Order Number 9224811

Studies in computational group theory

Finz, Harold Richard, Ph.D.

City University of New York, 1992

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

A

Studies in Computational Group Theory

by

Harold Finz

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

1992

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

4/30/92
Date

Michael Anshel
Chair of Examining Committee

5/1/92
Date

Martin Mostowski
Executive Officer

M. Anshel

B. Randol

A. Vasquez
Supervisory Committee

THE CITY UNIVERSITY OF NEW YORK

Abstract

Studies in Computational Group Theory

by

Harold Finz

Adviser: Professor Michael Anshel

We investigate connections among Gröbner bases of polynomial ideals, formal languages, and the word and conjugacy problems for a class of HNN groups. We focus on computational aspects.

Acknowledgements

I wish to acknowledge and thank those who helped to make this work possible. First and foremost, I want to thank Professor Michael Anshel, for his excellent guidance, his constant encouragement, and especially, for our innumerable conversations over the years. I wish also to thank Professor Randol and Professor Vasquez for their valuable insights and suggestions. I would also like to express my thanks to my colleagues at Citibank, especially Janet Weinglass, for their moral support over the past year, and to all my former colleagues at the City College and the Graduate Center.

Finally, I wish to thank my parents, and Debe Bednarchak, for all the encouragement and support they have given me.

Contents

Chapter 1

Gröbner Bases of Polynomial Ideals

1.1 An Overview.

The concept of a Gröbner basis for a polynomial ideal, as well as a method for constructing such a basis, was introduced in 1965 by Bruno Buchberger [13]. Here, we develop the theory of Gröbner bases for a polynomial ideal with coefficients in a field. We prove the equivalence of several characterizations of a Gröbner basis, and present a constructive procedure for obtaining such a basis from a given set of generators for the ideal. We also define a reduced Gröbner basis, and prove its existence and uniqueness. We then give some examples of the use of Gröbner basis methods to solve some interesting problems, and conclude with some brief remarks concerning the complexity of these methods.

The point of view taken here is that sets of polynomials can be formally viewed as defining term rewriting systems. For this reason, most of the proofs are combinatorial rather than purely algebraic in nature. This is in

part justified by the fact that one of the goals of this research is to gain insight into the underlying mechanisms in order to develop a usable package for working with Gröbner bases.

1.2 Background.

1.2.1 Term Orderings.

Let $\mathcal{F}$ be a field, let $X = \{x_1, x_2, \ldots, x_n\}$ be a finite set of indeterminates, and let $\mathcal{F}[X]$ be the ring of polynomials in the x_i. A *monomial* or *powerproduct* is an element of $\mathcal{F}[X]$ of the form $\prod_{i=1}^{n} x_i^{k_i}$, where the k_i are non–negative integers. A term is the product of a monomial and an element of $\mathcal{F}$. A polynomial written as a sum of distinct terms (i.e., no two terms differ only in their coefficients) is in *simplified sum-of-products form (SSPF).* All polynomials will be assumed to be in SSPF.

The *degree* of the monomial $\prod_{i=1}^{n} x_i^{k_i}$ is $\sum_{i=1}^{n} k_i$. The *total degree* of a polynomial f, denoted by $deg(f)$, is the maximum of the degrees of its monomials. $deg_{x_i}(f)$ is the highest power of x_i among all the monomials of f; i.e., the usual degree when viewing f as a polynomial in x_i, with coefficients in $\mathcal{F}[X \setminus \{x_i\}]$.

We give examples of two specific total orderings on monomials. The *Lexicographic* ordering on monomials m_1, m_2 is defined by:
$m_1 \ll m_2$ if there exists t (depending on m_1, m_2), $0 < t < n$, with $deg_{x_t}(m_1) < deg_{x_t}(m_2)$, and for $0 < i < t, deg_{x_i}(m_1) = deg_{x_i}(m_2)$. For example, in the bivariate case with $y = x_1, x = x_2$:

$$1 \ll y \ll y^2 \ll y^3 \ll \cdots \ll x \ll xy \ll xy^2 \ll \cdots \ll x^2 \ll x^2y \ll \cdots$$

The *Total degree* ordering is defined by $m_1 \ll_T m_2$ if either $deg(m_1) < deg(m_2)$, or else, the total degrees are the same, and $m_1 \ll m_2$. In the bivariate case with $y = x_1,\ x = x_2$:

$$1 \ll_T y \ll_T x \ll_T y^2 \ll_T yx \ll_T x^2 \ll_T y^3 \ll_T y^2x \ll_T \cdots$$

Following B. Buchberger, we will call an ordering $<$ on monomials *admissible* provided: $<$ is a linear ordering, $1 < t$ for all monomials $t \neq 1$, and for all monomials $s, t, u,\ s < t \Rightarrow s \cdot u < t \cdot u$.

The reflexive closure of either the lexicographic or the total degree orderings defined above are admissible well–orderings on monomials.

1.2.2 Reductions.

Let $<$ be an admissible ordering on monomials, and let $f \in \mathcal{F}[X]$. The *headterm* of f, $h(f)$ is the term of f that is maximal with respect to $<$, and the *reductum* of f, denoted by $r(f)$, is defined to be $f - h(f)$. We note that $h(f)$ is uniquely specified since $<$ is a total ordering on monomials, f is in SSPF, and f is a finite sum of terms.

If $f \neq 0$, f induces a relation $\rightarrow_f^*$, *"reduction by f"* on $\mathcal{F}[X]$ as follows:

Definition: $P_1 \rightarrow_f P_2$ if and only if $h(f)$ divides some term T of P_1, and

$$P_2 = P_1 - f \cdot \frac{T}{h(f)}.$$

We say that f *reduces* P_1 to P_2. Equivalently, $P_1 \rightarrow_f P_2$ if and only if $P_1 = a \cdot h(f) + b$, where a is a term, and $P_2 = -a \cdot r(f) + b$.

Example: Let

$$P_1 \ = \ 4x^3y^3 + x^2y^2,$$

$$f = x^2y - z + 1.$$

Then

$$\begin{aligned} P_1 &\rightarrow_f 4x^3y^3 + y(z-1) \\ &= 4x^3y^3 + yz - y. \end{aligned}$$

The notation $\rightarrow_f$ is intentionally suggestive, in the sense that $\rightarrow_f$ may be viewed as defining a partial, nondeterministic rewriting procedure on the set of monomials. In particular, given polynomials P and f, it is a simple matter to find a polynomial Q such that $P \rightarrow_f Q$ if such a Q exists, or to determine that no such Q exists. We may eliminate the nondeterminism quite easily, and in fact, any computer program will do so implicitly. This point will be pursued later.

If B is a finite subset of $\mathcal{F}[X]$, $P_1 \rightarrow_B P_2$ if and only if for some $f \in B$, $P_1 \rightarrow_f P_2$. We will write $\rightarrow$ instead of $\rightarrow_B$ if the set B is clear from context. Denote the reflexive transitive closure of $\rightarrow$ by $\rightarrow^*$ and the reflexive symmetric transitive closure of $\rightarrow$ by $\leftrightarrow^*$. As a notational device, if $a \rightarrow b$ (resp. $a \rightarrow^* b$), then we may write $b \leftarrow a$ (resp. $b \ ^*\!\leftarrow a$).

The *ideal* generated by the set $B = \{f_1, \ldots, f_s\}$ is the set

$$\left\{ \sum_{i=1}^{s} a_i f_i : a_i \in \mathcal{F}[X] \right\},$$

and is denoted by Ideal(B).

Proposition 1 *If $P_1 \rightarrow_B^* P_2$ then $P_1 - P_2 \in$ ideal(B).*

Proof: We use induction on the length of the reduction sequence. If the length $= 0$, $P_1 - P_2 = 0 \in$ ideal(B). Suppose $P_1 \rightarrow_f Q$. Then $P_1 = a \cdot h(f) +$

$b, Q = -a \cdot r(f) + b$, and $P_1 - Q = a \cdot (h(f) + r(f)) = a \cdot f$. By induction hypothesis, $Q - P_2 = R \in$ ideal(B). Therefore, $P_1 - Q + Q - P_2 = P_1 - P_2 \in$ ideal(B).

In general, the converse of Proposition 1 is false. For example, assume the ordering $x \gg y \gg z$ on the indeterminates, with respect to the total degree ordering, and let B $= \{f_1 = x - y, f_2 = x - z\}$. Then $y - z \in$ ideal(B), but $y - z$ cannot be reduced by either f_1 or f_2.

Proposition 2 *Suppose $<$ is an admissible ordering, and a and b are terms. If a is a multiple of b, and $a \neq b$, then $b < a$. Moreover, if $a \rightarrow_f p$, then $a >$ any term of p.*

Proof: Suppose $a = t \cdot b$, and $t \neq 1$. Then $1 < t$, and by admissibility, $b = 1 \cdot b < t \cdot b = a$. If $a \rightarrow_f p$, then $a = t \cdot h(f)$, for some term t, and $p = -t \cdot r(f)$. Let u be any term of $r(f)$. Now $h(f) > u$, and so $a = t \cdot h(f) > t \cdot u \geq t$.

Definition: A relation is *Noetherian* if there is no infinite sequence

$$P_1 \rightarrow P_2 \rightarrow \ldots$$

in which for all i, $P_i \neq P_{i+1}$.

Proposition 3 *Let $<$ be an admissible well-ordering on monomials, let $B = \{f_1, \ldots, f_s\}$ be a finite subset of $\mathcal{F}[X]$, and let $\rightarrow$ be a relation on $\mathcal{F}[X]$ as defined above. Then $\rightarrow$ is Noetherian.*

Proof: Suppose there is an infinite sequence of reductions $P_1 \rightarrow P_2 \rightarrow \cdots$, with $P_i = P_{i+1}$, for all i. Construct a tree as follows: For the root R, take any

term that is > the headterm of P_1, and let the descendants of R be the terms of P_1. For all i, the reduction $\rightarrow_{f_i}$ replaces some term a_i of P_i by a finite number of terms; the descendants of a_i are the terms of $-a_i \cdot r(f_i)/h(f_i)$. Since $<$ is admissible, by Proposition 2, each descendant of a_i is $< a_i$. We have, therefore, an infinite tree, in which each node has only a finite number of direct successors. Therefore, there must be an infinite path leading from the root. This path is an infinite decreasing sequence of monomials, which contradicts the fact that $<$ is a well–ordering.

1.3 Gröbner Bases.

1.3.1 Definitions.

Let P, P_1, P_2, Q be elements of $\mathcal{F}[X]$, and let $\rightarrow$ be a relation on $\mathcal{F}[X]$, defined by an admissible ordering $<$ on monomials.

P_1 and P_2 are *joinable* if there exists Q such that $P_1 \rightarrow^* Q \ ^*\!\leftarrow P_2$ and we say that P_1 and P_2 are *joinable to* Q. $\rightarrow$ is *confluent* if $P_1 \ ^*\!\leftarrow Q \rightarrow^* P_2$ implies P_1 and P_2 are joinable. $\rightarrow$ is *locally confluent* if $P_1 \leftarrow Q \rightarrow P_2$ implies P_1 and P_2 are joinable. $\rightarrow$ has the *Church-Rosser Property* if $P_1 \leftrightarrow^* P_2$ implies P_1 and P_2 are joinable. P is in *normal form* (or is *irreducible*) if $P \rightarrow Q$ implies $P = Q$. The *S-polynomial* $S(P_1, P_2)$ is defined by

$$S(P_1, P_2) = n_1 \cdot P_1 - n_2 \cdot P_2$$

where

$$T = l.c.m.(h(P_1), h(P_2)) \text{ and } n_i = \frac{T}{h(P_i)} \quad (\text{i} = 1, 2).$$

($l.c.m.(x, y)$ denotes the *least common multiple* of x and y.)

Definition: A *Gröbner Basis* for ideal(F) is a finite set F of polynomials in $\mathcal{F}[x_1, \ldots, x_n]$ with the property that $\rightarrow^*$ has the Church-Rosser property.

A Gröbner basis for Ideal(F) serves, not to remove, but to compensate for, the nondeterministic aspects of $\rightarrow_f$. We will make a precise statement in the following theorem. Additionally, pursuing the procedural interpretation of $\rightarrow_f$, confluence is a quite important property, for this corresponds to a natural requirement of any algebraic "simplification" system, that even if there is a choice to be made at some point in the simplification process, this choice should not affect the final outcome. Of course, the existence of a "final" state is a consequence of the Noetherian property of $\rightarrow_f$.

1.3.2 Alternate Characterizations of a Gröbner Basis.

We now demonstrate the fundamental result that bridges the purely computational notions of reduction and S-polynomial and the more abstract properties of term rewriting systems. This result will then serve as the basis for the constructive portions of the theory.

Theorem 1 *Let $F = \{f_1, \ldots, f_s\}$. Then the following are equivalent:*

1. *F is a Gröbner basis for $I = ideal(F)$.*
2. *$\rightarrow^*$ is confluent.*
3. *For all f,g in F, $S(f,g) \rightarrow_F^* 0$.*
4. *If $P_1 \ ^*\!\leftarrow P \rightarrow^* P_2$ and P_1, P_2 are in normal form with respect to $\rightarrow$, then $P_1 = P_2$.*

Proof: We will show that (1),(3) and (4) are each equivalent to (2).

(1) $\Rightarrow$ (2) is trivial.

(2) $\Rightarrow$ (1): We use induction on the length n of a reduction sequence $a \leftrightarrow^* b$. If $n = 0$, then $a = b$, and so a and b are trivially joinable. Assume that either $a \rightarrow x \leftrightarrow^* b$ or $a \leftarrow x \leftrightarrow^* b$. In each case, by induction hypothesis, there exists u joining x and b. In the first case, a and b are clearly joinable; $a \rightarrow x \rightarrow^* u \ ^*\leftarrow b$. In the second case, since $a \leftarrow x \rightarrow^* u$, and $\rightarrow$ is confluent, there exists v joining a and u. Therefore, $a \rightarrow^* v \ ^*\leftarrow u \ ^*\leftarrow b$, and so a and b are joinable.

(4) $\Rightarrow$ (2): Assume $b \ ^*\leftarrow a \rightarrow^* c$. Since $\rightarrow$ is Noetherian (Proposition 3), we can extend these reductions to irreducibles b' and c';

$$b' \ ^*\leftarrow b \ ^*\leftarrow a \rightarrow^* c \rightarrow^* c'.$$

By (4), a can't reduce to distinct irreducibles, so $b' = c'$, and therefore b and c are joinable.

(2) $\Rightarrow$ (4): Suppose $P_1 \ ^*\leftarrow f \rightarrow^* P_2$ where P_1, P_2 are irreducible. P_1 and P_2 are joinable, say to Q. Since P_1 and P_2 are irreducible, $P_1 = Q = P_2$.

We restate (2) $\Rightarrow$ (4) as Lemma 1, as we will refer to it later.

Lemma 1 *If $\rightarrow$ is confluent, then $P \in \mathcal{F}[X]$ can't reduce to distinct normal forms.*

In order to prove the equivalence of (3) and (2), we will employ a theorem due to M.H.A. Newman [53]: If $\rightarrow$ is Noetherian, then confluence is equivalent to local confluence. We first prove some lemmas.

Lemma 2 *Let f,g be in $\mathcal{F}[X]$. If $f - g \rightarrow^* 0$ then f and g are joinable.*

Proof: We show this by induction on the length of a reduction

$$f - g = a_0 \to a_1 \to a_2 \to \cdots \to a_N = 0.$$

If $N = 0$, then $f = g$. Suppose the result holds for reductions of length < N. For some $P \in F$, $a_0 \to_P a_1$. P replaces a term that is present in at least one of f and g; suppose it is a term of f. (The other cases are similar.) Then $f \to_P f'$, and $a_1 = f' - g$. By induction hypothesis, f' and g are joinable, so for some Q, $f \to f' \to^* Q \ {}^*\!\leftarrow g$.

Lemma 3 *Suppose $P \in$ ideal(F), and $\to$ is confluent. Then there is a reduction $P \to^* 0$.*

Proof: We prove this by induction on s, the number of terms of P. Write $P = \sum_{i=1}^{s} a_i f_i$. Define $P_0 = 0$, and for $0 < k \leq s$,

$$P_k = \sum_{i=1}^{k} a_i f_i.$$

Trivially, $P_0 \to^* 0$. Suppose there is a reduction $P_{k-1} \to^* 0$. Now,

$$P_k - P_{k-1} = a_k f_k \to_{f_k} 0,$$

and so by Lemma 2, there exists Q_k such that $P_k \to^* Q_k \ {}^*\!\leftarrow P_{k-1}$. Since $\to^*$ is confluent, $0 \ {}^*\!\leftarrow P_{k-1} \to^* Q_k$ implies that 0 and Q_k are joinable; i.e., $Q_k \to^* 0$. Therefore, $P_k \to^* Q_k \to^* 0$.

We may now prove (2) $\Rightarrow$ (3). If $f, g \in F$, then $S(f,g) \in$ ideal(F). By Lemma 3, there is a reduction $S(f,g) \to^* 0$.

Lemma 4 *If, for every pair of polynomals $f_1, f_2 \in F$, the corresponding S-polynomial $S(f_1, f_2) \to^* 0$, then $\to$ is locally confluent.*

Proof: Suppose $P_1 \ {}^*\!\leftarrow P \rightarrow^* P_2$. By Lemma 2, it is sufficient to show that $P_2 - P_1 \rightarrow^* 0$. There are 2 cases to consider: (1) f_1 and f_2 replace the same term of P; (2) f_1 and f_2 replace different terms of P.

Case 1: $P = a \cdot t + b$ where a and t are terms, b is some polynomial, and for some terms n_1, n_2,

$$t = \text{l.c.m.}(h(f_1), h(f_2)) = n_1 h(f_1) = n_2 h(f_2).$$

For $i = 1, 2$,

$$\begin{aligned} a \cdot t &= a \cdot n_i \cdot h(f_i) \\ &\rightarrow -a \cdot n_i \cdot r(f_i) \\ &= a \cdot n_i \cdot (h(f_i) - f_i). \end{aligned}$$

Therefore,

$$\begin{aligned} P_2 - P_1 &= a \cdot (n_2 \cdot h(f_2) - n_1 \cdot h(f_1) - n_2 \cdot f_2 + n_1 \cdot f_1) \\ &= a \cdot S(f_1, f_2) \\ &\rightarrow^* a \cdot 0 \\ &= 0, \end{aligned}$$

by hypothesis, and the admissibililty of $<$.

Case 2: For some terms e_1, e_2 and some polynomial b,

$$\begin{aligned} P &= e_1 h(f_1) + e_2 h(f_2) + b, \\ P &\rightarrow -e_1 r(f_1) + e_2 h(f_2) + b = P_1, \text{and} \\ P &\rightarrow e_1 h(f_1) - e_2 r(f_2) + b = P_2. \end{aligned}$$

Now, $P_2 - P_1 = e_1 f_1 - e_2 f_2$. This can fail to be reducible by at least one of the f_i if and only if $h(f_i)$ has "cancelled". This can occur in one of two

ways: Either $e_1h(f_1) = e_2h(f_2)$, or for some $i \neq j, e_jh(f_j) =$ some term of $e_ir(f_i)$. The first alternative has already been treated as case (1), since P is assumed to be in SSPF. So, assume $e_2h(f_2) = e_1a$, where a is some term of $r(f_1)$ (i.e., $h(f_2)$ is cancelled). Now, $h(f_1) > a$, and since $<$ is admissible, $e_1h(f_1) > e_1a = e_2h(f_2) >$ any term in $e_2r(f_2)$. Therefore, $h(f_1)$ cannot have also cancelled. So at least one of f_1, f_2 must apply to $P_2 - P_1$; say f_1. Then, again using the admissibility of $<$, $P2 - P_1 = e_1f_1 - e_2f_2 \rightarrow -e_2f_2 \rightarrow 0$.

The remaining case (3) $\Rightarrow$ (2) now follows from Newman's theorem and Lemma 4.

1.3.3 Construction of Gröbner Bases.

If I is an ideal in $\mathcal{F}[X]$ given explicitly by a finite set B of generators, then I has a Gröbner basis. Lemma 4 suggests a constructive proof of this. The essential idea is that if f_i and f_j are in B, and $S(f_i, f_j) \rightarrow_B^* a \neq 0$, then $S(f_i, f_j) \rightarrow_{B'}^* 0$, where $B' = B \bigcup \{a\}$. In algorithm GB below, the role of L is to keep track of the pairs of polynomials for which the corresponding S-polynomial has not been shown to reduce to 0.

Algorithm GB:
Input: B := $\{f_1, f_2, \ldots, f_s\}$
Output: a Gröbner basis F for ideal(B).
begin
    F := B;
    L := $\{(f_i, f_j) : 0 < i < j < s+1\}$;
    while (L $\neq \phi$) **do**
    begin
        Remove the first pair (f_i, f_j) from L
        Compute $S(f_i, f_j)$ and reduce this to an irreducible element a.
        if $a \neq 0$ **then**
        begin
            s := s+1;
            f_s := a;
            F := F $\bigcup \{f_s\}$;
            L := L $\bigcup \{(f_i, f_s) : i = 1, \ldots s-1\}$
        end
    end
end

Theorem 2 *Algorithm GB produces a Gröbner basis for ideal(B).*

Proof: We show that this algorithm terminates, and that upon termination, F is a Gröbner basis for ideal(B).

Termination: Let $F_0 = B$, and define F_n to be the value of B after B has been extended for the n^{th} time. Let H_n be the ideal generated by the leading powerproducts (i.e. the headterms without their coefficients) of the elements of F_n. We consider $|F|$ and $|L|$ as the "loop variants", where $|x|$ denotes the cardinality of the set x.

If L is not empty, then by Proposition 3, the reduction of the S-polynomial will produce an irreducible element a after a finite number of steps. There are two possibilities:

(1): $a = 0$, in which case $|L|$ decreases by 1, and $|F|$ is unchanged.

(2): $a \neq 0$: In this case, $|L|$ and $|F|$ both increase. The old value of F is F_n, for some n. Since $S(f_i, f_j) \rightarrow^* a$, with respect to the reductions induced by F_n, and a is irreducible, clearly $h(a)$ is not one of the generators of H_n. Since H_n is generated by powerproducts, we also have that $h(a)$ is not an element of H_n, and therefore H_n is properly contained in H_{n+1}.

By the ascending chain condition on ideals in $\mathcal{F}[X]$, case (2) can only occur finitely many times, and the algorithm terminates when $|L| = 0$.

Correctness: We have 2 loop invariants for the body of the WHILE loop: (1) ideal(B) = ideal(F), and (2) L consists precisely of those pairs of basis elements of F whose S-polynomials have not yet been proven to reduce to 0. (1) holds easily because only S-polynomials are added to the basis, and these are already in the ideal(B). (2) is clearly true before entering the loop. Suppose (f_i, f_j) is the pair of polynomials removed from L. $S(f_i, f_j) \rightarrow^* a$,

where a is irreducible. If $a = 0$, then (2) remains true at the end of the loop. If $a \neq 0$, $S \rightarrow_F^* a \rightarrow_a 0$. Now, F is replaced by $F \bigcup \{a\}$ before finishing the pass through the loop, and so (2) also remains true in this case. The loop is exited only when L is empty. By Lemma 4, F is a Gröbner basis.

