2X_2X_1X_2X_3X_2^{-1}X_2^{-1}\$$

W3: $X_1X_2X_3Kb$

At this point the $ is being scanned on worktape 1 and worktape 3 contains W".

The complexity of this operation is examined. At each step, worktape head 1 is moved right one position until the delimiter symbol is found. Thus, since worktape 1 contained at most m symbols, this operation requires at most m steps. Summarizing, if string W$ has length m+1, it can be cyclically reduced W in time at most (m) + (m+2) + (m+2) + (m) = 4m+4 steps.

2.2 Procedure for Determining Cyclic Permutations

The changes required to the above algorithm to cyclically reduce the input string V$U& are now summarized.

First, V$ is freely reduced exactly as outlined above. Worktapes tapes 1, 3, and 5 of the TM are used, where tape 5 will contain V", the cyclically reduced version of V, with head 5 positioned at the first blank square past the rightmost symbol of V". Next, the processing of the input tape continues by cyclically reducing U&. Here, tapes 2, 4 and 6 are used as the three worktapes, with tape 6 eventually containing U", the cyclically reduced version of U. As a trivial modification to the algorithm, an & is used as the delimiter symbol instead of a $. Note that worktape head

6 is positioned one square past the rightmost symbol of U".

To illustrate these changes to the algorithm, let $V = X_2^2X_1^2X_1^{-1}X_2X_3X_2^{-2}$ (the value of W in our previous example) and let $U = X_3^{-1}X_2X_2^{-1}X_2X_3X_1X_3$. Let the heads of tapes W1 through W6 be named H, J, K, L, M, N respectively. Initially,

INPUT: $IX_2X_2X_1X_1X_1^{-1}X_2X_3X_2^{-1}X_2^{-1}$ \$ $X_3^{-1}X_2X_2^{-1}X_2X_3X_1X_3$ &

W1, W2, W3, W4, W5, W6: b

After cyclically reducing V, the configuration of the tapes is:

INPUT: $X_2X_2X_1X_1X_1^{-1}X_2X_3X_2^{-1}X_2^{-1}$ \$ $IX_3^{-1}X_2X_2^{-1}X_2X_3X_1X_3$ &

W1: $X_2X_2X_1X_2X_3X_2^{-1}X_2^{-1}$H\$

W3: $KbX_2X_2X_1X_2X_3X_2^{-1}X_2^{-1}$\$

W5: $X_1X_2X_3$b (V'')

W2, W4, W6: b

Next, the contents of the tapes after cyclically reducing U is shown below:

INPUT: $X_2X_2X_1X_1^{-1}X_1X_2X_3X_2^{-1}X_2^{-1}$ \$ $_3^{-1}X_2X_2^{-1}X_2X_3X_1X_3$ &IbX_3 &Ib

W1, W3, W5: as before

W2: $X_3^{-1}X_2X_3X_1X_3$ J&

W4: $LbX_3^{-1}X_2X_3X_1X_3$&

W6: $X_2X_3X_1Nb$ (U'')

To proceed with the algorithm, U" and V" must have equal lengths, say n. As a first step, heads 5 and 6 (M,N) are moved left one position so they are positioned at the rightmost symbols of V' and U' respectively. Then, the TM scans left one position at a time on both tapes 5 and 6. This is continued until a blank square is encountered on either tape. If a blank is found on one tape and not on the other, U' and V' are of unequal lengths, and hence cannot be cyclic permutations. Therefore U and V are not conjugate elements.

```
M=M-1;
N=N-1;
DO WHILE (W5(M) ≠ b);
   IF W6(N)=b  FAILURE;
      ELSE DO;  N=N-1;
                M=M-1;
           END;
END;
```

On the other hand, if blanks are found on both tapes 5 and 6 at the same time, U' and V' are of equal length, and the TM goes on to the cyclic permutation phase of the algorithm. At this point both tape heads are moved right one position.

```
IF W5(M)=b & W6(N)=b
   THEN DO;  M=M+1;
             N=N+1;
        END;
```

To illustrate this "length checking" phase, consider the example acting upon tapes 5 and 6:

Initially	W5:	$X_1X_2X_3Mb$	(V')
	W6:	$X_2X_3X_1Nb$	(U')
step			
1	W5:	$X_1X_2MX_3$	(move left when non-
	W6:	$X_2X_3NX_1$	blanks are scanned)
2	W5:	$X_1MX_2X_3$	
	W6:	$X_2NX_3X_1$	
3	W5:	$MX_1X_2X_3$	
	W6:	$NX_2X_3X_1$	
4	W5:	$Mb\ X_1X_2X_3$	(move right when blanks
	W6:	$Nb\ X_2X_3X_1$	are scanned)
5	W5:	$MX_1X_2X_3$	
	W6:	$NX_2X_3X_1$	

The complexity of the checking of the length of U" and V" is simply length of U' (or V'), which has an upper bound of n, if $|U| = |V| = n$. Thus complexity = n+2.

A slight modification to the length checking phase of the algorithm is needed to ease the future operation of the TM. While passing left over tape 6 (U"), the TM will make a copy of U" on another tape, say tape 2. To do this, head J is moved in the same direction as head N and is moved whenever head N moves. However, when head N is reading a symbol, that symbol is written on tape 2. Recall that worktape head J is positioned at the & just right of the freely reduced version of U at the start of this copy operation. Note, we no longer need the freely reduced word on worktape 2.

Consider the following example:

Initially,	W2:	$X_3^{-1}X_2X_3X_1X_3J\&$	
	W5:	$X_1X_2X_3Mb$	
	W6:	$X_2X_3X_1Nb$	
step			
1	W2:	$X_3^{-1}X_2X_3X_1JX_3\&$	(move all 3 heads left)
	W5:	$X_1X_2MX_3$	
	W6:	$X_2X_3NX_1$	
2	W2:	$X_3^{-1}X_2X_3JX_1X_1\&$	(X_1 from W6 was
	W5:	$X_1MX_2X_3$	written onto W2 before
	W6:	$X_2nX_3X_1$	moving left)
3	W2:	$X_3^{-1}X_2JX_3X_3X_1\&$	(X_3 from W6 was
	W5:	$MX_1X_2X_3$	written onto W2 before
	W6:	$NX_2X_3X_1$	moving left)

4	W2:	$X_1^{-1}JX_2X_2X_3X_1\&$	(X_2 from W6 was
	W5:	$Mb\ X_1X_2X_3$	written onto W2 before
	W6:	$Nb\ X_2X_3X_1$	moving left)
5	W2:	$X_3^{-1}bJX_2X_3X_1\&$	(the b from W6 was
	W5:	$MX_1X_2X_3$	written onto W2 before
	W6:	$NX_2X_3X_1$	moving right)

Thus, if |U"| = |V"|, the tapes are configured in the following manner at the the length checking:

tape 2: $JU'\ \&$ (the head points to the leftmost symbol of U")

tape 5: MV' (the head points to the leftmost symbol of V")

tape 6: NU' (the head points to the leftmost symbol of U")

The final operation performed before checking for cyclic permutations consists of making V" & U"U" the contents of worktape 1. The reason for this becomes clear in the next section. For the moment, recall that worktape head H is currently positioned at the $ just right of the freely reduced version of V.

This phase of the algorithm begins by moving worktape head H right, and copying one position of V" at a time onto worktape 1 until a blank is encountered on

worktape 5. This takes |V"| steps, which is at most n. Next, an & is written onto worktape 1, taking 1 step. Next, U" is copied from worktape 2 onto worktape 1, using the & as the delimeter. This takes |U"| steps, which is at most n. Next, U" is copied from tape 6 onto tape 1, taking at most n steps. Here, the blank on tape 6 is used as the delimeter. Finally, tape head 1 is positioned at the leftmost symbol of V". This means passing the head left over V" & U"U" until the $ is found, and then moving tape head 1 right one square. This last operation requires |V"| + 1 + |U"| + |U"| steps, which is bounded by n+1+n+n = 3n+1.

```
H=H+1;
DO WHILE (W5(M) ≠ b);
   W1(H)=W5(M);
   H=H+1;
   M=M+1;
END;

W1(M)=&;
M=M+1;
DO WHILE (W2(J) ≠ &);
   W1(H)=W2(J);
   J=J+1;
   H=H+1;
END;
```

```
DO WHILE (W6(N) ≠ b);
   W1(H)=W6(N);
   H=H+1;
   N=N+1;
END;

DO WHILE (W1(H) ≠ $);
   H=H-1;
END;
H=H+1;
```

To illustrate the final phase of the setup algorithm, consider the example below:

Initially W1: (non-blank symbols) H$

W2: (non-blanks) $bJX_2X_3X_1\&$ (U')

W5: $MX_1X_2X_3$ (V')

W6: $NX_2X_3X_1$ (U')

For brevity, the leftmost non-blank symbols on tapes 1 and 2 will no longer be considered.

step			
	W1:	HX_1	(copy V' from W5
1	W5:	$X_1MX_2X_3$	onto W1)
	W2, W6: no change		
	.		
	.		
	.		
3	W1:	$\$X_1X_2HX_3$	
	W5:	$X_1X_2X_3Mb$	
4	W1:	$\$X_1X_2X_3H\&$	
	W5:	$X_1X_2X_3Mb$	(no change)
	W2:	$bJX_2X_3X_1\&$	
	W6:	$NX_2X_3X_1$	
5	W1:	$\$X_1X_2X_3\&HX_2$	(X_2 copied from W2)
	W2:	$X_2JX_3X_1$	
6	W1:	$X_1X_2X_3\&X_2HX_3$	(X_3 copied from W2)
	W2:	$X_2X_3JX_1$	
7	W1:	$\$X_1X_2X_3\&X_2X_3HX_1$	(X_1 copied from W2)
	W2:	$X_2X_3X_1Jb$	
	W6:	$NX_2X_3X_1$	
8	W1:	$\$X_1X_2X_3\&X_2X_3X_1HX_2$	(X_2 copied from W6)
	W6:	$X_2NX_3X_1$	
9	W1:	$\$X_1X_2X_3\&X_2X_3X_1X_2HX_3$	(X_3 copied from W6)
	W6:	$X_2X_3NX_1$	

10 W1: $\$X_1X_2X_3\&X_2X_3X_1X_2X_3HX_1$ (X_1 copied from W6)

W6: $X_2X_3X_1Nb$

11 W1: $\$X_1X_2X_3\&X_2X_3X_1X_2HX_3X_1$

.

. (move head left at each step)

.

20 W1: $H\$X_1X_2X_3\&X_2X_3X_1X_2X_3X_1$

21 W1: $\$HX_1X_2X_3\&X_2X_3X_1X_2X_3X_1$

= $\$H\ V'\ \&\ U'\ U'$

Note that on tapes 2, 5 and 6, the heads are now positioned to the right of the last X_i .

The following summarizes the complexity of cyclically reducing the input:

Reducing V using tapes 1, 3, 5	4n+4
Reducing U using tapes 2, 4, 6	4n+4
Length checking and copying U' onto tape 2	n+2
Making $V'\&U'U'$ on work tape 1	6n+2

Thus, the upper bound on the entire operation is 15n+12.

2.3 The Cyclic Permutation Problem for Free Groups

For notational convenience, U" and V" of the previous section will now be referred to as U and V respectively.

Fundamental Lemma: Let U and V be two words in the free generators of a free group F. Suppose U and V are cyclically reduced with $|U| = |V| = n$. Then there exist two words P and Q in the free generators of F such that $|P| + |Q| = n$ and

$U^2 = PVQ$, iff U and V are cyclic permutations of each other.

Proof: (1) Given that U and V are cyclic permutations of each other. There exist syllables (words) P and Q of U and V such that $U = PQ$ and $P = QP$. Thus, $U^2 = (PQ)(PQ) = P(QP)Q = PVQ$. Obviously $|P| + |Q| = n$ since $|P| + |Q| = |PQ| = |U| = n$.

(2) Now suppose that $U^2 = PVQ$. We have $|P| + |Q| = n$ since $2n = |U^2| = |PVQ| = |P| + |V| + |Q| = |P| + n + |Q|$. Let $|P|=p$, and $|Q|=q$. Since $U^2 = PVQ$ the first p symbols of U^2 must be the same syllables as the first p symbols of PVQ, namely P. Similarly, the last q symbols of U^2 must be the syllable Q. Thus $U = PQ$ since $P + Q = n$. Hence $V=QP$ since $U^2 = (PQ)(PQ) = P(QP)Q = PVQ$.

The above lemma is used to solve the cyclic permutation problem for free groups freely presented as follows: given U and V, the algorithm searches for the existence of V in the string U^2. If it exists, U and V are cyclic permutations, (hence, conjugate elements)

otherwise they are not.

2.4 Pattern Matching on a Turing Machine

What the problem has reduced to is a straightforward problem in pattern matching: finding an occurrence of one string within another in linear time. Knuth, Morris and Pratt (1977) present an algorithm which finds all occurrences of one string within another in running time proportional to the sum of the lengths of the strings. For the conjugacy problem, this would be $|V| + 2|U|$ which is $O(n)$. Fischer and Paterson (1974) show that the Knuth, Morris and Pratt algorithm can be implemented on a Multitape Turing Machine in linear time. What remains to be shown, then, is the Fischer and Paterson Turing Machine implementation for the conjugacy problem. We shall now carry out the application of that machine. We give an example of the implementation of this algorithm.

2.5 Backround

Given a text string $X=X_0X_1...X_t$, and a pattern string $Y=Y_0\ Y_1...Y_p$, where both X and Y are words over the same finite alphabet Σ, the task is to find i, where $t \leq i \leq p$ such that

$Y=[X_{i-p}...X_i]$; that is, Y is a consecutive substring in the text X. The method traditionally used to search for a matching pattern meant searching at every starting position of text, abandoning the search as soon as

an incorrect character is found. When this "mismatch" occurs, the pattern is "backed up" to its beginning and the text is "backed up" to the next character to be used as a start for comparisons. This traditional algorithm is inefficient, for example, consider aaaab in aaaaaaaaab. The complexity of this algorithm is $0(n^2)$.

2.6 Knuth, Morris, Pratt Algorithm

The key idea to the Knuth, Morris, Pratt algorithm is that if a segment of the text (in our case U^2) has successfully been matched with an initial segment of the pattern (V) before reaching an inequality, then it is unnecessary to read those symbols of the text again, since they are the same as the symbols of the pattern segment. The procedure used by the Knuth, Morris, Pratt algorithm carries out the first comparisons for the next relative position of the pattern V, by comparing V with a segment of itself, and these comparisons can be precomputed once at the beginning. This precomputation is very quick and has the same form as the main computation itself.

2.7 A Theoretical Description of the Algorithm

Given a word $Z=Z_0 \ldots Z_t$, the function P for $i=0,1,\ldots t$ is defined by $P(i) = \max \{s | Z_0 \ldots Z_s = Z_{i-s} \ldots Z_i \}$. It is not difficult to verify

that $P^K(i)$ = the K^{th} largest s such that $Z_0 \ldots Z_s = Z_{i-s} \ldots Z_i$, where $P^K(i)$ notes the composition of P with itself K times. Thus, $P^0(i) = i$ and $P^{K+1}(i) = P(P^K(i))$. Consider the following recursive definition for P:

$P(0) = -1$

$$P(i+1) = \begin{cases} = P^k(i) + 1 \text{ for } 0 \leq i \leq t \text{ where k is the smallest integer } >0 \text{ such that } Z_{[P^k(i)+1]} = Z_{i+1}. \\ = -1 \text{ if } P^k(i) = -1 \end{cases}$$

Described below are the implications of this recursive definition for P. Initially, define P(0) as -1. Next, consider P(i+1), to illustrate what the recursive definition implies, consider the following

string Z	a	b	a	b
position	0	1	2	3
P	-1	-1	0	?

Assume the values for P(0), P(1), and P(2) are given as above. The value for P(3), will be computed, which implies i=2.

The recursive definition says to check whether

$$Z_{[P^1(2)+1]} = Z_{2+1}?$$

$$\Rightarrow Z_{[0+1]} = Z_3$$

$$\Rightarrow Z_1 = Z_3$$

In this case, $Z_1 = Z_3$ (b=b). Thus, the definition for P says $P^k(i)+1$ is used as the value for P(i+1), which is

$P^1(2)+1 = 1$. Thus $P(3) = 1$ which implies

$Z_0Z_1 = Z_2Z_3$! Upon a close examination of the algorithm, one sees that information obtained previously was used; namely that $P(2) = 0$. The value of P(2) implies that $Z_0 = Z_2$. Thus, the definition merely says to check whether

$Z_1 = Z_3$ which would imply

Consider the outcome when events don't work out quite so well:

position	0	1	2	3	4	5	6
string z	a	b	a	a	b	a	b
P	-1	-1	0	0	1	2	?

In this example, the value for P(6) will be found, implying the i of the definition is 5. First we check to see if

$$Z_{[P^1(5)+1]} = Z_{5+1}$$

$$\Rightarrow Z_3 = Z_6. \quad (a \neq b).$$

In this case $Z_3 \neq Z_6$, the value of k is incremented by 1. This implies the algorithm will try to find a smaller value of s such that

$Z_0 \ldots Z_s = Z_{6-s} \ldots Z_6$.

$Z_{P^2(5)+1} = Z_6$

$\Rightarrow Z_{[P(P(5))+1]} = Z_6$

$\Rightarrow Z_{[P(2)+1]} = Z_6$

$\Rightarrow Z_{[0+1]} = Z_6$

$\Rightarrow Z_1 = Z_6$ (b=b)

In this case, $Z_1 = Z_6$, thus the value of 1 is assigned to P(6), implying $Z_0 Z_1 = Z_5 Z_6$. What the algorithm made use of in this case was the <u>second</u> largest value of t such that

$Z_0 \ldots Z_s = Z_{i-s} \ldots Z_i$; namely,

$Z_0 \ldots Z_0 = Z_{5-0} \ldots Z_5$ (a=a)! This second largest value of s was obtained by using

$P^2(5) = P(P(5)) = P(2) = 0$! Recall that the algorithm is always looking for a substring match starting from the <u>beginning</u> of the pattern (Z_0).

A complete example is shown below.

position	0	1	2	3	4	5	6	7	8
string Z	a	b	a	a	b	a	b	a	a
P	-1	-1	0	0	1	2	1	2	3

$P(2)=0 \Rightarrow Z_0 = Z_2$ (a=a)

$P(3)=0 \Rightarrow Z_0 = Z_3$ (a=a)

$P(4)=1 \Rightarrow Z_0Z_1 = Z_3Z_4$ (ab=ab)

$P(5)=2 \Rightarrow Z_0Z_1Z_2 = Z_3Z_4Z_5$ (aba=aba)

$P(6)=1 \Rightarrow Z_0Z_1 = Z_5Z_6$ (ab=ab)

$P(7)=2 \Rightarrow Z_0Z_1Z_2 = Z_5Z_6Z_7$ (aba=aba)

$P(8)=3 \Rightarrow Z_0Z_1Z_2Z_3 = Z_5Z_6Z_7Z_8$ (abaa=abaa)

Since $P(j+1)-1 \leq P(j)<j$ for all j, the value for $P(i+1)$ may be computed in time less than $C.(P(i) - P(i+1) + 2)$ for some constant C, provided that the string Z and the values of $P(j)$ for $j<i$ are readily accessible. To show this, consider the worst case example: $P(i+1)$ will be assigned the value of -1 because there is no K such that $Z_{[P^K(i)+1]} = Z_{i+1}$. Here, the following comparisons are made:

$Z_{P(i)+1}$	with Z_{i+1}	
$Z_{P^2(i)+1}$	with Z_{i+1}	at most
.	. .	$C.(P(i)+1)$
.	. .	comparisons
.	. .	

$Z_{P(0)+1}$ with Z_{i+1}

When all of these fail, -1 is assigned to P(i+1). Thus, at most C.(P(i)+1 +1) = C.(P(i)+2) operations are required. The expressions yield the following:

C.(P(i) - P(i+1)+2)

=C.(P(i) - (-1)+2)

=C.(P(i)+3) which is greater than C.(P(i)+2).

Therefore, the K of the recursive definition of P satisfies P(i)-P(i+1)+2$\geq$K$\geq$1. Thus, the total running time of the algorithm is bounded by C.(P(0) - P(t)+2C(t+1)) = 0(t).

For the original problem, define V as $V_1 \ldots V_n$ and U as $U_1 \ldots U_n$. Therefore, to solve the conjugacy problem, the pattern string V, a new symbol & and the text string U^2 are concatenated in that order and the values of P for the string $V\&U^2$ are computed in time 0(n+2n) = 0(n). Because of the &, P can never have a value greater than n (=|V|). The values of i for which P(i)=n mark the positions where V matches a substring of U^2, or more precisely

$$P(n+1+i) = n=>U_{i-(n-1)}\cdots U_i = V_1\cdots V_n$$

2.8 The Recursive Algorithm

The linear bound obtained by the Knuth, Morris, Pratt algorithm depends on the use of a random access machine as well as assigning unit cost to the operation of a memory access. Considering that the space required to represent the P array alone would require (n log n) bits when represented as a sequence of binary integers, a linear time Turing Machine implementation seems quite surprising.

A linear-time Turing Machine is implemented by representing P as a table, Δ, of differences in unary notation rather than an array of binary integers. Define $P(-1) = -1$ and let

$$\Delta(i) = 1+P(i)-P(i+1) \quad -1 \leq i < n$$

Then $$P(i) = i - \sum_{j=-1}^{i-1} \Delta(j),$$ and

$$\sum_{j=-1}^{n-1} \Delta(j) = n-P(n).$$

And since $n-P(n) \leq n+1$, the delta array can be represented using unary notation in linear space.

The following is a recursive algorithm for computing P. (We will call this algorithm X.)

Stage (0). Set P(0) =-1; go to Stage (1,1)

Stage (i+1,K).

1. If $Z_{P^K(i)+1} = Z_{i+1}$, set $P(i+1) = P^K(i)+1$ and go to stage (i+2,1).

2. If $P^K(i)=-1$, set P(i+1)=-1 and go to stage (i+2,1).

3. Otherwise, go to stage (i+1, K+1).

The objective is now to rewrite algorithm X without making an explicit reference to P by using the delta array. To do this, three new variables p, s, and d are required. At the beginning of stage (i+1,K), these variables will satisfy

(X1) $p = P^K(i)$

(X2) $s = P^K(i) - P^{K+1}(i)$

(X3) $d = P(i) - P^K(i)$

Algorithm Y maintains these conditions while computing the Δ array.