1.3.4 Reduced Gröbner Bases.

We have just shown that a Gröbner basis for a polynomial ideal provides a means of constructing a set of unique normal forms for polynomials modulo that ideal. We will show later that a Gröbner basis also may be used to solve a very fundamental question regarding an ideal, that is, the question of membership in that ideal. Since a Gröbner basis for an ideal has such useful properties, it is natural to question whether or not an ideal has a unique Gröbner basis. It is easy to see that the answer is "no", as it will be shown that if B is a Gröbner basis for an ideal I, and $g \in I$, then $B \bigcup \{g\}$ is also a Gröbner basis for I. However, it is possible to define the notion of a *reduced* Gröbner basis, which is uniquely determined by the ideal.

Definition: B is a *reduced* Gröbner basis for $I = \text{ideal}(B)$ if and only if B is a Gröbner basis for I, and $\forall f \in B$, f has leading coefficient $= 1$, and f is in normal form modulo B $\setminus \{f\}$.

We show first that an ideal I possesses a reduced Gröbner basis, and we present an algorithm for its construction. We then show that it is uniquely determined by I.

Lemma 5 *Let B be a Gröbner basis for I, $f \in B$, $g \in B$, and $P \rightarrow_f R$. Suppose $f \rightarrow_g f'$. Define $B' = B \bigcup f' \setminus \{f\}$. Then there exists Q such that $P \rightarrow_{B'}^* Q \ ^*\!\leftarrow R$.*

Proof: If g replaces $h(f)$ then, whenever $\rightarrow_f$ is applied, we can instead apply $\rightarrow_g$. If g replaces some other term of f, then we may apply $\rightarrow_{f'}$ instead of $\rightarrow_f$. For simplicity of notation, we will assume g replaces $h(f)$. We show that there is an *f-free* reduction P $\rightarrow^*$ Q.

Define $P = P_0$, $R = R_0$. We will inductively define sequences $P_i, R_i, i \geq 0$, as follows. By hypothesis, $P_0 \rightarrow^* R_0$. If $P_i \rightarrow^* R_i$ is not f-free, then there exist x_i, y_i such that

$$P_i \rightarrow^* x_i \rightarrow_f y_i \rightarrow^* R_i$$

where $P_i \rightarrow^* x_i$ is f-free, and also, $P_{i+1} \leftarrow_g x_i \rightarrow^* R_i$. Therefore,

$$P_{i+1} \rightarrow^* R_{i+1} \ {}^*\!\leftarrow R_i.$$

Choose i minimal such that $P_i \rightarrow^* R_i$ is f-free. Now, $i > 0$ since $P_0 \rightarrow_f R_0$. By construction,

$$P = P_0 \rightarrow^* x_0 \rightarrow_g P_1 \rightarrow^* x_1 \rightarrow_g P_2 \rightarrow^* \cdots P_i \rightarrow^* R_i$$

is an f-free reduction to R_i, and

$$R = R_0 \rightarrow^* R_1 \rightarrow^* \cdots \rightarrow^* R_i,$$

which proves the lemma.

Proposition 4 *If B is a Gröbner basis for an ideal I, and $g \in I$, then $B \cup \{g\}$ is also a Gröbner basis for I.*

Proof: By Theorem 2, it is enough to show that for any $f \in B$, $S(f,g) \rightarrow_B^* 0$. Now,

$$\begin{aligned} S(f,g) &= n_1 \cdot f - n_2 \cdot g \\ &= n_1 \cdot f - n_2 \cdot (\sum_{i=1}^{n} a_i f_i), \end{aligned}$$

which is in I. By Lemma 3, there is therefore a reduction $S(f,g) \rightarrow_B^* 0$.

Lemma 6 *Let B be a Gröbner basis for I, $f,g \in B$, $f \rightarrow_g f'$, and define $B' = B \cup \{f'\} \setminus \{f\}$, as in Lemma 5. If $P \rightarrow_B^* P'$, with P' irreducible, then $P \rightarrow_{B'}^* P'$.*

Proof: By Proposition 4, $B \cup \{f'\}$ is also a Gröbner basis for I, and $P \rightarrow^* P'$ with respect to this larger basis as well, since by Lemma 1, P can't be reduced to distinct irreducibles. As in Lemma 5, either $\rightarrow_g$ or $\rightarrow_{f'}$ is applicable any time $\rightarrow_f$ is, and again applying Lemma 1, $P \rightarrow_{B'}^* P'$.

Theorem 3 *Suppose B is a Gröbner basis for I, $f \neq g \in B$, and $f \rightarrow_g f'$. Then $B' = (B \setminus \{f\}) \cup \{f'\}$ is also a Gröbner basis for I.*

Proof: If $f \rightarrow f'$, then for some term a, $f' = f - a \cdot g$, so we have easily that ideal(B) = ideal(B'). By Theorem 1, to show that B' is a Gröbner basis, it is enough to show that every S-polynomial reduces by B' to 0. If f_1, f_2 are in B, both different from f, then $S(f_1,f_2) \rightarrow_B^* 0$. By Lemma 6, there is an f-free reduction, so $S(f_1,f_2) \rightarrow_{B'}^* 0$. Now,

$$S(f_1,f') = (f - ag) \cdot n_1 - f_1 \cdot n_2$$

for some monomials n_1, n_2. This is in ideal(B), so $S(f_1,f') \rightarrow_B^* 0$. Again using Lemma 6, this can be replaced by an f-free reduction.

Corollary 1 *If B is a Gröbner basis for the ideal I, then we can construct, from B, a reduced Gröbner basis.*

Proof: Take $f \in B$, reduce f modulo $B \setminus \{f\}$ to f', and replace f by f' in B. By Theorem 3, B remains a Gröbner basis for I after each reduction step.

Repeat this until each f is irreducible modulo $B \setminus \{f\}$, and then normalize by replacing each polynomial

$$\sum_{i=1}^{n} a_i f_i \text{ by } \frac{1}{a_1} \cdot \sum_{i=1}^{s} a_i f_i.$$

To show that this must terminate after a finite number of steps, we note that $|B|$ is finite and constant. Each f_i may only be replaced a finite number of times, since $\rightarrow^*$ is Noetherian.

Theorem 4 *If ideal(B) = ideal(B') and B and B' are reduced Gröbner bases, then B = B'.*

Proof: Let H, H' be the ideals generated by the head-monomials of B, B' respectively. We first show that $H = H'$. Let $a \in H$, $a = h(f)$. Since $f \in B$, and ideal(B) = ideal(B'), $f \rightarrow^*_{B'} 0$. So, there is some $g' \in B'$ such that $h(g')$ divides $h(f)$, and therefore some $a' \in H'$ divides a. We therefore have the following: Every element of H is divisible by some element of H', and vice versa. Also, no element of H (resp. H') divides any other element of H (resp. H'), since both B and B' are reduced. Take $h_1 \in$ H. $h_1 = c_1 h_1'$, and either $h_1' = c_2 h_1$ or $h_1' = c_2 h_2$. The second case implies that h_2 divides h_1, contradicting the fact that B is reduced. The first case implies that $c_1 c_2 = 1$. Since the leading coefficients are all 1, this implies $h_1 = h_1'$.

Now, rename the elements of B if necessary so that $h(f_i) = h(f_i')$, for $f_i \in B$, $f_i' \in B'$. Consider $f_i - f_i'$. This is in ideal(B), so there is some $f_j$in B that reduces $f_i - f_i'$. Now, if $f_i - f_i'$ is not 0, $f_j$must reduce a term that was either in f_i or f_i'. Since B was a reduced basis, f_j cannot reduce f_i. But, if f_j reduced f_i', then f_j' would also have reduced f_i', since $h(f_j) = h(f_j')$. Therefore we must have $f_i = f_i'$.

1.4 Applications.

1.4.1 Deciding Membership in a Polynomial Ideal.

We shall show how Gröbner basis methods may be used to resolve one of the most fundamental decision problems involving polynomial ideals — deciding membership in the ideal.

Theorem 5 *Let $B = \{f_1, \ldots, f_s\}$ be a finite subset of $\mathcal{F}[X]$, $I = ideal(B)$, $P \in \mathcal{F}[X]$. There is an effective procedure for deciding whether $P \in I$.*

Proof: Let B' be a Gröbner basis for I. Reduce P to a normal form P' with respect to B'. We show that $P \in I$ if and only if $P' = 0$.

From Proposition 1, we already know that if $P \rightarrow^*_{B'} 0$, then $P \in I$. Suppose, then, $P \in I$. Since B' is a Gröbner basis, $\rightarrow^*$ is confluent, and so from Lemma 3, there is some reduction $P \rightarrow^* 0$. Now, 0 is irreducible, and by Lemma 1, P can't reduce to distinct irreducibles, so $P = 0$.

Theorem 6 *If $P \in I$, there is an effective procedure to determine $a_1, \ldots, a_s$ in $\mathcal{F}[X]$ such that*

$$P = \sum_{i=1}^{s} a_i f_i.$$

Proof: First, we show how to express elements of B' that were not in B as $\mathcal{F}[X]$-linear combinations of elements of B. If f is a new basis element, $f = S(f_i, f_j)$, reduced to normal form, and we assume inductively that the expressions for f_i and f_j as linear combinations of elements of B are known. Now, $S(f_i, f_j) = f_i \cdot n_i - f_j \cdot n_j$, and if $g \rightarrow_{f_m} g'$, then $g' = g - z \cdot f_m$, for some term z. So, by "remembering" the appropriate n_i, n_j, and z at each

stage of the Gröbner basis calculation, we can express this new f in the form $\sum_{i=1}^{s} a_i f_i$.

Now, since $P \in I$, we can find a sequence $g_0, g_1, \ldots, g_n$ of elements of B' such that

$$P = P_n \rightarrow_{g_n} P_{n-1} \rightarrow_{g_{n-1}} \cdots \rightarrow_{g_1} P_0 = 0.$$

For each i, (i=1, ...n), $P_{i-1} = P_i - z_i \cdot g_i$, where z_i is computed in the course of the reduction. By simple back–substitution, we obtain

$$P_0 = P_n - \sum_{i=1}^{n} z_i g_i,$$

and therefore, $P = \sum_{i=1}^{n} z_i g_i$.

Theorem 5 can be strengthened to provide another characterization of a Gröbner basis:

Theorem 7 *A subset $B \subseteq I$ is a Gröbner basis for an ideal I if and only if*

$$p \rightarrow_B^* 0 \iff p \in I.$$

Proof: We have already proven ($\Rightarrow$) in Theorem 5. Assume then that $p \rightarrow_B^* 0 \iff p \in I$. To show that B is a Gröbner basis for I, it suffices to show that for every pair $f, g \in B$, $S(f,g) \rightarrow_B^* 0$. But $S(f,g) \in I$, so by hypothesis, $S(f,g) \rightarrow_B^* 0$.

1.4.2 A Basis for the Kernel of a Polynomial Ring Homomorphism.

We consider the problem of constructing a basis for ker ϕ, where ϕ is a homomorphism between two polynomial rings over a field $\mathcal{F}$, and which is the identity on $\mathcal{F}$.

We first state a result attributed to W. Trinks [60]:

Theorem 8 (Trinks) *Let $I \subseteq \mathcal{F}[x_1,\ldots,x_m,y_1,\ldots,y_n]$ be an ideal, and let B be a Gröbner basis for I with respect to a lexicographic ordering $<$, where $y_1 < \cdots < y_n < x_1 < \cdots < x_m$.*
Then $B \bigcap \mathcal{F}[y_1,\ldots,y_n]$ is a Gröbner basis for $I \bigcap \mathcal{F}[y_1,\ldots,y_n]$.

Theorem 9 *Let $\phi\colon \mathcal{F}[x_1,\ldots x_m] \longrightarrow \mathcal{F}[t_1,\ldots,t_n]$ be an algebra homomorphism, given by $\phi(x_i) = u_i(t_1,\ldots,t_n)$ $(i = 1,\ldots m)$, and $\phi(a) = a$ $(\forall a \in \mathcal{F})$. Then there is an effective procedure for constructing a basis for ker ϕ.*

Proof: Let $X = \{x_1,\ldots,x_m\}$, $T = \{t_1,\ldots,t_n\}$, and factor ϕ as

$$\phi = \phi' \cdot i\colon \mathcal{F}[X] \xrightarrow{i} \mathcal{F}[X,T] \xrightarrow{\phi'} \mathcal{F}[T]$$

where i is the natural inclusion, $\phi'(x_i) = \phi(x_i)$, $\phi'(t_i) = t_i$, and ϕ' is the identity on F. Let I be the ideal generated by $\{x_1 - u_1,\ldots,x_m - u_m\}$ in $\mathcal{F}[X,T]$, where $u_i = u_i(T)$. Let B be a Gröbner basis for I with respect to the lexicographic ordering $\ll$ on monomials, and the ordering $x_1 \ll \cdots \ll x_m \ll t_1 \ll \cdots \ll t_n$ on the indeterminates.

Clearly, $\ker \phi = \mathcal{F}[X] \cap \ker\phi'$. We assert that $\ker \phi' = I$. By definition of ϕ, $I \subseteq \ker \phi'$. Suppose $\phi'(f) = 0$. We may write $f = g + h$, where $g \in I$ and $h = h(t_1,\ldots,t_n)$. To see this, suppose $f = g + h$, with $g \in I$, and suppose some x_i appears in some term of h. Then $h = x_i \cdot a + b = (x_i - u_i)\cdot a + u_i \cdot a + b$, and therefore $f = (g + (x_i - u_i)\cdot a) + u_i \cdot a + b$. Now, $u_i \cdot a + b$ has at least one fewer occurrence of an x_i than h, and so we may repeat this with $u_i \cdot a + b$, until there

are no more x_i terms in h. Now, $0 = \phi'(f) = \phi'(g) + \phi'(h) = 0 + h$, since ϕ' is the identity on $\mathcal{F}[X]$. Therefore, $h = 0$, so $f = g \in \text{ideal}(\{x_i - u_i(T)\}) = I$, proving our assertion.

Now, $\ker \phi = \mathcal{F}[X] \cap \ker \phi'$, and so it follows from Trinks' Theorem that $\mathcal{F}[X] \cap B$ is a Gröbner basis for $\mathcal{F}[X] \cap I = \mathcal{F}[X] \cap ker\phi' = ker\phi$. ∎

1.4.3 The Word Problem for Commutative Semigroups.

Mayr and Meyer [49], citing the work of H. Simmons [58] have shown that the word problem for commutative semigroups may be reduced to the problem of deciding membership in a polynomial ideal. Our definitions and treatment of this reduction are taken from [49].

Let $S = \{s_1, \ldots, s_v\}$ be a finite alphabet. A *semi-Thue system* over S is a finite set of productions $l_i \rightarrow r_i$, where $l_i, r_i \in S^*$. A word $\beta \in S^*$ is *derived* in one step from $\alpha \in S^*$ via the production $(l_i \rightarrow r_i) \in P$, denoted by $(\alpha \rightarrow \beta(P))$ if and only if for some $\gamma, \delta \in S^*$,

$$\alpha = \gamma l_i \delta \text{ and } \beta = \gamma r_i \delta.$$

As usual, $\rightarrow^*$ denotes the reflexive transitive closure of the relation $\rightarrow$, and a *derivation* of $a_n \in S^*$ from $a_0 \in S^*$ is a sequence $a_0 \rightarrow a_1 \rightarrow \cdots \rightarrow a_n$.

A *Thue system* or *semigroup presentation* is a symmetric semi-Thue system P; by which we mean that $(l \rightarrow r) \in P \Rightarrow (r \rightarrow l) \in P$. In a semigroup, since $\rightarrow$ is symmetric, derivability defines an equivalence relation $\equiv$, given by

$$\alpha \equiv \beta(P) \iff \alpha \rightarrow^* \beta(P).$$

A *commutative semigroup* is a Thue system P in which

$$(\forall s, t \in S),\ (st \rightarrow ts) \in P.$$

Let $\Phi : S^* \times \{1, \ldots, w\} \rightarrow \mathcal{N}$ denote the Parikh mapping, defined by $\Phi(a, i) =$ the number of occurrences of s_i in a.

Let $X = \{x_1, \ldots, x_v\}$, and let $P = \{\alpha_i \equiv \beta_i\ (i = 1, \ldots, w)\}$ be a finite commutative semigroup presentation, with $\alpha_i, \beta_i \in X^*$ $(i = 1, \ldots, w)$. We

identify $\alpha \in X^*$ with the unary monomial $\alpha = x_1^{\Phi(\alpha,1)} \cdots x_v^{\Phi(\alpha,v)}$, and define $I_Z(P)$ and $I_Q(P)$ by

$$I_Z(P) = \left\{ \sum_{i=1}^{w} g_i(\beta_i - \alpha_i) : g_i \in Z[X] \ (i = 1, \ldots, w) \right\} \text{ and}$$

$$I_Q(P) = \left\{ \sum_{i=1}^{w} g_i(\beta_i - \alpha_i) : g_i \in Q[X] \ (i = 1, \ldots, w) \right\}.$$

$I_Z(P)$ and $I_Q(P)$ are the ideals in $Z[X]$ and $Q[X]$, respectively, generated by the polynomials $\{\beta_1 - \alpha_1, \ldots, \beta_w - \alpha_w\}$.

The following two lemmas, which Mayr and Meyer attribute in part to H. Simmons [58], establish the reduction of the word problem in a commutative semigroup to the problem of deciding membership in a particular class of polynomial ideals. Their proofs may be found in [49].

Lemma 7 *If $\alpha \equiv \beta(P)$, then $\beta - \alpha \in I_Z(P)$.*

Lemma 8 *If $\beta - \alpha \in I_Q(P)$, then $\alpha \equiv \beta(P)$.*

1.4.4 Some Basic Complexity Results

Let $B = \{f_1, \ldots, f_m\}$ be a set of polynomials in two indeterminates. Buchberger has shown that an upper bound on the number of steps of the Gröbner basis algorithm is $\frac{3}{2} \cdot (m + 2 \cdot (D+2)^2)^4$, where D is the maximum degree of the polynomials in B [62].

In the case of three indeterminates, F. Winkler has proven that an upper bound for the degrees of the polynomials generated during execution of the Gröbner basis algorithm is $(8D+1) \cdot 2^d$, where D is the maximum degree of the f_i, and d is the minimum degree of the f_i [62].

T. Dubé has proven [20] that if B is a set of polynomials in n variables, with degree at most d, then a reduced Gröbner basis for ideal(B) has degree at most $2((d^2/2) + d)^{2^{n-1}}$.

Mayr and Meyer have shown in [49] that the word problem for finitely presented commutative semigroups is exponential–space complete, and that this word problem is log-lin reducible to the membership problem for polynomial ideals.

D. T. Huynh has proven [34] that for every positive integer n, there is an ideal basis B with $O(n)$ cardinality such that any Gröbner basis equivalent to B has degree and cardinality at least 2^{2^n}.

1.5 Gröbner Basis Computations

The problem of computing a Gröbner basis has been proven to be intractible in general. This is not too surprising, given the scope of fundamental problems that are easily reducible to problems that can be solved using Gröbner basis methods. However, problems of significant size have been shown to be solvable using Gröbner basis methods, and as a result, the calculation of a Gröbner basis is becoming a standard tool in the computer algebra repertoire.

Several computer algebra packages (Macsyma, Scratchpad II, Mathematica, Maple) now provide support for Gröbner basis computations. As a general rule, the broader the scope of a computer algebra package, the less efficient it tends to be at any one task. The goal of creating a comprehensive integrated environment is certainly reasonable, but so is the maxim that any program "should do one thing, and do it well."

While doing research for this thesis, the author first implemented a Gröbner basis algorithm for polynomials in $\mathcal{Q}[X]$, using the muMath computer algebra system, on an Apple IIe microcomputer. It was no surprise to find that that this implementation was far too slow to be of much use, other than to help determine that the immediate performance bottleneck was the headterm calculation.

The second implementation was written using Pascal, using data structures that facilitated headterm computations. Arbitrary precision integer arithmetic was not implemented, due to time and space constraints, and so due to coefficient blow-up, some computations could only be performed with coefficients in Z/pZ (p prime). This version was ported to the IBM-PC,

whose 8087 coprocessor provided extended precision integer arithmetic, enabling larger problems to be solved over $\mathcal{Q}$. This implementation was fast enough to enable testing of various applications and speed-up heuristics.

An unexpected side-effect of this system was its convincing demonstration that full-featured, general-purpose systems may just be inappropriate for real computations. The calculation of the implicit form for Enneper's surface is an example of moderate size; the Gröbner basis computation took 12 minutes on a 6-Mhz IBM-AT; the same computation was cancelled after 15 minutes of CPU time (several hours real time) on an IBM mainframe computer, running Scratchpad II.

1.5.1 A Modified Gröbner Basis Algorithm

Because of the inherent difficulty in computing a Gröbner basis in general, any implementation typically attempts to make use of heuristics. The microcomputer-based Gröbner basis system described above has been extensively used to perform variable elimination (an application of Trinks' theorem). We discuss two modifications to Algorithm GB presented above that made these calculations feasible in a few minutes on a microcomputer. Some examples of nontrivial computations are provided below; these give a rough indication of the size of the problems that these methods can handle.

"Headterm" Reduction

In the Gröbner basis construction given earlier, each S-polynomial is reduced to an irreducible element a, which is added to the basis if and only if $a \neq 0$. By the definition of reduction, any term of a may be reduced. We obtain a

modified algorithm by insisting that only the headterm of a be considered for reduction; this is referred to as "headterm" or "weak" reduction. The proof of the correctness of the Gröbner basis algorithm remains valid, since headterm reduction only potentially misses some 0–reductions, possibly increasing the size of the basis. However, in this possibly larger basis, all S–polynomials reduce to 0, and hence, it is a Gröbner basis.

In order to obtain the (unique) reduced basis, headterm reduction is insufficient. In [GROBNER2.PAS], the construction of the reduced basis is carried out as a second step, using "strong" reduction, after the generation of the Gröbner basis.

In practice, the effect of headterm reduction is dramatic. The term "reduction" is perhaps misleading, in the sense that if $f \rightarrow_g f'$, then f' may contain more terms than f. There is a tradeoff between a faster 0–reduction and smaller intermediate polynomials, but in practice, the latter has a definite advantage.

We wish to point out that the notion of headterm reduction is closely related to yet another characterization of a Gröbner basis. [20,14,26]

Theorem 10 *A subset B of an ideal $I \subseteq \mathcal{F}[X]$ is a Gröbner basis for I if and only if*

$$\forall p \in I,\ \exists f \in B \text{ such that } h(f)|h(p).$$

A Modified Lexicographic Ordering

Given an ideal $I \subseteq \mathcal{F}[X]$, it is faster to compute a Gröbner basis with respect to the total degree ordering on monomials than with respect to lexicographic ordering. When the purpose of computing a Gröbner basis is to eliminate

variables from a system of polynomial equations, the lexicographic ordering is appropriate. However, in some cases we may know how many variables we are trying to eliminate. We call this number c the *cutoff*.

If $m = \prod_{i=1}^{n} x_i^{k_i}$ is a term, write $m = m_1 \cdot m_2$, where $m_1 = \prod_{i=1}^{c} x_i^{k_i}$, and $m_2 = \prod_{i=c+1}^{n} x_i^{k_i}$. Let $\ll$ and $\ll_T$ denote lexicographic and total degree orderings, respectively. Define $m < n$ if either $m_1 \ll_T n_1$, or $m_1 = n_1$, and $m_2 \ll_T n_2$.

For example, in the computation involving Enneper's surface in the following section, we expect to eliminate u, v, and obtain a polynomial in x, y, z alone, and so we order the indeterminates by $u > v > x > y > z$, and define the cutoff $c = 2$. Assuming headterm reduction, this ordering guarantees that each reduction step lowers the total degree in u and v.

1.5.2 A Geometric Example

In this section, we present an example involving a non–trivial Gröbner basis computation. This example arose in connection with both the author's work in mathematical computer graphics, and in extensive conversations with Debe Bednarchak about minimal surfaces.

In CAD ("Computer-Aided Design") and other areas of computer graphics, curves and surfaces are typically represented in parametric form. Some problems are more easily handled if the implicit form for the surface or curve is available.

For example, ray–tracing methods for surface rendering (with hidden surface removal) are based on determining the points of intersection of the surface to be displayed with a family of rays having a common source. Re-

gardless of the actual method used to find, for each ray, the point of intersection closest to this common source, one needs a way of deciding whether or not a point $p = (x_0, y_0, z_0) \in E^3$ is actually on the surface. If the implicit form for this surface is $F(x, y, z) = 0$, then $F(p) \neq 0$ implies that p is not on the surface. Usually, ad hoc conditions, depending on the surface, are used to deal with the case $F(p) = 0$. (For example, if the surface is a sphere, $F(p) = 0$ if and only if p is on the sphere.)

We emphasize that the geometric context should be seen simply to serve as a motivation for the example presented. (It should also be pointed out that developing a reasonably fast, interactive package for drawing these surfaces, including resolving the hidden-surface removal problem on a microcomputer, is in itself a fascinating problem.)

Suppose a curve C in $\mathcal{Q}^2$ is given by

$$C = \{x = u(t),\ y = v(t) : (t \in \mathcal{Q})\}.$$

We assume the coordinate functions are polynomials in t. By an "implicit form" of C, we mean simply a function $F(x, y)$ such that $F(u(t), v(t))$ is identically 0. I.e., if $p = (u(t_0), v(t_0))$ is a point on C, then $F(u(t_0), v(t_0)) = 0$. Analogously, if a surface S in $\mathcal{Q}^3$ is given by

$$S = \{x = f(u, v),\ y = g(u, v),\ z = h(u, v) : (u, v \in \mathcal{Q})\},$$

then by an "implicit form" of S, we mean a function $F(x, y, z)$ such that $F(f(u, v), g(u, v), h(u, v))$ is identically 0.

To find the implicit form of a curve, we view the parametric equations for the curve as defining a homomorphism ϕ: $\mathcal{Q}[x, y] \rightarrow \mathcal{Q}[t]$. If $F(x, y) \in \ker \phi$, then $F(u(t), v(t)) = 0$, and this is precisely what we want. We use the

method described in Theorem 9 to find a basis for $\ker \phi$. The implicitization problem for a surface is handled in an analogous manner.

Implicit Form for Enneper's Surface

The following well-known result from differential geometry concerning minimal surfaces provides the setting for the first example. A concise survey may be found in [31].

Theorem 11 (Local Enneper–Weierstrass representation) *Let D be a simply connected domain in the complex plane, $f(z)$ an analytic function, and $g(z)$ a meromorphic function on D. Then*

$$\begin{aligned} x_1(p) &= \mathrm{Re}\left(\int_{p_0}^{p}(1-g(z)^2)f(z)\,dz\right) \\ x_2(p) &= \mathrm{Re}\left(i\int_{p_0}^{p}(1+g(z)^2)f(z)\,dz\right) \\ x_3(p) &= \mathrm{Re}\left(\int_{p_0}^{p}2f(z)g(z)\,dz\right) \end{aligned}$$

is a conformal minimal immersion of D into $\mathbf{R}^3$.

The surface defined by choosing $f(z) = 1$, $g(z) = z$ is known as *Enneper's surface*. By letting $p_0 = 0$, $p = u + iv$, the Enneper–Weierstrass representation provides the following parametrization of this surface:

$$\begin{aligned} x_1(u,v) &= u - \frac{1}{3}u^3 + uv^2 \\ x_2(u,v) &= v - \frac{1}{3}v^3 + vu^2 \\ x_3(u,v) &= u^2 - v^2 \end{aligned}$$

The *implicitization problem* consists of finding a function F such that $F(x_1, x_2, x_3) = 0$. It will be convenient to rename $\{x_1, x_2, x_3\}$ as $\{x, y, z\}$. The

goal then is to eliminate the parameters u, v from the system of polynomial equations

$$\begin{aligned} x &= u - \frac{1}{3}u^3 + uv^2 \\ y &= v - \frac{1}{3}v^3 + vu^2 \\ z &= u^2 - v^2. \end{aligned}$$

We proceed by trying to find a polynomial $F(x, y, z)$ in the ideal generated by

$$\left\{u - \frac{1}{3}u^3 + uv^2 - x,\ v - \frac{1}{3}v^3 + vu^2 - y,\ u^2 - v^2 - z\right\}.$$

Using Trink's theorem, it suffices to compute a Gröbner basis for this ideal using the lexicographic ordering on monomials, with respect to the ordering $u > v > x > y > z$. As stated above, we use an "elimination" ordering, with cutoff=2. A Gröbner basis corresponding to this ideal is given in Figure [1]. The graph of Enneper's surface is shown in Figure [2].

[B1] $+1u^2 - 1v^2 - 1z$

[B2] $+1u^3 - 3u - 3uv^2 + 3x$

[B3] $+1u^2v + 1v - \frac{1}{3}v^3 - 1y$

[B4] $+1uv^2 - \frac{1}{2}uz + \frac{3}{2}u - \frac{3}{2}x$

[B5] $+1v^3 + \frac{3}{2}vz + \frac{3}{2}v - \frac{3}{2}y$

[B6] $+1v^2z - \frac{3}{2}u^2 + \frac{3}{2}ux + \frac{3}{2}v^2 - \frac{3}{2}vy + \frac{1}{2}z^2$

[B7] $+1uvz + \frac{3}{4}vx - \frac{3}{4}uy$

[B8] $+1uvy - \frac{2}{3}uz^2 + 2uz - 2xz - 1v^2x$

[B9] $+1uvx-\frac{2}{3}vz^2-2vz+1yz-1v^2y$

[B10] $+1uxz+1vyz+1ux+1x^2-1vy-1y^2+\frac{2}{9}z^3-2z$

[B11] $+1uz^3+\frac{27}{8}vxy-\frac{9}{8}uy^2-3uz^2+\frac{9}{4}xz^2-\frac{9}{4}ux^2+\frac{9}{4}xz$

[B12] $+1vz^3+\frac{9}{8}vx^2-\frac{27}{8}uxy+3vz^2-\frac{9}{4}yz^2+\frac{9}{4}vy^2+\frac{9}{4}yz$

[B13] $+1v^2x^2+\frac{2}{3}uxz^2-2uxz+2x^2z-\frac{2}{3}vyz^2-2vyz+1y^2z-1v^2y^2$

[B14] $+1ux^3+\frac{16}{9}uxz^2-\frac{5}{9}x^2z^2-\frac{16}{9}vyz^2+\frac{5}{9}y^2z^2+\frac{8}{81}z^5-\frac{8}{9}z^3-2vx^2y+$

$2uxy^2-1x^2z-1vy^3-1y^2z$

[B15] $+1vx^2yz+\frac{1}{3}x^4-\frac{16}{3}uxy^2-\frac{128}{27}uxz^2+\frac{70}{27}x^2z^2+\frac{17}{3}vx^2y+1vy^3z-\frac{2}{3}y^4+$

$\frac{32}{27}vyz^3+\frac{128}{27}vyz^2-\frac{52}{27}y^2z^2-\frac{8}{243}z^6-\frac{40}{243}z^5+\frac{7}{27}x^2z^3-\frac{1}{27}y^2z^3+\frac{8}{27}z^4+\frac{7}{3}x^2z+$

$\frac{5}{3}y^2z+\frac{40}{27}z^3-\frac{8}{3}ux^3+\frac{1}{3}x^2y^2+\frac{7}{3}vy^3$

[B16] $+1uxy^4+\frac{1}{3}vx^4y-\frac{32}{27}vx^2yz^2-\frac{107}{81}x^2y^2z^2-1vx^2y^3+\frac{37}{81}x^2y^2z-$

$\frac{8}{81}x^4z^2+\frac{176}{81}uxy^2z^2+\frac{1024}{729}uxz^4-\frac{688}{729}x^2z^4+\frac{7}{81}y^4z^2-\frac{256}{729}vyz^4-\frac{160}{729}y^2z^4+$

$\frac{64}{6561}z^8+\frac{320}{6561}z^7-\frac{56}{729}x^2z^5+\frac{80}{729}y^2z^5-\frac{64}{729}z^6-\frac{56}{81}x^2z^3-\frac{16}{27}y^2z^3-\frac{320}{729}z^5-$

$\frac{64}{81}vy^3z^2-\frac{1}{3}vy^5-\frac{1}{3}y^4z-\frac{64}{81}ux^3z-\frac{64}{81}x^4z+\frac{64}{81}vx^2yz+\frac{128}{81}x^2z^2$

[B17] $+1z^9-\frac{405}{4}x^2z^4+\frac{2187}{64}x^4y^2+\frac{729}{64}y^6+\frac{2187}{32}y^4z^2+\frac{405}{4}y^2z^4+\frac{729}{64}x^4z-$

$\frac{729}{32}x^2y^2z-\frac{27}{4}x^2z^6+\frac{27}{4}y^2z^6-\frac{243}{4}x^2z^5-\frac{243}{4}y^2z^5-18z^7-\frac{1215}{64}x^4z^3-\frac{3159}{32}x^2y^2z^3-$

$\frac{1215}{64}y^4z^3-\frac{2187}{32}x^4z^2+\frac{243}{4}x^2z^3+\frac{729}{64}y^4z+\frac{243}{4}y^2z^3+81z^5-\frac{729}{64}x^6-\frac{2187}{64}x^2y^4$

[B18] $+1vx^6y-6x^2y^4z^2-\frac{64}{9}vy^5z^2-\frac{31}{27}y^6z^2+\frac{88}{243}y^4z^5+\frac{1832}{243}y^4z^3-1vx^2y^5+$

$\frac{32}{9}x^2y^4z+1vy^7+\frac{235}{27}y^6z-\frac{32}{9}vx^4yz^2-\frac{41}{9}x^4y^2z^2-1vx^4y^3-\frac{89}{9}x^4y^2z-$

$\frac{8}{27}x^6z^2-\frac{688}{243}x^4z^4-\frac{1280}{243}vx^2yz^4-\frac{1888}{243}x^2y^2z^4+\frac{64}{2187}x^2z^8+\frac{320}{2187}x^2z^7-\frac{56}{243}x^4z^5-$

$$\frac{32}{243}x^2y^2z^5 - \frac{2624}{2187}x^2z^6 - \frac{1528}{243}x^4z^3 - \frac{2464}{243}x^2y^2z^3 - \frac{320}{243}x^2z^5 - 16vx^2y^3z^2 - \frac{64}{27}ux^5z -$$
$$\frac{64}{27}x^6z + \frac{64}{27}vx^4yz + \frac{128}{27}x^4z^2 - \frac{304}{27}ux^3y^2z + \frac{304}{27}vx^2y^3z + \frac{608}{27}x^2y^2z^2 - \frac{1024}{243}ux^3z^3 +$$
$$\frac{1024}{243}vx^2yz^3 + \frac{2048}{243}x^2z^4 - \frac{2560}{243}vy^3z^4 - \frac{736}{243}y^4z^4 + \frac{128}{2187}y^2z^8 + \frac{640}{2187}y^2z^7 - \frac{5248}{2187}y^2z^6 -$$
$$\frac{640}{243}y^2z^5 - \frac{208}{27}uxy^4z + \frac{208}{27}vy^5z + \frac{416}{27}y^4z^2 - \frac{2048}{243}uxy^2z^3 + \frac{2048}{243}vy^3z^3 + \frac{4096}{243}y^2z^4$$

Figure [1]: Gröbner basis corresponding to the paramteric form of Enneper's surface. The implicit form appears as [B17].

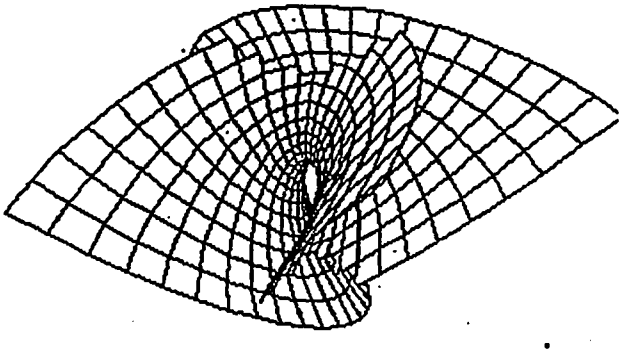


Figure [2]: Enneper's surface.

The computation of this Gröbner basis took approximately 3 minutes on a 25 MHz IBM-compatible 80386 PC, with an 80387 math co-processor. This particular computation has served as part of the validation suite when testing new versions of the Gröbner basis calculation routines.

It should be pointed out that [GROBNER2.PAS] has been unable to obtain the implicit form for the minimal surface defined by choosing $f(z) = 1$, $g(z) = z^2$, as the heights of the rational coefficients of some of the terms in the intermediate polynomials involved in the calculation become too large (more than 17 digits). Two views of the graph of this surface are shown in Figure [3].

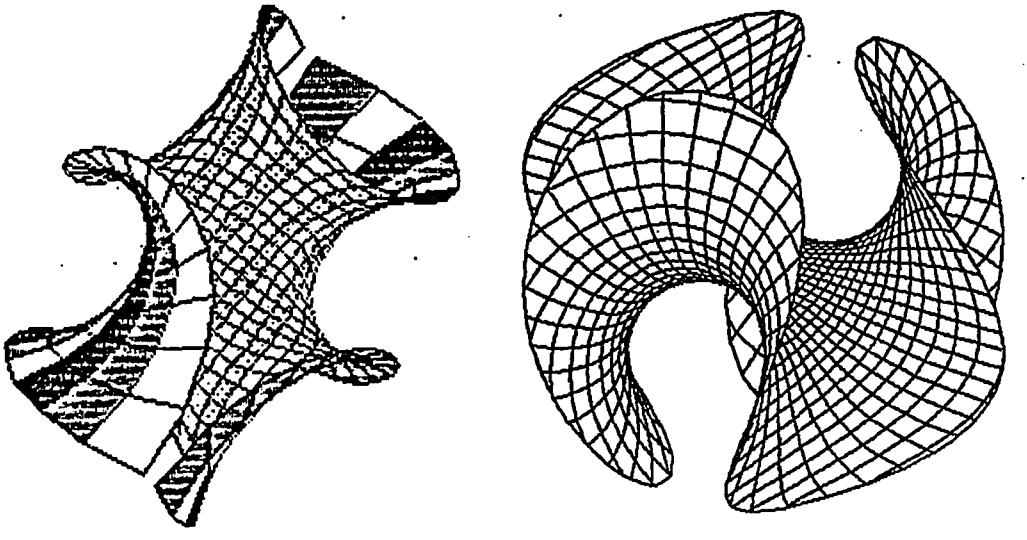


Figure [3]: The minimal surface obtained by letting $f(z) = 1$, $g(z) = z^2$. In the view on the right, the portion of the surface lying outside a sphere of radius = 1.6 has been removed.

Implicit Form for a Lissajous Curve

Let C be the curve given by $x(t) = \cos(3t)$, $y(t) = \cos(5t)$. Despite the fact that the coordinate functions are not polynomials in the parameter t, they may be viewed as polynomials in $\cos(t)$ and $\sin(t)$, which themselves satisfy the Pythagorean identity, which is a polynomial relation.

Let $u = \cos(t)$, $v = \sin(t)$. Then $x = u^3 - 3uv^2$, $y = u^5 - 10u^3v^2 + 5uv^4$, and $u^2 + v^2 = 1$. We consider the ideal I generated by

$$\{u^3 - 3uv^2 - x, u^5 - 10u^3v^2 + 5uv^4 - y, u^2 + v^2 - 1\},$$

and try to eliminate u, v, by computing a Gröbner basis for I with the ordering $u \gg v \gg x \gg y \gg z$ on the indeterminates.

The unique polynomial in the Gröbner basis in $\mathcal{F}[x, y]$ is

$$x^5 - \frac{5}{4}x^3 + \frac{5}{16}x - \frac{1}{4}y^3 + \frac{3}{16}y.$$

This Gröbner basis took 2 seconds to compute, running [GROBNER2.PAS] on a 25 MHz 80386 PC, with an 80387 co-processor.

This is a relatively easy Gröbner basis computation, but it is related to a computation that remains a challenge. Let S be the surface given as a 1–parameter family of spheres, whose centers lie on the curve in $\mathcal{Q}^3$ given by

$$x(t) = \cos(3t),\ y(t) = \cos(4t),\ z(t) = \cos(5t).$$

The graphs of C and S are shown in Figure [4].

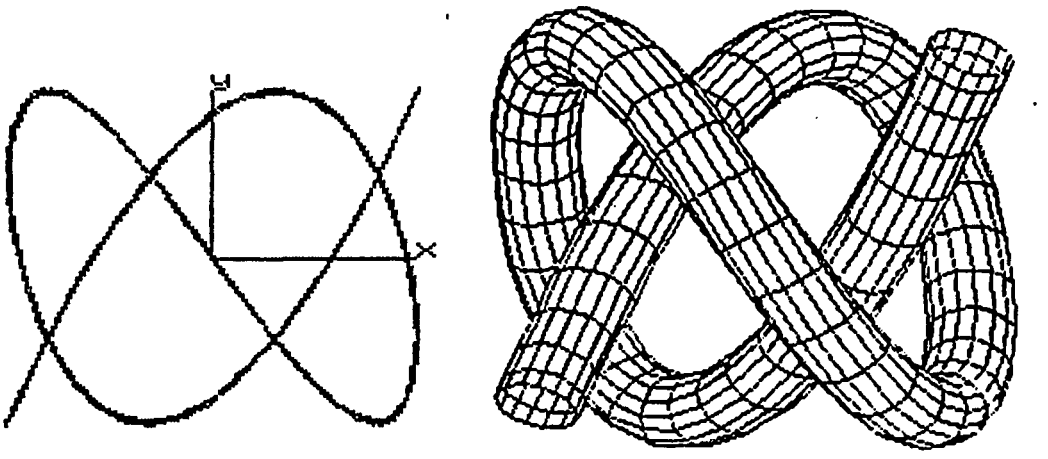


Figure [4]: The graphs of the curve $C = (\cos(3t), \cos(5t))$ and the envelope of the 1–parameter family of spheres centered around the curve $x(t) = \cos(3t)$, $y(t) = \cos(4t)$, $z(t) = \cos(5t)$.

The Gröbner basis method has been successful for finding the implicit form for the "tube–surface" when the curve is a circle; i.e., the surface is a torus. However, to date, knottier problems, such as this, blow up.

Chapter 2

HNN Extensions

2.1 Introduction

In this chapter, we deal with problems relating to certain classes of HNN extensions. HNN extensions were introduced by G. Higman, B.H. Neuman, and H. Neuman in 1949 [30], and are among the basic constructions of combinatorial group theory. The HNN construction arises in algebraic topology in the context of presenting the fundamental group of an arcwise connected space to which a "handle" has been attached. We will, however, be considering these groups from a purely combinatorial perspective.

The problems we will consider are the *word problem* and the *conjugacy problem.* The word problem for a group G is to decide whether a given word w in the generators of G defines the identity element of G. The conjugacy problem for G is to decide, given words u and v in G, whether there exists a word $x \in G$ such that $u = x^{-1}vx$. If such an x exists, we say that u and v are conjugate, and denote this by $u \sim v$. The word problem is easily seen to be a special case of the conjugacy problem, in which $v = 1$, and so it is

to be expected that the conjugacy problem for G will be harder, in general, than the word problem.

We first present the general groundwork on which our study of HNN extensions is based. We then focus attention on certain subclasses of HNN extensions, and investigate the complexity of the corresponding word and conjugacy problems.

2.2 Background

The following definitions and fundamental results are based on [46] and [50].

Definition: Let G be a group, let $\{H_i : (i \in I)\}$, $\{K_i : (i \in I)\}$ be families of subgroups of G for index set I, and let $\{\Phi_i : H_i \rightarrow K_i \ (i \in I)\}$ be a family of isomorphisms. The *HNN extension with base G, stable letters* $\{a_i \ (i \in I)\}$ *and associated subgroups* H_i *and* K_i is the group

$$G^* = \langle\ G, a_i\ ;\ a_i^{-1} h_i a_i = \Phi_i(h_i) \quad (h_i \in H_i,\ i \in I)\ \rangle.$$

It is understood that the set of stable letters is disjoint from the set of generators of G. A stable letter a_i and its formal inverse a_i^{-1} are called a *matched stable pair.*

Higman, Neumann, and Neumann proved in their original paper that G is embedded in G^* under the mapping $g \rightarrow g$. J. L. Britton [12] subsequently proved a fundamental result that, together with this embedding theorem, may be used to reduce the word problem for an HNN extension G^* to the word problems for the base group G and the associated subgroups H_i, K_i $(i \in I)$.

We first define a *reduced* sequence.

Definition: Let $\{ g_i \ (i = 0, \ldots n)\}$ be elements of G^* which contain no occurrence of either a_{i_j} or $a_{i_j}^{-1}$. A sequence

$$g_0,\ a_{i_1}^{\epsilon_1},\ g_1,\ \ldots a_{i_n}^{\epsilon_n},\ g_n \quad (n \geq 0)$$

is said to be *reduced* if there is no consecutive subsequence $a_{i_j}^{-1}, g_j, a_{i_j}$, with $g_j \in H_{i_j}$, or $a_{i_j}, g_j, a_{i_j}^{-1}$, with $g_j \in K_{i_j}$.

Lemma 9 (Britton) *If the sequence* $g_0,\ a_{i,1}^{\epsilon_1},\ g_1,\ \ldots\ a_{i,n}^{\epsilon_n},\ g_n \quad (n \geq 0)$ *is reduced and* $n > 0$, *then* $g_0 \cdot a_{i_1}^{\epsilon_1} \cdot g_1 \cdots a_{i_n}^{\epsilon_n} \cdot g_n \neq 1$ *in* G.

Suppose a word $w = 1$ in G^*. If w does not contain a stable letter, then the embedding of G in G^* implies that $w = 1$ in G^* if and only if $w = 1$ in G. If, on the other hand, w contains a stable letter, then according to Britton's lemma, a matched stable pair may be removed from w. This removal of a matched stable pair, by application of the appropriate relator, is termed a *reduction*, or a *pinch*.