Algorithm Y

Stage (0) Set $\Delta(-1) = 1$

$p = -1$

$s = 0$

d = 0

go to stage (1,1)

Stage (i+1,K)

Case 1. IF $Z_{p+1} = Z_{i+1}$

THEN BEGIN; Δ (i)=d

p = p+1

s = s + Δ (p)

d = 0

go to stage (i+2,1).

END;

Case 2. ELSE IF p=-1

THEN BEGIN; Δ (i)=d+1

p = p

s = s

d = 0

go to stage (i+2,1)

END;

Case 3. ELSE

BEGIN; p=p-s

$$s=s-\sum_{j=p-s}^{p-1} \Delta(j)$$

d=d+s

go to stage (i+1,K+1).

END;

It remains to be verified that conditions (X1)-

(X3) hold after stage (0), and are preserved after the remaining stages, and that the Δ are computed correctly.

At the end of stage 0:

$p = P^1(0) = -1$

$s = P^1(0) - P^2(0) = -1 - (-1) = 0$

$d = P(0) - P^1(0) = 0$

$\text{delta}(-1) = 1 + P(-1) - P(0)$

$= 1 + (-1) - P(0)$

$= -(-1) = 1$

Stage $(i+1, K)$-

Case 1.0 $Z_{p+1} = Z_{i+1}$

At the end of this stage, go to stage $(i+2, 1)$. Hence, $i \leftarrow i+1$, $K \leftarrow 1$.

$P(i+1) = P^K(i) + 1$ (1.0.1)

$= p+1$

1.1 $\Delta(i) = 1 + P(i) - P(i+1)$ from definition of delta

$= 1 + P(i) - (P^K(i) + 1)$ from (1.0.1)

$= P(i) - P^K(i)$

$= d$ from definition of d

1.2 $p = P^K(i)$ at end of this stage $i \leftarrow i+1$; $K=1$, so

$= P^1(i+1)$

$= P^K(i) + 1$ from $Z_{P^K(i)+1} = Z_{i+1}$

$= p+1$

1.3 $s = s+\Delta(p)$ going to stage $(i+2,1)$,

$$s = P^1(i+1) - P^2(i+1) \quad (1.3.1)$$

$$s = P^K(i)-P^{K+1}(i) + \Delta(P^K(i))$$
$$s = [P^K(i)-P^{K+1}(i)] + [1+P(P^K(i)) - P(P^K(i)+1)]$$
$$s = [P^K(i)-P^{K+1}(i)] + [1+P^{K+1}(i) - P(P(i+1))]$$
$$s = P^K(i)+1 - P^2(i+1)$$
$$s = P^1(i+1) - P^2(i+1) \quad (1.3.1)$$

1.4 $d=P(i)-P^K(i) \rightarrow P(i+1)-P(i+1)=0$

Case 2.0 $P=-1$ implies $P^K(i)=-1 = P(i+1)$

and go to stage $(i+2,1)$ => i <- i+1; K <- 1

2.1 $\Delta(i) = 1+P(i) - P(i+1)$

$$= 1+P(i) - (-1)$$
$$= 2+P(i) \quad (2.1.1)$$

$d+1 = [P(i) - P^K(i)]+1$

$$= [P(i) - (-1)]+1$$
$$= [P(i) +1] +1$$
$$= P(i) +2 \quad (2.1.1)$$

Hence $\Delta(i) = d+1$

2.2 p=p, i <- i+1; K <- 1

show p = $P^K(i)$

= $P^1(i+1)$

but $P^K(i)=-1$

and $P(i+1)=-1$

so p=p!

2.3 s = s, again i <- i+1; K <- 1

show s = $P^K(i)-P^{K+1}(i)$

= $P^1(i+1)-P^2(i+1)$

= -1 -P(-1) (P(P(i+1)) = P(-1))

= -1 -(-1)

= 0

but s = $P^K(i)-P^{K+1}(i)$

= -1 - P($P^K(i)$)

= -1 - P(-1)

= -1 - (-1)

= 0

so s = s

2.4 d=0 i <- i+1; K <- 1

show d = P(i) - $P^K(i)$

= P(i+1) - P(i+1)

= 0

therefore d = 0

Case 3.0 Otherwise, go to stage (i+1,K+1)

from (i+1,K)

3.1 p=p-s i <- i; K <- K+1

show $p = P^K(i)$

$= P^{K+1}(i)$ (3.1.1)

p=p-s yields

$= P^K(i) - [P^K(i) - P^{K+1}(i)]$

$= P^{K+1}(i)$ (3.1.1)

Case 3.2 $s = s - \sum_{j=p-s}^{p-1} \Delta(j)$ i<-i;K<-K+1

show $s = P^K(i) - P^{K+1}(i)$

so $s = P^{K+1}(i) - P^{K+2}(i)$ (3.2.1)

$s = s - \sum_{j=p-s}^{p-1} \Delta(j)$ yields

$= s - ([1+P(p-s) - P(p-s+1)]$

$+ [1+P(p-s+1) - P(p-s+2)]$

$+ \ . \ . \ .$

$+ [1+p(p-1) - P(p)]$

$= s-(s+P(p-s) - P(p))$

$= P(p) - P(p-s)$

$= P(P^K(i)) - P(P^K(i) - (P^K(i) - P^{K+1}(i)))$

$= P^{K+1}(i) - P^{K+2}(i)$ (3.2.1)

3.3 d=d+s i <- i; K <- K+1

show

$$d = P(i) - P^{K}(i)$$
$$= P(i) - P^{K+1}(i) \qquad (3.3.1)$$

d = d+s yields

$$= [P(i) - P^{K}(i)] + [P^{K}(i) - P^{K+1}(i)]$$
$$= P(i) - P^{K+1}(i) \qquad (3.3.1)$$

2.9 The Actual Implementation

Still to be shown is the implementation algorithm Y on a Turing Machine. Fischer and Paterson (1974) implement the algorithm on a multihead, multitape Turing Machine using four tapes, each being one way infinite to the right. This algorithm is summarized pointing out that tapes with two heads can be replaced by several tapes with only one head per tape without time loss. (Fischer, Meyer, Rosenberg, 1972).

The input tape, (worktape 1 in the working example) has two heads, A and B. Worktape 2 has two heads C and D and holds the values of the Δ array and the value of d. Worktape 3 is used as a counter and contains the value of s. Finally, worktape 4 is used as a scratch tape. Note that p is represented by the positions of heads A and C on tapes 1 and 2 respectively.

To illustrate how this Turing Machine works, the previous example is continued. The tapes are configured as follows before starting stage 0. Note that

heads A and B on W1 are at the position head H was at. Heads C and D of W2, S on W3 and H on W4 are all moved right to the first blank encountered. For notational convenience, the non-blank symbols to the left of these heads will not be shown.

tape

W1 $\$ABX_1X_2X_3\&X_2X_3X_1X_2X_3X_1$

W2 bCD

W3 bS

W4 bH

Note that if I and J are tape heads, IJX denotes X being scanned both heads.

Continuing with the example:

W1 $A\$BX_1X_2X_3\&X_2X_3X_1X_2X_3X_1$

W2 C01D0

W3 S0

W4 H

At the end of stage 0, A is positioned one cell to the left of the leftmost symbol of V (that is, A is reading the $), B is positioned under the leftmost symbol of V (just right of the $), C is under the leftmost symbol on tape 2, which was set to 0, D is positioned under the rightmost symbol of tape 2, which also is set to zero, and S is positioned under the lone symbol on tape 3, which is set to 0. Note that the contents of tape 2 at the conclusion of stage 0 are 010, the 1 being the value of Δ (-1), and the rightmost 0 the initial value of d.

Since p is set to -1, this implies moving head A left to read the $ on W1 (the -1st symbol), and writing a 0 onto W2 where head C is. Head C will always be positioned at the 0 preceeding the value of some Δ (i). Head B on W1 did not move. Head D on W2 moved right one position, writing a 1 (the value of Δ (-1)) and moved right again, writing a 0 (the value of d). Head D writes the 1's that make up the value of Δ (i) in unary notation, and then writes the current value of d. Head S on W3 wrote the value of s (0) on the tape. Finally, head H on W4 did not move.

At the start of stage (i+1,K), head A is scanning Z_p, and B is under Z_i. Let 1^0 denote the empty symbol, and let 1^k denote a string of k 1's. Using this notation, the contents of tape 2 at the start of

stage (i+1,K) is $01^{\Delta(-1)}$ $01^{\Delta(0)}$ $01^{\Delta(1)}0$ . . . $01^{\Delta(i-1)}$ 01^{d} . Head C is on the 0 immediately before $1^{\Delta(p)}$ (which is the (p+2)nd 0 from the left). Head D is on the rightmost non-blank square. Finally tape 4 contains the value for s.

Next, all the possible operations required during stage (i+1,K) are analyzed. The test to see whether $Z_{p+1} = Z_{i+1}$ is done in 3 steps, since head A is only one symbol away from Z_{p+1} and head B is only one symbol away from Z_{i+1} . The test for p=-1 becomes a test to see if head A is scanning the \$. The algorithm ends when B is scanning a blank symbol while looking at Z_i

First consider case 1; $Z_{p+1} = Z_{i+1}$. D is shifted right and a 0 is written (Δ (i)=d,d=0). Heads A and B are shifted right (p= p+1, i= i+1), and head C moves right to the next 0 (p= p+1). As C is advanced, a 1 is added to tape (W3) and move head S right ($s=s+\Delta p$).

```
DO WHILE (W1(B)  ≠  b);

   IF W1(A+1) = W1(B+1)

      THEN DO;  D=D+1;

                W2(D)=0;

                A=A+1;

                B=B+1;

                C=C+1;

                DO WHILE (W2(C)  ≠  0);

                   W3(S)=1;

                   S=S+1;

                   C=C+1;

                END;

      END;
```

Next, consider case 2; p=-1. Heads A and C are left alone (p=p). B moves right one cell (i=i+1), and D moves right two cells, printing a 1 followed by a zero ($\triangle$ (i)=d+1, d=0).

```
IF W1(A) = $

   THEN DO;  B=B+1;

             D=D+1;

             W2(D)=1;

             D=D+1;

             W2(D)=0;

          END;
```

Finally, consider case 3. Since the values of p and d are modified by the value of s, and since the contents of s are modified at the same time, the value of s is first copied onto tape 4. Then, head C is moved left over s zeroes (the value of s which is on tape 4). For each 0 passed over by head C, head D moves one cell right and prints a one (d=d+s) and head A moves left one cell (p=p-s). Then, for each 1 passed over by C (delta values), the value of s on tape 3 is decremented by one (print a blank symbol and move head S right).

At the conclusion of this stage, the contents of tape 4 (the copy of the old value of s) will be erased.

```
S=S-1;
DO WHILE (W3(S) ≠ b);
   W4(H)=W3(S);
   H=H+1;
   S=S-1;
END;
S=S+1;
DO WHILE (W4(H)=1);
   C=C-1;
   DO WHILE (W2(C)=1);      /*pass over 1's*/
      W3(S)=b;
      S=S+1;
      C=C-1;
   END;
                            /*pass over 0's*/
   D=D+1;
   W2(D)=1;
   A=A-1;
```

```
      H=H-1;
   END;
   H=H+1;
   DO WHILE (W4(H)=1);
      W4(H)=b;
      H=H+1;
   END;
```

At this point, the example is continued. The contents of each tape at the conclusion of each stage is shown below.

at end of stage

(1,1) W1: $A\$X_1BX_2X_3\&X_2X_3X_1X_2X_3X_1$

(case 2) W2: C0101D0

W3: SO

W4: H

(2,1) W1: $A\$X_1X_2BX_3\&X_2X_3X_1X_2X_3X_1$

(case 2) W2: C010101D0

W3: SO

W4: H

(3,1) W1: $A\$X_1X_2X_3B\&X_2X_3X_1X_2X_3X_1$

(case 2) W2: C01010101D0

W3: SO

W4: H

(4,1) W1: $A\$X_1X_2X_3\&BX_2X_3X_1X_2X_3X_1$

(case 2) W2: C01010101D0

W3: SO

W4: H

(5,1) W1: $A\$X_1X_2X_3\&X_2BX_3X_1X_2X_3X_1$

W2: C0101010101D0

W3: SO

W4: H

(6,1) W1: $\$AX_1X_2X_3\&X_2X_3BX_1X_2X_3X_1$

(case 1) W2: 01C010101010D0

W3: 1S

W4: H

(7,1) W1: $\$X_1AX_2X_3\&X_2X_3X_1BX_2X_3X_1$

(case 1) W2: 0101C01010100D0

W3: 11S

W4: H

(8,1) W1: $\$X_1X_2AX_3\&X_2X_3X_1X_2BX_3X_1$

(case 1) W2: 010101C010101000D0

W3: 111S

W4: H

(9,1) W1: $A\$X_1X_2X_3\&X_2X_3X_1X_2BX_3X_1$

(case 3) W2: C0101010101010000011D1

W3: S

W4: H (it was 111H => 11H => 1H => H)

(9,2) W1: $\$AX_1X_2X_3\&X_2X_3X_1X_2X_3BX_1$

(case 1) W2: 01C0101010101000011 1D0

W3: 1S

W4: H

(10,1) W1: $\$AX_1X_2X_3\&X_2X_3X_1X_2X_3X_1B$

(case 2) W2: 01C0101010101000011101D0

W3: 1S

W4: H

Here, the TM terminates since head B is scanning a blank on W1.

2.10 Complexity of the Algorithm

The time spent in each of these three cases is now totaled. For each value of i, case 2 will be executed at most one time during stage (i+1,K). Each execution takes constant time (2 steps), so that the total over all stages is clearly O(n), where n is the length of V

and also the length of U.

When case 3 is executed at stage (i+1,K), it takes time C.s for some constant C, where s is the value on tape 3 at the start of the stage ($s=(P^K(i) - P^{K+1}(i))$. Note that $s \geq \sum_{j=p-s}^{p-1} \Delta(j)$, so C.s is clearly the bound in this case. Let k_i be the largest value of k for which a stage (i+1,k) is executed. Then stages (i+1,1), ... (i+1, k_{i-1}) all execute case 3 and stage (i+1, k_i) will then execute either case 1 or 2. Therefore, the total time spent in case 3 is

$$\sum_{k=1}^{k_i-1} C(P^k(i) - P^{k+1}(i)) = C(P(i) - P^{k_i}(i)) \leq C(P(i) - P(i+1)+1)$$

Summing over all stages,

$$C(P(0) - P(1) +1)$$

$$+ C(P(1) - P(2) +1)$$

$$+ \quad .$$

$$\quad .$$

$$\quad .$$

$$+ C(P(n-1) - P(n) +1) = C(P(0) - P(n) +n)$$

$$\leq C(n+2)$$

The constant C takes into account the time required to copy the original value on tape 3 onto tape 4, as well as the actual processing time performed during each execution of case 3.

The time spent on Case 1 is bounded by the number of times head C is shifted right. This is at most the eventual length of tape 2 plus the number of times head C is shifted left. But the shifting of head C left only occurs in case 3, and is therefore bounded by $0(n)$. Since the eventual length of tape 2 is $n+2 + \sum_{j=-1}^{n-1} \Delta(j) < 2n+3$ (which is still $0(n)$), the total time spent in case 1 is also $0(n)$.

At this point, the algorithm has executed on the string V&UU, where the delta array is contained on tape 2. The final phase of the algorithm searches for a value of i where $P(i)$ is equal to n ((length of V)). This implies V is contained within U^2, thus implying that U and V are conjugate elements. Recall that $P(i)=i - \sum_{j=-1}^{i-1} \Delta(j)$. This implies $i=n + \sum_{j=-1}^{i-1} \Delta(j)$ if $P(i)=n$. To make use of this, tape 5 is used as a counter to keep a running total of $n+ \sum_{j=-1}^{i-1} \Delta(j)$. Tape 5 has two heads, I and N. The essential idea here is to compare the position of head I (which represents i) to the position of head

N (which represents the value of $n+\sum_{j=-1}^{i-1} \Delta(j)$). If the two heads are at precisely the same position, then P(i)=n.

To implement this, head A of tape 2 is positioned at the $ preceding V&UU, taking at most |V&UU|+1 steps (the length of W1). Note that |V&UU| is still linear. Next, heads I and N are positioned under a 0 on work-tape 5. The value of n (length of V) is put onto work-tape 5. Head A is moved right until the & is reached. For each "non-&" symbol scanned, a 1 is written onto tape 5 and head N is moved <u>right</u>. When the & is scanned, head N is moved left one position and a 0 is written. Now |V| is on tape 5. Next, head C of tape 2 is positioned under the leftmost 0 of the tape. Again, this takes at most linear time since tape 2 contains the delta array.

To illustrate the final portion of the algorithm, the example is continued. Initially, the tapes are configured as follows:

W1: AX_1X_2X_3$&X$_2X_3X_1X_2X_3X_1$

W2: 01C01010101010000111101D0

W5: INO

After performing the steps outlined above to put the value of n on tape 5:

W1: $X_1X_2X_3A\&X_2X_3X_1X_2X_3X_1$

W2: C0101010101010000111101D0

W5: I11NO

Here, we begin to check whether $i=n+\sum_{j=-1}^{i-1}\Delta(j)$. Head C is moved right to the next 0 passing over $\Delta(j)$. For each 1 head C passes over, a 1 is written onto tape 5 and head N is moved right one position. When head C is scanning the 0, a 0 is written onto tape 5 where head N is positioned. N is now at position $n+\sum_{j=-1}^{i-1}\Delta(j)$. Thus, the TM checks to see whether head I is scanning the 0. If it is, the TM halts with successful results; i.e., $i=n+\sum_{j=-1}^{i-1}\Delta(j)$ implying U and V are conjugate elements. Otherwise, head I is moved right one position. This part of the algorithm is repeated until either successful results are obtained, or until all of the symbols on worktape 2 (the Δ array) have been exhausted.

```
DO WHILE (W2(C)  ≠  b);
   C=C+1;
   DO WHILE (W2(C)  ≠  0);
      W5(N)=1;
      N=N+1;
      C=C+1;
   END;
   W5(N)=0;

   IF W5(I)=0   SUCCESS;
      ELSE I=I+1;
END;
FAILURE;
```

Continuing the example, the configuration of tapes 2 and 5 is shown when heads I and N are about to be checked for scanning the same symbol on tape 5.

1. W2: 01C01010101010000111010

   W5: I111NO

2. W2: 0101C010101010000111010

   W5: 1I111NO

3. W2: 010101C0101010000111010

   W5: 11I111NO

4. W2: 01010101C01010000111010

   W5: 111I111NO

5. W2: 0101010101C010000111010

   W5: 1111I111NO

6. W2: 010101010101C0000111010

   W5: 11111I111NO

7. W2: 01010101010C000111010

   W5: 111111I11NO

8. W2: 0101010101010C00111010

   W5: 1111111I1NO

9. W2: 010101010101000C0111010

   W5: 11111111INO

At this point, heads I and N are both scanning the 0 on tape 5, and thus the TM halts with successful

results. Since head I is scanning the 9th symbol on tape 5, this implies that symbols 7, 8, and 9 are identical with symbols 1, 2, and 3 on W1. This, in fact, is true since

$$W1: \quad \$X_1X_2X_3\&X_2X_3X_1X_2X_3X_1$$

position 0 1 2 34 5 6 7 8 9 10

The time bound of this final phase of the algorithm is $|V|+1$ + length of tape 2. This length is the

$$\sum_{j=-1}^{|V\&UU-1|} \Delta(j) + |V\&UU|+1+1$$

(the 0's preceeding each delta)

Since the sum of the Δ's is $\leq |V\&UU|+1$, this bound is linear in the length of V&UU, and hence makes the entire algorithm linear. Note, Rivest (1977) shows that there do not exist pattern matching algorithms whose worst case behavior is sublinear (that is, linear whose constant is less than one). Thus, this result is the best possible.

3. THE TIME COMPLEXITY OF THE POWER PROBLEM FOR FREE GROUPS

Given a presentation of a free group, G, on n free generators X_1, X_2, ... X_n, the Power problem for G is defined as deciding whether a given word V in the generators is derivable from a power of U, another word in the generators; i.e., is $V=U^K$ for some finite value of K. The existence of a linear upper time bound is shown for this problem on a multitape Turing Machine.

3.1 Introduction

The first part of the solution to the Power problem is to freely reduce words U and V to words U' and V'. To illustrate why this is done, consider the following: U=aba and $V=abb^{-1}ba$. When checking to see if V is a power of U, i.e., if $V=U^K$, since |V|=5 and |U|=3 (where |W| is the length of W), the naive algorithm will halt with failure. But this is wrong; V is freely reducible to aba, thus yielding $V=U^K$ where K=1.

The algorithm used to freely reduce U and V is the algorithm used to solve the word problem for free groups.

Before continuing, our algorithm must verify that no trivial relators can be generated by concatenating U with itself. To illustrate, consider $U = aba^{-1}$,

$V = abba^{-1}$. Here $U^2 = aba^{-1}aba^{-1} = abba^{-1} = V$. Therefore, the algorithm checks whether the first and last symbols of U are inverses. If so, we check whether the first symbol of U is equal to the first symbol of V, and whether the last symbol of U is equal to the last symbol of V. Note that simple cyclic reduction is not enough here; consider $U = aba^{-1}$, and $V = a^{-1}bba$. If we cyclically reduce U and V, we incorrectly conclude that V is a power of U. We call the above procedure refined reduction.

Once U and V have been reduced to U' and V', we determine whether $V' = (U')^K$ for some finite, integral value of K. The linear time pattern matching algorithm of Chapter 2 is applied.

The pattern matching algorithm finds all occurrences of one string within another, in our case, all occurrences of the word U' within the word V' in linear time. Assume $|U'| = u$, and $|V'| = v$. Then, if the first occurrence of U' in V' occurs at position u (the first u symbols of V' match all of the symbols of U'), and if the second occurrence of U' in V' occurs at position 2u, etc., and the last occurrence of U' in V' occurs at position Ku, where Ku=v, then a successful solution to the Power problem exists, namely $V = U^K$. We show the application of the pattern matching algorithm to be a linear one, thus solving the Power

problem in linear time on a multitape Turing Machine.

3.2 The Model

The multitape Turing Machine model which solves the Power problem is shown to have a linear upper bound for its time complexity. The model makes use of five worktapes. The input tape and worktapes initially have the following configuration:

Input Tape: U $ V &

Worktapes 1 thru 5: $ blanks

Output Tape: blanks

3.3 Free Reduction of Input

During this phase of the algorithm, U is freely reduced yielding U' (the freely reduced version of U) on worktape 1, and V is freely reduced, yielding V' (the freely reduced version of V) on worktape 2. The free reduction process is the same as the process outlined previously (in Section 2.1), using $ from the input tape as the delimiter symbol in freely reducing U, and & as the delimiter in freely reducing V. This phase requires $|U|+|V|+2$ computational steps leaving the tapes in the following state:

Input Tape: U $ V &

Worktape 1: $ U'

Worktape 2: $ V'

Worktapes 3-5: $ blanks

The worktape heads for tapes 1 and 2 are positioned under the rightmost symbols of U' and V' respectively.

To illustrate the remaining parts of the algorithm, each phase of the algorithm is traced as it executes on the following example: $U=X_1X_2^{-1}X_2^2X_3$, $V=X_1X_2X_3X_1X_2^{-1}X_2^2X_3$. At the conclusion of Phase 1, the worktapes look like:

W1: $\$X_1X_2H1X_3$

W2: $\$X_1X_2X_3X_1X_2H2X_3$

W3-W5: $ blanks

3.4 Refined Reduction

Recall that we first test whether the first and last symbols of U are inverses. If so, we test whether the first symbols of U and V are identical, and whether

the last symbols of U and V are identical. To ease the task of illustration, let head H1 be positioned at the rightmost symbol of U, head J1 be positioned at the leftmost symbol of U, head H2 be positioned at the rightmost symbol of V, and head J2 be positioned at the leftmost symbol of V. The algorithm for refined reduction is defined below:

```
DO WHILE (W1(H1)≠$);

IF W1(H1) = -W1(J1)                              /* INVERSES */

 & W1(H1) =  W2(H2)

 & W1(J1) =  W2(J2)

   THEN DO;     W1(H1) = b;  /* blank */

                W1(J1) = b;

                W2(H2) = b;

                W2(J2) = b;

                H1 = H1-1;

                J1 = J1+1;

                H2 = H2-1;

                J2 = J2+1;

        END;
```

```
END;
```

3.5 Determination Of The Length of U'

Here, the number of symbols of U' are counted. This length, called u, is saved on worktape 3.

To do this, the current symbol on tape 1 is read. If it is not the $, a 1 is written onto tape 3, moving worktape head 1 (H1) left one position, and moving worktape head 3 (H3) right one position. If the current symbol on worktape 1 is the $, worktape head H1 is moved right one position, so that H1 is positioned under the leftmost symbol of U'. A $ is written onto worktape 3, and head H3 is moved left one position. This leaves H3 positioned under the rightmost 1 on tape 3. Phase 2 leaves u, the length of U', expressed in unary notation on tape 3.

```
DO WHILE (W1(H1)≠$);

        W3(H3)=1;

        H1=H1-1;

        H3=H3+1;

END;


H1=H1+1;
```

W3(H3)=$;

H3=H3-1;

Since at each step of this phase, head 1 moves left, and finally moves right one position, exactly $|U'|+1$ computational steps are performed.

To illustrate phase 2, our example is continued. The states of tapes 1 and 3 are shown below at the conclusion of each computation.

step 0 - (How we start phase 2)

W1: $\$X_1X_2H1X_3$

W3: $H3b (b is a blank)

step 1 - W1: $\$X_1H1X_2X_3$

W3: $1H3b

step 2 - W1: $\$H1X_1X_2X_3$

W3: $11H3b

step 3 - W1: $H1\$X_1X_2X_3$

W3: $111H3b

step 4 - W1: $\$H1X_1X_2X_3$

(last

step) W3: \$11H31\$

3.6 Procedure for the Pattern Matching Algorithm

For the Fischer and Paterson Turing Machine to work, the input must be in the form $U'\&V'$. To do this, first recall the configurations of the tapes at the end of Phase 2:

Worktape 1: \$U' Head 1 is positioned under the leftmost symbol of U'.

Worktape 2: \$V' Head 2 is positioned under the rightmost symbol of V'.

Worktape 3: \$u Head 3 is positioned under the rightmost 1 of u.

Since our goal is to make worktape 1 contain $U'\&V'$,

- head H1 is moved past the rightmost symbol of U' on worktape 1,
- an & is written onto worktape 1, and
- V' is copied from worktape 3 onto worktape 1.

This entire operation is done in time $|U'| + |V'|$.

To copy V' onto tape 1, the head on tape 2 (H2) must be positioned under the leftmost symbol of V'. To do this, each symbol of V' is read. If the symbol being read is not a \$, the head is moved left one position. When the \$ is finally scanned, H2 is moved right one position. This operation requires $|V'|+1$ steps.

```
DO WHILE (W2(H2)≠$);
   H2=H2-1;
END;
H2=H2+1;
```

Initially,

W2: $\$X_1X_2X_3X_1X_2H2X_3$

step 1: W2: $\$X_1X_2X_3X_1H2X_2X_3$

step 2: W2: $\$X_1X_2X_3H2X_1X_2X_3$

step 3: W2: $\$X_1X_2H2X_3X_1X_2X_3$

step 4: W2: $\$X_1H2X_2X_3X_1X_2X_3$

step 5: W2: $\$H2X_1X_2X_3X_1X_2X_3$

step 6: W2: $H2\$X_1X_2X_3X_1X_2X_3$

(start repositioning)

step 7: W2: $\$H2X_1X_2X_3X_1X_2X_3$

(last step)

Next, the head on W1 is placed one tape cell <u>past</u> the rightmost symbol of U'. This is so an & can be written onto W1. This is done in real time on the length of U'. Simply stated, head H1 is moved right one symbol at time until a blank is encountered, at which time the & is written onto W1 and H1 is moved right one position. Consider the algorithm and example below:

```
DO WHILE (W1(H1)≠b);
   H1=H1+1;
END;
W1(H1)=&;
H1=H1+1;
```

step 0: W1: $\$H1X_1X_2X_3$

step 1: W1: $\$X_1H1X_2X_3$

step 2: W1: $\$X_1X_2H1X_3$

step 3: W1: $\$X_1X_2X_3H1b$

step 4: W1: $\$X_1X_2X_3\&H1b$

Finally, V' is copied from W2 onto W1. This leaves W1 containing $U'\&V'$, with H1 positioned past the last symbol of V'. H1 must then be repositioned so that it is scanning the leftmost symbol of U'. This operation is done in $|V'|+|U'|+|V'|+1$ steps, as seen below:

Algorithm

```
DO WHILE (W2(H2)≠b);
   W1(H1)=W2(H2);
   H1=H1+1;
   H2=H2+1;
END;

DO WHILE (W1(H1)≠$);
   H1=H1-1;
END;
H1=H1+1;
```

step 0: W1: $\$X_1X_2X_3\&H1b$

W2: $\$H2X_1X_2X_3X_1X_2X_3$

step 1: W1: $\$X_1X_2X_3\&X_1H1b$

W2: $\$X_1H2X_2X_3X_1X_2X_3$

step 2: W1: $\$X_1X_2X_3\&X_1X_2H1b$

W2: $\$X_1X_2H2X_3X_1X_2X_3$

step 3: W1: $X_1X_2X_3\&X_1X_2X_3H1b$

W2: $X_1X_2X_3H2X_1X_2X_3$

step 4: W1: $X_1X_2X_3\&X_1X_2X_3X_1H1b$

W2: $X_1X_2X_3X_1H2X_2X_3$

step 5: W1: $X_1X_2X_3\&X_1X_2X_3X_1X_2H1b$

W2: $X_1X_2X_3X_1X_2H2X_3$

step 6: W1: $X_1X_2X_3\&X_1X_2X_3X_1X_2X_3H1b$

W2: $X_1X_2X_3X_1X_2X_3H2b$

step 7: W1: $X_1X_2X_3\&X_1X_2X_3X_1X_2H1X_3$

step 8: W1: $X_1X_2X_3\&X_1X_2X_3X_1H1X_2X_3$

step 9: W1: $X_1X_2X_3\&X_1X_2X_3H1X_1X_2X_3$

step 10: W1: $X_1X_2X_3\&X_1X_2H1X_3X_1X_2X_3$

step 11: W1: $X_1X_2X_3\&X_1H1X_2X_3X_1X_2X_3$

step 12: W1: $X_1X_2X_3\&H1X_1X_2X_3X_1X_2X_3$

step 13: W1: $X_1X_2X_3H1\&X_1X_2X_3X_1X_2X_3$

step 14: W1: $\$X_1X_2H1X_3\&X_1X_2X_3X_1X_2X_3$

step 15: W1: $\$X_1H1X_2X_3\&X_1X_2X_3X_1X_2X_3$

step 16: W1: $\$H1X_1X_2X_3\&X_1X_2X_3X_1X_2X_3$

step 17: W1: $H1\$X_1X_2X_3\&X_1X_2X_3X_1X_2X_3$

(all done; move right 1 position)

step 18: W1: $\$H1X_1X_2X_3\&X_1X_2X_3X_1X_2X_3$

3.7 Pattern Matching

The Fischer-Paterson pattern matching Turing Machine of Chapter 2 is applied here. To illustrate how this Turing Machine works, the example is continued. The tapes are configured as follows just before starting stage 0. Note, head H1 corresponds to heads A and B, and head H2 corresponds to heads C and D below.

tape

W1 $\$ABX_1X_2X_3\&X_1X_2X_3X_1X_2X_3$

W2 CD

W4 S

W5 H

At the end of stage 0:

W1 $A\$BX_1X_2X_3\&X_2X_3X_1X_2X_3X_1$

W2 C01D0

W4 SO

W5 H

Continuing, we show the contents of each tape in our example at the conclusion of each stage of execution.

at the end of stage

0 W1 - $A\$BX_1X_2X_3\&X_1X_2X_3X_1X_2X_3$

W2 - C01D0

W4 - SO

W5 - H

1 W1 - $A\$X_1BX_2X_3\&X_1X_2X_3X_1X_2X_3$

(case 2) W2 - C0101D0

(applied) W4 - SO

W5 - H

2 W1 - A$$X_1X_2BX_3$&$X_1X_2X_3X_1X_2X_3$

(case 2) W2 - C010101D0

W4 - SO

W5 - H

3 W1 - A$$X_1X_2X_3B$&$X_1X_2X_3X_1X_2X_3$

(case 2) W2 - C01010101D0

W4 - SO

W5 - H

4 W1 - AX_1X_2X_3$&$BX_1X_2X_3X_1X_2X_3$

(case 1) W2 - 01C0101010D0

W4 - 1S

W5 - H

5 W1 - $$X_1AX_2X_3$&$X_1BX_2X_3X_1X_2X_3$

(case 1) W2 - 0101C010100D0

W4 - 11S

W5 - H

6 W1 - $$X_1X_2AX_3$&$X_1X_2BX_3X_1X_2X_3$

(case 1) W2 - 010101C01000D0

W4 - 111S

W5 - H

7,1 W1 - A$$X_1X_2X_3$&$X_1X_2BX_3X_1X_2X_3$

(case 3) W2 - C01010101000011D1

W4 - S

W5 - H

(Note that the contents of W5 have also been erased at the conclusion of this stage.)

7,2 W1 - AX_1X_2X_3$&$X_1X_2X_3BX_1X_2X_3$

(case 1) W2 - 01C01010100001110

W4 - 1S

W5 - H

8 W1 - $$X_1AX_2X_3$&$X_1X_2X_3X_1BX_2X_3$

(case 1) W2 - 0101C01010000111D0

W4 - 11S

W5 - H

9 W1 - $X_1X_2AX_3\&X_1X_2X_3X_1X_2BX_3$

(case 1) W2 - 010101C010000111100D0

W4 - 111S

W5 - H

Since Z_{i+1} is a blank (recall is represented by the position of head B on W1), the Turing Machine halts here.

3.8 Procedure For Finding Powers

Next, at most $|U'|+|V'|+1$ symbols on tape W5 will be scanned. To insure that this is the maximum number of symbols scanned, a $ is written on tape W5 at position $|U'|+|V'|+2$. To do this, the number of symbols on tape W1 (which still contains $U'\&V'$) are counted and the head on W5 is moved accordingly. Recall the configuration tapes W1 and W5 are in at the conclusion of the Fischer-Paterson algorithm:

(0) W1: $\$X_1X_2X_3\&X_1X_2X_3X_1X_2BX_3$

W5: bbbbbbbbbbbHb

First a $ is written on tape W5 and H is moved left one position:

(1) W1: $\$X_1X_2X_3\&X_1X_2X_3X_1X_2BX_3$

W5: bbbbbbbbbbHb$

The $ will correspond to position $|U'|+|V'|+2$ when finished with this phase. Now, if the symbol scanned by B on W1 is not $, heads B and H are moved left one position.

(2) W1: $\$X_1X_2X_3\&X_1X_2X_3X_1BX_2X_3$

W5: bbbbbbbbbHbb$

This "reverse-scan" operation continues until an encounter with $ on W1, at which time we halt.

(3) W1: $X_1X_2X_3\&X_1X_2X_3BX_1X_2X_3$

W5: bbbbbbbbHbbb$

(4) W1: $X_1X_2X_3\&X_1X_2BX_3X_1X_2X_3$

W5: bbbbbbbHbbbb$

(5) W1: $X_1X_2X_3\&X_1BX_2X_3X_1X_2X_3$

W5: bbbbbbHbbbbb$

(6) W1: $X_1X_2X_3\&BX_1X_2X_3X_1X_2X_3$

W5: bbbbbHbbbbbb$

(7) W1: $X_1X_2X_3B\&X_1X_2X_3X_1X_2X_3$

W5: bbbbHbbbbbbb$

(8) W1: $X_1X_2BX_3\&X_1X_2X_3X_1X_2X_3$

W5: bbbHbbbbbbbb$

(9) W1: $X_1BX_2X_3\&X_1X_2X_3X_1X_2X_3$

W5: bbHbbbbbbbbb$

(10) W1: $BX_1X_2X_3\&X_1X_2X_3X_1X_2X_3$

W5: bHbbbbbbbbbb$

(11) W1: $B\$X_1X_2X_3\&X_1X_2X_3X_1X_2X_3$

(stop) W5: Hbbbbbbbbbbb$

The blank being scanned at the end of this phase corresponds to position 0 (the $) with respect to the symbols on tape W1.

At this point, we want to check that P(i)=u for all i in V' where i=Ku, and K is a positive integer. To implement this, consider the following: since $P(i) = i - \sum_{j=0}^{i-1} \Delta(j)$, and we want to check for values of P(i)=u, then if $P(i)=u=i-\sum_{j=0}^{i-1} \Delta(j)$, $i=u+\sum_{j=0}^{i-1} \Delta(j)$! We use this in solving the Power problem.

Tape 5 is used as a counter to keep a running total of $u+\sum_{j=0}^{i-1} \Delta(j)$. Tape 5 will have two tape heads, I and SUM. Here, the position of head I (which represents the positions of i of V') is compared to the position of head SUM (which represents the value of $u+\sum_{j=0}^{i-1} \Delta(j)$). If I corresponds to a position Ku, and if both heads I and SUM are at <u>precisely</u> the same position, we conclude that P(i)=u. Head I is then moved exactly u positions to the right on tape 5. The algorithm continues in this manner until one of the following conditions occurs:

a. $P(i) \neq u$ and $i = Ku$ for some integer value of K. This implies failure.

b. If $P(i) = u$ and $i = Ku$ for some integer value of K, the algorithm continues, unless the delta array is exhausted, in which case halt with success.

c. If the delta array is exhausted, and $P(i) \neq u$, halt with failure.

Head I must be placed at position 2u+1 on tape 5. This corresponds to i=u (the u^{th} symbol of V'). Recall that tape 3 contains u expressed in unary notation. The tape head, called H3, is scanning the rightmost 1 of U. A $ is written on tape 5, and heads I and SUM are initially positioned at this $. If the symbol read by H3 is a 1, head I is moved right one position, and H3 is moved left one position. This continues until H3 is scanning a $. H3 is then moved right one position, and head I right one position. Note that head I has moved right exactly u+1 positions. Thus, head I must be moved u positions more to the right. So we continue scanning the symbols read by H3, now moving H3 to the right, instead of left. Again, if H3 is scanning a 1, heads I and H3 are both moved right one position. Once H3 is scanning a blank, H3 is moved left one position, not moving Head I. Head I is now at position 2u+1.

Presented below is this procedure:

```
DO WHILE (W3(H3)=1);
   I=I+1;
   H3=H3-1;
END;
H3=H3+1;
I=I+1;

DO WHILE (W3(H3)=1);
   I=I+1;
   H3=H3+1;
END;
H3=H3-1;
```

The placement of head I at position 2u+1 is illustrated below.

```
step 0  W3:  $11H31

        W5:  ISUM$bbbbbbbbbb$

     1  W5:  SUM$Ibbbbbbbbbb$

        W3:  $1H311

     2  W5:  SUM$bIbbbbbbbbb$

        W3:  $H3111

     3  W5:  SUM$bbIbbbbbbbb$

        W3:  H3$111

        (start moving H3 right)

     4  W5:  SUM$bbbIbbbbbbb$

        W3:  $H3111

     5  W5:  SUM$bbbbIbbbbbb$

        W3:  $1H311

     6  W5:  SUM$bbbbbIbbbbb$

        W3:  $11H31

     7  W5:  SUM$bbbbbbIbbbb$
```

W3: $111H3b

8 W5: SUM$bbbbbbIbbbb$

W3: $11H31

Next, head SUM is placed at position u on tape 5, because the position of SUM represents $u+\sum_{j=0}^{i-1}\Delta(j)$. To do this, head SUM is moved right one position for each 1 scanned by H3. H3 will continue reading and moving left. When the delimiter is found, H3 is moved right, and thus SUM will be left at position u on tape 5.

Algorithm

```
DO WHILE (W3(H3)=1);
   SUM=SUM+1;
   H3=H3-1;
END;
   H3=H3+1;
```

The placement of head SUM at position u on W5 is illustrated below:

step 0 W5 - SUM$bbbbbbIbbbb$

W3 - $11H31

1 W5 - $SUMbbbbbbIbbbb$

W3 - $1H311

2 W5 - $bSUMbbbbbIbbbb$

W3 - $H3111

3 W5 - $bbSUMbbbbIbbbb$

W3 - H3$111

4 W5 - $bbSUMbbbbIbbbb$

W3 - $H3111

3.9 Finding Powers

Now, SUM is located at position u, and I is located at position 2u+1 (which corresponds to the uth symbol of V'). To check if P(i)=u, SUM is moved to position $u + \sum_{j=0}^{i-1} \Delta(j) = u + \sum_{j=0}^{2u} \Delta(j)$. If I and SUM are then at precisely the same position, I is moved to position 3u+1 (the 2*uth symbol of V') and SUM is moved to position $u + \sum_{j=0}^{i-1} \Delta(j) = u + \sum_{j=0}^{3u} \Delta(j)$. In this case,

this involves moving head SUM right exactly $(\triangle(i-u)+\ldots+\triangle(i-1)) = (\triangle(2u+1)+\ldots+\triangle(3u))$ positions. This procedure continues until one of the following occurs:

1. $P(i)=u$ (heads I and SUM are at the same position on W5) and $i=|U'|+|V'|+1$ (the last symbol of V'). This would imply that V' is a power of U'.

2. $P(i)\neq u$ $(i\neq u+\sum_{j=0}^{i-1}\triangle(j))$ for some position i. This implies that V' is <u>not</u> a power of U'.