Definition: A *P-reduction* of a word w is the application to w of one of the following:

1. replace a subword of the form $a_i^{-1} g a_i$ by $\Phi_i(g) \quad (g \in H_i)$
2. replace a subword of the form $a_i g a_i^{-1}$ by $\Phi_i^{-1}(g) \quad (g \in K_i)$

The *P–projection* of a word $w \in G^*$ is the image in G^* of $\pi(w)$, where π is the homomorphism defined by $\pi(G) = 1,\ \pi(a_i) = a_i \ (i \in I)$. w and w' are said to be *P–parallel* if their P–projections are equal elements of G^*.

Britton's lemma and the embedding theorem can be seen to provide an algorithm for solving the word problem for G^*, assuming the generalized word problems for G and the associated subgroups H_i and K_i ($i \in I$) may be effectively decided. Let $w \in G^*$. We may assume w is freely reduced. If w contains no stable letters, then simply apply the decision procedure for G. Otherwise, look for a subword of either the form $a_{i_j}^{-1} \cdot g_j \cdot a_{i_j}$ with $g_j \in H_{i_j}$, or of the form $a_{i_j} \cdot g_j \cdot a_{i_j}^{-1}$, with $g_j \in K_{i_j}$; i.e., a subword whose presence indicates that the corresponding sequence for w is not reduced. If no such subword can be found, then w is not 1, by Britton's lemma, and we are done. Otherwise, replace the subword by $\Phi_{i_j}(g_j)$ or by $\Phi_{i_j}^{-1}(g_j)$, respectively. Freely reduce, and obtain a word w'. Recursively apply this procedure to w'. Since each replacement eliminates two stable letters from w, this process must terminate.

This procedure is clearly effective. In order to reduce the word problem for G^* to the generalized word problems for G and the subgroups H_i and K_i, (which we assume to be effectively decidable), we must first detect a matched stable pair, in which the letters are adjacent in the P–projection; this is a straightforward pattern-matching problem. We must also decide if the subword between these letters is in the appropriate associated subgroup; this is accomplished by solving the word problem in either H_i or K_i. Replacement and free reduction are also straightforward.

In order to deal with the conjugacy problem for HNN extensions, we will make use of a theorem due to D.J.Collins, known as Collins' Lemma [50]. We first provide some more definitions.

Definition: A word $w = g_0 a_{i_1}^{\epsilon_1} \cdots a_{i_n}^{\epsilon_n} g_n \quad (n \geq 0)$ is *reduced* if the sequence

$g_0,\ a_{i_1}^{e_1}, \ldots a_{i_n}^{e_n},\ g_n$ is reduced.

Lemma 10 *Let $u = g_0 a_{i_1}^{e_1} \cdots a_{i_n}^{e_n} g_n$ and $v = h_0 a_{j_1}^{f_1} \cdots a_{j_m}^{f_m} h_m$ be reduced words, and suppose $u = v$ in G^*. Then $n = m$, and $a_{i_k}^{e_k} = a_{j_k}^{f_k}$, $(k = 1 \ldots n)$. (I.e., if two reduced words in G^* are equal, then they are P–parallel.)*

Proof: Since $u = v$,

$$uv^{-1} = g_0\ a_{i_1}^{e_1} \cdots \overbrace{a_{i_n}^{e_n}\ g_n\ h_m^{-1}\ a_{j_m}^{-f_m}} \cdots a_{j_1}^{-f_1}\ h_0^{-1} = 1,$$

and so some P–reduction must be possible. Now, u and v are reduced, and so the only subword of uv^{-1} to which a P–reduction can be applicable is $a_{i_n}^{e_n} g_n h_m^{-1} a_{j_m}^{-f_m}$; this implies that $a_{i_n}^{e_n} = a_{j_m}^{-f_m}$. We may repeat this process, removing one stable letter from both u and v^{-1} at each iteration. If $n \neq m$, say $n > m$, this process would yield an equality

$$1 = g_0\ a_{i_1}^{e_1} \cdots a_{i_k}^{e_k}\ g_k h_0^{-1} \quad (k > 0),$$

contradicting the fact that u was reduced. ∎

Definition: Let $z \in G^*$. The *stable length* of z, $\|z\|$, is the total number of occurrences of stable letters and their inverses in any reduced word of G^* which represents z.

The fact that $\|z\|$ is well-defined follows directly from Lemma 10.

Definition: A word $w = g_0 a_{i_1}^{e_1} \cdots a_{i_n}^{e_n} g_n$ is *cyclically reduced* if all cyclic permutations of the sequence $g_0,\ a_{i_1}^{e_1}, \ldots, a_{i_n}^{e_n} g_n$ are reduced.

Proposition 5 *If $w \in G^*$, and P–reduction is effective, then we may effectively produce $z \in G^*$ such that z is cyclically reduced, and $w \sim z$.*

Proof: Let $w_1, \ldots w_n$ be the list of all cyclic permutations of w. If $\forall i$, w_i is reduced, then we may let $z = w$. Otherwise, suppose w_j is not reduced, and that w_j' is the word formed by applying a P–reduction to w_j. Now, $w \sim w_j = w_j'$, since w is conjugate to any cyclic permutation of itself. We now recursively apply this procedure to w_j'. Since $\|w_j'\| < \|w\|$, this process terminates in a finite number of steps. ∎

Theorem 12 (Collins' Lemma) *Let*

$$G^* = \langle\, G, a_i \;;\; a_i^{-1} h_i a_i = \Phi_i(h_i) \quad (h_i \in H_i,\ i \in I) \,\rangle$$

be an HNN extension, and let $u = a_{i_1}^{e_1} g_1 \cdots a_{i_n}^{e_n} g_n \quad (n \geq 1)$ *and* v *be conjugate P–cyclically reduced elements of* G. *Then* $\|u\| = \|v\|$, *and* $u = z^{-1} v^* z$, *where* v^* *is a cyclic permutation of* v *starting with* $a_{i_1}^{e_1}$, u *and* v^* *are P–parallel, and*

$$z \in \begin{cases} H_{i_1} & \text{if } e_1 = 1, \text{ and} \\ K_{i_1} & \text{if } e_1 = -1. \end{cases}$$

Collins' Lemma provides a natural framework for solving the conjugacy problem for general HNN extensions [50]. This will be discussed in detail later in this chapter.

2.3 The Word Problem for HNN(Z)

We now consider a class of HNN extensions in which the base G is the infinite cyclic group $\langle b;\ \rangle$, the index set I is finite, and none of the associated subgroups H_i and K_i are trivial. This class is called HNN(Z) in [5]. We note that the presentation in this case is considerably simpler. For each stable letter a_i, the subgroup H_i of G is infinite cyclic, and we may choose $p_i > 0$ such that $H_i = \langle b^{p_i}; \rangle$. Each stable letter also introduces only one relator, as Φ_i is determined by its action on b^{p_i}. Define q_i by $\Phi_i(b^{p_i}) = b^{q_i}$. Then G^* has presentation

$$G^* = \langle\ b, a_i;\ a_i^{-1} b^{p_i} a_i = b^{q_i}\ \ (i \in I)\ \rangle.$$

2.3.1 A Turing Machine for the Word Problem

In this section we present a Turing machine that solves the word problem for the class HNN(Z) of HNN extensions with several stable letters, and for which the base group is the free group of rank 1. We prove the correctness of this Turing machine, and prove a basic result regarding the complexity of the word problem for these groups.

We will be referring to the program listing `Word_Problem_Machine`, provided below, in which certain crucial *begin-end* blocks have been numbered. It is understood that a *loop* consists of iterated execution of a *block*, and so the labels {-n-} in the program can be used to refer both to loops and to blocks.

We provide a brief overview of the program notation. Each tape is bi-infinite, with one read/write head, and is implemented as a Pascal *object*.

A tape responds to a message by changing its internal state variables, or in the case of sym, returning a value (namely, the symbol currently scanned by the read/write head). The notation for message passing is the same as that used to reference a field of a record.

For example, if T is a tape, then in our notation:

`T.Move(Left)` moves the read/write head of T one cell to the left.

`T.WriteAndMove(Blank,Left)` writes a blank in the currently scanned cell, and then moves the read/write head of T one cell to the left.

`T.Sym` is the currently scanned symbol on tape T.

We prove the correctness of function `Word_Problem_Machine` by presenting appropriate loop invariants. Loop invariants, according to S. Baase [9, page 15], are "conditions and relationships that are satisfied by the variables and data structures at the end of each iteration of the loop." The proof given here is an informal proof, in light of the formal proof methods for *while programs* such as this. [47]

`Word_Problem_Machine` uses 6 tapes, `I,A,S,R,T,U`. These variables are global. We now list the crucial components of `Word_Problem_Machine`, along with the preconditions and postconditions to which we will be referring in the correctness proof for `Word_Problem_Machine`.

- **Word_Problem_Machine**

Preconditions: The read head on the input tape I is positioned over the leftmost non-blank symbol, if there is one. I may be blank. R contains the symmetrized set of relators for the HNN group G. S, A, T, and U are initially blank.

Postconditions: Word_Problem_Machine returns TRUE if the input word defines the identity in G, and returns FALSE otherwise.

Tapes used:

I (the input tape)

S (a stack containing the processed input)

A (the P-projection of S)

R (the tape containing the relators)

T,U (worktapes used in the rewriting process).

```
function Word_Problem_Machine :boolean;
var   x, alpha,beta :symbol;
begin {-1-}
  x := I.Sym;
  while x <> blank do begin {-2-}
    if FreeMatch(x,S.Sym) then Freely_Reduce(x)
    else begin {-3-}
       S.Push(x);
       if x in Stable_Letters then begin {-4-}
         if not FreeMatch(x,A.Sym) then A.Push(x)
         else begin {-5-}
           A.Push(x);  message('free stable match');
           S.Move(left);
           beta := S.Sym;
           repeat
               S.Move(left);
               T.Push(beta)
           until S.Sym in Stable_Letters;
           alpha := S.Sym;
           Scan_R_Tape(alpha,beta);
           while T.Sym in Base_Letters do begin {-6-}
             while (R.Sym in Base_Letters) and
                   (T.Sym in Base_Letters) do begin {-7-}
                   T.WriteAndMove(blank,left);
                   R.Move(right)
             end; {-7-}
             if not (R.Sym in Base_Letters) then begin
                  Apply_Relator;
                  if T.Sym = blank then Pinch
             end
             else Clean_Up  { Too few b's to pinch. }
           end; {-6-}
         end; {-5-}
         message('Rewind R');  R.Rewind;
       end; {-4-}
    end; {-3-}
    I.Move(right);  { Advance Input }
    x := I.Sym;
  end; {-2-}
  Word_Problem_Machine := S.Sym = blank;
  if S.Sym = blank then Accept else Reject
end; {-1-}
```

- **FreeMatch**

FreeMatch(x,y) is true if and only if x and y are inverse symbols of each other. FreeMatch is kept representation–dependant for simplicity, but it is clear that the decision that it makes could be simply incorporated into the Turing machine's finite state control.

```
function FreeMatch(x,y :symbol) :BOOLEAN;
begin
  FreeMatch := (odd(y) and (y-1 = x)) or
               (odd(x) and (x-1 = y))
end;
```

- **Skip_To_Next_Relator**

Preconditions: R.sym is the leftmost symbol of a relator.

Postconditions: R.sym is the leftmost symbol of the next relator on r.

Tapes Affected: R

```
procedure Skip_To_Next_Relator;
begin
  R.Move(right);
  while not (R.Sym in Stable_Letters ) do R.Move(right);
  R.Move(right);
  while not (R.Sym in Stable_Letters ) do R.Move(right);
end;
```

- **Scan_R_Tape**

Preconditions: R.Sym is the leftmost non-blank symbol on R. There is a unique relator on R beginning with ab.

Postconditions: R.Sym is the leftmost base letter on the LHS of the relator $ab^l a^{-1} b^r$

Tapes Affected: R

```
procedure Scan_R_Tape(alpha,beta :symbol);
begin
   { scan R for  alphabet[alpha]+alphabet[beta] }
   while R.Sym in Stable_Letters do
   begin
     if R.Sym <> alpha then Skip_To_Next_Relator
     else begin
       R.Move(right);
       if R.Sym <> beta then
          Skip_To_Next_Relator
       else { don't move. }
     end
   end
end;
```

- **Clean_Up**

Preconditions: U.sym is the rightmost non-blank symbol on U. U contains b^m, where b is a base letter. S.sym is a – the first syllable of the subword of s that is a candidate for "pinching".

Postconditions: S.sym is the rightmost non-blank symbol on S. U is empty.

Tapes Affected: S, U

```
procedure Clean_Up;
begin
   message('Too few b''s to pinch.');
   while U.sym <> blank do     { Erase U }
       U.WriteAndMove(blank,left);
   while S.sym <> blank do     { Reset S }
         S.Move(right);
   S.Move(left);
end;
```

- **Pinch**

Preconditions: S.sym is a - the first syllable of the subword of s that will be pinched. U.sym is the rightmost non- blank symbol on U. U contains b^m, where b is a base letter.

Postconditions: S.sym is the rightmost non-blank symbol on S. U is empty.

Tapes Affected: S, U, A

```
procedure Pinch;
begin
   { S.sym is currently the leftmost symbol of the   }
   { 3-syllable subword that is about to be replaced.}
   message('Pinch; remove top 3 syllables from S');
   S.Move(right);              { use this as a marker  }
   while S.sym <> blank do     { Clear to end of tape  }
      S.WriteAndMove(blank,right);
   while S.sym = blank do      { return to placeholder }
      S.Move(left);
   S.WriteAndMove(blank,left);    { erase placeholder }
   message('now transfer contents of U to S');
   while U.Sym <> blank do
      begin
        if Freematch(S.Sym,U.Sym) then S.Pop
        else S.Push(U.Sym);
        U.WriteAndMove(blank,left)
      end;
   message('remove the stables from a.');
   A.Pop; A.Pop;
end;
```

- **Apply_Relator**

Preconditions: r.sym is the stable letter to the left of the LHS of this relator. U.sym is the rightmost non-blank symbol on U.

Postconditions: R.Sym is the leftmost base letter on the LHS of this relator.

Tapes Affected: U, R

```
procedure Apply_Relator;
begin
   message('Copy RHS of relator to U.');
   R.Move(right);
   while (R.Sym in Base_Letters) do
   begin
      U.Push(R.Sym);
      R.Move(right);
   end;
   message('skip back to beginning of this relator');
   repeat
      R.Move(left)
   until (R.Sym in Stable_Letters);
   repeat
      R.Move(left);
   until (R.Sym in Stable_Letters);
   R.Move(right);
end;
```

- **Freely_Reduce**

Preconditions: A, S have the stack property. x and S.Sym are free inverses of each other. A and S are P-parallel.

Postconditions: A and S are P-parallel.

Tapes Affected: A, S

```
procedure Freely_Reduce(x :symbol);
begin
   message('freely reduce');
   S.Pop;
   if FreeMatch (x,A.Sym) then
     begin
       message('Also freely reduce A stack');
       A.Pop;
     end;
end;
```

Theorem 13 `Word_Problem_Machine` *solves the word problem for* G.

Proof: We suppose that the input word w is on tape I, and that R contains the symmetrized set of relators for G. Suppose G has presentation

$$G = \langle\ b, a_1, \ldots, a_n;\ a_i b^{e_i} a_i^{-1} = b^{f_i}\ \rangle.$$

For each relation $ab^e a^{-1} = b^f$, we create 4 rewriting rules:

$$ab^e a^{-1} \to b^f,\ ab^{-e} a^{-1} \to b^{-f},\ a^{-1} b^f a \to b^e,\ a^{-1} b^{-f} a \to b^{-e}.$$

Assume that the subroutines work as documented; i.e., that the preconditions are met when the routines are called, and that the postconditions are satified upon exit. (These are separate lemmas, proof of which will not be given, since this is not going to be a formal proof.) By an abuse of language, we will identify a tape with the group word represented by the non-blank symbols on that tape, provided the tape contains at most one contiguous

block of non-blanks. A blank tape represents the empty word. If T is a tape, then $|T|$ denotes the number of non-blank symbols on T.

We claim that the following assertions regarding Loop 2 are loop invariants:

A1: S is freely reduced

A2: S is P-reduced; i.e., unpinchable.

A3: S + (the unprocessed input on I) = w, viewed as words in G.
("+" is used here to denote concatenation.)

A4: A is the P–projection of S

Before proving these assertions, we will show how the correctess of the algorithm follows from these invariants.

At the beginning of each execution of block 2, x is the next input symbol to be processed. Precisely one input symbol is processed for each execution of block 2, and processing halts when the input is exhausted. This guarantees that block 2 of `Word_Problem_Machine` is only executed finitely many times.

We claim that, after the final execution of block 2, S is empty iff $w = 1$ in G. Suppose S is empty. By assertion (A3), since there is no input left to process, $w = 1$ in G, where as usual, we identify the empty word with the identity element of G. Suppose S is not empty, and $w = 1$ in G. By (A3), S = 1 in G. There are two cases to consider. If S contains a stable letter, then by Britton's lemma, S contains a matched stable pair which can be pinched; but this contradicts assertion (A2), that S is P–reduced. If S does not contains a stable letter, then S is equal to the identity element in the free subgroup

of G generated by the base letter b, and since S is not the empty word, S is not freely reduced, contradicting (A1). ∎

Initially, these assertions are true at the beginning of the program. We will show that these assertions are true after each execution of block 2, and are therefore true loop invariants.

Write $S = uy$, where u is a word, and y is a single symbol, possibly a blank. We assume that the 4 loop assertions are satisfied, and that x is the next symbol to be processed.

There are only two places in the algorithm in which the contents of S can change; the beginning of block 2, and the "pinch" step. We will show that the assertions remain valid under these changes.

Consider first the beginning of block 2. Exactly one of 2 changes occurs to S:

- if $y = x^{-1}$, S is replaced by u
- if $y \neq x^{-1}$, S is replaced by $uyx = Sx$.

We consider each assertion in turn.

A1. Since S is freely reduced, so is U, and if $y \neq x^{-1}$, then sx is also freely reduced.

A2. We may only observe here that no attempt to pinch will be made unless x is a stable letter, and we will not know whether or not S can be pinched until we execute block 5. By looking at the beginning of block 4, we see that we will not attempt a pinch unless $S = ux^{-1}b^{\pm n}$ for some subword u. Note that we are using the fact that A and S are p–parallel.

A3. This is trivial; If $S = uy$, and the unprocessed input $= xv$, then

$(uy)(xv) = u(yx)v$.

A4. Observe that procedure `Freely_Reduce` preserves the property that A and S are p–parallel. If no free reduction occurs, note that x is pushed onto S, and then, at the beginning of block 4, x is pushed onto A, regardless of the outcome of `FreeMatch(x,A.Sym)`.

Summing up these observations, assertions A1,A3, and A4 are preserved at the beginning of block 2, and block 5 will be executed if and only if $S = v\alpha\beta^n\alpha^{-1}$, where v is some word, possibly the empty word, α is a stable letter, β is a base letter, and $n > 0$.

The heart of the algorithm is the loop executing block 6. The beginning of block 5 copies the base letters in the top three syllables of S onto T, and positions R's read head over the leftmost base letter on the left-hand side of the unique applicable relator.

So, prior to entering loop 6, the following conditions hold:

1. $S = v\alpha\beta^n\alpha^{-1}$ (v is some word)
2. A is the p–projection of S
3. $T = \beta^n$
4. U is empty
5. R contains the relator $\alpha\beta^L\alpha^{-1}\beta^r$, and R.sym is the first β in the left–hand side of this relator.

If $n = qL$, for some positive integer q, then we will replace $s = v\alpha\beta^n\alpha^{-1}$ by vb^{qr}. Loop 6, therefore, is seen to decide the generalized word problem for the subgroup $\langle b; \beta^L \rangle$ of the base group $\langle b; \rangle$. Furthermore, while deciding membership in this subgroup, the program builds up the word b^{qr} on the

tape U, as a side effect; this simplifies the rewriting of S, in case a pinch is possible.

Claim 1 *Block 6 is executed at least once.*

Proof: T contains no stable letters, so block 6 is executed if and only if $|T| > 0$. If $|T| = 0$, then the top two symbols of S would constitute a matched stable pair. This is impossible, since these would have been freely reduced. (Assertion A1.)

Claim 2 *Whenever block 6 is executed, block 7 is executed at least once.*

Proof: Prior to execution of block 6, R is scanning the first β in the left–hand side of a relator. On each pass of loop 6, except for the last, R.sym is repositioned here by procedure `Apply_Relator`.

Let q be the number of times block 6 is executed. Since block 7 decreases $|T$, block 6 will be executed only finitely many times. By Claim 1, $q > 0$.

Loop 7 erases m = min($|T|$, L) symbols from T, and writes these m symbols to U.

Claim 3 *After executing loop 7 (q-1) times,* $|T| = n - (q-1) \cdot L$, *and* $|U| = (q-1) \cdot r$.

Claim 4 *After executing loop 7, "not (R.sym in Base_Letters)" is true if and only if* $m = L$.

Proof: R.Move(right) in block 7 is executed m times. R is initially positioned over the first β, so after m moves, if $m < L$, then R is scanning a β on the left–hand side of the relator; if $m = L$, then R is scanning α^{-1}.

Claim 5 *After executing loop 7, if R.sym is a base letter, then T.sym = blank.*

Proof: This follows from the fact that T contains only base letters and blanks.

Claim 6 *At the start of the q^{th} execution of block 6, $|T| \leq L$.*

Proof: Block 7 is executed whenever block 6 is, and block 7 only removes L symbols from T. If $|T| > L$, then block 6 will be executed more than q times, contrary to the definition of q.

After the q^{th} (and final) execution of loop 7, we will know whether or not it is possible to pinch S. According to Claim 4, there are 2 cases to consider:

case 1. $m = L$. `Apply_Relator` is invoked once more, and adds r β's to U, while erasing m β's from T. Therefore,

$$\begin{aligned} |T| &= n - (q-1)L - L = 0 \Rightarrow n = qL \\ |U| &= (q-1)r + r = qr. \end{aligned}$$

Since $|T| = 0$, T.sym = blank, so the program will pinch. As we pointed out before, pinching should occur if and only if $n = qL$.

case 2. $m \neq L$. Then $m = |T| \leq L$ (by definition of m), and so $|T| < L$. Therefore L does not divide n, and so we do not pinch.

In either case 1 or case 2, the work tapes will be cleared, and S will be restored to stack status.

We now return to the discussion of the loop invariants. If "`Pinch`" is not executed, then S is not changed, so there is nothing to say regarding A1,A3, and A4.

Suppose Pinch is executed. A3 is satisfied because Pinch applies a group relator to a word in G. Pinch removes the top two stable letters from S and also from A; since A4 was initially true, A4 is true after pinching. A1 was true before pinching, and since free reduction takes place while transferring the contents of U to S, A1 is true after pinching as well.