The final phase of the algorithm is presented below:

```
/* count up the values of the first u delta values
   (△(0)  . . .  △(u-1)) */
   C=C+1;
   DO WHILE W3(H3)=1;
      DO WHILE W2(C)=1;
         C=C+1;          /* add in a value of delta */
         SUM=SUM+1;
      END;
      IF W2(C)=0         /* finished with a delta value */
         THEN DO;
               C=C+1;
               H3=H3+1;
            END;
   END;
/* add 1 more delta value to the SUM */
   DO WHILE W2(C)=1;
      C=C+1;
```

```
      SUM=SUM+1

   END;

   /* advance C */

   C=C+1;

/* since H3 is now reading a delimeter, position H3 at the
   1 in the direction opposite the direction H was trave-
      ling in  */     H3=H3-1;

/* add to SUM
   △(i-u) +  . . .  △(i-1) */

   REPEAT;

      DO WHILE W3(H3)=1;     /* add to SUM u values
                                of delta */
         DO WHILE W2(C)=1;

            C=C+1;

            SUM=SUM+1

         END;

         IF W2(C)=0

            THEN DO; C=C+1;
```

```
                    H3=H3-1;

                END;

    END;

    H3=H3+1;


/* see if I and SUM are reading the same value;
                        i.e. P(i) + u */

    W5(SUM)=0;

    IF W5(I)=0

       THEN DO;    /* yes - P(i)=u; see if we're done */

            IF W5(I+1)=$

               THEN HALT with SUCCESS;

               ELSE DO; DO WHILE W3(H3)=1;

                        I=I+1; /* If I was at po-

                               sition Ku, advance I to

                               position (k+1)u */

                        H3=H3+1;

                       END;

                        H3=H3-1; /*reverse direction*/

                        W5(SUM)=b; /*reset to a blank*/
```

```
                    END;

               END;

          ELSE    /* P(i)≠u => FAILURE */

             HALT with FAILURE

     END;    /* End of REPEAT */
```

Presented below is the action of this algorithm on our example. The state of the tapes at the conclusion of each step is shown, plus the statement in the algorithm that got us there.

step

```
 0 - (initially)

     W2 - C0101010100000111000

     W3 - $H3111b

     W5 - $bbSUMbbbIbbbb$

 1 - W2 - 0C101010100000111000      /* C=C+1 */

     W3 - $H3111b

     W5 - $bbSUMbbbbIbbbb$
```

```
2 - W2 - 01C0101010000111000     /* DO WHILE W2(C)=1;
    W3 - $H3111b                    C=C+1
    W5 - $bbbSUMbbbIbbbb$           SUM=SUM+1 */

3 - W2 - 010C101010000111000     /* IF W2(C)=0
    W3 - $1H311b                    THEN C=C+1
    W5 - $bbbSUMbbbIbbbb$           H3=H3+1 */

4 - W2 - 0101C01010000111000     /* DO WHILE W2(C)=1
    W3 - $1H311b                    C=C+1;
    W5 - $bbbbSUMbbIbbbb$           SUM=SUM+1 */

5 - W2 - 01010C1010000111000     /* IF W2(C)=0
    W3 - $11H31b                    THEN C=C+1;
    W5 - $bbbbSUMbbIbbbb$           H3=H3+1 */

6 - W2 - 010101C010000111000     /* DO WHILE W2(C)=1
    W3 - $11H31b                    C=C+1
    W5 - $bbbbbSUMbIbbbb$           SUM=SUM+1 */

7 - W2 - 0101010C10000111000     /* IF W2(C)=0
```

```
     W3 - $111H3b                          C=C+1

     W5 - $bbbbbSUMbIbbbb$                 H3=H3+1 */

     /* End of DO WHILE W3 (H3) = 1 */

 8 - W2 - 01010101C0000111000      /* DO WHILE W2(C)=1

     W3 - $111H3b                      C=C+1;

     W5 - $bbbbbbSUMIbbbb$             SUM=SUM+1 */

 9 - W2 - 010101010C000111000      /* advance C

     W3 - $111H3b                      C=C+1 */

     W5 - $bbbbbbSUMIbbbb$

10 - W2 - 010101010C000111000      /* H3=H3-1 */

     W3 - $11H31b

     W5 - $bbbbbbSUMIbbbb$

11 - W2 - 0101010100C00111000      /* IF W2(C)=0

     W3 - $1H311b                      C=C+1

     W5 - $bbbbbbSUMIbbbb$             H3=H3-1 */

12 - W2 - 01010101000C0111000      /* IF W2(C)=0
```

```
     W3 - $H3111b                    C=C+1

     W5 - $bbbbbbSUMIbbbb$           H3=H3-1 */

13 - W2 - 0101010100000C111000     /* IF W2(C)=0

     W3 - H3$111b                    C=C+1

     W5 - $bbbbbbSUMIbbbb$           H3=H3-1 */

14 - W2 - 0101010100000C111000     /* H3=H3+1 */

     W3 - $H3111b

     W5 - $bbbbbbSUMIbbbb$

15 - W2 - 0101010100000C111000     /* W5 (SUM)=0 */

     W3 - $H3111b

     W5 - $bbbbbbSUMI0bbb$

16 - W2 - 0101010100000C111000     /* W5(I)=0

     W3 - $1H311b                    DO WHILE W3(H3)=1

     W5 - $bbbbbbSUM0Ibbb$           I=I+1; H3=H3+1 */

17 - W2 - 0101010100000C111000     /* DO WHILE W3(H3)=1

     W3 - $11H31b                    I=I+1
```

```
     W5 - $bbbbbbSUM0bIbb$          H3=H3+1 */

18 - W2 - 0101010100000C111000    /* DO WHILE W3(H3)=1
     W3 - $111H3b                    I=I+1
     W5 - $bbbbbbSUM0bbIb$          H3=H3+1 */

19 - W2 - 0101010100000C111000    /* H3=H3-1
     W3 - $11H31b                    W5 (SUM)=b */
     W5 - $bbbbbbSUMbbbIb$

     /* REPEAT FOREVER AGAIN */

20 - W2 - 0101010100001C11000     /* DO WHILE W2(C)=1
     W3 - $11H31b                    C=C+1
     W5 - $bbbbbbbSUMbbIb$          SUM=SUM+1 */

21 - W2 - 0101010100011C1000      /* DO WHILE W2(C)=1
     W3 - $11H31b                    C=C+1
     W5 - $bbbbbbbbSUMbIb$          SUM=SUM+1 */

22 - W2 - 0101010100111C000       /* DO WHILE W2(C)=1
     W3 - $11H31b                    C=C+1
```

```
     W5 - $bbbbbbbbbSUMIb$ -          SUM=SUM+1 */

23 - W2 - 010101010000111OC00      /* IF W2(C)=0
     W3 - $1H311b                       C=C+1
     W5 - $bbbbbbbbbSUMIb$              H3=H3-1 */

24 - W2 - 01010101000011100C0      /* IF W2(C)=0
     W3 - $H3111b                       C=C+1
     W5 - $bbbbbbbbbSUMIb$              H3=H3-1 */

25 - W2 - 0101010100001110000C     /* IF W2(C)=0
     W3 - H3$111b                       C=C+1
     W5 - $bbbbbbbbbSUMIb$              H3=H3-1 */

     /* End of DO WHILE W3 (H3)=1 */

26 - W2 - 0101010100001110000C     /* H3=H3+1
     W3 - $H3111b                       W5 (SUM)=0 */
     W5 - $bbbbbbbbbSUMI0$
     -

27 - HALT with success
     since W5(I+1)=$
```

and W5(I)=0

The algorithm is linear in the length of $|U'|+|V'|$ since at each step of the algorithm either head C is advanced on tape 2, or head I is moved u positions to the right on tape 5. Since tape 2 is of length $O(|U'|+|V'|)$, and tape 5 is of length exactly $|U'|+|V'|+2$, the algorithm is linear.

3.10 A Simpler Algorithm

Since its initial writing, a simpler algorithm has been found that tests whether $V = U^k$. The algorithm does not make use of the Fischer-Paterson Turing Machine. Instead, it merely compares a symbol of U with a symbol from V repeatedly, until either the delimeters on the tapes for U and V are found simultaneously (success) or until a mismatch occurs (failure). This is done by re-using the symbols of U for comparisons with the symbols of V in real-time in the length of V.

Consider the refined reductions of both U and V on tapes W1 and W2 respectively. Let H1 be positioned at the rightmost symbol of U, J1 be positioned at the leftmost symbol of U, H2 be positioned at the rightmost symbol of V, and J2 be positioned at the leftmost symbol of V. The algorithm is presented below:

```
/* Initially, let H = J1, and let J = H1, where H and J are
   variables retaining the names of the worktape heads.*/

     DO WHILE (W2(J2) ≠ b);
          IF W1(H) = W2(J2)
             THEN DO;     H = H+1;
                          J = J-1;
                          J2=J2+1;
                  END;
             ELSE IF W1(H) = b       /* Reuse U */
                     THEN DO;     TMP = H;
                                  H = J;
                                  J = TMP;
                                  H = H+1;
                                  J = J-1;
                          END;
                     ELSE FAILURE;
     END;
```

SUCCESS

4. THE TIME COMPLEXITY OF THE POWER-CONJUGACY PROBLEM FOR FREE GROUPS

Given the free group, G, with the n generators $X_1, X_2, \ldots X_n$, the Power-Conjugacy problem for G is defined as deciding whether a given word V in the generators is equivalent to a conjugate of a power of U, another word in the generators; i.e., is $V=W^{-1}U^kW$ for some finite value of k. A linear upper time bound is shown to exist for the solution of this problem on a multitape Turing Machine.

4.1 Introduction

To determine if V is a conjugate of a power of U, we first apply a theorem from Magnus, Karrass, and Solitar (1966): given two words R,S in the n generators of a free group freely presented on $X_1, X_2, \ldots X_n$, R is a conjugate of S ($S=W^{-1}RW$) iff the cyclic reduction of R is a cyclic permutation of the cyclic reduction of S.

Thus, both U and V must first be freely reduced to unique equivalent corresponding words U' and V'. The algorithm used to reduce U and V comes from the solution of the Conjugacy problem for free groups (see Chapter 2).

Once this is completed, the next portion of the algorithm is started. For ease of notation, we shall

refer to U' as U and V' as V from this point on. Next, we check to see if $|V|=|U^k|$ for some integral value of k. If no such k exists, the algorithm halts with failure. But, if V is a cyclic permutation of U^k for some value of k, then $|V|=|U^k|$!

Next, the algorithm uses a lemma from Chapter 2. If U,V are words over the generators $X_1,X_2,...X_n$, and $|V|=|U|=n$, then $U^2=PVQ$ iff $U=PQ$ and $V=QP$; i.e., U is a cyclic permutation of V. Using this lemma for the Power-Conjugacy problem, we will be checking to see if V is a substring of U^kU^k.

At this point, we have reduced the Power-Conjugacy problem to a straight forward problem in pattern recognition. We want to use a pattern matching algorithm which finds occurrences of one string within another; in our case, the occurrence of V within U^kU^k. We make use of the linear pattern matching algorithm originated by Knuth, Morris and Pratt (1977) and implemented on a multitape Turing Machine by Fischer and Paterson (1974).

Summarizing, we demonstrate that each of the above parts of the algorithm can be implemented in linear time on a multitape Turing Machine, which therefore shows the solution to the Power-Conjugacy problem has a linear upper time bound.

4.2 Cyclic Reduction of Input

The first phase of the solution to the problem is to cyclically reduce the input in linear time. The algorithm used to do this is part of the solution to the Conjugacy Problem for Free Groups (Chapter 1). Rather than repeat the algorithm here, we go on to the next part of the algorithm. Note that the TM contains 6 worktapes, W1-W6, and that W5 and W6 contain the cyclic reductions of V and U respectively.

4.3 Is $|V|=|U^k|$ for some finite value of k

At this point, we must check to see if $|U^k|=|V|$ for some finite value of k. If not, the TM halts the computation concluding that U is not a conjugate of V. If so, the algorithm goes on to its pattern matching phase.

The key idea behind this length checking phase is to insure that for each symbol in U^k there exists a non-trivial symbol in V. Symbols from U are compared with symbols from V until one of the following conditions occur:

1. Both U and V have been exhaustively scanned. (The $ delimeters have been found on both tapes W5 and W6.) This implies $|V|=|U^k|$.

2. V has been exhaustively scanned, but U has not. This implies $|V| \neq |U^k|$, thus yielding failure.

Note that in the event of success, tape W1 will contain \$V& and tapes W3 and W4 will both contain a copy of U^k. Tape heads H3 and H4 will be positioned at the rightmost symbols on W3, W4 respectively, while head H1 will be positioned just past the & on tape W1:

Initially

W1: $V'H1\$$ (V' = the free reduction of V)

W3: $H3bV'\$$

W4: $H4bU'\$$

W5: $bV''H5\$$ (V'' is cyclically reduced)

W6: $bU''H6\$$ (U'' is cyclically reduced)

Note the non-trivial contents of tapes W1, W3, and W4 will be ignored by the following algorithm. This is because the TM no longer has any need for this information.

```
/* Length Checking algorithm */

/* Step 1 - Position H5 and H6 at the leftmost symbols of
   V'' and U'' respectively */

   DO WHILE (W5(H5) ≠ b);
      /* pass left over the non-blank symbols of V'' */
      H5=H5-1;
   END;
   H5=H5+1;  /* position H5 at leftmost symbol of V'' */

   DO WHILE (W6(H6) ≠ b);
      /* pass left over the non-blank symbols of U'' */
      H6=H6-1;
   END;
   H6=H6+1;  /* position H6 at the leftmost symbol of U'' */

/* Step 2 - Position H1 so that is is ready to write V''
   onto tape W1 */

   H1=H1+1;  /* H1 is now positioned at the next avail-
```

```
                    able tape cell of W1        */

/* Step 3 - Compare the non-trivial symbols of U'' with
   non-trivial symbols of V'' */

   REPEAT;
      /* "Count off" n (where n=|U''|) non-trivial
         symbols in V'' */

      DO WHILE (W6(H6) ≠ $);
         IF W5(H5) ≠ $
            THEN DO;      /* non-trivial symbols found
                             in both U'' and V'' -      */

                          W1(H1)=W5(H5);
                          /* copy the symbol from V''(W5)
                             onto tape W1 */

                          H1=H1+1;
                          /* position H1 at next available
                             tape cell */

                          W3(H3)=W6(H6);
                          W4(H4)=W6(H6);
                          /* copy the symbol from U''  (W6)
                             onto both tapes W3 and W4 */

                          H3=H3+1;
                          H4=H4+1;
                          /* position H3 and H4 at next
```

```
                    available tape cells */

                H5=H5+1;
                H6=H6+1;
                /* position H5 and H6 at next
                   symbols of V'' and U'' */
            END;

        ELSE HALT with Failure;
             /* |V''| ≠ |(U'')^k| for a finite
                integral value of k */

END; /* end of DO WHILE (W6(H6) ≠ $) */

/* A scan of U'' has been completed -
   Check for a successful completion of the algorithm */

   IF W5(H5) = $
      THEN EXIT;  /* SUCCESS! */
      /* Both U'' and V'' have been
         completely exhausted */

      ELSE DO;
      /* H6 must be repositioned at the leftmost symbol of
         U'' so the TM may continue checking V'' */

      DO WHILE (W6(H6) ≠ b);
         H6=H6-1;
      END;
      H6=H6+1;
```

```
      /* H6 is now positioned at the leftmost symbol of U'' */

      END;    /* end of ELSE DO */

END;    /* end of REPEAT */

EXIT;   /* This point is reached upon successful
           end of length checking algorithm */

/* The TM writes the & delimeter symbol onto tape W1,
   and a $ onto tapes W3 and W4 */

   W1(H1)=&;
   H1=H1+1;    /* position H1 at next available tape
                  cell on W1 */

   W3(H3)=$    /* write delimeters onto both W3 and W4 */
   W4(H4)=$
```

Perhaps the easiest way to observe the execution of this algorithm is by presenting two examples: one that illustrates success, and the other failure.

Example 1 - (Success)

Initially:

W1: XH1$b (X is extraneous and will no longer be
W3: H3bX illustrated)
W4: H4bX
W5: $bX_1X_2X_1X_2H5\$$ ($V'' = X_1X_2X_1X_2$)
W6: $bX_2X_1H6\$$ ($U'' = X_2X_1$)

```
/* step 1 - position H5 and H6 at the leftmost symbols of
   V'' and U'' respectively */

   W5:  bX1X2X1H5X2$      /* DO WHILE (W5(H5) ≠ b);
        bX1X2H5X1X2$         H5=H5-1;
        bX1H5X2X1X2$         END;
        bH5X1X2X1X2$         H5=H5+1; */
        H5bX1X2X1X2$
        bH5X1X2X1X2$

   W6:  bX2HbX1$       /* DO WHILE (W6(H6) ≠ b);
        bHBX2X1$          H6=H6-1;
        HbbX2X1$          END;
        bHbX2X1$          H6=H6+1; */

   W1:  H1$b      /* No change */
   W3:  H3b       /* No change */
   W4:  H4b       /* No change */

/* Step 2 - reposition head H1 */

   W1:  $H1b      /* H1=H1+1 */

   W3:  H3b               /* No change */
   W4:  H3b               /* No change */
   W5:  bH5X1X2X1X2$      /* No change */
   W6:  bH6X2X1$          /* No change */

/* Step 3 - Compare the non-trivial symbols of U'' with
   non-trivial symbols of V'' (REPEAT) */
```

```
/* Count off 2 (2=|U''|) non-trivials in V'' */
   W1:  $X1X2H1          /* DO WHILE (W6(H6) ≠ $);
   W3:  X2X1H3              IF W5(H5) ≠ $
   W4:  X2X1H4              THEN DO; W1(H1)=W5(H5);
   W5:  bX1X2H5X1X2$                 H1=H1+1;
   W6:  bX2X1H6$                     W3(H3)=W6(H6);
                                     W4(H4)=W6(H6);
                                     H3=H3+1;
                                     H4=H4+1;
                                     H5=H5+1;
                                     H6=H6+1;
                                END;
                           END; */

/* A scan of U'' (W6) has been completed - after
   unsuccessfully checking for completion of the
   algorithm, reposition H6 at the leftmost symbol of
   U'' (W6) */

   W6:  bX2HbX1$      /* IF W5(H5)=$ THEN EXIT; (NO)
        bHbX2X1$         ELSE DO; DO WHILE (W6(H6) ≠ b);
        HbbX2X1$                     H6=H6-1;
        bHbX2X1$                  END;
                                  H6=H6+1;
                              END;          */
   W1:  $X1X2H1           /* No change */
   W3:  X2X1H3            /* No change */
   W4:  X2X1H4            /* No change */
```

```
   W5:  bX1X2H5X1X2$      /* No change */

/* The TM now repeats execution of the REPEAT loop */

/* Count off 2 more symbols of V'' */

   W1:  $X1X2X1X2H1       /* DO WHILE (W6(H6) ≠ $);
   W3:  X2X1X2X1H3           IF W5(H5) ≠ $
   W4:  X2X1X2X1H4           THEN DO; W1(H1)=W5(H5);
   W5:  bX1X2X1X2H5$                  H1=H1+1;
   W6:  bX2X1Hb$                      W3(H3)=W6(H6);
                                      W4(H4)=W6(H6);
                                      H3=H3+1;
                                      H4=H4+1;
                                      H5=H5+1;
                                      H6=H6+1;
                                 END;
                              END;                          */

/* Again, a check for completion, and EXIT with success */

/* IF W5(H5) = $ THEN EXIT; */
   W5:  bX1X2X1X2H5$

/* Write delimeter symbols onto tape W1, W3, W4
   leaving the Turing Machine in the following state: */

   W1:  $X1X2X1X2&H1      /* W1(H1)=&
   W3:  X2X1X2X1H3$          H1=H1+1;
   W4:  X2X1X2X1H4$          W3(H3)=$;
```

W5: $bX_1X_2X_1X_2H5\$$ W4(H4)=$; */

W6: $bX_2X_1H6\$$

Example 2 - (Failure)

Initially

W1: H1$b

W3: H3b

W4: H4b

W5: $bX_1X_2X_1H5\$$ ($V'' = X_1X_2X_1$)

W6: $bX_2X_1H6\$$ ($U'' = X_2X_1$)

/* Step 1 - position H5 and H6 at the leftmost symbols of V'' and U'' respectively */

W5: $bX_1X_2H5X_1\$$ /* DO WHILE (W5(H5) ≠ b);

$bX_1H5X_2X_1\$$ H5=H5-1;

$bH5X_1X_2X_1\$$ END;

$H5bX_1X_2X_1\$$ H5=H5+1; */

$bH5X_1X_2X_1\$$

W6: $bX_2H6X_1\$$ /* DO WHILE (W6(H6) ≠ b);

$bH6X_2X_1\$$ H6=H6-1;

$H6bX_2X_1\$$ END;

$bH6X_2X_1\$$ H6=H6+1 */

W1: H1$b /* No change */

W3: H3b /* No change */

W4: H4b /* No change */

```
/* Step 2 - Reposition head H1 */

   W1:  $H1b      /* H1=H1+1 */

   W3:  H3b             /* No change */
   W4:  H4b             /* No change */
   W5:  bH5X1X2X1$      /* No change */
   W6:  bH6X2X1$        /* No change */

/* Step 3 - Compare the non-trivial symbols of U''
   with non-trivial symbols of V'' (REPEAT) */

/* Count off 2 non-trivials in V'' */

   W1:  $X1X2H1         /* DO WHILE (W6(H6) ≠ $);
   W3:  X2X1H3             IF W5(H5) ≠ $
   W4:  X2X1H4             THEN DO; W1(H1)=W5(H5);
   W5:  bX1X2H5X1$                  H1=H1+1;
   W6:  bX2X1H6$                    W3(H3)=W6(H6);
                                    W4(H4)=W6(H6);
                                    H3=H3+1;
                                    H4=H4+1;
                                    H5=H5+1;
                                    H6=H6+1;
                                END;
                           END;                 */

/* A pass over U'' (W6) has been completed - after
   unsuccessfully checking for completion of the algorithm,
   reposition H6 at the leftmost symbol of U'' (W6) */
```

```
W6:  bX2H6X1$      /* IF W5(H5)=$ THEN EXIT; (NO)
     bH6X2X1$        ELSE DO; DO WHILE (W6(H6) ≠ b);
     H6bX2X1$                    H6=H6-1;
     bH6X2X1$                  END;
                               H6=H6+1;
                             END;                    */

W1:  $X1X2H1       /* No change */
W3:  X2X1H3        /* No change */
W4:  X2X1H4        /* No change */
W5:  bX1X2H5X1$    /* No change */

/* The TM now repeats execution of the REPEAT loop */
/* Try to count off 2 more symbols of V'' -
   Each step in this execution is shown */

W1:  $X1X2X1H1     /* DO WHILE (W6(H6) ≠ $);
W3:  X2X1X2H3         IF W5(H5) ≠ $);
W4:  X2X1X2H4         THEN DO; W1(H1)=W5(H5)
W5:  bX1X2X1H5$                H1=H1+1
W6:  bX2H6X1$                  W3(H3)=W6(H6);
                               W4(H4)=W6(H6);
                               H3=H3+1;
                               H4=H4+1;
                               H5=H5+1;
                               H6=H6+1;
                             END;          */

/* When attempting to execute the DO WHILE (W6(H6) ≠ $)
```

again, W5(H5)=$, and so the TM Halts with failure! */

```
/* DO WHILE (W6(H6) ≠ $);
      IF W5(H5) ≠ $ THEN DO;  .
                              .
                              .
                           END;
                     ELSE HALT with failure; */
```

The complexity of this length checking algorithm is now analyzed. Step 1, which repositions heads H5 and H6, is performed in real time in the sum of the lengths of U'' and V''. Step 2, positioning head H1 just past the $ requires one operation. Finally, step 3. in the case of success, (which has a higher complexity than that of failure), (2*k) complete passes over U'' are made. Hence the complexity of step 3 is $(2*k*|U''|)$, which is linear in the length of U''. Thus, the complexity of the entire length checking algorithm is $O(|U''|) + O(|V''|)$.

4.4 The Cyclic Permutation Problem for Free Groups

For notational convenience, U'' and V'' are referred to as U and V respectively.

At this stage of the algorithm, V$ on tape W1 and U^k on tapes W3 and W4. What must be shown now to solve the Power-Conjugacy problem is that V is a cyclic per-

mutation of U^k.

Using the lemma from Chapter 2, the cyclic permutation problem is solved as follows: given V and U^k, the TM searches for the existence of V in the string U^kU^k. If it exists, U and V are cyclic permutations of each other, otherwise they are not.

4.5 $V\&U^kU^k$ on a Turing Machine Tape

Recall from section 3 that the tapes are left in the following state:

W1: $V&H1b

W3: $U^kH3\$$

W4: $U^kH4\$$

Tape W1 will contain the string $\$V\&U^kU^k$ so that it can be used as input to the Fischer-Paterson Turing Machine. This will require us to reposition heads H3 and H4 at the leftmost non-blank symbols of W3 and W4 (U^k), and then to copy the contents of W3 onto W1, followed by copying the contents of W4 onto W1. The linear algorithm which will do this is presented below.

```
/* Position H3 at the leftmost non-blank symbol of U^k */
   DO WHILE (W3(H3) ≠ b);
      H3=H3-1;
```

```
   END;
   H3=H3+1;

/* Position head H4 at the leftmost non-blank symbol of U^k */
   DO WHILE (W4(H4) ≠ b);
      H4=H4-1;
   END;
   H4=H4+1;

/* Copy U^k from W3 onto W1 */
   DO WHILE (W3(H3) ≠ $);
      W1(H1)=W3(H3);
      H1=H1+1;
      H3=H3+1;
   END;

/* Copy U^k from W4 onto W1 */
   DO WHILE (W4(H4) ≠ $);
      W1(H1)=W4(H4);
      H1=H1+1;
      H4=H4+1;
   END;

/* Finally, position H1 at the leftmost symbol of V on W1 */
   DO WHILE (W1(H1) ≠ $);
      H1=H1-1;
   END;
   H1=H1+1;
```

The complexity of this algorithm is now analyzed. Repositioning head H3 at the leftmost symbol of U^k on tape W3 requires $|U^k|+1$ steps. Repositioning head H4 also requires $|U^k|+1$ steps. Copying U^k from W3 onto W1 requires $|U^k|$ steps, as does copying U^k from W4 onto W1. Finally, positioning H1 at the leftmost symbol of V on W1 requires $|V|+2*|U^k|+1$ steps. Thus, this algorithm is linear in the sum of the lengths of V and U^k.

4.6 Conclusion

At this point, the string $V\&U^kU^k$ has been assembled on tape W1. The Fischer-Paterson machine is then run on this string in linear time to determine whether or not V is a substring of U^kU^k, thus determining if V is a cyclic permutation of U^k. Hence, the Power-Conjugacy problem for free groups is solvable in time linear in the sum of the lengths of V and U^k.

5. GROUPS SOLVABLE BY DEHN'S ALGORITHM.

The DA-groups are shown to have a word problem which is solvable in logspace on a deterministic multitape Turing Machine.

5.1 Finite State Control

The algorithm to solve the word problem for DA-presentations states that a subword V of the given word W must be found, where V is identical with a subword of some relator R_w such that $|V|>\frac{1}{2}|R_w|$. V is replaced by the inverse of the remainder of R_w, thus obtaining the shorter word equivalent to W. This procedure is repeated until W cannot be further reduced by this process or by free reductions. Consider the following problem involving the relators in which the symmetric set of relators $R_u=\{ab,a^2, \ldots\}$. Here, we have two relators which begin with the same generator ("a"). Let the input word W be a^2b; when scanning the first letter of W, (the "a"), it is not known which of the two relators (ab or a^2) will be used in reducing the length of W. Only when the next symbol is scanned (the "a") will we be able to reduce a^2b to b.

If the "next state" function in the Turing Machine is to direct us to a state which corresponds to the detection of a subword of a relator R_w, this function must be defined in such a way as to eliminate all

ambiguity deriving from the aforementioned problem. This can be done by defining a tree structure for the set R of relators $\{R_u\}$.

First, a finite list of the subwords of the R_u and their corresponding replacements must be assembled, where the lengths of the replacements are less than one half of the lengths of the subwords they are replacing. For each relator $R_w \in \{R_u\}$, one entry in the "subword replacement list" is formed. Assuming that the relators are in the form R_w ($R_w = r_1 r_2 \cdots r_m, m>2$), the subword replacement list entry will be

$$r_1 r_2 \cdots r_{\left\lceil \frac{m+1}{2} \right\rceil} = r_m^{-1} r_{m-1}^{-1} \cdots r^{-1}_{\left\lceil \frac{m+1}{2} \right\rceil + 1}$$

Note $\lceil X \rceil$ is defined as the first integer which is greater than or equal to X. If m=2, the subword replacement list entry is $r_1 r_2 = 1$.

As an example consider the relator $b^3 a^{-2}$. The entry in the list for this entry is $b^3 = a^2$. For the relator $a^2 b^{-2}$, we form the entry $a^2 b^{-1} = b$.

This list of subwords and their replacements is formed by using all of the relators in the symmetric set.

The "subword tree" T_L for this list L has for its points all initial segments of subwords in L including

the subword 1 of length 0. We denote a point by the initial segment (sub-subword) it represents. Two points P and Q are connected by a directed line from P to Q iff $|P| + 1 = |Q|$ and P is an initial segment of Q. Note that each subword in our list L is the endpoint of a unique path in T_L starting at 1.

The notion of a "subword tree" is illustrated with the following example. Let $G=\langle a,b;\ b^2a^{-1}\rangle$. The symmetric set R for G is:

b^2a^{-1}	(original relator, (R_1))
ab^{-2}	(inverse of R_1)
$a^{-1}b^2$	cyclic permutations
$ba^{-1}b$	of R_1
$b^{-1}ab^{-1}$	cyclic permutations
$b^{-2}a$	of $\overline{R_1}$

The finite list of subwords and their replacements for this symmetric set of relators is

$b^2 = a$ $\quad ba^{-1} = b^{-1}$

$ab^{-1} = b$ $\quad b^{-1}a = b$

$a^{-1}b = b^{-1}$ $\quad b^{-2} = a^{-1}$

Next, the subwords are partitioned into classes.

Each class is denoted by the generator(s) the members of the class begin with.

$P_a = (ab^{-1})$

$P_b = (b^2, ba^{-1})$

$P_{a^{-1}} = (a^{-1}b)$

$P_{b^{-1}} = (b^{-1}a, b^{-2})$

$P_{ab^{-1}} = (ab^{-1})$

$P_{b^2} = (b^2)$

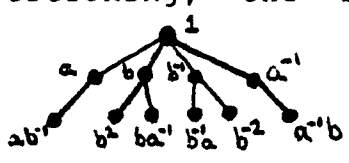
$P_{ba^{-1}} = (ba^{-1})$

$P_{a^{-1}b} = (a^{-1}b)$

$P_{b^{-1}a} = (b^{-1}a)$

$P_{b^{-2}} = (b^{-2})$

Finally, from this partitioning, the following subword tree is formed:



Using this subword tree, we will show how to define the next state function for our Turing Machine.

5.2 The State Symbol Alphabet

Consider the worst-case example of the word problem for DA-groups; no reduction of the input W occurs. Thus, W is encoded onto the worktape using only n Turing Machine cells. As an illustrative example, consider only a two symbol input alphabet. Thus, the number of words of length 2^n that can possibly exist is $2^{(2^n)}$. Thus, for some k, called the base, $k^n = 2^{(2^n)}$. The base k repesents an encoding and solving for k, we get:

$$n \log k = 2^n,$$

$$\log k = \frac{2^n}{n},$$

$$k = 2^{\frac{2^n}{n}}.$$

For example, let $|W| = 2^n = 16$, thus $n = 4$. The base to be used for encoding W is 16. This implies that binary strings of length less than or equal to 16 can be coded in base 16 using only 4 (log 16) symbols.

Generalizing this for an input alphabet of length m which includes the generating symbols and their

inverses,

$$k = 2^{\frac{2^n \log_2 m}{n}}$$

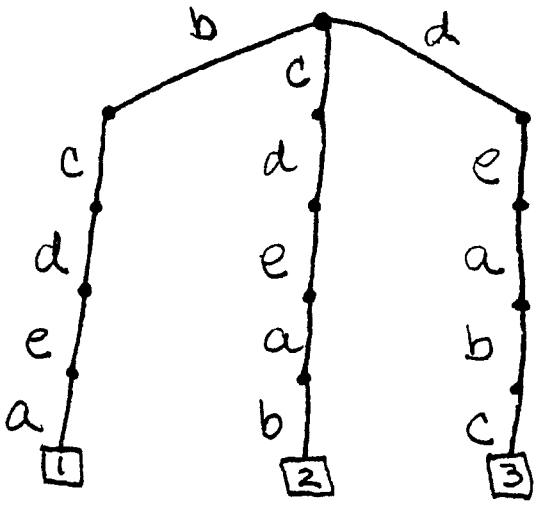
The algorithm for the solution to the word problem is simply described below:

1. Count $|W| = 2^n$, and compute n.
2. Compute k.
3. Compute the lexigraphic value of the input, base k.
4. When LHS's are found:
   - Delete the LHS already encoded.
   - Go to beginning of the encoded worktape.
   - Recompute the lexigraphic value of the remaining encoding, and the new RHS.
   - If another LHS is found, repeat this procedure recursively until no LHS's are found.
5. Continue by reading new input symbols.

5.3 Two Data Structures

Note that when subwords that appear on the subword replacement list have been replaced, it is possible for a new subword (also on the replacement list) to appear that must be replaced. In these cases, the finite control must contain within it a function which defines how two consecutive state symbols are to be "combined". Thus, we form a "combination list" - a list of pairs of state symbols with corresponding replacement state symbols. The length of the combination list is k^2.

At certain times, the Turing Machine examines the rightmost two state symbols on a worktape to find the substring of maximum length that lies on a path of the subword tree. The rightmost two worktape symbols will (possibly) be recoded so that the rightmost symbol on the worktape is the substring of maximal length corresponding to a path on the subword tree. We define for each of the k^2 pairs of encoded strings a pair of replacement strings. This set of definitions is called the Z-list. As an example, consider the following: the rightmost two symbols on the worktape are (ab) (cde) (where (X) is a one symbol encoding of X), and a subtree of the subword tree is



Note that the TM can enter either of two possible states: path 1 (bcde) or path 2 (cde), since the rightmost symbols of the worktape fall on both paths. In this case, the Z-list will map the encoding (ab) (cde) to (a) (bcde) because (bcde) is the substring of maximal length that corresponds to a path on the subword tree.

5.4 The Turing Machine Algorithm

Initially, symbols from the input tape will be "combined" onto the worktape, and new states will be determined depending upon the structure of the subword tree. New input symbols that follow some path on the subword tree will automatically be combined using the combination list. If a new input symbol is encountered which does not follow a path on the subword tree, the

Turing Machine must determine a new path to follow. The determination will be done using the Z-list.

If, however, a subword on the subword replacement list is detected, the Turing Machine must, first replace the subword with its shorter replacement. After replacement, the Turing Machine must check to see if any other subword replacements can be made because of the introduction of the new symbol. This will be done by backing up the heads on the worktape and combining consecutive symbols (if possible). If another subword to be replaced by a shorter subword is found, the replacement is made and the Turing Machine backs up once again. Once no more subword replacements can be made, the Turing Machine will use the Z-list to determine the current path on the subword tree it will follow to continue reducing the input.

When no more input is encountered, the worktape is examined to see if any symbols are contained on it. If the tape is empty, the word problem has been solved for this input word.

5.5 Complexity

Given an input word of length 2^n, the number of symbols encoded on the worktape is less than or equal to the n. This is the logspace bound we have been searching for. Note, too that the running time of this

algorithm is polynomial in the length of the input, since the worktape heads back up to the beginning of the worktape whenever subword replacements are made.

6. THE WORD PROBLEM FOR THE GROUPS OF DEHN'S ALGORITHM

Given a presentation of a group $G = \langle \Sigma, R \rangle$ where Σ is a set of n generators and R is a symmetric set of relators, the word problem for the DA-groups is defined as deciding whether or not a given word in the generators in Σ is equivalent to the identity. A linear upper time bound is presented for the solution to this problem on a multitape Turing Machine model which can write symbols of length zero.

6.1 Introduction

The word problem for the DA groups is solvable in a manner similar to the solution of the word problem for free groups (chapter 1). That is, a monotonic reduction process is used to successively reduce the length of the word to zero. Given a word W in the generators, the algorithm searches for the occurrence of a subword of one of the relators. The size of this subword is greater than half of the relator. The algorithm replaces this subword with the inverse of the remaining portion of the relator. Specifically, given a relator $r = a_1 a_2 \ldots a_k$ ($k \geq 2$ and k odd), the algorithm searches for an occurrence of $a_1 a_2 \ldots a_{\frac{k+1}{2}}$ and replaces this with $a_k^{-1} a_{k-1}^{-1} \ldots a_{\frac{k+1}{2}+1}^{-1}$. When k is even, the search is for $a_1 a_2 \ldots a_{\frac{k}{2}+1}$ and replace

this with $a_k^{-1} a_{k-1}^{-1} \cdot \cdot \cdot a_{\frac{k}{2}+2}^{-1}$.

Thus, if the relator was of odd length, the length of the text would be reduced by one, and if even, reduced by two. For clarity, the portion of the relator that acts as a pattern is denoted as a "left hand side" (LHS), and its replacement portion is denoted as the "right hand side". Also, let the length of LHS be denoted j, and let the length of RHS be at most j-1.

The algorithm must continue seeking occurrences of LHS's of relators. Once a LHS has been replaced by a RHS of length at most j-1 and a null symbol of length zero, the pointer to the text must then be backed up at least j-1 positions from the beginning of the RHS. Consider the following example:

Let LHS = abc j=3

RHS = de

text = xyzabc

replace

LHS with RHS xyzde

backup

j-1 symbols xyzde

Note the pointer was backed up j-1 positions because the algorithm must continue searching for LHS patterns of length j. Let the number of relators (including trivial relators) be m, and let the number of LHS's be m, and let the length of the longest of these LHS's be p. The algorithm must back up p-1 positions each time a replacement is made. This is done to insure that all possible LHS's can be considered for future LHS searches.

The algorithm continues in this fashion, searching for LHS's, replacing them with a corresponding RHS, and backing up p-1 positions each time. Once no more replacements can be made, the algorithm terminates.

Essentially, if there are m relators, m dynamically changing pattern recognition problems are being solved in parallel. Each instance of a pattern recognition problems is solvable in time 0 {1 (length of a LHS) + length of text} by using a Fischer-Paterson Turing Machine (see Chapter 2). By careful examination of the worst case occurrence for this algorithm, the algorithm is shown to have an m*0(p*n) upper time bound, where

m = # of relators

p = length of longest LHS

n = length of input text

6.2 Complexity

Given input word W of length n, and a symmetric set with m relators, where the length of the LHS of the longest relator is p, the complexity of the algorithm is now examined. Initially, all LHS's in the symmetric set are processed in parallel using the Fischer-Paterson TM in time O(p). Next, the algorithm searches the input for the first occurrence of a relator. In the worst case, this relator occurs at positions n-p+1 through n. This was found by examining the n input symbols in time m*O(n) (for each "sub-machine"). Replacing the LHS with a RHS requires

- backing up from the n^{th} symbol to the first symbol of the found LHS (backup at most p-1 symbols).
- copying at most p-1 symbols of the RHS onto the tape.
- writing 1 null symbol.

This requires at worst 2(p-1)+1 = 2p-1 steps on each of

the m sub-machines.

Once the LHS has been replaced, the TM searches for more occurrences of LHS's. To insure that all possible LHS's are looked for, the algorithm backs up the input tape head. Since p is the length of the longest relator, the algorithm must continue by analyzing the p-1 symbols of text that are left of the newly inserted RHS, and the first symbol of the RHS. Consider the following text:

$$\cdots X_1X_2 \cdots X_{p-1}r_1r_2 \cdots r_{p-1} \cdots$$

$$\{ \text{RHS} \quad \}$$

In this example, the algorithm will continue by analyzing $X_1X_2 \ldots X_{p-1}r_1$ (p symbols). The algorithm last wrote a null symbol to the right of the RHS. Thus, we back up p-1 positions to r_1, and then move left p-1 additional symbols to X_1.

The process of searching, backup, replacement, and backup continues until the input has been exhausted. The complexity of the entire process is summarized below:

occurence of pattern	event	pattern occurs at	complexity

0	pre scan pattern	-	p
1	search	n-p+1 thru n	n
	backup		p-1
	replace		p-1+1 (null)
	backup		2(p-1)
2	search	n-p thru n-1	2(p-1)
	backup		p-1
	replace		p-1+1
	backup		2(p-1)
.			
.			
.			
n-(2p-1)+1	search	p thru 2p-1	2(p-1)
	backup		p-1
	replace		p-1+1
	backup		2(p-1)
n-(2p-2)+1	search	p-1 thru 2p-2	p-1+p-1
	backup		p-1
	replace		p-1+1
	backup		p-1+p-2
n-(2p-3)+1	search	p-2 thru 2p-3	p-1+p-2
	backup		p-1
	replace		p-1+1
	backup		p-1+p-3

.			
.			
.			
n-(p+1)+1	search	2 thru p+1	p-1+2
	backup		p-1
	replace		p-1+1
	backup		p-1+1
n-(p)+1	search	1 thru p	p-1+1
	backup		p-1
	replace		p-1+1
	backup		p-1
n-(p-1)+1	search	1 thru p-1	p-1
	backup		p-2
	replace		p-2+1
	backup		p-2
.			
.			
.			
n-1	search	1 thru 2	2
	backup		1
	replace		1+1
	backup		1

$$
\begin{aligned}
\underline{\text{Total}} = \quad & p \\
& + n+4(p-1)+1 \\
& + (n-(2p-1))(6(p-1)+1) \\
& + (p-1)(4(p-1)+1) \\
& + \sum_{i=1}^{p-1} n + \sum_{j=1}^{p-2} j \\
& + \sum_{k=2}^{p-1} k + (3 * \sum_{l=1}^{p-2} 1) + (p-2) \\
= \quad & p \\
& + n+4p-3 \\
& + (n-2p+1)(6p-5) \\
& + (p-1)(4p-3) \\
& + \frac{(p-1)(p)}{2} + \frac{(p-2)(p-1)}{2} \\
& + \frac{(p-1)(p)}{2} - 1 + 3 * \frac{(p-2)(p-1)}{2} + p - 2 \\
= \quad & n+5p-3 \\
& + 6np-5n-12p^2+10p+6p-5 \\
& + 4p^2-3p-4p+3 \\
& + \frac{p^2}{2} - \frac{p}{2} + \frac{p^2-3p}{2} \\
& + \frac{p^2}{2} - \frac{p}{2} - 1 + \frac{3p^2}{2} - \frac{9p}{2} + 3 + p - 2
\end{aligned}
$$

$= -4n-8p^2+6np+15p+3p^2-7p-4$

$= -4n+6np-5p^2+8p-4$

$= 0(n*p)$ for each submachine

$= m*0(n*p)$ for the entire machine

6.3 The Machine

Again, assume the number of relators in the symmetric set is m. The multitape Turing Machine which solves the word problem for a particular DA group will have the following properties:

- One input tape, with one right-moving, read-only head. This tape contains the original, unaltered input text string we are trying to reduce.

- 6m worktapes, partitioned into 6 distinct classes; each class contains m tapes

  - Class Z - "Modified" Input Tapes

    Each tape initially contains a LHS of a relator, followed by a delimeter. The text is concatenated onto each of these tapes during the first phase of the algorithm. When LHS's are found, the RHS's will be written onto these tapes. These tapes contain the "dynamically" changing input

string. Each tape contains 2 read-write heads, both moving left and right. These heads correspond to heads A and B of the Fischer-Paterson Turing Machine.

- Class Y - The Δ Tapes

Each tape contains two heads, both of which are 2-way, read-write. They correspond to heads C and D of the Fischer-Paterson machine. These tapes hold the Δ and d values for each "submachine".

- Class S - The S Tapes

Each tape contains one head, 2-way and read-write, called S. Each tape is used as a counter and holds the value of s of each Fischer-Paterson submachine.

- Class T - The Scratch Tapes

Each tape is used as a scratch tape analagous to tape T in the Fischer-Paterson machine. Each tape contains one read-write, 2-way head.

- Class X - More Delta Tapes

These tapes are identical copies of the Y tapes, with heads E and F which are

equivalent to heads C and D of class Y. They are used in determining whether a LHS has been matched.

- Class SUM - The "Matching" Tapes

These tapes contain two 2-way, read-write heads, I and N which are used to determine if $P(i)=|LHS|$. Recall that for a string $Z_0Z_1\ldots Z_n$, the function P for i=0,1,...n is defined $P(i)=\max\{t \mid Z_{i-t}\ldots Z_i=Z_0\ldots Z_t\}$ $(-1\leq t<i) = i-\sum_{j=-1}^{i-1}\Delta(j)$. $\Delta(i)$ is defined as $1+P(i)-P(i+1)$.

I represents the position in the text, and N represents the running sum $|LHS|+\sum_{j=0}^{i-1}\Delta(j)$. When heads I and N are at the same position, we conclude that a match of a LHS with some text has been found. (A detailed explanation of the Fischer-Paterson machine and its theoretical background appear in chapter 2).

- One "control" tape, which contains one read-write, 2-way head. This tape contains the unary value of m, and is used to synchronize the action of the m submachines.

Appendix 1 provides a PL/1 implementation of a TM

which solves the word problem for a particular small cancellation group, $\langle a,b,c,d;\ a^{-1}b^{-1}abc^{-1}d^{-1}cd\rangle$, as well as sample output. The program is commented, and can be read as if it were written in the high-level Turing Machine language previously presented.

7. THE CONJUGACY PROBLEM FOR THE PRESENTATIONS OF DA-GROUPS

Given a DA presentation of a group $G = \langle \Sigma, R \rangle$ where Σ is a set of generators, the conjugacy problem for G is defined as deciding whether or not two words U and V are conjugate elements; that is, does there exist a word W such that $W^{-1} U W = V$? A linear upper time bound is shown to exist for the solution to this problem using a multitape Turing Machine as the computational model, where null symbols of length zero can be written. The solution is derived from the solution to two other decision problems in the theory of group presentations, namely the conjugacy problem for presentations of free groups (Chapter 2) and the word problem for presentations of DA-groups (Chapter 6).

7.1 Introduction

Given a DA presentation for a group $G = \langle \Sigma, R \rangle$ where Σ is the set of generators and R is a symmetric set of relators, the conjugacy problem for G can be transformed to deciding whether the cyclic R-reductions of U and V are cyclic permutations of each other (see Chapter 1). Then the algorithm to solve the conjugacy problem is straightforward: cyclically reduce the input words U and V, and apply the fundamental lemma from the

conjugacy problem for free groups:

Lemma: Given two words U, V $\in \Sigma^*$, $|U| = |V| = n$, and U and V are cyclically reduced, then $U^2 = PVQ$ iff U and V are cyclic permutations of each other (Chapter 2).