We are assuming inductively that prior to processing the symbol x, $S = v\alpha\beta^n$ could not be pinched. In particular, v was unpinchable. After pinching, $S = v\alpha\beta^{qr}$, which can be pinched if and only if v can be, since pinching removes stable letters. We have just shown that the program will pinch $S + x$ if and only if n is a multiple of the length of the approriate relator; if pinching does not occur, then $S + x$ is the new contents of tape S. ∎

2.4 The Conjugacy Problem for the Class *VA*

A group G is said to be a *VA–group* if G admits a presentation

$$G = \langle\ b, a_i;\ a_i^{-1} b^{p_i} a_i = b^{q_i}\ \ (i \in I,\ p_i > 0, q_i > 0, \text{and} \gcd(p_i, q_i) = 1)\ \rangle,$$

VA–groups are also known as *groups of vector addition systems,* or simply *vector groups.*

Decision problems for these groups have been investigated by Anshel [5,7], Anshel and Stebe [8], Hurwitz [33], and more recently, Tretkoff [59]. The connection between group-theoretic investigations and reachability problems for vector addition systems was shown by Anshel in 1976.

We will show that the conjugacy problem for a VA-group G is effectively reducible to the problems of (1) integer factorization, (2) determining the existence of a solution to a system of linear Diophantine equations, and (3) deciding the word problem in a commutative semigroup.

Remark: A program developed by the author, [HNNZ7.PAS], implements a solution to the word problem for HNN(Z). It is based on the Word Problem Turing machine presented in [TMO22.PAS], and has proven to be quite useful in the study groups in HNN(Z).

The word problem in a commutative semigroup is reducible to a Gröbner basis computation and subsequent reduction to normal form. These computations have been implemented by the author in [GROBNER2.PAS].

The problem of solving a system of linear Diophantine equations may be reduced to the problem of diagonalizing an integer matrix, as discussed in [37]. A program based on the method in [37, page 179] for finding the elementary divisors of an integer matrix, along with the associated change of

basis matrices, has been developed by the author [EDNF16.C]. Frumkin [23] and von zur Gathen and Sieveking [25] have presented polynomial time algorithms that solve this problem. Improved algorithms are given in [40,17,35].

We again use a Pascal–like syntax for presenting the algorithm. Words in G are represented as sequences of syllables, where a syllable is either b to some power, or a stable letter or its inverse. For example, b^3, a^{-1}, b^{-2}, a is an allowable sequence, but a^{-2}, b^4, a^2 is not; the latter sequence is represented as $a^{-1}, a^{-1}, b^4, a, a$.

The algorithm uses the `syllable` and `group_word` data objects that have been implemented in [HNNZ7.PAS]. Associated with these data objects are the following functions and methods:

- Member functions and procedures:

  `w.syllable_length` is the number of syllables in the group_word w.

  `w.cycle` cyclically permutes w; the first syllable of w is removed and placed at the end of w.

  `w.P_Reduce` applies all possible free reductions and P–reductions to w. Its implementation requires only simple pattern-matching and substitution.

  `w.syllable(n)` is the n^{th} syllable of w.

- Predicates:

  `u.P_Reduced` is true if and only if u is P_reduced.

  `Conjugate(u,v)` is true if and only if $u \sim v$.

`StableLetter(s)` is true if and only if the syllable `s` is a stable letter, or the inverse of a stable letter.

`P_parallel( u,v )` is true if and only if u and v are P_parallel.

2.4.1 Collins' Lemma

As mentioned earlier, Collins' Lemma may be seen to describe a process for deciding whether two words in G are conjugate. The functions `Conjugate` and `Collins` taken together implement this procedure. By Collins' Lemma, if u and v are conjugate and P–cyclically reduced, then $\|u\| = \|v\|$; i.e., they have the same stable length. If $u \sim v$ and u contains a stable letter, Collins' Lemma further asserts that u is conjugate to some cyclic permutation of v that is P–parallel to u, by an element of the base group — in this case, $\langle\, b\,;\,\rangle$. The role of function `Collins` is to consider all the cyclic permutations of v that are P–parallel to u, and for each P–parallel pair u,v, determine if the group equation $u = b^{-n}vb^{n}$ has a solution $n \in Z$. `Collins(x,y)` is true if and only if there is a solution corresponding to at least one of these permutations.

Collins' Lemma does not apply in case $\|u\| = 0$. This case, however, has been shown by Anshel [5] to be reducible to the *reachability problem for self-dual vector addition systems*, which is reducible to the word problem for commutative semigroups, and which, in turn, has been shown by Mayr and Meyer to be reducible to the problem of deciding membership in a polynomial ideal [49].

This last problem is dealt with extensively in the chapter on Gröbner bases. The function `reachable`, which will be discussed in detail below,

handles the reduction of the $\|u\| = 0$ case to a problem of deciding membership in a polynomial ideal.

```
function Conjugate(u,v :group_word) :boolean;
(*  Given u,v ∈ G, is u ~ v?    *)
var x,y :group_word;
begin
    x := P_Cyclically_Reduce(u);
    y := P_Cyclically_Reduce(v);
    if ||x|| ≠ ||y|| then
        Conjugate := false    (* by Collins' Lemma *)
    else
        if ||x|| = 0 then
            Conjugate := Reachable(x,y)
        else
            Conjugate := Collins(x,y)
end;


function Collins(x,y :group_word) :boolean;
var i,n :integer;
    solution_found :boolean;
begin
    n := y.syllableLength;
    i := 0;
    solution_found := false;
    while ( (i < n) and (not solution_found) ) do
    begin
        if StableLetter( y.syllable(1) ) then
```

```
        begin
            if P_parallel( x,y ) then
                if exponent_equation( x,y ) then
                    solution_found := true
        end
        else begin
            y.cycle;
            i := i + 1;
        end
    end
end;
```

In "P_Cyclically_Reduce" below, N counts the number of cyclic permutations of u that have been verified to be P–reduced. Clearly, if the while loop terminates, then P_Cyclically_Reduce is correct, by definition of cyclically reduced. To see that the loop terminates, note that whenever N is reset to 0, $\|u\|$ decreases by 1.

```
function P_Cyclically_Reduce(u :group_word):group_word
var N :integer;
    x :group_word
begin
    x := u;
    while N < x.syllableLength do
    begin
        if x.P_Reduced then
        begin
            N := N + 1
```

```
            x.cycle
        end
        else begin
            x.P_reduce;
            N := 0;
        end;
    end;
    P_Cyclically_Reduce := x;
end;
```

2.4.2 Case 1: The "Exponent Equations"

We proceed to derive necessary conditions for u and v to be conjugate by an element of G, where u and v are P–cyclically reduced, P–parallel elements of G, with $\|u\| = \|v\| > 0$.

Suppose $u = b^n v b^{-n}$. Let

$$u = a_{i_m}^{e_m} b^{x_m} \cdots a_{i_1}^{e_1} b^{x_1} = a_{i_m}^{e_m} b^{x_m} \bar{u}$$

and

$$v = a_{i_m}^{e_m} b^{y_m} \cdots a_{i_1}^{e_1} b^{y_1} = a_{i_m}^{e_m} b^{y_m} \bar{v}.$$

Then

$$\bar{u}^{-1} b^{-x_m} a_{i_m}^{-e_m} b^n a_{i_m}^{e_m} b^{y_m} \bar{v} b^{-n} = 1,$$

and so by Britton's lemma, a P–reduction is possible. Since u and v are P–reduced, the only subword that can be pinched is $a_{i_m}^{-e_m} b^n a_{i_m}^{e_m}$, and therefore

$$n = \begin{cases} p_{i_1} s_i \text{ and } \bar{u}^{-1} b^{q_s - x_m + y_m} \bar{v} b^{-n} = 1 & \text{if } e_1 = 1, \text{ and} \\ q_{i_1} s_i \text{ and } \bar{u}^{-1} b^{p_s - x_m + y_m} \bar{v} b^{-n} = 1 & \text{if } e_1 = -1. \end{cases}$$

Define

$$h_j = \begin{cases} p_{i_j} & \text{if } e_j = 1 \\ q_{i_j} & \text{if } e_j = -1 \end{cases} \qquad \text{and} \qquad \tilde{h}_j = \frac{p_{i_j} q_{i_j}}{h_j}.$$

We define sequences n_i and s_i, $(i = 0, \ldots, m)$ Let $n_m = n$. Then

$$\begin{aligned} u &= b^{n_m} v\, b^{-n} \text{ and} \\ b^{x_m} \bar{u} &= a_{i_m}^{-e_m} b^{n_m} a_{i_m}^{e_m} b^{y_m} \bar{v}. \end{aligned}$$

Again, applying Britton's lemma, we must have $n_m = h_m s_m$, for some integer s_m, and since $a_{i_m}^{-e_m} b^{n_m} a_{i_m}^{e_m} = b^{\tilde{h}_m s_m}$,

$$\bar{u} = b^{\tilde{h}_m s_m - x_m + y_m} \bar{v}.$$

Define $n_{m-1} = \tilde{h}_m s_m - x_m + y_m$; then

$$\bar{u} = b^{n_{m-1}} \bar{v} b^{-n}.$$

After m iterations of this process, all stable letters will have been removed, at which point we obtain $\bar{u} = \bar{v} = 1$, $n_1 = h_1 s_1$, $n_0 = \tilde{h}_1 s_1 - x_1 + y_1$, and since

$$1 = b^{\tilde{h}_1 s_1 - x_1 + y_1} \cdot 1 \cdot b^{-n},$$

$n_0 = n$.

Pulling all this together, we arrive at the following system of $2m+2$ linear integer equations in $n, n_0, n_1, \ldots n_m,\ s_1, \ldots s_m$:

$$\begin{aligned}
n_m &= n \\
n_m &= h_m s_m \\
n_{m-1} &= \tilde{h}_m s_m - x_m + y_m \\
&\vdots \\
n_2 &= h_2 s_2 \\
n_1 &= \tilde{h}_2 s_2 - x_2 + y_2 \\
n_1 &= h_1 s_1 \\
n_0 &= \tilde{h}_1 s_1 - x_1 + y_1 \\
n_0 &= n.
\end{aligned}$$

By taking these equations in pairs, we may eliminate all the n_i, yielding the following system of $m+1$ equations in $n,\ s_1, \ldots s_m$:

$$\begin{aligned}
\tilde{h}_1 s_1 - n &= x_1 - y_1 \\
\tilde{h}_2 s_2 - h_1 s_1 &= x_2 - y_2 \\
&\vdots \\
\tilde{h}_m s_m - h_{m-1} s_{m-1} &= x_m - y_m \\
n - h_m s_m &= 0,
\end{aligned}$$

Expressed more conveniently in terms of matrices,

$$\begin{pmatrix} \tilde{h}_1 & 0 & 0 & 0 & \dots & -1 \\ -h_1 & \tilde{h}_2 & 0 & 0 & \dots & 0 \\ 0 & -h_2 & \tilde{h}_3 & 0 & \dots & 0 \\ \vdots & & \ddots & \ddots & & \vdots \\ 0 & 0 & \dots & -h_{m-1} & \tilde{h}_m & 0 \\ 0 & 0 & \dots & 0 & -h_m & 1 \end{pmatrix} \cdot \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ \\ s_m \\ n \end{pmatrix} = \begin{pmatrix} x_1 - y_1 \\ x_2 - y_2 \\ \vdots \\ \\ x_m - y_m \\ 0 \end{pmatrix}.$$

We remark that $h_j, \tilde{h}_j, x_i$, and y_i are determined at the outset, by inspection of the syllables of the words u, v.

Finally, the value of the function exponent_equation (which is used in function Collins) is "true" if and only if this system has a solution $(s_1, \dots s_m,\ n) \in Z^{m+1}$.

2.4.3 Case 2: The Reachability Problem

We now consider the case in which $\|x\| = \|y\| = 0$, and show how the conjugacy problem in this case is reducible to the word problem for finitely presented commutative semigroups. Suppose $x = b^m$, $y = b^n$, for some integers m, n; we wish to determine if there exists $z \in G$ such that $x = z^{-1}yz$. We first show that if such a z exists, then we may assume that it is a freely reduced word in the stable letters.

Lemma 11 *If $b^m = z^{-1}b^n z$, for some $z \in G$ then there exists $\bar{z}$ such that either $\bar{z} = 1$, or else $\bar{z}$ is a freely reduced word in the stable letters and their inverses, and $x = \bar{z}^{-1}y\bar{z}$.*

Proof: We prove this by induction on $\|z\|$. If $\|z\| = 0$, then $z = b^e$ for some $e \in Z$, and so we may choose $\bar{z} = 1$, since $yz = zy$. Suppose $\|z\| \geq$

1, $x = z^{-1}yz$, and assume the conclusion of the lemma for all $\tilde{z} \in G$ such that $\|\tilde{z}\| < \|z\|$. Freely reduce z. If this decreases $\|z\|$, then apply the induction hypothesis. We may assume then that z is freely reduced, and in addition, that z begins with a stable letter. For, if $z = b^e z'$ for some $e \neq 0$, where z' begins with a stable letter, then $b^m = z'^{-1}b^{-e}b^n b^e z' = z'^{-1}b^n z$, and so we may trivially replace z by z'.

Write $z = aw$, where a is a stable letter. Now, $w^{-1}a^{-1}b^n awb^{-m} = 1$, and so by Britton's lemma, this is not reduced, and some P–reduction must be applicable. Since z is reduced, w and w' are also reduced, and so the only subword to which a P–reduction can apply is $a^{-1}b^n a$. Suppose $a^{-1}b^n a = b^x$. Then $b^m = w^{-1}b^x w$, and since $\|w\| = \|z\| - 1$, there exists $\bar{w}$ such that $b^m = \bar{w}^{-1}b^x\bar{w}$, and $\bar{w}$ is a word in the stable letters. Therefore,

$$b^m = \bar{w}^{-1}a^{-1}b^n a\bar{w} = (a\bar{w})^{-1}b^n a\bar{w}.$$

The result follows by setting $\bar{z} = a\bar{w}$.

We refer to [6] for the following definitions.

Definition: For a group G in HNN(Z), the set of *exponents* of

$$G = \langle\ b, a_i;\ a_i^{-1}b^{p_i}a_i = b^{q_i}\ (i = 1, \ldots, k)\ \rangle$$

is the set of integers $\{p_1, q_1, \ldots, p_k, q_k\}$. If m, n are positive integers, then we say that *m is reachable from n with respect to the exponents of G* if either $m = n$, or else, if there exists a sequence of integers $n = n_0, n_1, \ldots, n_\alpha = m$ with the property that $(\forall i\ (0 \leq i < \alpha),\ \exists j)$ such that either

$$p_j | n_i, \text{ and } n_{i+1} = n_i \frac{q_j}{p_j}, \text{ or}$$

$$q_j | n_i, \text{ and } n_{i+1} = n_i \frac{p_j}{q_j}.$$

Equivalently, m is reachable from n if and only $m = n$, or else, if there exists a sequence $i_1, \ldots, i_\alpha$ such that

$$m = n \cdot \left(\frac{q_{i_1}}{p_{i_1}}\right)^{\epsilon_1} \cdots \left(\frac{q_{i_\alpha}}{p_{i_\alpha}}\right)^{\epsilon_\alpha} \quad \text{where} \quad \epsilon_j = \pm 1,$$

and, for $1 \leq j \leq \alpha$,

$$n \cdot \prod_{k=1}^{j} \left(\frac{q_{i_k}}{p_{i_k}}\right)^{\epsilon_k} \in Z.$$

Lemma 12 *$b^m \sim b^n$ if and only if m is reachable from n with respect to the exponents of G.*

Proof: We observe that a word $a_j^{-1} b^n a_j$ may be P–reduced if and only if $p_j | n$, in which case $a_j^{-1} b^n a_j = b^{n \cdot q_j / p_j}$. Similarly, $a_j b^n a_j^{-1}$ may be P–reduced if and only if $q_j | n$, in which case $a_j b^n a_j^{-1} = b^{n \cdot p_j / q_j}$.

If m is reachable from n, and

$$m = n \cdot \left(\frac{q_{i_1}}{p_{i_1}}\right)^{\epsilon_1} \cdots \left(\frac{q_{i_\alpha}}{p_{i_\alpha}}\right)^{\epsilon_\alpha} \quad (\epsilon_j = \pm 1),$$

with $n \cdot \prod_{k=1}^{j} \left(\frac{q_{i_k}}{p_{i_k}}\right)^{\epsilon_k} \in Z \quad (1 \leq j \leq \alpha)$, construct z as follows: $\|z\| = \alpha$, and the i^{th} letter of z is $a_i^{\epsilon_i}$. By the observation above, $b^m = z^{-1} b^n z$.

Now, suppose $b^m \sim b^n$. By Lemma 11, we may assume that $b^m = z^{-1} b^n z$, where z is a freely reduced word in the stable letters and their inverses. If $\|z\| = 0$, then $m = n$, and there is nothing to show. Otherwise, we may assume, without loss of generality, that $z^{-1} a^{-1} b^n a z b^{-m} = 1$, for some stable letter a corresponding to the relator $a^{-1} b^p a = b^q$. By Britton's lemma, some pinch is possible. The only possible subword that may be pinched is $a^{-1} b^n a$, and therefore, $p | n$, and $z^{-1} b^{n \cdot q/p} z b^{-m} = 1$. We may repeat this for each stable letter, and the result follows. ∎

We now reduce the reachability problem to a word problem in a commutative semigroup using a technique reported in [52] attributed to a personal communication from A. Rosenberg.

Let $D = \{d_1, \ldots, d_\nu\}$ be the set of the prime divisors of the set of exponents of G, and define $A: \mathcal{N} \to \mathcal{N}^\nu$ by

$$A(x) = \begin{cases} (e_1, \ldots, e_\nu) & \text{if } x = \prod_{i=1}^{\nu} d_i^{e_i} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Let $\hat{d} = \prod_{i=1}^{\nu} d_i$, and define $\Delta : \mathcal{N} \to \mathcal{N}$ by

$$\Delta(x) = \begin{cases} 1 & \text{if } x = 0 \\ x/\mathrm{L(x)} & \text{if } x > 0, \end{cases}$$

where $\mathrm{L}(x)$ is the greatest integer dividing x which is relatively prime to $\hat{d}$.

Formally, $\mathrm{L}(x\mathrm{L}(x) = \begin{cases} x & \\ \mathrm{L}(x/\gcd(x,\hat{d}) & \end{cases}$

$$\mathrm{L}(x) = \begin{cases} x & \text{if } \gcd(x,\hat{d}) = 1 \\ \mathrm{L}(x/\gcd(x,\hat{d}) & \text{otherwise.} \end{cases}$$

A simple example will serve to illustrate these definitions. Suppose $G = \langle\, b, a_1, a_2;\ a_1^{-1}b^2a_1 = b^3,\ a_2^{-1}b^2a_2 = b^5 \,\rangle$. Then $D = \{2,3,5\}, \hat{d} = 30$, and $\Delta(700) = 100$, since $\mathrm{L}(700) = \mathrm{L}(70) = \mathrm{L}(7) = 7$.

Definition: The commutative semigroup presentation P over the alphabet $\{x_1, \ldots, x_\nu\}$, associated with G, is given as follows: For any $z \in Z$ such that $z > 0$, and $z = \Delta(z) = \prod_{i=1}^{\nu} d_i^{e_i}$, define

$$\mathrm{W}(z) = x_1^{e_1} \cdots x_\nu^{e_\nu}.$$

(Recall that ν is the number of distinct prime divisors of the exponents of G, and so for $i = 1, \ldots, k$, $\mathrm{W}(p_i)$ and $\mathrm{W}(q_i)$ are defined.) P contains, for each relator $a^{-1}b^{p_i}a = b^{q_i}$, the productions $\mathrm{W}(p_i) \to \mathrm{W}(q_i)$ and $\mathrm{W}(q_i) \to \mathrm{W}(p_i)$.

Theorem 14 establishes the reducibility of the conjugacy problem for this case ($\|x\| = 0$) to the word problem for P.

Theorem 14 *$b^m \sim b^n$ in G if and only if $L(m) = L(n)$, and $W(\Delta(m)) \equiv W(\Delta(n))$.*

To prove Theorem 14, we first prove two more lemmas.

Lemma 13 *m is reachable from n if and only if $L(m) = L(n)$, and $\Delta(m)$ is reachable from $\Delta(n)$.*

Proof: m is reachable from n if and only if

$$m = n \cdot \prod_{i=1}^{\alpha} \left(\frac{p_i}{q_i}\right)^{\epsilon_i}$$

if and only if

$$\Delta(m)\, L(m) = \Delta(n)\, L(n) \cdot \prod_{i=1}^{\alpha} \left(\frac{p_i}{q_i}\right)^{\epsilon_i}$$

if

$$\Delta(m) = \Delta(n) \cdot \prod_{i=1}^{\alpha} \left(\frac{p_i}{q_i}\right)^{\epsilon_i} \text{ and } L(m) = L(n).$$

Now, we only need to show that if

$$\Delta(m)\, L(m) = \Delta(n) \cdot \prod_{i=1}^{\alpha} \left(\frac{p_i}{q_i}\right)^{\epsilon_i} L(n),$$

then $L(m) = L(n)$. Now, $\gcd(L(m), \hat{d}) = 1$, and $L(m)$ divides the left hand side, so $L(m)|L(n)$. Since "reachable" is a symmetric relation, a similar argument implies that $L(n)|L(m)$, and so $L(n) = \pm L(m)$. Since $L(n)$ and $L(m)$ are positive, $L(n) = L(m)$.

Lemma 14 *Let* $\Delta(m) = m, \Delta(n) = n$. *Then* $\mathrm{W}(m) \equiv \mathrm{W}(n)$ *if and only if* m *is reachable from* n.

Proof: Essentially, this is just another way of saying that a production corresponds directly to a "reachable" step. Note that since $\Delta(m) = m$ and $\Delta(n) = n$, $\mathrm{W}(m)$ and $\mathrm{W}(n)$ are defined.

Let $\mathrm{W}(m) = x_1^{e_1} \cdots x_\nu^{e_\nu}$, $\mathrm{W}(p) = x_1^{f_1} \cdots x_\nu^{f_\nu}$ and $\mathrm{W}(q) = x_1^{g_1} \cdots x_\nu^{g_\nu}$, and let $\rightarrow_f$ denote the production $\mathrm{W}(p) \rightarrow \mathrm{W}(q)$ in $\mathcal{P}$.

$\mathrm{W}(m) \rightarrow_f \mathrm{W}(\bar{m})$ if and only if for $i = 1, \ldots, \nu$, $e_i \geq f_i$ and $\mathrm{W}(\bar{m}) = x_1^{e_1 - f_1 + g_1} \ldots x_\nu^{e_\nu - f_\nu + g_\nu}$, if and only if $\bar{m} = m \cdot \frac{q}{p}$ and $p | m$, if and only if $\bar{m}$ is reachable from m in one step.

We now easily obtain the proof of Theorem 14.

Proof: $b^m \sim b^n$ in $G \iff m$ is reachable from n (Lemma 12)
$\iff \Delta(m)$ is reachable from $\Delta(n)$, and $\mathrm{L}(m) = \mathrm{L}(n)$ (Lemma 13)
$\iff \mathrm{W}(\Delta(m)) \equiv \mathrm{W}(\Delta(n))$, and $\mathrm{L}(m) = \mathrm{L}(n)$ (Lemma 14). ∎

By Theorem 14, the conjugacy problem is reduced to the problem of factoring the exponents of G and deciding the word problem in $\mathcal{P}$. We observe that this factorization and calculation of a Gröbner basis corresponding to $\mathcal{P}$ depends only on the group G, and not on the input words.