Thus, once U and V are cyclically R-reduced, the string U^2 is formed on a worktape of the Turing Machine, and the problem is reduced to finding an occurrence of V within the string U^2. This is a straightforward application of the Fischer-Paterson pattern matching machine of Chapter 2, and its complexity is linear in the sum of the lengths of V and U.

Thus, only the analysis of the cyclic reduction of U and V remains. The inherent difficulties in doing this for the DA groups are shown.

7.2 Free Reduction of Input

To cyclically reduce a given word W in the DA group G, W must first be freely reduced. The free reduction W' of W is defined as a word which does not contain the Left Hand Side (LHS) of a relator embedded within it.

The Multitape Turing Machine which performs this function is the TM which solves the word problem for the DA groups (Chapter 6). Given a DA group G with m

relators, p is the length of the longest LHS, and n is the length of the input word W, the TM can freely reduce W in time m*0(p*n). Thus, this portion of the conjugacy problem for DA groups has linear complexity.

7.3 Cyclic Reduction of Input

A cyclically reduced word W in Σ^* is a freely reduced word which does not end with the prefix of a LHS and begin with the remaining suffix of that same LHS. As an example, let abcd be a relator. Then the LHS = abc and the RHS = d^{-1}. If a freely reduced word W is defined as c...ab or bc... a, where ... can be any sequence as long as W is still freely reduced, then W is not cyclically reduced. This is because W (in both cases) ends with a prefix of a LHS (ab, a) and begins with the remaining suffix of that LHS (c, bc).

Recall that the complexity of freely reducing a word W has been shown to be m*0(p*n), where

m = number of relators

p = length of longest LHS

n = length of input word W.

Recall that the complexity of finding a pattern V within a string U^2 has been shown to be 0(|V| + 2*|U|) = 0(3n) = 0(n) where n= length of V and U. Thus, the

complexity of the entire problem is reduced to showing that the complexity of the cyclic reduction phase of the algorithm is $0(n)$. We show this complexity is actually $m*0(p^2*n) = 0(n)$.

To cyclically reduce a given word W, we first freely reduce it using the algorithm of Chapter 6 in time $m*0(p*n)$. This yields a freely reduced word W' of the form:

$$\$X_1\ X_2\ \ldots\ X_{p-1}\ \ldots\ X_{n-(p-2)}\ \ldots\ X_n\$$$

More occurrences of LHS's can only occur between the <u>rightmost</u> p-1 symbols ($X_{n-(p-2)}$ thru X_n) and the <u>leftmost</u> p-1 symbols (X_1 thru X_{p-1}), since W' is freely reduced. Thus, the Turing Machine only considers these $2(p-1)$ symbols in time $m*0(p*(2(p-1))) = m*0(p^2)$!

If a LHS of a relator is found while scanning these $2(p-1)$ symbols, consider what must occur. The LHS is replaced by a RHS of length at most p-1. Thus, further possible occurrences of LHS's must involve symbols from the newly inserted RHS. Since the length of the longest LHS is p, <u>we must consider the p-1 symbols to the left of this RHS, and all the symbols thru the p-1 symbols to the right of the RHS</u>. Therefore, the TM backs up its heads p-1 symbols to the left of the RHS, and scans thru the p-1<u>st</u> symbol to the right of the RHS. This implies scanning at most $3(p-1)$

symbols. Therefore, the complexity here is $m*0(p*(3(p-1))) = m*0(p^2)$!

We continue scanning at least 3(p-1) symbols until no more replacements can be made. Each time a LHS is found, it is replaced by a RHS of shorter length, and at least 3(p-1) symbols beginning with the p-1st to the left of the RHS are scanned. Since at each step, 0(p) symbols are scanned in time $0(p^2)$, and since the length of the freely reduced word W' is n, the entire algorithm has complexity $m*0(p^2*n) = 0(n)$!

7.4 The Cyclic R-Reduction Algorithm

Presented below is the Turing Machine algorithm that cyclically reduces a string W in time $m*0(p^2*n)$. Section 7.5 shows how this complexity result is derived.

```
CYCLIC:  PROCEDURE (RECURSIVE);
```

/*Turing Machine algorithm to cyclically reduce a word for the small cancellation groups.

Assume W, the word to be cyclically reduced, consists of the symbols $\$X_1X_2...X_n\$$.

Let head A initially be positioned under $X_{n-(p-2)}$. Let head B initially be positioned under X_{p-1}.

Assume W has been freely reduced. Then the only possible occurrences of LHS's occur between (in the cyclic sense) $X_{n-(p-2)} \cdot\cdot\cdot X_nX_1 \cdot\cdot\cdot X_{p-1}$. We mark X_{p-1} as the rightmost symbol to be scanned. If X_{p-1} is the symbol x, it is marked as Rx. */

```
W(B) = Rx;
```

/* Now we begin the major portion of the algorithm. We scan, starting at $X_{n-(p-2)}$, all the symbols through the rightmost symbol (marked Rx).

Let HEAD be a variable that takes on the value A if head A is scanning the "current" symbol, else HEAD = B. Note, if HEAD = A, we say $\overline{\text{HEAD}}$ = B, and if HEAD = B, we say $\overline{\text{HEAD}}$ = A.

Note too, that heads A & B will be moving in opposite directions over the tape containing W. They

will always be the same number of symbols from the opposite delimeter symbols ($). */

```
HEAD = A; /* initially, A is right moving */

REPEAT;

    CALL FORWARD (HEAD);

    /* FORWARD scans "right" until a LHS is found. If
no LHS is found, FORWARD will halt the execution of the
algorithm. "Right" implies reversing the value of HEAD
when the delimeter is found.

    If a LHS is found, FORWARD returns leaving HEAD
positioned at the rightmost symbol of the LHS. Thus,
we backup |LHS|-1 symbols. */

    CALL BACKUP (HEAD, |LHS| -1);

    /* BACKUP leaves HEAD positioned under the first
symbol of the "found" LHS. Now the LHS is replaced by
the RHS and null symbols */

    CALL REPLACE (HEAD, RHS);

    /* REPLACE leaves HEAD positioned under the
rightmost null symbol just written.

    Next, we must "mark" the new rightmost symbol to
be scanned. This will be at least p-1 symbols to the
```

"right" of the null symbol just written. Note, if the Rx is still right of the p-1st symbol to the right, MARK does nothing. However if MARK passes over Rx, the p-1st symbol becomes Rx. */

```
CALL MARK (HEAD, p-1);

/* Finally, HEAD must now be reversed 2(p-1) +
|RHS| -1 non-null symbols in order to consider all
possible occurences of LHS's.*/

CALL BACKUP (HEAD, 2(p-1) + |RHS| -1);

END; /* END of REPEAT */
```

```
   FORWARD:  PROCEDURE (HEAD);

   /* subroutine FORWARD scans "right" for Rx until

   • a LHS is found, in which case FORWARD merely
     returns,

   • no LHS is found, in which case FORWARD halts.  */

DO WHILE (W(HEAD) ≠ Rx );

   IF W(HEAD) = $

      THEN DO; /* reverse heads */

               HEAD = ¬HEAD;

               HEAD = HEAD + 1;

               ¬HEAD = ¬HEAD - 1;

           END;

      ELSE DO;
```

```
                Scan W(HEAD);
                IF (a LHS has been found)
                   THEN RETURN;
                   ELSE DO;  /* advance the heads */
                             HEAD = HEAD +1;
                             HEAD' = HEAD' -1;
                         END;
                END;
END;

/* Rx has been found...
   IF Rx causes a LHS to be found,
      return; otherwise halt -
      W is fully reduced */

Scan W(HEAD);
IF (a LHS has been found)
   THEN RETURN;
   ELSE STOP;
```

```
END FORWARD;
```

```
     BACKUP:  PROCEDURE (HEAD, LENGTH);

     /* Subroutine BACKUP  reverses  the  direction  of
HEAD and moves HEAD LENGTH symbols "left" */

     /* If Rx is found while backing up, a new type  of
processing will be done....*/

DO I = 1 to LENGTH;

   IF W(HEAD) = Rx

      THEN CALL NEWSTATE;

      ELSE DO; HEAD = HEAD-1;

               HEAD = HEAD + 1;


   IF W(HEAD) = $

      THEN DO; /* Reverse */

               HEAD = HEAD

               HEAD = HEAD -1;

               HEAD = HEAD + 1;

          END;

          END; /* End of ELSE -DO */

END; /* end of DO-I */
```

```
/* HEAD is now correctly positioned */

RETURN;

END BACKUP;
```

```
     REPLACE:  PROCEDURE (HEAD, RHS);
/* Subroutine REPLACE replaces the previously found LHS
with the corresponding RHS (including null symbols)*/
DO I = 1 TO |RHS|;
  W(HEAD) = RHS[I];
            /* let RHS be an array
               of symbols 1 thru |RHS| */
  HEAD = HEAD + 1;
  HEAD_ = HEAD_ -1;
  IF W(HEAD) = $ /* reverse */
    THEN DO;  HEAD = HEAD_;
              HEAD = HEAD +1;
              HEAD_ = HEAD_ -1 ;
         END;
END;  /* End of DO-I */
END REPLACE ;
```

```
     MARK: PROCEDURE (HEAD, LENGTH);
/* Subroutine MARK moves HEAD right LENGTH symbols.  If
the  Rx  symbol is found in the process, it is replaced
by x, and the rightmost  symbol  found  is  marked  Rx.
Conversely, if the Rx symbol is not found, no "marking"
is done. */

DO I = 1 TO LENGTH;

   IF W(HEAD) = Rx

      THEN DO; FLAG = 1;

                W(HEAD) = x;

           END;

   HEAD = HEAD +1;

   HEAD = HEAD -1 ;


   IF W(HEAD) = $ /* Reverse */

   THEN DO;  HEAD = HEAD;

             HEAD = HEAD + 1;

             HEAD = HEAD -1;

        END;

END; /* end of DO-I */
```

```
END MARK;
```

```
NEWSTATE: PROCEDURE;
/* Subroutine NEWSTATE is called by BACKUP if Rx is
found.  Had BACKUP not called NEWSTATE, subroutine
FORWARD would not scan the proper number of symbols.
This condition occurs when a number of BACKUP's
resulted in causing all of W to have been "reverse-
scanned".

    In this situation, W must be freely reduced
(again), followed by having this entire algorithm "re-
execute." This results in minimizing the number of
steps to be executed.  */

/* Note, W(HEAD) is Rx, so first...*/

W(HEAD) = X;

Freely reduce W;

Position A and B under

   X(n-(p-2)) and X(p-1) respectively;

CALL CYCLIC;


END NEWSTATE;


END CYCLIC; /* all done */
```

7.5 Complexity

It seems rather surprising that cyclically reducing W can be done in time linear in the length of W. We examine the complexity of this procedure here via a worst-case analysis.

Let W (freely reduced) be

$$X_1 \ldots X_{p-1}\ X_p \ldots X_{n-(p-2)} \ldots X_n\ \$$$

Initially, symbols $X_{n-(p-2)}$ thru X_n followed by X_1 thru X_{p-1} are scanned. Assume no LHS occurs until symbol X_{p-1} is scanned. Thus, we are freely reducing 2(p-1) symbols. From the previous chapter, the complexity of freely reducing a set of n symbols has complexity m*O(p*n).

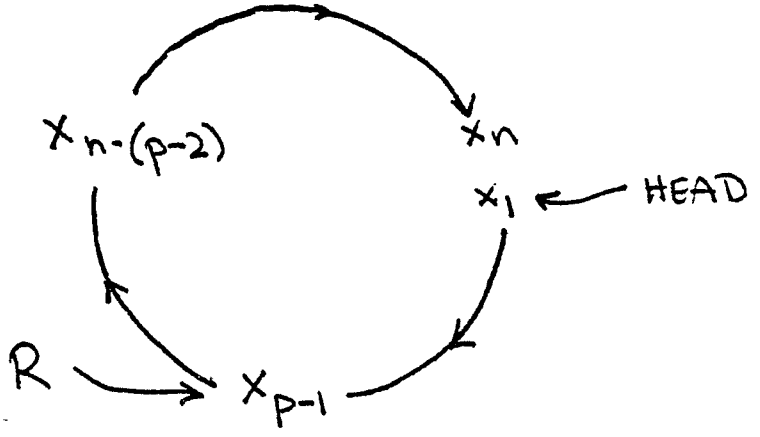
Thus, freely reducing the 2(p-1) symbols has complexity m*O(p *(2p-1)) = m*O(p^2).

Next, HEAD is backed up at most 2(p-1) symbols. From here on, 3(p-1) symbols are scanned. Again, assume the LHS is not found until the 3(p-1)st symbol is scanned. Replacing the LHS with the RHS results in this entire operation having complexity m*O(p*(3p-1)) = m*O(p^2).

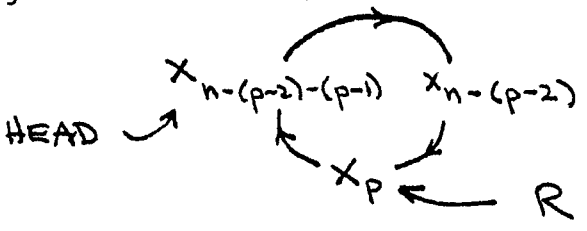
Again, HEAD is backed up at most 3(p-1) symbols, and the process is repeated. Clearly, each iteration

in the process results in reducing the length of W by at least 1. Thus, if |W|=n, the complexity of the entire cycle reduction process is at most $\underline{m}*\underline{0}(p^2*\underline{n})$.

Now consider cases other than the worst case. Assume that the LHS's are found after scanning only p symbols each time. To simplify the explanation, we write W on a circle, with HEAD pointing to the current symbol being scanned and R pointing to the rightmost symbol to be scanned. In the first case, the TM has the following configuration after scanning p symbols:

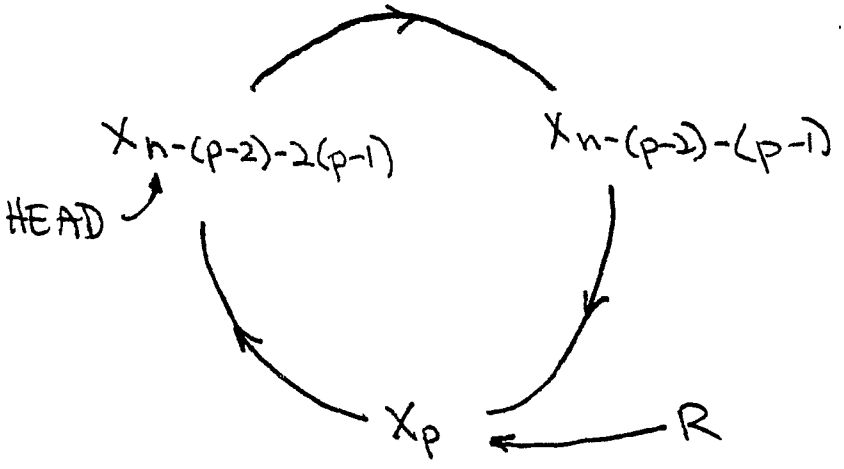


The complexity of scanning these p symbols ($X_{n-(p-2)} \cdots X_nX_1$), replacing them with the corresponding RHS, marking the new rightmost symbol, and backing up is $m*0(p*3p-1)) = m*0(p^2)$. The new TM configuration will be:



The $3(p-1)$ symbols $X_{n-(p-2)-(p-1)}$ thru X_n followed by X_1 thru X_p must now be examined.

Next, again assume that a LHS is found after examining only the first p symbols. The complexity of this operation will again be $m*0(p^2)$. Note, however, the resulting configuration:



At this point, R was not moved, so we must examine the $3(p-1)$ symbols to the right of HEAD, plus the remaining $2p-1$ symbols that were not scanned before. By accounting for previously unscanned symbols in later steps, we can safely assume that the complexity of reducing each symbol in W is $m*0(p*(3(p-1))!$

Note that backing up HEAD can result in "passing" the rightmost symbol marked by R. Effectively, all of W must be scanned in this case, because of all the reduction that has occurred. Rather than count how many symbols should be scanned, we merely start over by freely reducing W from left to right. This has

complexity m*0(p*n') where n' is the length of W after all the cyclic reduction (n'<n). Then, the cyclic reduction algorithm is restarted. This is less complex than going "around and around" W.

Note this extreme backup case results in linear complexity. This is because n' can actually be expressed as a function of p!

7.6 Summary

The steps of the algorithm are:

1 - Freely R-reduce U using the TM of the word problem for the DA groups in time m*0(p*n).

2 - Cyclically R-reduce U', in time $m*0(p^2*n)$.

3 - Freely R-reduce V as in (1) above in time m*0(p*n).

4 - Cyclically R-reduce V as in (2) above in time $m*0(p^2*n)$.

5 - Form the string \$ V & U^2, where V, U are both cyclically R-reduced and use the Fischer-Paterson machine of Chapter 1 to find an occurrence of V within U^2 in time 0(|U| + |V|). If such an occurrence exists, the TM halts successfully, otherwise failure.

Each component of the algorithm has linear complexity, and thus the entire machine runs in linear time.

8. CONCLUSIONS

A variety of problems in the theory of group presentations have been shown to have linear time and log space complexity bounds. Results were obtained by using linear time pattern matching algorithms on multitape Turing Machines.

A large number of decision problems in the theory of group presentations still remain to be analyzed for their computational complexity. For instance, the word and conjugacy problems for the HNN groups (Anshel and Stebe, 1974). Perhaps the solutions to these problems will lead to further research, as did Max Dehn's work in 1911.