2.5 The Conjugacy Problem: Examples.

We now consider two examples to illustrate the algorithm given for solving the conjugacy problem. Let

$$G = \langle b,\ a, c, d;\ a^{-1}b^{20}a = b^{21},\ c^{-1}b^{18}c = b^{25},\ d^{-1}b^{15}d = b^{14}\ \rangle.$$

Example 1. Decide whether $w_1 \sim w_2$, where

$$\begin{aligned} w_1 &= c\, b^{12}\, d^{-1}\, b^{10}\, c\, b^{-20}\, d^{-1}, \quad \text{and} \\ w_2 &= c\, b^{3}\, d^{-1}\, b^{-12}\, c\, b^{-10}\, d^{-1}\, b. \end{aligned}$$

w_1 and w_2 are P–cyclically reduced, P–parallel, and since $\|u\| > 0$, Case (1) applies. We observe immediately that there is one additional cyclic permutation of w_2 that is P–parallel to w_1,

$$w_3 = c\, b^{-10}\, d^{-1}\, b\, c\, b^{3}\, d^{-1}\, b^{-12}.$$

If w_1 and w_2 are conjugate, then the system

$$A \cdot \begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \\ n \end{pmatrix} = \begin{pmatrix} 9 \\ 22 \\ -10 \\ -1 \\ 0 \end{pmatrix} \qquad (\star)$$

must have an integer solution, where

$$A = \begin{pmatrix} 25 & 0 & 0 & 0 & -1 \\ -18 & 15 & 0 & 0 & 0 \\ 0 & -14 & 25 & 0 & 0 \\ 0 & 0 & -18 & 15 & 0 \\ 0 & 0 & 0 & -14 & 1 \end{pmatrix}.$$

Using the diagonalization procedure described in [37, page 179], and implemented in [EDNF16.C], we determine the integer matrices

$$P = \begin{pmatrix} 5 & 7 & 0 & 0 & 0 \\ 90 & 125 & 134 & 0 & 0 \\ 252 & 350 & 375 & 521 & 1 \\ -252 & -350 & -375 & -521 & 0 \\ 1512 & 2100 & 2250 & 3125 & 1512 \end{pmatrix} \quad \text{and}$$

$$Q = \begin{pmatrix} 1 & 105 & -29556455 & -88317615 & 502518380 \\ 0 & 1 & -281490 & -841120 & 4785890 \\ 0 & 0 & -84 & -251 & 1429 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 3 & 14 \end{pmatrix}$$

so that

$$P\,A\,Q = \begin{pmatrix} -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 25707 \end{pmatrix} = D.$$

Since P and Q are the products of elementary matrices, P and Q are themselves unimodular, and therefore invertible over Z. Now,

$$A \cdot \vec{x} = \vec{c} \iff PAQ \cdot Q^{-1}\vec{x} = P\vec{c},$$

and therefore, $(\star)$ has an integer solution if and only if $D\,\vec{y} = \vec{e}$ does, where

$$\vec{y} = Q^{-1} \cdot \begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \\ n \end{pmatrix} \quad \text{and} \quad \vec{e} = P \cdot \begin{pmatrix} 9 \\ 22 \\ -10 \\ -1 \\ 0 \end{pmatrix} = \begin{pmatrix} 199 \\ 2220 \\ 5697 \\ -5697 \\ 34183 \end{pmatrix}.$$

Since $34183 \bmod 25707 = 8476$, $D\,\vec{y} = \vec{e}$ has no integer solution, and so neither does $(\star)$.

We now consider the remaining cyclic permutation w_3 of w_2 that is P–parallel to w_1. If w_1 and w_3 are conjugate, then the system

$$A \cdot \begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \\ n \end{pmatrix} = \begin{pmatrix} 22 \\ 9 \\ -17 \\ 12 \\ 0 \end{pmatrix} \qquad (\star\star)$$

must have an integer solution. A and the diagonalizing matrices P and Q are the same as in the previous case, and so $(\star\star)$ has a solution if and only if

$$D \cdot \vec{y} = P \cdot \begin{pmatrix} 22 \\ 9 \\ -17 \\ 12 \\ 0 \end{pmatrix} = \begin{pmatrix} 173 \\ 827 \\ 8571 \\ -8571 \\ 51414 \end{pmatrix}$$

does. This system has a solution

$$\vec{y} = \begin{pmatrix} -173 \\ -827 \\ 8571 \\ -2857 \\ 2 \end{pmatrix} = Q^{-1} \cdot \begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \\ n \end{pmatrix}$$

and so w_1 and w_3 are conjugate elements in G. The solution to $(\star\star)$ is

$$\begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \\ n \end{pmatrix} = Q \cdot \begin{pmatrix} -173 \\ -827 \\ 8571 \\ -2857 \\ 2 \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \\ 1 \\ 2 \\ 28 \end{pmatrix},$$

and therefore, $w_1 = b^{28}\, w_3\, b^{-28}$.

Finally, since $w_3 = (c\, b^3\, d^{-1}\, b^{-12})^{-1}\, w_2\, (c\, b^3\, d^{-1}\, b^{-12})$,

$$w_1 = b^{28}\, (c\, b^3\, d^{-1}\, b^{-12})^{-1}\, w_2\, (c\, b^3\, d^{-1}\, b^{-12})\, b^{-28}.$$

Example 2. Decide whether $w_1 \sim w_2$, where

$$\begin{aligned} w_1 &= b^{1250} \quad \text{and} \\ w_2 &= b^{1225}. \end{aligned}$$

As in Example 1, w_1 and w_2 are P–parallel and P–cyclically reduced, but since $\|u\| = 0$, Case (2) applies.

The set of prime divisors of the set of exponents of G is $\{2,3,5,7\}$. For the associated semigroup presentation $\mathcal{P}$, we choose generators u,x,y,z corresponding, respectively, to 2,3,5,7, and so

$$\mathcal{P} = \langle u,x,y,z;\ u^2y = xz,\ ux^2 = y^2,\ uz = xy \rangle.$$

By Theorem 14, $w_1 \sim w_2$ if and only if $uy^4 \equiv y^2z^2\ (\mathcal{P})$. We use the method of Gröbner bases to decide the word problem in $\mathcal{P}$.

The polynomial ideal in $\mathcal{Q}[u,x,y,z]$ associated with the commutative semigroup $\mathcal{P}$ is

$$I = \text{Ideal}(\ u^2y - xz, ux^2 - y^2, uz - xy\ ).$$

The reduced Gröbner basis for this ideal, produced by [GROBNER2.PAS] is:

$$\begin{aligned} B_1 &= u^2y - xz \\ B_2 &= ux^2 - y^2 \\ B_3 &= uz - xy \\ B_4 &= uy^3 - x^3z \\ B_5 &= uxy^2 - xz^2 \end{aligned}$$

$$\begin{aligned} B_6 &= x^3y - y^2z \\ B_9 &= x^2z^2 - y^4 \\ B_{10} &= x^2y^3 - xz^3 \\ B_{11} &= x^4z - xyz^2 \\ B_{12} &= xy^5 - y^2z^3 \\ B_{13} &= y^7 - xz^5. \end{aligned}$$

Now, 1225 is reachable from 1250 with respect to the exponents of G if and only if $uy^4 \equiv y^2z^2\ (P)$, by Lemma 14.

The Gröbner basis for I provides the following reduction of $y^2z^2 - uy^4$ to normal form:

$$y^2z^2 - uy^4 \ \rightarrow_{B4}\ y^2z^2 - x^3yz \ \rightarrow_{B6}\ y^2z^2 - y^2z^2 = 0.$$

By Theorem 5, $y^2z^2 - uy^4 \in I$, and so by Lemma 8, $y^2z^2 \equiv uy^4\ (P)$. Therefore, w_1 and w_2 are conjugate elements of G.

We now proceed to determine $z \in G$ such that $w_1 = z^{-1}\, w_2\, z$. As has already been shown in Section 1.4.1, by Theorem 6, $y^2z^2 - uy^4$ may be expressed as a $\mathcal{Q}[u,x,y,z]$–linear combination of $\{B_1, B_2, B_3\}$:

$$y^2z^2 - uy^4 = x^2y \cdot B_1 + (z^2 - uy^2) \cdot B_2 - zx^2 \cdot B_3.$$

Applying the method used in the proof of Lemma 8, we determine the following sequence of productions in P:

$$uy^4 \rightarrow u^2x^2y^2 \rightarrow x^3yz \rightarrow ux^2z^2 \rightarrow y^2z^2.$$

From this, we obtain

$$2 \cdot 5^4 \times \frac{2 \cdot 3^2}{5^2} \times \frac{3 \cdot 7}{2^2 \cdot 5} \times \frac{2 \cdot 7}{3 \cdot 5} \times \frac{5^2}{2 \cdot 3^2} = 5^2 \cdot 7^2,$$

and therefore,

$$\begin{aligned} w_1 &= c\,d^{-1}\,a\,c^{-1}\,w_2\,c\,a^{-1}\,d\,c^{-1} \\ &= (c\,a^{-1}\,d\,c^{-1})^{-1}\,w_2\,(c\,a^{-1}\,d\,c^{-1}). \end{aligned}$$

Chapter 3

Formal Languages

3.1 The Chomsky Hierarchy

Let Σ be a finite set, called the *alphabet*, and let Σ^* be the set of all finite words over Σ, including the empty word λ. The *length* of a word w, $|w|$, is defined to be the total number of symbols in w. Formally, $|\ | : \Sigma^* \rightarrow \mathbf{N}$ is defined by: $|\lambda| = 0, |a| = 1$ $(\forall a \in \Sigma)$, and $|aw| = 1 + |w|$ $(\forall a \in \Sigma,\ w \in \Sigma^*)$.

A *language over* Σ is any subset of Σ^*. Languages may be specified in terms of *grammars*, or *automata*. One focus of formal language theory is to study the classes of languages obtained from restricted grammars and from restricted automata, and to study correspondences between classes so obtained.

3.1.1 Grammars

A *Phrase Structure Grammar*, also known as an *unrestricted* grammar, is a 4-tuple $G = (\Sigma, T, P, S)$, where Σ is a finite alphabet, T (the set of *terminals*) is a subset of Σ, P (the set of *productions*) is a subset of $\Sigma^* \times \Sigma^*$, and S

(the start symbol) is an element of $\Sigma \setminus T$. Elements of $\Sigma \setminus T$ are called *nonterminals.*

If (α, β) is a production, we will write $\alpha \rightarrow \beta$. If u,v are words in Σ^*, we write $u \rightarrow v$ if and only if there exists a production $\alpha \rightarrow \beta$ in P and words $x, y \in \Sigma^*$, such that $u = x\alpha y, \quad v = x\beta y$. We denote by $\rightarrow^*$ the reflexive transitive closure of the relation defined by P.

For a grammar Γ we define $L(\Gamma)$, the language *generated* by Γ, to be the set $\{w : w \in T^* \text{ and } S \rightarrow^* w\}$. If Γ is a phrase structure grammar, $L(\Gamma)$ is called a *recursively enumerable (r.e.)* language. $L(\Gamma)$ is said to be a *recursive* language if both $L(\Gamma)$ and its complement with respect to words in the terminal symbols ($T^* \setminus L(\Gamma)$) are r.e. languages. If every production of a grammar Γ is of the form $L \rightarrow R$ where the length of L is at most the length of R, then Γ is called *Context-Sensitive*, and $L(\Gamma)$ is called a *context-sensitive* language (CSL). The term "context-sensitive" is given due to the fact that every CSL is generated by a grammar in which every production is of the form $xAy \rightarrow xwy$, where A is a nonterminal, x, y, w are words in Σ^*, and w is not the empty word. Γ is a *context-free grammar* if every production is of the form $A \rightarrow w$, where A is a nonterminal, and $w \in \Sigma^*$; i.e., a non-terminal A may be replaced regardless of the "context" in which it appears.

In a context-free grammar Γ, if the right-hand-side of every rule contains at most one non-terminal, which, if present, is the last symbol of the string, then the grammar is called *regular*, and $L(\Gamma)$ is called a *regular* language.

A theorem due to Kleene characterizes the class of regular languages over an alphabet T as the minimal set of languages which contains the empty set and all the singleton sets $\{a\}$, where $a \in T$, and which is closed under the operations of union (denoted by $\cup$), concatenation (denoted by juxtaposi-

tion), and *Kleene star.* The Kleene star L^* of a language L is the set of all strings that can be obtained by concatenating zero or more strings from L. An immediate consequence of Kleene's theorem is that all finite sets are regular.

Regular languages may be described by *regular expressions.* We define regular expressions over Σ, and the languages they denote as follows:

- $\emptyset$ and λ are regular expressions, denoting the empty set and the set $\{\lambda\}$, respectively.
- $\forall a \in \Sigma$, a is a regular expression, denoting the set $\{a\}$.
- If r, s are regular expressions denoting the languages R and S, respectively, then $(r + s), (rs)$, and (r^*) are regular expressions, denoting the sets $R \bigcup S$, RS, and R^*, respectively.

The precedence of the operations is, in order: *, concatenation, and $+$, with $+$ having the lowest precedence. This convention enables us to non-ambiguously interpret a regular expression in which some pairs of parentheses have been omitted.

The regular sets, context-free languages, context-sensitive languages, and the recursively enumerable languages, form a properly ascending chain of language classes, sometimes referred to as the Chomsky Hierarchy. These language classes may also be categorized in terms of automata, or *machines.*

3.1.2 Finite State Automata

The definitions for the formal computational models are based on those in [32].

Definition: A *finite automaton*, or *finite state machine*, is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where:

1. Q is a finite set of *states*,
2. Σ is a finite *input alphabet*,
3. $q_0 \in Q$ is the *initial state*,
4. $F \subseteq Q$ is the set of *final states*, and
5. $\delta : Q \times \Sigma \to Q$ is the *state transition function.*

Informally, one may consider a finite automaton to be a device with a fixed control mechanism, and which at any given moment is in one of a finite number of states. This device begins its operation in a known initial state, and processes a sequence of input symbols, one at a time. As each input symbol is received, the device responds by changing its state according to the rules contained in its finite control. The machines we are considering have no output; the focus is on whether a given string of symbols causes the machine to reach a final, or *accepting*, state.

δ is extended to a function $\hat{\delta} : Q \times \Sigma^* \to Q$ as follows:

- $\hat{\delta}(q, \lambda) = q$, and
- $\hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w),\ a)$, for all $w \in \Sigma^*,\ a \in \Sigma$.

Since $\hat{\delta}(q, a) = \delta(\hat{\delta}(q, \lambda), a)\delta(q, a)$, δ and $\hat{\delta}$ agree on arguments for which they are both defined, and so may be represented by the same symbol. For convenience, we will write δ instead of $\hat{\delta}$.

A finite automaton is said to be *deterministic*, in the sense that the "next state" function δ is in fact a function.

Definition: A *nondeterministic finite automaton*, or *NFA*, is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where Q, Σ, q_0, F have the same meaning as for a finite automaton, but $\delta : Q \times \Sigma \to 2^Q$, where 2^Q is the power set of Q.

As in the deterministic case, we extend δ to a function $\hat{\delta}$: $Q \times \Sigma^* \to 2^Q$, defined on strings, as follows:

- $\hat{\delta}(q, \lambda) = \{q\}$, and
- $\hat{\delta}(q, wa) = \{p \mid p \in \delta(r, a) \text{ for some } r \in \hat{\delta}(q, w)\}$.

Again, since $\hat{\delta}(q, a) = \delta(q, a)$, we will use the simpler notation δ rather than $\hat{\delta}$.

A string x is *accepted* by a finite automaton $M = (Q, \Sigma, \delta, q_0, F)$ if $\delta(q_0, x) \in F$. The *language accepted by* M is the set $L(M) = \{x | \delta(q_0, x) \in F\}$; i.e., the set of all strings accepted by M.

The original use of finite state machines is attributed to McCulloch and Pitts, who in 1943, used these systems to model neural nets. Kleene proved the equivalence of regular expressions and the neural nets of McCulloch and Pitts in 1956. Chomsky and Miller [16] showed that the languages generated by regular grammars are the same as the languages accepted by finite automata.

Nondeterministic finite automata were introduced in 1959 by Rabin and Scott, who proved that deterministic and nondeterministic finite automata were equivalent, in the sense that they both accepted precisely the class of regular languages.

3.1.3 Pushdown Automata

The pushdown automaton was introduced by Oettinger [54], and Schutzenberger [57]. It may be thought of as a nondeterministic finite state machine which may manipulate an internal *stack*, or "last-in–first-out" queue. State transitions may be made based on both the input symbol and on the symbol on the top of the stack. Formally:

Definition: A *pushdown automaton* M is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, where:

1. Q is a finite set of *states*,
2. Σ is a finite *input alphabet*,
3. Γ is a finite *stack alphabet*,
4. $q_0 \in Q$ is the *initial state*,
5. $Z_0 \in \Gamma$ is the *start symbol*,
6. $F \subseteq Q$ is the set of *final states*, and
7. δ is a mapping from $Q \times (\Sigma \cup \{\lambda\}) \times \Gamma$ into finite subsets of $Q \times \Gamma^*$.

δ may be thought of as defining two types of transitions. The first type, $\delta(q, a, Z) = \{(p_1, \gamma_1), (p_2, \gamma_2), \ldots, (p_m, \gamma_m)\}$ is interpreted as meaning that if M is in state q, currently scanning input symbol a, with Z the top symbol on the stack, then for some i $(1 \leq i \leq m)$, M may enter state p_i, replace symbol Z on the stack by $\gamma_i \in \Gamma^*$, and advance the input head to the next symbol.

The second type permits M to simply manipulate the stack without advancing the input head. $\delta(q, \lambda, Z) = \{(p_1, \gamma_1), (p_2, \gamma_2), \ldots, (p_m, \gamma_m)\}$ is interpreted to mean that if M is in state q, and Z is the top symbol on the stack, then for some i $(1 \leq i \leq m)$, M may enter state p_i, and replace symbol Z on the stack by $\gamma_i \in \Gamma^*$.

There are 2 ways to define the language accepted by a pushdown automaton M. The *language accepted by empty stack* is the set of words for which some sequence of moves of M results in an empty stack. The *language accepted by final state* is the set of words which, for some sequence of moves of M, cause M to enter a final state $q \in F$.

The relationships between context-free languages and pushdown automata were originally shown by Chomsky [15] and Evey [22].

Theorem 15 (Chomsky, Evey) *The following statements are equivalent:*

(1) L is a context-free language

(2) L is accepted by a PDA that accepts by final state

(3) L is accepted by a PDA that accepts by empty stack.

Definition: A pushdown automaton is deterministic if:

(1) $(\forall q \in Q, Z \in \Gamma) \quad \delta(q, \lambda, Z) \neq \emptyset \Rightarrow (\forall z \in \Sigma)\ \delta(q, a, Z) = \emptyset$;

(2) There does not exist $q \in Q, Z \in \Gamma$, and $a \in (\Sigma \cup \{\lambda\})$ such that $\delta(q, a, Z)$ contains more than one element.

A language accepted by deterministic PDA is termed a *deterministic context sensitive language*, or *DCSL*. The DCSL's form a proper subclass of the context-free languages.

3.1.4 Turing Machines

Our general computational model will be a *multitape Turing machine*, as defined in [2]. We begin with an informal description of the model.

A multitape Turing machine (TM) consists of k *tapes*, which are infinite to the right. Each tape is marked off into *cells*, each of which holds one of a finite number of *tape symbols*. Associated with each tape is a *tape head*, which may read and write one cell at a time, and which may move to the left and right. Operation of the Turing machine is determined by a *finite control*, which is always in one of a finite number of *states*.

A computational step of a Turing machine is determined by the current state of the finite control and the tape symbols which are currently scanned by the tape heads. In one step, the Turing machine may do any or all of the following operations:

1. Change the state of the finite control.
2. Write new tape symbols into the currently scanned tape cells, on any or all of the tapes.
3. Move any or all of the tape heads, independantly, one cell to the left or right. A tape head may remain stationary.

Formally, a *multitape Turing machine* is a 7–tuple $(Q, T, I, \delta, \mathtt{b}, q_0, q_f)$, where:

1. Q is a finite set of *states*,
2. T is a finite set of *tape symbols*,

3. I is the set of *input symbols*; $I \subseteq T$,

4. $\mathtt{b} \in T \setminus I$ is the *blank*,

5. $q_0 \in Q$ is the *initial state*,

6. $q_f \in Q$ is the set of *final*, or *accepting* state, and

7. δ, the *next-move (partial) function*, is a mapping of a subset of $Q \times T^k$ to $Q \times (T \times \{\mathrm{L}, \mathrm{R}, \mathrm{S}\})^k$.

The correspondence between the formal definition and the informal characterization of a Turing machine is made by interpreting

$$\delta(q, a_1, \ldots a_k) = (q', (a_1', d_1) \ldots (a_k', d_k))$$

to mean that if the Turing machine is in state q, with tape head i scanning tape symbol a_i, then in one move, the Turing machine enters state q', changes symbol a_i to a_i' on tape i, and moves tape head i in direction d_i. ($i = 1, \ldots k$, $d_i \in \{\mathrm{L}, \mathrm{R}, \mathrm{S}\}$). The directions L,R, and S denote, respectively, move one cell to the *L*eft, move one cell to the *R*ight, and remain *S*tationary.

The *language accepted by M* is the set of words in Σ^* that cause M to enter a final state, when M starts in state q_0, scanning the leftmost symbol of w. A *nondeterministic* Turing machine (NTM) is defined in a manner analogous to a NFA; the next-move function δ is defined to be a mapping of a subset of $Q \times T^k$ to (finite) subsets of $Q \times (T \times \{\mathrm{L}, \mathrm{R}, \mathrm{S}\})^k$. The language L(M) accepted by a NTM M is defined by: $w \in \mathrm{L(M)}$ if and only if there is some computation of M, with input w, which causes M to reach a final state. Deterministic and nondeterministic Turing machines accept the same class of languages, the *recursively enumerable* languages.

3.1.5 Linear-Bounded Automata

A deterministic Turing machine that obeys a space bound of $n+1$ is termed a *deterministic linear bounded automaton*, or DLBA, while a non-deterministic TM obeying the same space bound is termed a *non-deterministic linear bounded automaton*, or *NLBA*. The set of languages recognizable by NLBA's is precisely the set of context-sensitive languages. By Savitch's theorem (1970), the CSL's can be recognized by DTM's obeying a space bound of $(n+1)^2$.

A major open question, known simply as *the LBA problem*, asks: Can every CSL be recognized by a DLBA? Or, otherwise stated, is DSPACE(n) = NSPACE(n)? According to Garey and Johnson [24], the LBA problem was "perhaps the most famous open problem in complexity theory before the *P vs NP* problem arose".

Another major question, raised by Kuroda [43] in 1964, was whether or not the context-sensitive languages were closed under complementation. This question remained open until 1988, when N. Immerman proved the following stronger result: [36]

Theorem 16 (Immerman) *For any* $s(n) \geq \log(n)$,

$$NSPACE\,[s(n)] = co\text{-}NSPACE\,[s(n)].$$

Since the class of CSL's is precisely NSPACE(n), the fact that the CSL's are closed under complementation follows immediately from Immerman's theorem.

3.2 Languages Associated With Groups.

The Chomsky formal language hierarchy provides a framework for introducing complexity measures on finitely presented groups given by generators and defining relators. We consider the problem of determining relationships between the structure of the group and the complexity of formal languages associated with the group. One such language, the *Word Problem*, is the set of words which are equal to the identity element in the group. We also show how canonical forms of elements in finitely generated groups may be specified by *Cayley Languages*, which are minimal Schreier coset representative systems for G mod the trivial subgroup.

3.2.1 The Word Problem.

In 1911, Max Dehn formulated the well-known *word problem* for a presentation defining a group G: "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." [48, page 24] The word problem has been one of the most extensively studied algorithmic problems of combinatorial group theory. Although the word problem has been solved for many classes of presentations, there exist groups having a finite presentation for which no "there exits no general and effective procedure to decide for any given word W in the generators whether it represents the unit element of the group, or not." [48, pg 25]

Suppose now that G is a finitely generated group presented by a set of generators and defining relators. Specifically, suppose G is presented as $\langle X;\ R\rangle$, where $X = \{x_1, \ldots, x_n\}$. Let X^{-1} denote the set $\{x_1^{-1}, \ldots, x_n^{-1}\}$, and let $G^{\#} = (X \bigcup X^{-1})^*$. $G^{\#}$ may be characterized algebraically as the free

monoid with free set of generators $\{x_1, \ldots, x_n, x_1^{-1}, \ldots, x_n^{-1}\}$.

We may view any word in $G^{\#}$ as an element of G, associating the identity element 1 of G with the empty word λ. Formally, this association is made via a monoid epimorphism $h^{\#} : G^{\#} \rightarrow G$ onto the group G, with the properties that $h^{\#}(\lambda) = 1$, and $h^{\#}(x_j^{-1}) = (h^{\#}(x_j))^{-1}$, for $j = 1, \ldots n$. For our purposes, we will always let $h^{\#}(x_j) = x_j$, for $j = 1, \ldots n$, and so explicit references to $h^{\#}$ will often be omitted. The word problem for a group G is to determine, for a given word g in $G^{\#}$, whether or not g defines the identity element 1 of G. In accordance with the standard practice of naming sets by their membership predicates, we give the following definition:

Definition: The *Word Problem* arising from a given presentation of a group G, and a monoid homomorphism $h^{\#}$ (as specified above), is the set

$$\mathrm{WP(G)} = \{\ x \in G^{\#} :\ h^{\#}(x) = 1\ \};$$

i.e., WP(G) is the language consisting of all words in $G^{\#}$ which define the identity element 1 of G.

The notation WP(G), which omits reference to the presentation, is motivated by the following result, due to Anisimov: [51,29]

Proposition 6 *If any finitely generated presentation of a group G possesses a regular (resp. context-free) Word Problem, then all finitely generated presentation of G possess a regular (resp. context-free) Word Problem.*

It can be seen directly from the definition that WP(G) is a recursive set if and only if G has solvable word problem. Groups with regular word problem have been characterized by A.V. Anisimov to be precisely the finite groups [3].

Theorem 17 (Anisimov) *A finitely generated group G is finite if and only if G possesses a regular Word Problem.*

Proof: First, suppose that the Word Problem L for G is a regular set. Define an equivalence relation R on $G^{\#}$ as follows:

$$R(x,y) \quad \text{if and only if} \quad \forall z \in G^{\#}, \quad xz \in L \iff yz \in L.$$

By a theorem due to Nerode [32, page 65], L is a regular set if and only if R is of finite index. It is easily seen that if x and y are in the same R-equivalence class, then they represent equal elements of G. For, if $R(x,y)$, then $\forall z,\ xz = 1 \iff yz = 1$. Take z to be the formal inverse of x; then $yx^{-1} = 1$, and so $y = x$. Since there are only finitely many R-equivalence classes, G must be finite.

Conversely, suppose G is finite. We show how to construct a finite state machine that recognizes L, the Word Problem for G. For the set of states S, we take the elements of G. The identity element is both the initial state and the final state. The input alphabet Σ also consists of the elements of G, and the state transition function $\delta : G \times \Sigma \to G$ is defined as left-multiplication in G. Now,

$$w \in L \iff w = 1 \iff 1 \cdot w = 1 \iff \delta(1,w) = 1,$$

so this machine recognizes L. By Kleene's theorem, L is regular.

Muller and Schupp went on to classify groups having context-free Word Problem [51]. They prove that a finitely generated group G is virtually free (i.e., G contains a free subgroup of finite index) if and only if G is accessible and has context-free Word Problem. Dunwoody [21] has since shown that

all finitely presented groups are accessible. Hence, a finitely presented group G has context-free Word Problem if and only if it is virtually free.

Muller and Schupp also show that a finitely generated torsion-free group G is free if and only if G has context-free Word Problem.

3.2.2 Cayley Languages.

If H is a subgroup of G, a *coset representative system* for G/H is any set of elements of G which contains exactly one element from each coset of H. A coset representative system L for G/H is said to be a *Minimal Schreier Transversal* provided each coset is specified by a word of shortest possible length, and if in addition, L is *prefix-closed*; i.e., if w is a word in the generators and their inverses, and w is in L, then every initial segment of w is also in L. A minimal Schreier transversal for a finitely generated group always exists. [48, page 93]

We illustrate these definitions with an example.

Example: Let G be the free group on two generators, $\{a, b\}$, and let H be the normal subgroup of G generated by $\{abab, a^2, b^3\}$. Then G/H has the presentation $\langle\ a, b;\ abab = a^2 = b^3 = e\ \rangle$. The set $\{e, a, b, b^2, b^2a, ab\}$ is a coset representative system that is neither prefix-closed nor minimal (that is to say, not all cosets are specified by elements of minimal length). The set $\{e, a, b, b^{-1}, ab, ab^2\}$ is prefix-closed but not minimal, while the set $\{e, a, b, b^{-1}, ab, ba\}$ is both prefix-closed and minimal, and hence a minimal Schreier Transversal.

Definition: A *Cayley Language* for G is a minimal Schreier transversal corresponding to the trivial subgroup. An *enumeration* of G is any coset

representative system for $G/\{1\}$.

A Cayley Language has a natural geometric interpretation in terms of the graph of the group, also known as a *Cayley diagram.* Define the weight of an element to be the fewest number of edges between that element and the identity. A Cayley language then corresponds to a minimal spanning tree of a Cayley diagram for G, in which each edge has unit weight. We remark that the problem of finding a minimal spanning tree of a given Cayley diagram is solvable in polynomial time by Kruskal's algorithm or by Prim's algorithm. [56]

Let G be a finitely generated group, and let

$$\mathrm{G} = \langle\, a_1, \ldots, a_n; \quad r_1, r_2, \ldots \,\rangle$$

be a presentation for G. We assume the a_i are distinct symbols, and that if a_i is one of the generators, then its inverse a_i^{-1} appears as some a_j, where $j \neq i$. Define a total ordering $\ll$ on $G^{\#}$ as follows:

Definition: $w_1 \ll w_2$ if and only if either
(1) $|w_1| < |w_2|$, or
(2) $|w_1| = |w_2|$, $w_1 = u_1 a_i$, $w_2 = u_2 a_j$, and either $i < j$, or $i = j$ and $u_1 \ll u_2$.

It is clear that $G^{\#}$ can be effectively enumerated in this order, starting with the empty word. The enumeration begins with the empty word λ. If $w_1, w_2, \ldots, w_k$ are the words of length m, listed in ascending order, then the the words of length $m+1$, listed in ascending order, are:

$$w_1 a_1, \ldots, w_k a_1, \quad w_1 a_2, \ldots, w_k a_2, \quad \ldots, \quad w_1 a_n, \ldots, w_k a_n.$$

For the purpose of illustration, let $n = 2$, $a_1 = a$, $a_2 = b$. Then the enumeration corresponding to $\ll$ is:

$$\lambda, \quad a, b, \quad aa, ba, ab, bb, \quad aaa, baa, aba, bba, aab, bab, abb, bbb, \quad \ldots$$

This enumeration may be constructed in the usual way by means of a "successor" function. A single-tape Turing machine that computes, for $x \in G^{\#}$, SUCCESSOR(x) is given in Figure [5]. We note that the machine SUCCESSOR visits at most $n+1$ tape cells in computing the successor of a word w of length n.

1. Start with x on the input tape, scanning the left-most symbol of x.

2. While symbol scanned = a_n, move right.

3. If symbol scanned is not a blank, replace the symbol by its successor with respect to $\ll$, and move left to the leftmost symbol of x, leaving the tape otherwise unchanged.
   Otherwise, replace the blank by a_1, and now move left to the beginning, replacing all non-blank symbols (they are all a_n in this case) by a_1.

Figure [5]: Turing machine for SUCCESSOR(x).

In actual computations, it is helpful to have a more efficient function whose value is the k^{th} element of the enumeration. One implementation of such a function is given in Figures [6] and [7].

The calculation first establishes the actual length c of the k^{th} element, and then utilizes an order–preserving correspondence between words of length c

and a "bit-reversed" radix–n representation of the integers $0, \ldots, c^n - 1$, under which the alphabet symbols $a_1, \ldots, a_n$ map to $0, \ldots n-1$. The number of words of length less than c is

$$1 + \alpha + \cdots + \alpha^{c-1} = \frac{\alpha^c - 1}{\alpha - 1},$$

and therefore, the number of characters in the k^{th} word is the smallest integer c such that

$$k \leq \frac{\alpha^c - 1}{\alpha - 1}.$$

```
function WORD(n :longint) :string;
(* Returns the n^th word in the enumeration *)
var  c, i :integer;
     p, q :longint;
     w    :string;
begin
     if n < 0 then word := '' (* empty word *)
     else begin
          (* Determine c minimal such that (α^c - 1)/(α - 1) > n. *)
          c := charlength(n);

          (* q is the number of words of length < c. *)
          q := ( power( alpha,c ) -1 ) div ( alpha-1 ) -1;

          (* w is the bit-reversed representation of n - q, radix α,
          prepended as necessary with the first character of the
          group alphabet. *)
          w := alpha_rev(n - q);

          for i := 1 to c- length(w) do w := w + alphabet[1];
          WORD := w
     end;
end;
```

Figure [6]: WORD(n) is the n^{th} word under the ordering $\ll$.

```
var alphabet  :string;     (* the group alphabet     *)
    alpha     :integer;    (* length of alphabet.    *)

function charlength (n :longint) :integer;
(* the number of chars in the n^th word.           *)
var  b, c  :longint;
begin
  b := 1;  c := 0;
  while n >= 0 do begin
     b := b * alpha;
     c := c + 1;
     n := n - b;
  end;
  charlength := c;
end;

function alpha_rev(n :longint) :string;
(* n radix alpha, least signficant digit first.   *)
begin
  if n < alpha then alpha_rev := alphabet[n+1]
  else alpha_rev := alphabet[n mod alpha+1]
                    + alpha_rev(n div alpha)
end;

function WORD(n :longint) :string;
(* Returns the n^th word in the enumeration *)
var  cl,i :integer;
     p,q  :longint;
     w    :string;
begin
    if n < 0 then word := ''
    else begin
        cl := charlength(n);
        q  := ((power(alpha,cl)-1) div (alpha-1)-1);
        w  := alpha_rev(n - q);
        for i := 1 to cl - length(w) do
            w := w + alphabet[1];
        word := w
    end;
end;
```

Figure [7]: Implementation of WORD(n), from [HNNZ7.PAS].

Proposition 7 *(1)* $\ll$ *is a linear ordering,*
(2) $\forall w \in G^{\#}, \quad \lambda \ll w$, *and*
(3) $\forall w, x, y \in G^{\#}, \quad x \ll y \Rightarrow xw \ll yw.$

Lemma 15 *A finitely generated group with solvable word problem admits a recursive Cayley language.*

Proof: Let L = $\{x \in G^{\#} : \forall z \ll x, x \neq z$ (as elements of G) $\}$. L is evidently a minimal enumeration of G. To see that L is prefix-closed, suppose $w = uv \in L$, but $u \notin L$. Then there is some z, $z \ll u$, such that $z = u$ as elements of G. Now $w = zv$ in G, but by Proposition 7, $zv \ll uv$, which implies that w was not minimal. Hence, L is prefix-closed, and thus a Cayley language for G.

We now show how to decide, given w in $G^{\#}$, whether or not w is in L. List the words of $G^{\#}$ in ascending order, starting with λ, according to the ordering $\ll$, and stop when a word z is found with $z = w$ in G. If $z = w$ (as words) then $w \in L$; otherwise, $w \notin L$. ∎

The construction described in this lemma may be directly implemented, as shown below in Figure [8], and in Figure [9].

We also note that the "list-and-test" decision procedure described in the lemma determines a function SELECT: $G^{\#} \rightarrow G^{\#}$, where SELECT($w$) is the minimal $z \in G^{\#}$ (with respect to the ordering $\ll$) such that $z = w$. We say that a Cayley language is *computable* if it is recursive, and if it possesses such a "selector" function. We will call the image of SELECT the set of *normal forms.* A Pascal implementation of SELECT is given in Figure [10].

Algorithm: "Enumerate Cayley Language Version 1":
Output: a Cayley language L.

```
begin
    L = {λ}
    w = λ
    while TRUE do
    begin
        z = λ
        match_found = FALSE
        while not match_found do
        begin
            if wz^-1 = 1 then
                match_found = TRUE
            else
                z = SUCCESSOR(z)
        end
        if z = w then L = L ⋃ {w};
        w = SUCCESSOR(w)
    end
end
```

Figure [8]: Cayley Language enumeration corresponding to $\ll$.

```
procedure enumerate_Cayley_ver_1 (howmany :integer);
var i,j          :longint;
    z,w          :string;
    match_found :boolean;
    g,n          :group_word;
begin
   for i := -1 to howmany do begin
      w := word(i);   (* is w in L? *)
      j := -1;
      match_found := false;
      while not match_found do begin
         z := word(j);
         unary_to_exponential(w*formal_inverse(z),g);
         if word_problem(g, n) then
             match_found := true
         else
             inc(j);
      end;
      if j=i then writeln('accept ',w);  (*z=w*)
   end;
end;
```

Figure [9]: A direct implementation of the algorithm outlined in the proof of Lemma 15.

```
function SELECT(w :string) :string;
var match_found :boolean;
    i,j    :longint;
    z      :string;
    g,n    :group_word;
begin
    j := -1;
    match_found := false;
    while not match_found do
    begin
       z := word(j);
       { Convert string w z^{-1} to the group_word g }
       unary_to_exponential(w + formal_inverse(z), g);

       if word_problem(g, n) then
           match_found := true
        else
           inc(j);
    end;
    SELECT := z;
end;
```

Figure [10]: SELECT(w) is the representative of the Cayley language corresponding to the word $w \in G^{\#}$, relative to the ordering utilized by the function WORD(k).

A stronger form of Lemma 15 is now easy to state:

Lemma 16 *A finitely generated group G has solvable word problem if and only if G possesses a computable Cayley language.*

Proof: The "only if" part has just been shown. Suppose L is a Cayley language for G, with selector function SELECT as defined above. If $w = 1$ in G then SELECT(w) = SELECT$(1) = \lambda$, since L is a minimal enumeration of G, and by Proposition 7, $\lambda < x$, $\forall x \in G$. ∎

A group G may have more than one distinct Cayley language. We will give an example to show that Cayley language for G may depend on the choice of an ordering of the words of $G^{\#}$. In the course of giving this example, we will consider some practical aspects regarding construction of a Cayley language.

Let $G = \langle\ a, z; aza^{-1} = z^2\ \rangle$. G is a group in the class HNN(Z), and therefore has a solvable word problem. An implementation of a solution to its word problem is discussed in detail in the chapter on HNN extensions.

The algorithm of Lemma 15 constructs its minimum spanning tree of the Cayley diagram using Prim's method, in which at all stages of the construction, the edges that have been included form a tree. (This is in contrast to Kruskal's method, in which the included edges form a forest which may be extended to a minimum spanning tree.)

It is informative to view the construction from a geometric point of view. The ordering $\ll$ induces an enumeration in which the rightmost symbol varies the slowest. This results in a breadth- first traversal of a complete $n-$ary tree that is far easier to see than to describe. We will illustrate the traversal

instead. (See Figure [11].) The numbers at the nodes indicate the traversal order. Only the first three levels are shown.

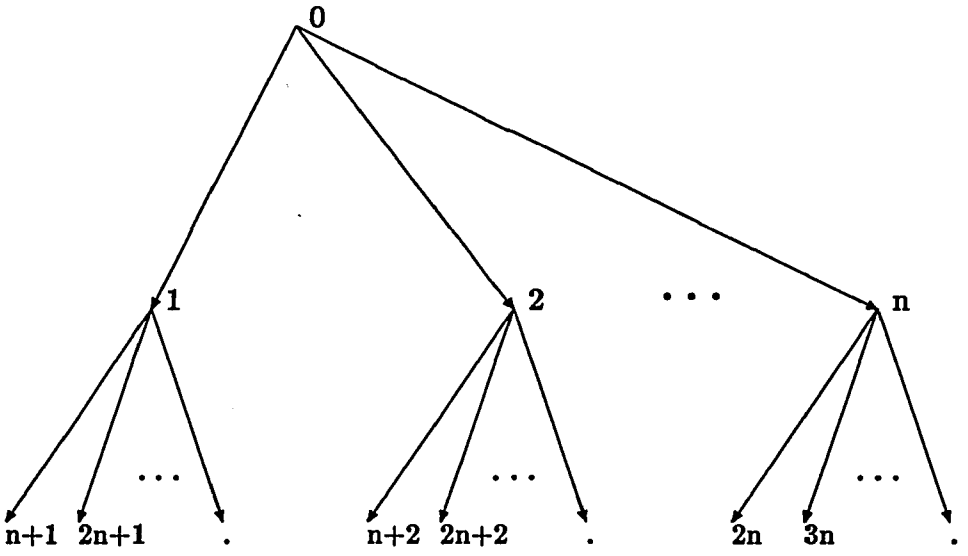


Figure [11]: Traversal corresponding to the ordering $\ll$.

In considering the execution of "`enumerate_Cayley_ver_1`" in generating a Cayley language L for G, we make the following straightforward observations. First, if $w \in G^{\#}$, and $w = w'\alpha$, where $|\alpha| = 1$, and $w' \notin L$, then $w \notin L$. Therefore, we need only test for inclusion in L words that are single-symbol extensions of words already known to be in L. Second, if $w \notin L$, then SELECT$(w) \ll w$.

Taken together, these observations indicate how the Cayley language may be built up more efficiently, at the cost of storing the language as it is built, using a "bootstrap" process. (Refer to Figures [12] and [13].)

Algorithm: "Enumerate Cayley Language Version 2":
Output: a Cayley language L.
begin
Let $a_1 \ll a_2 \ll \ldots \ll a_n$ be the alphabet symbols.
L = $\{\lambda\}$.
Mark λ as "untested".
while there are untested words in L **do**
begin
w = the minimal untested word in L;
for $x = a_1, \ldots, a_n$ **do**
begin
if $(\forall z \in L$ with $z \ll wx)$ $wxz^{-1} \neq 1$ **then**
begin
mark wx as "untested";
L := L $\bigcup\{wx\}$;
end
end
mark w as "tested"
end
end

Figure [12]: Alternate traversal corresponding to the ordering $\ll$.

```
procedure enumerate_Cayley_ver_2 (howmany :integer);
label breakout;
var   count,nxts,i,j :longint;
      z,w, next_to_sprout :string;
      g :group_word;
      w_in_CL :boolean;
begin
   C_L[0]  := '';  (* the empty word *)
   C_L_ptr := 0;   (* index of the last word in C_L   *)
   count   := 1;   (* the number of words considered. *)
   nxts    := 0;   (* index in C_L of the next word to
                      which symbols will be appended. *)
   while C_L_ptr < howmany do begin
      next_to_sprout := C_L[nxts];
      for i := 1 to alpha do begin
         w := next_to_sprout + alphabet[i];  (* <<' *)
         inc(count);
         w_in_CL := true;
         for j := 0 to C_L_ptr do begin
            z := C_L[j];
            unary_to_exponential(w+formal_inverse(z),g);
            if word_problem(g, normal_form) then begin
               w_in_CL := false;
               goto breakout;
            end;
         end;
         breakout:
         if w_in_CL then begin (* append w to C_L *)
            inc(C_L_ptr);
            C_L[C_L_ptr] := w;
         end;
      end; (* alpha loop *)
      inc(nxts);
   end;  (* howmany loop *)
end;
```

Figure [13]: This generates the Cayley language corresponding to $\ll'$. In practice, this runs faster than Version 1 at the cost of (1) storing the Cayley language as it is built up, and (2) choosing an ordering on $G^{\#}$ that reflects a more "natural" tree traversal.

This algorithm does not depend on the ordering of words in $G^{\#}$; therefore, an ordering should be chosen that will facilitate finding a minimal "unmarked" element of $G^{\#}$. We define an ordering $\ll'$ that corresponds to an different breadth-first tree traversal; here, we extend one node at at time, as much as possible. Again, this is hard to describe, and so we once more resort to a simple illustration (Figure [14]).

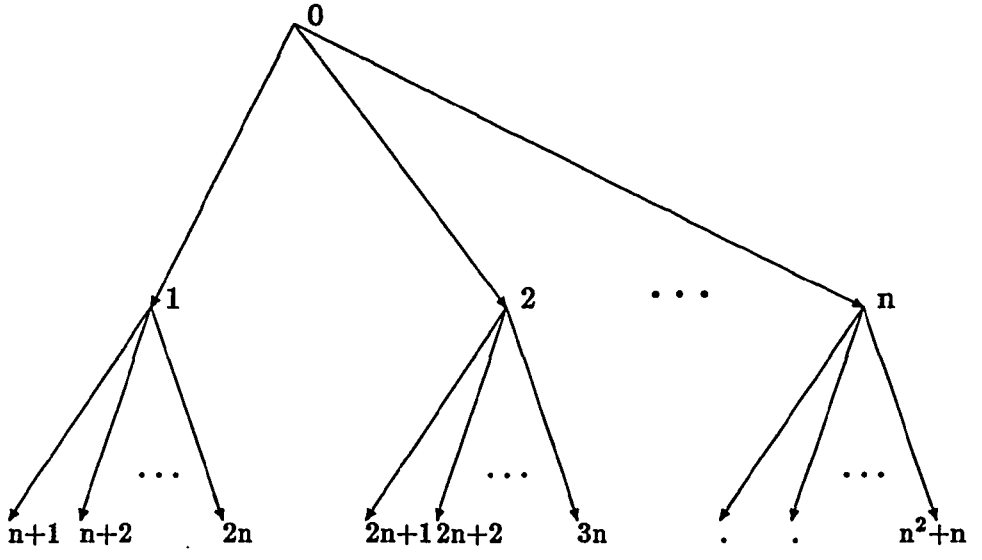


Figure [14]: Traversal corresponding to the ordering $\ll'$.

Formally, the ordering $\ll'$ may be defined as follows:

Definition: $w_1 \ll' w_2$ if and only if either (1) $|w_1| < |w_2|$, or (2) $|w_1| = |w_2|$, $w_1 = a_i u_1$, $w_2 = a_j u_2$, and either $i < j$, or $i = j$ and $u_1 \ll' u_2$.