APPENDIX

```
        SMALL: PROC OPTIONS(MAIN);

STMT LEVEL NEST
  1                 SMALL: PROC OPTIONS(MAIN);


                          /* TURING MACHINE FOR SMALL
                             CANCELLATION GROUPS IN LINEAR TIME   */

                          /* DEFINE THE ARRAYS AND STATIC POINTERS        */

                          /* THE NUMBER OF RELATORS = M = 24
                             ASSUME UP TO 50 SYMBOLS CAN BE PLACED ONTO A
                             Z TAPE - NOTE THIS REQUIREMENT CAN BE INCREASED
                             AT WILL.

                             EACH TEXT SYMBOL IS REPRESENTED BY 2 CHARACTERS:

                             A  IS REPRESENTED BY  A
                              -1
                             A   IS REPRESENTED BY  A-
                          */

  2     1               DCL Z(24,50) CHAR(2) STATIC;
  3     1               DCL (Y(24,50), X(24,50),
                             S(24,50), SUM(24,50))   CHAR(1) STATIC;

                          /* 1'S AND 0'S ARE REPRESENTED USING ONE CHARACTER EACH */

  4     1               DCL (A(24), B(24), C(24), D(24), E(24), F(24),
                             N(24), II(24), SS(24), R(24)    )
                             FIXED BIN(15) STATIC;


                          /* THE FOLLOWING ARE THE HEADS ON THE VARIOUS TAPES:

                          TAPE    HEAD
                          ----    ----

                          Z       A,B
                          Y       C,D
                          X       E,F
                          S       SS
                          SUM     N,II

                          NOTE THAT A(I) IMPLIES THAT HEAD A IS CURRENTLY
                          READING THE I-TH SYMBOL ON A TAPE.

                          */

  5     1               DCL (I,J,K,MM) FIXED BIN(15) STATIC;

                          /* THE COUNTERS */
```

```
        SMALL: PROC OPTIONS(MAIN);

STMT LEVEL NEST

                        /* READ IN THE INPUT WORD        */
  6     1               DCL INPUT CHAR(80) STATIC;
                        /* UP TO 40 2-CHARACTER INPUT SYMBOLS ARE ALLOWED.      */
  7     1               ON ENDFILE(SYSIN) GO TO END_SMALL;
  9     1         READ_FILE:
                        READ FILE(SYSIN) INTO(INPUT);
                         /* ASSUME THAT $ SURROUNDS THE INPUT TEXT */
```

```
        SMALL: PROC OPTIONS(MAIN);

STMT LEVEL NEST

                        /* INITIALIZE THE TAPES --
                        */


                        /*
                        THE PARTICULAR GROUP USED IN THE EXAMPLE IS THE GROUP
                        WITH THE SYMMETRIC SET OF THE RELATOR
                                A-B-A B C-D-C D

                        BELOW ARE THE LHS "RULES" FOR THE SYMMETRIC SET WITH
                        DELIMETERS $ AND &
                        */

 10     1               DCL LHS(24) CHAR(14) VAR INIT
                        ('$ A-B-A B C-& ',
                         '$ D A-B-A B & ',
                         '$ C D A-B-A & ',
                         '$ D-C D A-B-& ',
                         '$ C-D-C D A-& ',
                         '$ B C-D-C D & ',
                         '$ A B C-D-C & ',
                         '$ B-A B C-D-& ',
                         '$ D-C-D C B-& ',
                         '$ A D-C-D C & ',
                         '$ B A D-C-D & ',
                         '$ A-B A D-C-& ',
                         '$ D-A-B A D-& ',
                         '$ C B-A-B A & ',
                         '$ D C D-A-B & ',
                         '$ C-D C D-A-& ',
                         '$ A A-& ',
                         '$ A-A & ',
                         '$ B B-& ',
                         '$ B-B & ',
                         '$ C C-& ',
                         '$ C-C & ',
                         '$ D D-& ',
                         '$ D-D & '                );
```

```
        SMALL: PROC OPTIONS(MAIN);

STMT LEVEL NEST

                       /*
                       BELOW ARE THE RHS "RULES" FOR THE SYMMETRIC SET
                       */

 11     1              DCL RHS(24) CHAR(10) VAR STATIC INIT
                       ('  D-C-D   ',
                        '  C-D C   ',
                        '  D C B-  ',
                        '  C B-A-  ',
                        '  B-A-B   ',
                        '  A-B A   ',
                        '  B A D-  ',
                        '  A D-C-  ',
                        '  A-B-A   ',
                        '  B-A B   ',
                        '  A B C-  ',
                        '  B C-D-  ',
                        '  C-D-C   ',
                        '  D-C D   ',
                        '  C D A-  ',
                        '  D A-B-  ',
                        (8) (4) ' '    );
```

```
        SMALL: PROC OPTIONS(MAIN);

STMT LEVEL NEST

                        /*
                        MORE VARIABLES
                        */
 12     1               DCL M FIXED BIN(15) STATIC INIT(24);
                        /* M = THE NUMBER OF RELATORS (= 24) */

 13     1               DCL P FIXED BIN(15) STATIC INIT(5);
                        /* P = THE LENGTH OF THE LONGEST RELATOR (= 5) */

                        /* NOW INITIALIZE THE Z ARRAY   */
 14     1               DO I = 1 TO M;
 15     1   1              DO J = 1 TO LENGTH(LHS(I))/2;
 16     1   2                 Z(I,J) = SUBSTR(LHS(I),2*J-1,2);
                        /* THE LEFTMOST SYMBOLS ON EACH Z TAPE ARE THE LHS WITH
                           DELIMETERS $ AND &.                                  */
 17     1   2              END;
                           /* CONCATENATE THE INPUT TOO */
 18     1   1           DO K = 1 TO LENGTH(INPUT)/2
                             WHILE(SUBSTR(INPUT,2*K-1,2) ¬= ' ');
                        /* THE INPUT IS DONE WHEN A BLANK IS ENCOUNTERED */
 19     1   2              Z(I,J+K-1)   = SUBSTR(INPUT,2*K-1,2);
 20     1   2              END;
 21     1   1           END;

                        /* NEXT, INITIALIZE THE OTHER TAPES */
 22     1               DO I = 1 TO M;
 23     1   1              Y(I,1) = '0';        Y(I,2) = '1';   Y(I,3) = '0';
 26     1   1              C(I)   = 1;          D(I)   = 3;

 28     1   1              X(I,1) = '0';        X(I,2) = '1';   X(I,3) = '0';
 31     1   1              E(I)   = 1;          F(I) = 3;

 33     1   1              S(I,1) = '0';
```

```
      SMALL: PROC OPTIONS(MAIN);

STMT LEVEL NEST

 34     1    1              SS(I)  = 2;

                           /* INITIALIZE HEADS A AND B FOR THE Z TAPES */
 35     1    1              A(I) = 1;
 36     1    1              B(I) = 2;
                            /* AND HEADS II AND N FOR THE SUM TAPES      */
 37     1    1              II(I) = 1;
 38     1    1               N(I) = 1;
 39     1    1           END;

                           /* SUM TAPE INITIALIZATION (SEE CHAPTER 1 - THE CONJUGACY
                              PROBLEM FOR FREE GROUPS)     */

 40     1                 DO I = 1 TO M;
 41     1    1              DO WHILE(Z(I,A(I)+1) ¬= '&');
 42     1    2                   SUM(I,N(I))     = '1';
 43     1    2                   N(I) = N(I) + 1;
 44     1    2                   A(I) = A(I) + 1;
 45     1    2              END;

 46     1    1           N(I) = N(I) - 1;
 47     1    1           SUM(I,N(I)) = '0';
 48     1    1           DO WHILE( Z(I,A(I)) ¬= '$' );
 49     1    2              A(I) = A(I) - 1;
 50     1    2           END;

 51     1    1           END;
 52     1                PUT PAGE LIST('INITIAL CONFIGURATION');
 53     1                MM = -1;
 54     1                CALL OUTPUT;
```

```
        SMALL: PROC OPTIONS(MAIN);

STMT LEVEL NEST

                                /*  PROCESS 1 TEXT SYMBOL AT A TIME FROM EACH
                                   TAPE GIVEN THE INITIAL CONFIGURATION
                                   UNTIL THE & IS FOUND     */

 55     1                       DO I = 1 TO M;
 56     1     1                   DO WHILE(Z(I,B(I)) ¬= '&' );
 57     1     2                         CALL PATTERN;

                                /*

                                THE SUMMATION MUST BE INCREMENTED BY
                                THE NEW VALUE OF DELTA (BETWEEN THE 0'S)
                                AND THE HEADS MUST BE MOVED RIGHT.

                                */

 58     1     2                         E(I) = E(I) + 1;
 59     1     2                         DO WHILE( X(I, E(I)) ¬= '0' );
 60     1     3                            SUM( I, N(I)) = '1';
 61     1     3                            N(I) = N(I) + 1;

 62     1     3                            E(I) = E(I) + 1;
 63     1     3                         END;

 64     1     2                         SUM(I, N(I)) = '0';
 65     1     2                         II(I) = II(I) + 1;

 66     1     2                   END;
 67     1     1               END;          /* *** ALL PATTERNS ARE NOW PROCESSED *** */
 68     1                     PUT PAGE LIST('ALL PATTERNS ARE PRE-PROCESSED:');
 69     1                     CALL OUTPUT;

 70     1                     PUT PAGE LIST('TEXT IS NOW TO BE PROCESSED');
```

```
        SMALL: PROC OPTIONS(MAIN);

STMT LEVEL NEST

                        /* THE FISCHER-PATERSON PATTERN MATCHING SUBROUTINE */

 71     1           PATTERN:  PROC;
 72     2                DCL T FIXED BIN(15) STATIC;

                        /*

                        T IS EFFECTIVELY THE TAPE CLASS T OF THE FISCHER-PATERSON

                        */

 73     2                T = 0;

                        /* 1ST TEST */

 74     2           TEST1:
                        IF( Z(I, A(I)+1) = Z(I, B(I)+1) )
 75     2                THEN DO;

                        /*

                        HEADS A AND B ARE READING IDENTICAL SYMBOLS, SO,

                        */

 76     2    1                 D(I) = D(I) + 1;
 77     2    1                 Y(I, D(I)) = '0';

                        /*

                        MOVE HEADS D AND F RIGHT AND PRINT '0'

                        */
 78     2    1                 F(I) = F(I) + 1;
 79     2    1                 X(I, F(I)) = '0';
 80     2    1                 A(I) = A(I) + 1;

                        /*

                        MOVE HEADS A AND B RIGHT

                        */


 81     2    1                 B(I) = B(I) + 1;
 82     2    1                 C(I) = C(I) + 1;

                        /*

                        MOVE C RIGHT TO THE NEXT 0.
```

```
         SMALL: PROC OPTIONS(MAIN);

STMT LEVEL NEST

                                AS C IS ADVANCED, S IS INCREMENTED
                                ONCE FOR EACH '1' THAT C PASSES OVER

                                */

 83     2    1                      DO WHILE( Y(I, C(I)) = '1');
 84     2    2                         S(I, SS(I))  = '1';
 85     2    2                         SS(I) = SS(I) + 1;
 86     2    2                         C(I) = C(I) + 1;
 87     2    2                      END;
 88     2    1                     END;

                                /* CASE 2 */

 89     2                      ELSE

 89     2                       IF (Z(I, A(I)) = '5')
 90     2                         THEN DO;

                                /*

                                A IS POSITIONED AT THE LEFTMOST SYMBOL ON Z

                                A AND C ARE LEFT ALONE,
                                B MOVES RIGHT ONE SYMBOL,
                                D AND F MOVE RIGHT 2 SYMBOLS,
                                PRINTING A '1' FOLLOWED BY A '0'

                                */

 91     2    1                         B(I) = B(I) + 1;
 92     2    1                         D(I) = D(I) + 1;
 93     2    1                         Y(I,D(I)) = '1';
 94     2    1                         D(I) = D(I) + 1;
 95     2    1                         Y(I,D(I)) = '0';

 96     2    1                         F(I) = F(I) + 1;
 97     2    1                         X(I,F(I)) = '1';
 98     2    1                         F(I) = F(I) + 1;
 99     2    1                         X(I,F(I)) = '0';
100     2    1                         END;

                                /* CASE 3 */

101     2                      ELSE DO;
```

```
SMALL: PROC OPTIONS(MAIN);

STMT LEVEL NEST

                        /*

                        "COPY" THE CONTENTS OF S INTO "T"

                        C IS MOVED LEFT OVER  S  ZEROES.
                        FOR EACH '0' PASSED OVER BY C,
                        HEAD A MOVES ONE SQUARE TO THE LEFT AND
                        HEADS D AND F MOVE ONE SQUARE TO THE RIGHT
                        AND PRINT A '1'.

                        FOR EACH '1' PASSED OVER BY C,
                        S IS DECREMENTED.  SINCE THE VALUE OF S
                        IS MODIFIED BY THIS PROCESS,
                        ITS CONTENTS ARE FIRST COPIED INTO THE
                        TEMPORARY COUNTER  T  WHICH IS THEN
                        USED TO CONTROL THE ITERATION.

                        */
102     2    1          SS(I) = SS(I) - 1;
103     2    1          DO WHILE(S(I,SS(I)) = '1');
104     2    2             T = T + 1;
105     2    2             SS(I) = SS(I) -1;
106     2    2          END;
107     2    1          SS(I) = SS(I) + 1;       /* GO BACK TO WHERE WE CAME FROM */
108     2    1          DO WHILE(S(I,SS(I)) = '1');
109     2    2            SS(I) = SS(I) + 1;
110     2    2          END;
111     2    1          SS(I) = SS(I) - 1;
112     2    1          DO J = 1 TO T;
113     2    2             C(I) = C(I) -1;
114     2    2             DO WHILE(Y(I,C(I)) = '1');
115     2    3                   S(I,SS(I)) = ' ';
                                 /* DECREMENT S */
116     2    3                   SS(I) = SS(I) - 1;
117     2    3                   C(I) = C(I) -1;
118     2    3             END;
119     2    2             A(I) = A(I) - 1;
120     2    2             D(I) = D(I) + 1;
121     2    2             Y(I,D(I)) = '1';
122     2    2             F(I) = F(I) +1;
123     2    2             X(I,F(I)) = '1';
124     2    2          END;

125     2    1          GO TO TEST1;    /* ** GO TO NEXT STAGE ** */

126     2    1       END;       /* END OF ELSE DO */


                        /* ALL DONE */
```

```
SMALL: PROC OPTIONS(MAIN);

STMT LEVEL NEST

127     2                   END PATTERN;
```

```
        SMALL: PROC OPTIONS(MAIN);

STMT LEVEL NEST

                        /* PROCESS TEXT */

128     1           SCAN:

                        DO I = 1 TO M;


                        /* TEST IF WE'RE ALL DONE */

129     1   1            IF (Z(I,B(I)+1) = ' ') THEN GO TO END_OF_INPUT;
131     1   1            CALL PATTERN;

                        /*

                        PROCESS NEXT INPUT SYMBOL

                        UPDATE SUM - ADD NEXT DELTA VALUE ONTO SUM

                        */


132     1   1            E(I) = E(I) + 1;
133     1   1            DO WHILE(X(I,E(I)) = '1' );
134     1   2              SUM(I,N(I)) = '1';
135     1   2              N(I) = N(I) + 1;
136     1   2              E(I) = E(I) + 1;
137     1   2            END;

                         /* X(I,E(I)) IS NOW 0 */

138     1   1            SUM(I,N(I)) = '0';
139     1   1          END;


                        /*

                        WHEN HEADS II AND N ARE POSITIONED
                        AT PRECISELY THE SAME SYMBOL,
                        WE'VE GOT A MATCH.

                        */


140     1               DO I = 1 TO M;
141     1   1            IF SUM(I,III(I)) = '0'
142     1   1              THEN GO TO MATCH;
143     1   1          END;

                        /* NO MATCHES - MOVE II RIGHT ONE */
                        /* POSITION ON ALL THE TAPES      */
```

```
          SMALL: PROC OPTIONS(MAIN);

STMT LEVEL NEST

144      1               DO I = 1 TO N;
145      1    1            II(I) = II(I) + 1;
146      1    1          END;

                         /* START SCANNING NEXT TEXT SYMBOL */

147      1               GO TO SCAN;
```

```
          SMALL: PROC OPTIONS(MAIN);

STMT LEVEL NEST

                                 /* MATCH - */

148       1           MATCH:
                                 MM = I;
                                 /* THE I-TH MACHINE MATCHED */
149       1                      DO I = 1 TO M;

                                 /*

                                 MODIFY THE CONTENTS OF EACH TAPE.
                                 BACKUP (|LHS| - 1) SYMBOLS ON TAPE Z

                                 */

150       1     1                  DO J = 1 TO LENGTH(LHS(MM))/2-2;
151       1     2                    B(I) = B(I) - 1;
152       1     2                    Z(I,B(I)) = ' ';
153       1     2                  END;

                                 /* NOW COPY THE NEW RHS ONTO Z */

154       1     1                  DO J = 1 TO LENGTH(RHS(MM))/2-2;

155       1     2                     Z(I,B(I)) = SUBSTR(RHS(MM),2*J+1,2);
156       1     2                     B(I) = B(I) + 1;
157       1     2                  END;

                                 /*

                                 ELIMINATE THE BLANKS THAT ARE LEFT OF THE REMAINING TEXT.
                                 THIS IS EQUIVALENT TO PLACING "DONT-CARE" SYMBOLS ONTO THE TAPE.
                                 HERE, THE REMAINING TEXT IS BEING MOVED LEFT SO THAT IT WILL
                                 BE ADJACENT TO THE NEWLY INSERTED RHS.

                                 */

158       1     1                  DO WHILE( Z(I,B(I)) = ' ');
159       1     2                     DO J = B(I)+1 TO 50 WHILE( Z(I,J) ¬= '$');
160       1     3                        Z(I,J-1) = Z(I,J);
161       1     3                     END;

162       1     2                     Z(I,J-1) = Z(I,J);
163       1     2                     Z(I,J) = ' ';
164       1     2                  END;

                                 /* NOW BACKUP B (|RHS|-1) SYMBOLS
                                    SO THE NEW RHS CAN BE SCANNED */

165       1     1                  B(I) = B(I) - 1;
166       1     1                  DO J = 1 TO LENGTH(RHS(MM))/2-2;
```

```
        SMALL: PROC OPTIONS(MAIN);

STMT LEVEL NEST

167     1    2              B(I) = B(I) -1;
168     1    2          END;

                        /*

                        A & C ARE MOVED LEFT TO POINT TO THE BEGINNING OF THE PATTERN

                        */

169     1    1          DO WHILE(Z(I,A(I)) ¬= '$');
170     1    2             A(I) = A(I) - 1;
171     1    2             C(I) = C(I) - 1;
172     1    2             DO WHILE( Y(I,C(I)) = 'L' );
173     1    3                  C(I) = C(I) - 1;
174     1    3             END;
175     1    2          END;

                        /* S MUST BE 0 */

176     1    1          S(I,1) = '0';
177     1    1          DO J = 2 TO 50;
178     1    2             S(I,J) = ' ';
179     1    2          END;
180     1    1          SS(I) = 2;

                        /* D,F,II & N MUST BE MOVED LEFT |LHS| SYMBOLS */

181     1    1          DO J = 1 TO LENGTH(LHS(MM))/2-1;

182     1    2          Y(I,D(I)) = ' ';
183     1    2          D(I) = D(I) - 1;
184     1    2          X(I,F(I)) = ' ';
185     1    2          F(I) = F(I) - 1;
186     1    2             DO WHILE(Y(I,D(I)) = 'L');
                              /* ERASE */
187     1    3                Y(I,D(I)) = ' ';
188     1    3                X(I,F(I)) = ' ';
                              /* BACKUP */
189     1    3                D(I) = D(I) - 1;
190     1    3                F(I) = F(I) - 1;
191     1    3             END;


                        /* E MOVES LEFT OVER |LHS| DELTA VALUES */
192     1    2          E(I) = E(I) - 1;
193     1    2          DO WHILE( X(I,E(I)) = 'I' );
194     1    3             SUM(I,N(I)) = ' ';
195     1    3             N(I) = N(I) - 1;
196     1    3             E(I) = E(I) - 1;
197     1    3          END;
```

```
        SMALL: PROC OPTIONS(MAIN);

STMT LEVEL NEST

198      1    2            III(I) = III(I) - 1;

199      1    2           END;
200      1    1            III(I) = II(I) + 1;
201      1    1            SUM(I,N(I)) = '0';
```

```
SMALL: PROC OPTIONS(MAIN);

STMT LEVEL NEST

                        /* NOW BACKUP |P|-1 SYMBOLS OR TO THE &
                           WHICHEVER COMES FIRST          */

202     1    1          DO J = 1 TO P-1 WHILE(
                            Z(I,B(I)) ¬= '&'  );
203     1    2               B(I) = B(I) - 1;
204     1    2               Y(I,D(I)) = ' ';
205     1    2               D(I) = D(I) - 1;
206     1    2               X(I,F(I)) = ' ';
207     1    2               F(I) = F(I) - 1;
208     1    2               DO WHILE(Y(I,D(I)) = '1');
209     1    3                  Y(I,D(I)) = ' ';
210     1    3                  D(I) = D(I) - 1;
211     1    3                  X(I,F(I)) = ' ';
212     1    3                  F(I) = F(I) - 1;
213     1    3               END;
214     1    2               II(I) = II(I) - 1;
215     1    2               E(I) = E(I) - 1;
216     1    2               DO WHILE (X(I,E(I)) = '1' );
217     1    3                  SUM(I,N(I)) = ' ';
218     1    3                  N(I) = N(I) - 1;
219     1    3                  E(I) = E(I) - 1;
220     1    3               END;

221     1    2          END;

222     1    1          SUM(I,N(I)) = '0';

223     1    1      END;



                        /* NOW START ALL OVER */

224     1               PUT PAGE LIST('AFTER MATCHING');

225     1               CALL OUTPUT;
226     1               MM = 0;
227     1               GO TO SCAN;
```

```
        SMALL: PROC OPTIONS(MAIN);

STMT LEVEL NEST

                        /*  ALL DONE - PRINT THE MACHINE STATUS AND ITS RESULT */

228      1              END_OF_INPUT:

                            DO WHILE( Z(I,B(I)) ¬= '&' );
229      1    1                B(I) = B(I) - 1;
230      1    1           END;

231      1                B(I) = B(I) + 1;          /* MOVE OVER ONE */

232      1                IF (Z(I,B(I)) ¬= '$')

                          /*

                          IF TEXT APPEARS BETWEEN THE $ AND THE &,
                          THE INPUT HAS NOT BEEN REDUCED TO THE IDENTITY.

                          ON THE OTHER HAND, IF NO TEXT APPEARS BETWEEN THE
                          DELIMETERS, THE INPUT HAS BEEN REDUCED TO THE IDENTITY.

                          */
233      1                   THEN PUT SKIP(5) EDIT('FAILURE')(A);
234      1                   ELSE PUT SKIP(5) EDIT('SUCCESS')(A);

                          /* REINITIALIZE THE TAPES FOR THE NEXT INPUT WORD */

235      1                Z(*,*) = ' ';
236      1                Y(*,*) = ' ';
237      1                X(*,*) = ' ';
238      1                S(*,*) = ' ';
239      1                SUM(*,*) = ' ';

240      1                A(*) = 0;
241      1                B(*) = 0;
242      1                C(*) = 0;
243      1                D(*) = 0;
244      1                E(*) = 0;
245      1                F(*) = 0;
246      1                N(*) = 0;
247      1                II(*)= 0;
248      1                SS(*) = 0;
249      1                R(*) = 0;
250      1                MM = 0;
251      1                GO TO READ_FILE;
```

```
        SMALL: PROC OPTIONS(MAIN);

STMT LEVEL NEST

                        /* OUTPUT ROUTINE */

252     1           OUTPUT: PROC;
253     2                   IF (MM < 0) THEN
254     2                      DO I = 1 TO M;
255     2   1                     CALL OUTPUT2;
256     2   1                  END;

257     2                   ELSE
257     2                     DO;
258     2   1                    I = MM;
259     2   1                    CALL OUTPUT2;
260     2   1                  END;


261     2           OUTPUT2: PROC;
262     3                    PUT SKIP(2) EDIT
                             ('Z  :',( Z(I,J) DO J = 1 TO 50 )) (R(FMTZ));
263     3                    PUT SKIP EDIT('A','B')
                               (X(A(I)*2+2),A,SKIP(0),X(B(I)*2+2),A);
264     3                    PUT SKIP EDIT(
                               'Y  :',( Y(I,J) DO J = 1 TO 50 ) ) (R(FMT_XY));
265     3                    PUT SKIP EDIT('C','D')
                               (X(C(I)+3),A,SKIP(0),X(D(I)+3),A);
266     3                    PUT SKIP EDIT(
                               'X  :',(X(I,J) DO J = 1 TO 50)) (R(FMT_XY));
267     3                    PUT SKIP EDIT('E','F')
                               (X(E(I)+3),A,SKIP(0),X(F(I)+3),A);
268     3                    PUT SKIP EDIT('S  :', (S(I,J) DO J = 1 TO 50)) (R(FMT_XY));
269     3                    PUT SKIP EDIT('S')(X(SS(I)+3),A);
270     3                    PUT SKIP EDIT('SUM:', (SUM(I,J) DO J = 1 TO 50) )
                               (R(FMT_XY));
271     3                    PUT SKIP EDIT('I','N')
                               (X(II(I)+3),A,SKIP(0),X(N(I)+3),A);


272     3           FMTZ:
                           FORMAT( A(4),  (50) A(2) );

273     3           FMT_XY:
                           FORMAT( A(4), (50) A(1) );

274     3                  END OUTPUT2;


275     2                  END OUTPUT;

276     1           END_SMALL:
```

```
        SMALL: PROC OPTIONS(MAIN);

STMT LEVEL NEST

                        END SMALL;
```

```
INITIAL CONFIGURATION

Z  :$ A-B-A B C-C A-B-A B C-C B-A-B A $
     A B
Y  :010
     C D
X  :010
     E F
S  :0
      S
SUM:11110
     I   N

Z  :$ D A-B-A B C A-B-A B C-C B-A-B A $
     A B
Y  :010
     C D
X  :010
     E F
S  :0
      S
SUM:11110
     I   N

Z  :$ C D A-B-A C A-B-A D C-C B-A-B A $
     A B
Y  :010
     C D
X  :010
     E F
S  :0
      S
SUM:11110
     I   N

Z  :$ D-C D A-B-C A-B-A B C-C B-A-B A $
     A B
Y  :010
     C D
X  :010
     E F
S  :0
      S
SUM:11110
     I   N

Z  :$ C-D-C D A-C A-B-A B C-C D-A-B A $
     A B
Y  :010
     C D
X  :010
     E F
S  :0
      S
SUM:11110
     I   N

Z  :$ B C-D-C D C A-B-A B C-C B-A-B A $
     A B
Y  :010
```

```
     C D
X   :010
     E F
S   :0
      S
SUM:11110
     I   N

Z   :$ A B C-D-C & A-B-A B C-C B-A-B A $
     A B
Y   :010
     C D
X   :010
     E F
S   :0
      S
SUM:11110
     I   N

Z   :$ B-A B C-D-& A-B-A B C-C B-A-B A $
     A B
Y   :010
     C D
X   :010
     E F
S   :0
      S
SUM:11110
     I   N

Z   :$ D-C-D C B-& A-B-A B C-C B-A-B A $
     A B
Y   :010
     C D
X   :010
     E F
S   :0
      S
SUM:11110
     I   N

Z   :$ A D-C-D C & A-B-A B C-C B-A-B A $
     A B
Y   :010
     C D
X   :010
     E F
S   :0
      S
SUM:11110
     I   N

Z   :$ B A D-C-D & A-B-A B C-C B-A-B A $
     A B
Y   :010
     C D
X   :010
     E F
S   :0
      S
```

```
SUM:11110
    I   N

Z  :$ A-B A B-C-C A-B-A B C-C B-A-B A $
    A B
Y  :010
    C D
X  :010
    E F
S  :0
     S
SUM:11110
    I   N

Z  :$ B-A-B A B-C A-B-A B C-C B-A-B A $
    A B
Y  :010
    C D
X  :010
    E F
S  :0
     S
SUM:11110
    I   N

Z  :$ C B-A-B A C A-B-A B C-C B-A-B A $
    A B
Y  :010
    C D
X  :010
    E F
S  :0
     S
SUM:11110
    I   N

Z  :$ D C B-A-B C A-B-A B C-C B-A-B A $
    A B
Y  :010
    C D
X  :010
    E F
S  :0
     S
SUM:11110
    I   N

Z  :$ C-D C B-A-C A-B-A B C-C B-A-B A $
    A B
Y  :010
    C D
X  :010
    E F
S  :0
     S
SUM:11110
    I   N

Z  :$ A A-C A-B-A B C-C B-A-B A $
    A B
```

```
Y   :010
     C D
X   :010
     E F
S   :0
      S
SUM:10
     IN

Z   :$ A-A & A-B-A B C-C B-A-B A $
     A B
Y   :010
     C D
X   :010
     E F
S   :0
      S
SUM:10
     IN

Z   :$ B B-& A-B-A B C-C B-A-B A $
     A B
Y   :010
     C D
X   :010
     E F
S   :0
      S
SUM:10
     IN

Z   :$ B-B & A-B-A B C-C B-A-B A $
     A B
Y   :010
     C D
X   :010
     E F
S   :0
      S
SUM:10
     IN

Z   :$ C C-& A-B-A B C-C B-A-B A $
     A B
Y   :010
     C D
X   :010
     E F
S   :0
      S
SUM:10
     IN

Z   :$ C-C & A-B-A B C-C B-A-B A $
     A B
Y   :010
     C D
X   :010
     E F
S   :0
```

```
      S
SUM:10
      IN

Z   :$ 0 0-0 & A-B-A B C-C B-A-B A $
      A B
Y   :010
      C D
X   :010
      E F
S   :0
      S
SUM:10
      IN

Z   :$ 0-0 & A-B-A B C-C B-A-B A $
      A B
Y   :010
      C D
X   :010
      E F
S   :0
      S
SUM:10
      IN
```

```
ALL PATTERNS ARE PRE-PROCESSED:

Z  :$ A-B-A B C-C & A-B-A B C-C B-A-B A $
     A             B
Y  :0101010101010
     C             D
X  :0101010101010
               E F
S  :0
      S
SUM:1111111110
          I   N

Z  :$ D A-B-A B & A-B-A B C-C B-A-B A $
     A             B
Y  :0101010101010
     C             D
X  :0101010101010
               E F
S  :0
      S
SUM:1111111110
          I   N

Z  :$ C D A-B-A & A-B-A B C-C B-A-B A $
     A             B
Y  :0101010101010
     C             D
X  :0101010101010
               E F
S  :0
      S
SUM:1111111110
          I   N

Z  :$ D-C D A-B-& A-B-A B C-C B-A-B A $
     A             B
Y  :0101010101010
     C             D
X  :0101010101010
               E F
S  :0
      S
SUM:1111111110
          I   N

Z  :$ C-D-C D A-& A-B-A B C-C B-A-B A $
     A             B
Y  :0101010101010
     C             D
X  :0101010101010
               E F
S  :0
      S
SUM:1111111110
          I   N

Z  :$ B C-D-C D & A-B-A B C-C B-A-B A $
     A             B
Y  :0101010101010
```

```
    C           D
X  :0101010101010
              E F
S  :0
     S
SUM:1111111110
         I   N

Z  :$ A B C-D-C & A-B-A B C-C B-A-B A $
    A           B
Y  :0101010101010
    C           D
X  :0101010101010
              E F
S  :0
     S
SUM:1111111110
         I   N

Z  :$ B-A B C-D-C & A-B-A B C-C B-A-B A $
    A           B
Y  :0101010101010
    C           D
X  :0101010101010
              E F
S  :0
     S
SUM:1111111110
         I   N

Z  :$ D-C-D C B-& A-B-A B C-C B-A-B A $
    A           B
Y  :0101010101010
    C           D
X  :0101010101010
              E F
S  :0
     S
SUM:1111111110
         I   N

Z  :$ A D-C-D C & A-B-A B C-C B-A-B A $
    A           B
Y  :0101010101010
    C           D
X  :0101010101010
              E F
S  :0
     S
SUM:1111111110
         I   N

Z  :$ B A D-C-D & A-B-A B C-C B-A-B A $
    A           B
Y  :0101010101010
    C           D
X  :0101010101010
              E F
S  :0
     S
```

```
SUM:1111111110
          I   N

Z   :$ A-B A D-C-E A-B-A B C-C B-A-B A $
     A           B
Y   :0101010101010
     C           D
X   :0101010101010
               E F
S   :0
      S
SUM:1111111110
          I   N

Z   :$ B-A-B A D-E A-B-A B C-C B-A-B A $
     A           B
Y   :0101010101010
     C           D
X   :0101010101010
               E F
S   :0
      S
SUM:1111111110
          I   N

Z   :$ C B-A-B A E A-B-A B C-C B-A-B A $
     A           B
Y   :0101010101010
     C           D
X   :0101010101010
               E F
S   :0
      S
SUM:1111111110
          I   N

Z   :$ D C B-A-B E A-B-A B C-C B-A-B A $
     A           B
Y   :0101010101010
     C           D
X   :0101010101010
               E F
S   :0
      S
SUM:1111111110
          I   N

Z   :$ C-D C B-A-E A-B-A B C-C B-A-B A $
     A           B
Y   :0101010101010
     C           D
X   :0101010101010
               E F
S   :0
      S
SUM:1111111110
          I   N

Z   :$ A A-E A-B-A B C-C B-A-B A $
     A     B
```

```
Y   :0101010
     C     D
X   :0101010
         E F
S   :0
      S
SUM:1110
       IN

Z   :$ A-A & A-B-A B C-C B-A-B A $
     A     B
Y   :0101010
     C     D
X   :0101010
         E F
S   :0
      S
SUM:1110
       IN

Z   :$ B B-& A-B-A B C-C B-A-B A $
     A     B
Y   :0101010
     C     D
X   :0101010
         E F
S   :0
      S
SUM:1110
       IN

Z   :$ B-B & A-B-A B C-C B-A-B A $
     A     B
Y   :0101010
     C     D
X   :0101010
         E F
S   :0
      S
SUM:1110
       IN

Z   :$ C C-& A-B-A B C-C B-A-B A $
     A     B
Y   :0101010
     C     D
X   :0101010
         E F
S   :0
      S
SUM:1110
       IN

Z   :$ C-C & A-B-A B C-C B-A-B A $
     A     B
Y   :0101010
     C     D
X   :0101010
         E F
S   :0
```

```
     S
SUM:1110
      IN

Z   :$ D D-& A-B-A B C-C B-A-B A $
     A     B
Y   :0101010
     C     D
X   :0101010
        E F
S   :0
     S
SUM:1110
      IN

Z   :$ D-D & A-B-A B C-C B-A-B A $
     A     B
Y   :0101010
     C     D
X   :0101010
        E F
S   :0
     S
SUM:1110
      IN
```

TEXT IS NOW TO BE PROCESSED

```
AFTER MATCHING

Z  :$ A-B-A B C-E D-C-D C B-A-B A $
    A           B
Y  :0101010101010
    C           D
X  :0101010101010
              E F
S  :0
     S
SUM:1111111110
          I   N
```

```
AFTER MATCHING

Z  :$ D-C-D C B-C A-B-A A-B A $
    A           B
Y  :0101010101010
    C           D
X  :0101010101010
              E F
S  :0
     S
SUM:1111111110
         I   N
```

```
AFTER MATCHING

Z   :$ A A-E A-B-B A $
     A     B
Y   :0101010
     C     D
X   :0101010
       E F
S   :0
      S
SUM:1110
      IN
```

```
AFTER MATCHING

Z   :$ B-B & A-A $
     A     B
Y   :01010101
     C     D
X   :01010101
         E F
S   :0
      S
SUM:1110
       IN
```

```
AFTER MATCHING

Z   :$ A-A & $
     A      B
Y   :01010101
     C      D
X   :01010101
         E F
S   :0
      S
SUM:1110
       IN


SUCCESS
```

```
INITIAL CONFIGURATION

Z  :$ A-B-A B C-C D A-B-A B C D A-B-A $
    A B
Y  :010
    C D
X  :010
    E F
S  :0
     S
SUM:11110
    I   N

Z  :$ B A-B-A B C D A-B-A B C D A-B-A $
    A B
Y  :010
    C D
X  :010
    E F
S  :0
     S
SUM:11110
    I   N

Z  :$ C D A-B-A B C D A-B-A B C D A-B-A $
    A B
Y  :010
    C D
X  :010
    E F
S  :0
     S
SUM:11110
    I   N

Z  :$ D-C D A-B-C D A-B-A B C D A-B-A $
    A B
Y  :010
    C D
X  :010
    E F
S  :0
     S
SUM:11110
    I   N

Z  :$ C-D-C D A-C D A-B-A B C D A-B-A $
    A B
Y  :010
    C D
X  :010
    E F
S  :0
     S
SUM:11110
    I   N

Z  :$ B C-D-C D C D A-B-A B C D A-B-A $
    A B
Y  :010
```

```
     C D
X   :0l0
     E F
S   :0
      S
SUM:lllll0
     I   N

Z   :$ A B C-D-C & D A-B-A B C D A-B-A $
     A B
Y   :0l0
     C D
X   :0l0
     E F
S   :0
      S
SUM:lllll0
     I   N

Z   :$ B-A B C-D-& D A-B-A B C D A-B-A $
     A B
Y   :0l0
     C D
X   :0l0
     E F
S   :0
      S
SUM:lllll0
     I   N

Z   :$ D-C-D C B-& D A-B-A B C D A-B-A $
     A B
Y   :0l0
     C D
X   :0l0
     E F
S   :0
      S
SUM:lllll0
     I   N

Z   :$ A D-C-D C & D A-B-A B C D A-B-A $
     A B
Y   :0l0
     C D
X   :0l0
     E F
S   :0
      S
SUM:lllll0
     I   N

Z   :$ B A D-C-D & D A-B-A B C D A-B-A $
     A B
Y   :0l0
     C D
X   :0l0
     E F
S   :0
      S
```

```
SUM:11110
    I   N

Z  :$ A-B A B-C-C D A-B-A B C D A-B-A $
    A B
Y  :010
    C D
X  :010
    E F
S  :0
     S
SUM:11110
    I   N

Z  :$ B-A-B A B-C D A-B-A B C D A-B-A $
    A B
Y  :010
    C D
X  :010
    E F
S  :0
     S
SUM:11110
    I   N

Z  :$ C B-A-B A C D A-B-A B C D A-B-A $
    A B
Y  :010
    C D
X  :010
    E F
S  :0
     S
SUM:11110
    I   N

Z  :$ D C B-A-B C D A-B-A B C D A-B-A $
    A B
Y  :010
    C D
X  :010
    E F
S  :0
     S
SUM:11110
    I   N

Z  :$ C-D C B-A-C D A-B-A B C D A-B-A $
    A B
Y  :010
    C D
X  :010
    E F
S  :0
     S
SUM:11110
    I   N

Z  :$ A A-C D A-B-A B C D A-B-A $
    A B
```

```
Y   :010
     C D
X   :010
     E F
S   :0
      S
SUM:10
     IN

Z   :$ A-A & D A-B-A B C D A-B-A $
     A B
Y   :010
     C D
X   :010
     E F
S   :0
      S
SUM:10
     IN

Z   :$ B B-& D A-B-A B C D A-B-A $
     A B
Y   :010
     C D
X   :010
     E F
S   :0
      S
SUM:10
     IN

Z   :$ B-B & D A-B-A B C D A-B-A $
     A B
Y   :010
     C D
X   :010
     E F
S   :0
      S
SUM:10
     IN

Z   :$ C C-& D A-B-A B C D A-B-A $
     A B
Y   :010
     C D
X   :010
     E F
S   :0
      S
SUM:10
     IN

Z   :$ C-C & D A-B-A B C D A-B-A $
     A B
Y   :010
     C D
X   :010
     E F
S   :0
```

```
      S
SUM:10
      IN

Z   :$ 0 0-0 0 A-B-A B C D A-B-A $
      A B
Y   :010
      C D
X   :010
      E F
S   :0
      S
SUM:10
      IN

Z   :$ 0-0 0 0 A-B-A B C D A-B-A $
      A 0
Y   :010
      C 0
X   :010
      E F
S   :0
      S
SUM:10
      IN
```

```
ALL PATTERNS ARE PRE-PROCESSED:

Z  :$ A-B-A B C-C D A-B-A B C D A-B-A $
    A           B
Y  :0101010101010
    C           D
X  :0101010101010
              E F
S  :0
     S
SUM:1111111110
          I   N

Z  :$ D A-B-A B C D A-B-A B C D A-B-A $
    A           B
Y  :0101010101010
    C           D
X  :0101010101010
              E F
S  :0
     S
SUM:1111111110
          I   N

Z  :$ C D A-B-A C D A-B-A B C D A-B-A $
    A           B
Y  :0101010101010
    C           D
X  :0101010101010
              E F
S  :0
     S
SUM:1111111110
          I   N

Z  :$ D-C D A-B-C D A-B-A B C D A-B-A $
    A           B
Y  :0101010101010
    C           D
X  :0101010101010
              E F
S  :0
     S
SUM:1111111110
          I   N

Z  :$ C-D-C D A-C D A-B-A B C D A-B-A $
    A           B
Y  :0101010101010
    C           D
X  :0101010101010
              E F
S  :0
     S
SUM:1111111110
          I   N

Z  :$ B C-D-C D C D A-B-A B C D A-B-A $
    A           B
Y  :0101010101010
```

```
     C            D
X  :0101010101010
                E F
S  :0
      S
SUM:1111111110
          I   N

Z  :$ A B C-D-C & D A-B-A B C D A-B-A $
     A            B
Y  :0101010101010
     C            D
X  :0101010101010
                E F
S  :0
      S
SUM:1111111110
          I   N

Z  :$ B-A B C-D-C & D A-B-A B C D A-B-A $
     A            B
Y  :0101010101010
     C            D
X  :0101010101010
                E F
S  :0
      S
SUM:1111111110
          I   N

Z  :$ D-C-D C B-A & D A-B-A B C D A-B-A $
     A            B
Y  :0101010101010
     C            D
X  :0101010101010
                E F
S  :0
      S
SUM:1111111110
          I   N

Z  :$ A D-C-D C & D A-B-A B C D A-B-A $
     A            B
Y  :0101010101010
     C            D
X  :0101010101010
                E F
S  :0
      S
SUM:1111111110
          I   N

Z  :$ B A D-C-D & D A-B-A B C D A-B-A $
     A            B
Y  :0101010101010
     C            D
X  :0101010101010
                E F
S  :0
      S
```

```
SUM:1111111110
          I   N

Z  :$ A-B A B-C-& D A-B-A B C D A-B-A $
    A             B
Y  :0101010101010
    C             D
X  :0101010101010
              E F
S  :0
     S
SUM:1111111110
          I   N

Z  :$ B-A-B A D-& D A-B-A B C D A-B-A $
    A             B
Y  :01̄0101010101O
    C             D
X  :0101010101010
              E F
S  :0
     S
SUM:1111111110
          I   N

Z  :$ C B-A-B A & D A-B-A B C D A-B-A $
    A             B
Y  :0101010101010
    C             D
X  :0101010101010
              E F
S  :0
     S
SUM:1111111110
          I   N

Z  :$ D C B-A-B & D A-B-A B C D A-B-A $
    A             B
Y  :0101010101010
    C             D
X  :0101010101010
              E F
S  :0
     S
SUM:1111111110
          I   N

Z  :$ C-D C B-A-& D A-B-A B C D A-B-A $
    A             B
Y  :0101010101010
    C             D
X  :0101010101010
              E F
S  :0
     S
SUM:1111111110
          I   N

Z  :$ A A-& D A-B-A B C D A-B-A $
    A     B
```

```
Y  :0101010
    C     D
X  :0101010
       E F
S  :0
     S
SUM:1110
      IN

Z  :$ A-A C D A-B-A B C D A-B-A $
    A     B
Y  :0101010
    C     D
X  :0101010
       E F
S  :0
     S
SUM:1110
      IN

Z  :$ B B-C D A-B-A B C D A-B-A $
    A     B
Y  :0101010
    C     D
X  :0101010
       E F
S  :0
     S
SUM:1110
      IN

Z  :$ B-B C D A-B-A B C D A-B-A $
    A     B
Y  :0101010
    C     D
X  :0101010
       E F
S  :0
     S
SUM:1110
      IN

Z  :$ C C-C D A-B-A B C D A-B-A $
    A     B
Y  :0101010
    C     D
X  :0101010
       E F
S  :0
     S
SUM:1110
      IN

Z  :$ C-C C D A-B-A B C D A-B-A $
    A     B
Y  :0101010
    C     D
X  :0101010
       E F
S  :0
```

```
     S
SUM:1110
      IN

Z   :$ 0 0-0 0 A-B-A B C 0 A-B-A $
     A     B
Y   :0101010
     C     D
X   :0101010
        E F
S   :0
     S
SUM:1110
      IN

Z   :$ 0-0 0 0 A-B-A B C 0 A-B-A $
     A     B
Y   :0101010
     C     D
X   :0101010
        E F
S   :0
     S
SUM:1110
      IN
```

TEXT IS NOW TO Be PROCESSED

```
AFTER MATCHING

Z   :$ D A-B-A B G C-D C C D A-B-A $
     A           B
Y   :0101010101010
     C           D
X   :0101010101010
               E F
S   :0
      S
SUM:1111111110
          I   N
```

```
AFTER MATCHING

Z   :$ C D A-B-A C C-D C D C B-$
      A           B
Y   :0101010101010
      C           D
X   :0101010101010
               E F
S   :0
       S
SUM:1111111110
           I   H


FAILURE
```

Bibliography

[1] Aho, A., Hopcroft, J., Ullman, J., The Design and Analysis of Computer Algorithms, Addison-Wesley, New York, 1974.

[2] Anshel, M., Stebe, P., The Solvability of the Conjugacy Problem for Certain HNN Groups, Bull. Amer. Math. Soc., 80(1974), p. 266-270.

[3] Britton, J. L., The Word Problem for Groups, Proc. London Math. Soc., 8, No. 32 (third series), pp. 493-506, 1958.

[4] Boone, W. W., Certain Simple Unsolvable Problems in Group Theory, Indig. Math., Vols. 16, 17, 19, 1955.

[5] Dehn, M., Uber Unendliche Diskontinuierliche Gruppen, Math. Ann., Vol. 71, p. 116-114, 1911.

[6] Fischer, M. J., Paterson, M. S., String - Matching and Other Products, SIAM-AMS Proceedings, Vol. 7, 1974, pp. 113-125.

[7] Fischer, P. C., Meyer, A. R., A. L. Rosenberg, On Real-Time Simulation of Multihead Tape Units, J.A.C.M., 19(1972), pp. 590-607.

[8] Greendlinger, M., Dehn's Algorithm for the Word Problem, Comm. Pure and Appl. Math., Vol 13, 1960a.

[9] Knuth, D. E., Morris, J. H., Pratt, V. R., Fast Pattern Matching in Strings, SIAM Journal of Computing, Vol. 6, No. 2, June 1977, pp. 323-350.

[10] Knuth, D. E., Fundamental Algorithms - The Art of Computer Programming, Vol. 1., Addison-Wesley, Reading, Mass., 1969.

[11] Magnus, W., Das Identitas Problem fur Gruppen mit einer definierenden Relation, Math. Ann., Vol. 106, p. 295-307, 1932.

[12] Magnus, W., Karrass, A., Solitar, D., Combinatorial Group Theory: Presentations of Groups in Terms of Generators and Relations, John Wiley & Sons, New York, 1966.

[13] Novikov, P. S., On the Algorithmic Unsolvability of the Word Problem in Group Theory, Trudy Mat. Inst. Im Steklov, No. 44, pp. 143, Izdat. Akad. Nauk. SSSR, Moscow (Russian), MR17, 706, 1955.

[14] Novikov, P. S., On the Algorithmic Unsolvability of the Word Problem in Group Theory, English translation in "Russian Translations", Series 2, Volume 9, 1958. Edited by the American Math. Soc.

[15] Rabin, M. O., Recursive Unsolvability of Group Theoretic Problems, Annals of Math., Vol 67, p. 172-174, 1958.

[16] Rotman, J. J., The Theory of Groups - An Introduction (Second Edition), Allyn and Bacon, Boston, Mass., 1973.

[17] Rivest, R. L., On the Worst-Case Behavior of String-Searching Algorithms, SIAM Journal of Computing, Vol. 6, No. 4, December 1977, pp. 669-674.

[18] Stockmeyer, L., The Complexity of Decision Problems in Automata, Theory and Logic, Project MAC Technical Report 133, 1974.