The sequence of words of $G^{\#}$ that Algorithm "enumerate_Cayley_ver_2" produces is in ascending order relative to $\ll'$. As observed before, only the elements of the Cayley language L need be stored. A simple linear list may be used to store L, and L will remain sorted by simply appending elements shown to be in L at the end of L. A Pascal implementation of this is provided in Figure [13].

Using the procedures listed in Figure [9] and in Figure [13], we obtain the words of length at most 4 in the Cayley language corresponding to the orderings $\ll$ and $\ll'$. These lists are given in Figure [15] below.

$\ll$	$\ll'$	$\ll$	$\ll'$	$\ll$	$\ll'$
z	z	$a^{-1}z^{-1}a$	$az^{-1}a$	za^2z^{-1}	$z^{-1}a^3$
z^{-1}	z^{-1}	za^2	a^2z	$z^{-1}a^2z^{-1}$	$z^{-1}a^{-1}z^{-1}a$
a	a	$z^{-1}a^2$	a^2z^{-1}	a^3z^{-1}	$z^{-1}a^{-1}z^{-1}a^{-1}$
a^{-1}	a^{-1}	a^3	a^3	$z^{-2}a^{-1}z^{-1}$	$z^{-1}a^{-2}z$
z^2	z^2	z^2a^{-1}	$a^{-1}za$	$z^{-1}a^{-2}z^{-1}$	$z^{-1}a^{-2}z^{-1}$
az	za	$a^{-1}za^{-1}$	$a^{-1}za^{-1}$	$a^{-3}z^{-1}$	$z^{-1}a^{-3}$
$a^{-1}z$	za^{-1}	$z^{-2}a^{-1}$	$a^{-1}z^{-1}a$	a^2za	az^3
z^{-2}	z^{-2}	$a^{-1}z^{-1}a^{-1}$	$a^{-1}z^{-1}a^{-1}$	$a^{-2}za$	$azaz$
az^{-1}	$z^{-1}a$	za^{-2}	$a^{-2}z$	$a^2z^{-1}a$	aza^2
$a^{-1}z^{-1}$	$z^{-1}a^{-1}$	$z^{-1}a^{-2}$	$a^{-2}z^{-1}$	$a^{-2}z^{-1}a$	az^{-3}
za	az	a^{-3}	a^{-3}	aza^2	$az^{-1}az^{-1}$
$z^{-1}a$	az^{-1}	z^4	z^4	$a^{-1}za^2$	$az^{-1}a^2$
a^2	a^2	az^3	z^3a^{-1}	$az^{-1}a^2$	a^2z^2
za^{-1}	$a^{-1}z$	zaz^2	$z^2a^{-1}z$	$a^{-1}z^{-1}a^2$	a^2za
$z^{-1}a^{-1}$	$a^{-1}z^{-1}$	a^2z^2	z^2a^{-2}	za^3	a^2z^{-2}
a^{-2}	a^{-2}	$azaz$	zaz^2	$z^{-1}a^3$	$a^2z^{-1}a$
z^3	z^3	$a^{-1}zaz$	$zaza$	a^4	a^3z
az^2	z^2a^{-1}	za^2z	za^2z	z^3a^{-1}	a^3z^{-1}
zaz	zaz	$z^{-1}a^2z$	za^2z^{-1}	$za^{-1}za^{-1}$	a^4
a^2z	za^2	a^3z	za^3	$a^{-2}za^{-1}$	$a^{-1}za^2$
$za^{-1}z$	$za^{-1}z$	$z^2a^{-1}z$	$za^{-1}za$	$z^{-3}a^{-1}$	$a^{-1}za^{-2}$
$a^{-2}z$	za^{-2}	$a^{-1}za^{-1}z$	$za^{-1}za^{-1}$	$z^{-1}a^{-1}z^{-1}a^{-1}$	$a^{-1}z^{-1}a^2$
z^{-3}	z^{-3}	$za^{-2}z$	$za^{-2}z$	$a^{-2}z^{-1}a^{-1}$	$a^{-1}z^{-1}a^{-2}$
az^{-2}	$z^{-2}a^{-1}$	$z^{-1}a^{-2}z$	$za^{-2}z^{-1}$	z^2a^{-2}	$a^{-2}za$
$z^{-1}az^{-1}$	$z^{-1}az^{-1}$	$a^{-3}z$	za^{-3}	$a^{-1}za^{-2}$	$a^{-2}za^{-1}$
a^2z^{-1}	$z^{-1}a^2$	z^{-4}	z^{-4}	$z^{-2}a^{-2}$	$a^{-2}z^{-1}a$
$z^{-1}a^{-1}z^{-1}$	$z^{-1}a^{-1}z^{-1}$	az^{-3}	$z^{-3}a^{-1}$	$a^{-1}z^{-1}a^{-2}$	$a^{-2}z^{-1}a^{-1}$
$a^{-2}z^{-1}$	$z^{-1}a^{-2}$	$z^{-1}az^{-2}$	$z^{-2}a^{-1}z^{-1}$	za^{-3}	$a^{-3}z$
aza	az^2	a^2z^{-2}	$z^{-2}a^{-2}$	$z^{-1}a^{-3}$	$a^{-3}z^{-1}$
$a^{-1}za$	aza	$az^{-1}az^{-1}$	$z^{-1}az^{-2}$	a^{-4}	a^{-4}
$az^{-1}a$	az^{-2}	$a^{-1}z^{-1}az^{-1}$	$z^{-1}a^2z^{-1}$		

Figure [15]: Enumerations of words of length at most 4 in the group $< a, z;\ aza^{-1} = z^2 >$, based on the orderings $\ll$ and $\ll'$.

A comparison of these lists shows that the sets are not the same. There are 4 elements of G that are represented by different words in the Cayley languages corresponding to the two orderings, listed in Figure [16]. The pairing of equal elements of G was determined using the "SELECT" function in [HNNZ7.PAS], using the ordering $\ll$.

Version 1 ($\ll$)		Version 2 ($\ll'$)
$a^{-1}z^{-1}az^{-1}$	=	$z^{-1}a^{-1}z^{-1}a$
$a^{-1}za^{-1}z$	=	$za^{-2}z^{-1}$
$a^{-1}zaz$	=	$za^{-1}za$
$z^{-1}a^2z$	=	$zaza$

Figure [16]: Different words in Enumerations 1 and 2.

Even if the ordering is fixed, we may construct different Cayley languages by using different orderings of the generators. This can be illustrated with the free abelian group of rank 2, $F_2 =< a, b; >$. We use the ordering $\ll'$, since the tree traversal it induces is somewhat simpler to visualize.

If the generators are ordered as $a \ll' a^{-1} \ll' b \ll' b^{-1}$, then the Minimal Schreier transversal constructed is

$$(a^*|(a^{-1})^*)(b^*|(b^{-1})^*),$$

while under the ordering $a \ll' b \ll' a^{-1} \ll' b^{-1}$, the Minimal Schreier transversal is the set

$$a^*(b^*|(b^{-1})^*) \bigcup bb^*(a^{-1})^* \bigcup a^{-1}(a^{-1})^*(b^{-1})^*.$$

We conclude this section with a generalization of Lemma 15. Given a

group G with subgroup H, the *generalized word problem* for the pair (G, H) is to decide, given a word $w \in G$, whether $w \in H$..

Lemma 17 *If G is a finitely generated group with subgroup H, and (G,H) has solvable generalized word problem, then G/H admits a recursive minimal Schreier transversal.*

Proof: Let L = $\{x \in G^{\#} : \forall z \ll x, zx^{-1} \notin H\}$. L evidently contains precisely one element of $G^{\#}$ from each coset in G/H, and this element is the minimal representative of its coset with respect to the ordering $\ll$.

To see that L is prefix-closed, suppose $w = uv \in L$, but $u \notin L$. Then there is some z, $z \ll u$, such that $zu^{-1} \in H$. By Proposition 7, $zv \ll uv$. Now, $zvw^{-1} = zvv^{-1}u^{-1} = zu^{-1}$, which is in H. Since $zv \ll w$, this implies $w \notin L$. Hence, L is prefix-closed, and is therefore a minimal Schreier transversal for G/H.

We now show how to decide, given w in $G^{\#}$, whether or not w is in L. List the words of $G^{\#}$ in ascending order, starting with λ, according to the ordering $\ll$, and stop when a word z is found with $zw^{-1} \in H$. If $z = w$ (as words) then $w \in L$; otherwise, $w \notin L$. ∎

Note that this Lemma reduces to Lemma 15 in case H is the trivial subgroup {1}.

3.2.3 Free and Free Abelian Groups.

We now focus attention on two simple classes of groups, the free groups and the free abelian groups. These classes of groups serve as good examples for contrasting the Cayley languages with the Word Problem.

In [51] it is shown that the free groups F_r of rank $r \geq 1$ on a free set of generators have group languages that are context-free but not regular, while the free abelian groups A_s of rank $s \geq 2$ on a free set of generators have group languages that are context-sensitive but not context-free. From the perspective afforded by their group languages, the free abelian groups A_s are more complex than the free groups F_r. This seems to indicate that the group languages do not faithfully reflect the groups' structural complexity.

This situation is in marked contrast with that of the Cayley languages. Both F_r and A_s possess regular Cayley languages. In the case of $F_r = \langle a_1, \ldots a_r; \rangle$, we may take for a Cayley language $L(F)$ the freely reduced words; i.e., those words containing no subwords of the form $a_i^{-1}a_i$ or $a_i a_i^{-1}$. In the case of

$$A_s = \langle b_1, \ldots b_s; \quad b_i^{-1}b_j^{-1}b_i b_j, \ (i,j = 1, \ldots s, \ i \neq j) \rangle,$$

we may take for a Cayley language $L(A)$ the freely reduced monotonic words (i.e., words in which the symbol subscripts form a non-decreasing sequence). It is easily seen that both $L(F)$ and $L(A)$ are Cayley languages. $L(F)$ is the complement of the language

$$G^{\#}(b_1 b_1^{-1})G^{\#} \bigcup G^{\#}(b_1^{-1}b_1)G^{\#} \bigcup \ldots \bigcup G^{\#}(b_s b_s^{-1})G^{\#} \bigcup G^{\#}(b_s^{-1}b_s)G^{\#},$$

and the complement of a regular language is again a regular language. $L(A)$

is given as the concatenation of the regular expressions

$$((a_1)^* \bigcup (a_1^{-1})^*), ((a_2)^* \bigcup (a_2^{-1})^*), \ldots ((a_r)^* \bigcup (a_r^{-1})^*)$$

and is therefore a regular language.

3.2.4 Confluent Groups

We consider two classes of groups, the *confluent groups* and *groups of Dehn's Algorithm* for which the word problem is solvable by means of straightforward pattern matching and string rewriting procedures.

We first consider confluent groups, or groups possessing a finite *complete* presentation. The concepts here are analogous to those developed in the polynomial case. Confluent groups may be viewed as generalizations of finitely generated commutative monoids, in the sense that they may be presented in such a way that their relators define a term rewriting system which yields an algorithm for deciding the word problem.

We first define the analog of an "admissible ordering on monomials", as defined in the Gröbner basis chapter. Let G be generated by $X = \{x_1, \ldots, x_n\}$.

Definition: A total ordering $<$ on X^* is a *reduction order* provided that
(1) $<$ is a linear order,
(2) $\lambda < x_i \quad (1 \leq i \leq n)$, and
(3) $u < v \Rightarrow aub < avb, \quad (\forall a, b, u, v \in X^*)$.

Proposition 8 *The orderings $\ll$ and $\ll'$ on $G^\#$ as defined earlier in the Cayley Language section are reduction orders.*

Definition: A group G has a *confluent* presentation with respect to an ordering $\ll$ if there exists a finite set $P = \{(\alpha_i, \beta_i) \ (i \in I)\}$ of pairs of words of $G^{\#}$, with $\alpha_i = \beta_i$ in G, $\beta_i \ll \alpha_i$, and if the reflexive transitive closure of $\rightarrow_P$, defined by

$$w \rightarrow_P w' \iff ((\exists i \in I) \ (w = u\alpha_i v, \ w' = u\beta_i v)),$$

is Church–Rosser. We denote the reflexive transitive closure of $\rightarrow_P$ by $\rightarrow_P^*$.

As in the Gröbner basis chapter, the relation $\rightarrow_P$ may be interpreted as a rewriting process. Given w, $\exists w'$ such that $w \rightarrow_P w'$ if and only if for some i, α_i is a subword of w. The properties of a reduction order guarantee that any sequence $w \rightarrow_P w_1 \rightarrow_P \cdots$ terminates in a finite number of steps. Finally, the finite termination and the Church–Rosser property provide a normal form algorithm for elements of G.

If G has a confluent presentation with respect to a reduction order, then the word problem for G is solvable; $w = 1 \iff w \rightarrow_P^* \lambda$.

Definition: The class of *regular aperiodic sets* is the smallest family of sets containing the finite sets, and which is closed under union, concatenation, and complementation.

Theorem 18 *Let G be a finitely generated group with confluent presentation. Then G admits a regular aperiodic Cayley Language.*

Proof: Since G has a complete presentation, G has solvable word problem. Let L be the Cayley language constructed in the proof of Lemma 15. We claim that L is precisely the set of all normal forms, or irreducible words of $G^{\#}$.

Suppose $w \in L$, and $w \rightarrow w'$. Then $w'w^{-1} = \lambda$, and $w' \ll w$, since $\rightarrow$ is induced by a reduction order. But this contradicts the fact that the Cayley language L contains only minimal representatives.

Now, suppose that w is irreducible. The construction of Lemma 15 produces a word $z = \text{SELECT}(w) \in L$, with $wz^{-1} = 1$, and $w \not\gg z$. Consider the word $wz^{-1}z$. Since G is confluent, $wz^{-1} \rightarrow^* \lambda$ and $z^{-1}z \rightarrow^* \lambda$. By property (3) of a reduction order, there exist reduction sequences $wz^{-1}z \rightarrow^* 1 \cdot z = z$ and $wz^{-1}z \rightarrow^* w \cdot 1 = w$. But then, $w = z$, since $wz^{-1}z$ can't reduce to distinct irreducibles. Hence $w \in L$.

Writing the set of rewrite rules as $P = \{(\alpha_i, \beta_i)\ (i \in I)\}$, the set of irreducible words is the complement of

$$\bigcup_{i \in I} G^{\#} \alpha_i G^{\#},$$

which is a regular set if $|P|$ is finite. Since $G^{\#}$ itself is the complement of the empty set, the set of irreducible words is seen to be a regular aperiodic language. ∎

As an example, we consider the free abelian groups A_s of finite rank s. The Cayley language for A_s presented earlier was regular, but nor regular aperiodic. However, A_s possesses a confluent presentation, which essentially freely reduces, and commutes the generators so that their indices form a non-decreasing sequence:

$$\{a_j^{\epsilon_1} a_i^{\epsilon_2} \rightarrow a_i^{\epsilon_1} a_j^{\epsilon_2}\ (1 \leq i < j \leq s,\ \epsilon_i = \pm 1)\}$$
$$\bigcup\ \{a_i a_i^{-1} \rightarrow 1,\ a_i^{-1} a_i \rightarrow 1\ (1 \leq i \leq s)\}.$$

With respect to this presentation, the set of irreducible words is obtained

by excluding from $G^{\#}$ both the freely reduced words, and the sets

$$G^{\#}(b_i|b_i^{-1})(b_j|b_j^{-1})G^{\#}, \quad (\forall j < i).$$

3.2.5 Groups of Dehn's Algorithm

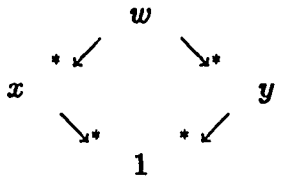
A group G of Dehn's Algorithm, or a *DA–Group*, is a finitely generated group whose word problem is solvable by a *monotonic reduction process.* [19] By this, we mean that there is a finite subset of $G^{\#} \times G^{\#}$, $\{(\alpha_i, \beta_i) : i \in I\}$, with $|\alpha_i| > |\beta_i|$, $\alpha_i = \beta_i$ in G, and in addition, any non-empty word $w \in G^{\#}$ that is equal to the identity contains a subword α_k, for some $k \in I$. Dehn's algorithm for deciding whether or not $w = 1$ in G consists of successively replacing subwords α_k of w by β_k, until no further substitution is possible. Since each substitution decreases the length of w, this process terminates after a finite number of steps. For a DA–group, $w = 1$ if and only if this process terminates with the empty string.

A free group of finite rank, $F_r = \langle a_1, \ldots a_r; \rangle$ is a DA–group. The list of allowable replacements is the set $\bigcup_{i=1}^{r}\{(a_i a_i^{-1}, \lambda),\ (a_i^{-1} a_i, \lambda)\}$.

In 1912, Dehn proved that the word problem for the fundamental groups of closed, (two-dimensional) orientable surfaces of genus $g \geq 2$ is also solvable by means of such a monotonic reduction process [48]. Domanski and Anshel [19] proved in 1985 that the word problem for DA–groups is solvable in linear time on a multi-tape Turing machine.

DA–groups are said to be *Church-Rosser on the congruence class of the*

identity; that is, the following diamond condition holds for any $w = 1$:



We emphasize that if $w \neq 1$, then w may reduce to distict irreducibles; i.e., Dehn's algorithm solves the word problem, but does not, in general, provide a normal form algorithm. Therefore, we do not expect that the DA–groups necessarily possess a regular aperiodic minimal Schreier cross-section, as in the case of the confluent groups.

Theorem 19 *Let G be a finitely generated DA–group. Then G admits a context-sensitive Cayley language.*

Proof: Let L be the Cayley language as constructed in Lemma 15. Given $w \in G^{\#}$, deciding whether $w \in L$ requires space for solving the word problem, and for enumerating words z of $G^{\#}$, $z \ll w$. Domanski and Anshel [19] have shown that G has word problem solvable in linear time, and hence in linear space, on a multi-tape Turing machine. The enumeration may be performed using $|w| + 1$ tape cells, so the total space required is linear in $|w|$. Hence, L is a context–sensitive language. ∎

Tretkoff [59] attributes to Otto the result that groups with finite Church–Rosser presentations with length–decreasing rules have context–free word problem. By the Muller–Shupp theorem, these groups are all virtually free. Madlener and Otto have shown that groups that are confluent over the congruence class of the identity with respect to a rewriting relation in which all

rules are length–decreasing properly contain the virtually free groups. Therefore, DA–groups do not, in general, admit context-free Cayley languages.

We note that Theorem 19 is in fact a corollary of the following stronger result:

Theorem 20 *If G is a group having deterministic context-sensitive Word Problem, then G admits a context-sensitive Cayley language.*

Proof: We show that under the hypotheses, the Cayley language L that is constructed in Lemma 15 is a CSL. Recall that L is defined as follows:

$$w \notin L \quad \text{if and only if} \quad \exists z \ll w \text{ such that } wz^{-1} = 1.$$

Let $n = |w|$. We have already shown (in the discussion preceding Figure [5]) that SUCCESSOR(w) $\in$ DSPACE(n). Since WP(G) is a DCSL, there is a deterministic Turing machine, WPM, which halts on input x if and only if $x = 1$. (We follow the convention that, rather than entering a rejecting state, WPM loops if $x \neq 1$.) In Figure [17], we describe a nondeterministic Turing machine M $\in$ NSPACE(n) which accepts $\Sigma^* \setminus L$.

Essentially, M works on input w as follows: In the special case of $w = \lambda$, M rejects w by looping, since every Cayley language contains the empty word. If $\lambda \ll w$, then M starts enumerating the elements of $G^{\#}$, and nondeterministically chooses a word z. M will only reach step 4 if $z \ll w$. M reaches step 6 and halts if and only if M has already reached step 4, and WPM halts on wz^{-1}; i.e., if and only if a word $z \ll w$ has been found, with $z = w$. The actual number of tape cells used by M is at most $4|n + 2|$, including end-of-string markers.

Since M $\in$ NSPACE(n), $\Sigma^* \setminus L$ is context-sensitive. By Immerman's result, its complement L is also a context-sensitive language. ∎

M has three tapes; the Input tape, initially containing a word w, and two work tapes, z and x. M starts in state 0.

(trivial case)	0.	If $w = \lambda$, go to 0. (loop; $w \in L$.)
(initialize)	1.	$z = \lambda$.
(form successor)	2.	$z =$ SUCCESSOR(z).
(verify order)	3.	if $z = w$, go to step 3 (loop); otherwise, go to either step 2 or to step 4.
(form test word)	4.	$x = wz^{-1}$
(solve WP)	5.	run WPM on input x.
(w is not in L)	6.	halt; $w \notin L$.

Figure [17]: A Turing Machine accepting the complement of the Cayley language of Lemma 15.

3.2.6 Polycyclic–by–Finite Groups

Definition: A group G is *polycyclic* if there exists a finite normal series

$$1 = G_0 \lhd G_1 \lhd \cdots \lhd G_n = G$$

such that G_{i+1}/G_i is cyclic, for $0 \leq i < n$. A group G is *polycyclic–by–finite* if G has a polycyclic subgroup of finite index which is normal in G.

Definition: A set $X \subseteq \Sigma^*$ is *bounded* if $\exists w_1, \ldots, w_n \in \Sigma^*$ such that $X \subseteq w_1^* \cdots w_n^*$.

Gilman [27] observes that a polycyclic–by–finite group G has a bounded regular cross-section. We show here that a polycyclic by finite group G possesses a regular aperiodic cross–section.

Lemma 18 *If $a \in \Sigma$ then a^* is a regular aperiodic language.*

Proof: $a^* = \Sigma^* \setminus (\Sigma^*(\Sigma \setminus \{a\})\Sigma^*)$, since the complement of a^* in Σ^* is the set of words containing something other than an a.

Theorem 21 *If G is an extension of a polycyclic group by a finite group, then G possesses a regular aperiodic cross–section.*

Proof: We first show that the assertion is true if G is simply polycyclic, using induction on the length of a polycyclic series for G.

Let G have polycyclic series $1 = G_0 \lhd G_1 \lhd \cdots \lhd G_n = G$. If $n = 1$, then either G is finite, in which case the assertion holds trivially, or else G is a free group of rank 1, which we have already shown to have a regular aperiodic cross–section.

Inductively, assume $n > 1$, and G_{n-1} has regular aperiodic cross–section denoted by a regular expression R_{n-1}, and G_n/G_{n-1} is cyclic, generated by $g_n G_{n-1}$. As sets, G_n is the disjoint union $\bigcup g_n^i G_{n-1}$.

Case 1: $|G_n/G_{n-1}|$ is finite, say of order m. Then a cross–section for G is the language

$$\bigcup_{i=0}^{m-1} g_n^i R_{n-1},$$

which is regular aperiodic, given by the regular expression

$$R_{n-1} \mid g_n R_{n-1} \mid \cdots \mid g_n^{m-1} R_{n-1}.$$

Case 2: G_n/G_{n-1} has infinite order. Then G has cross–section

$$(g_n)^* R_{n-1} \bigcup (g_n^{-1})^* R_{n-1},$$

and by Lemma 18 above, this is also regular aperiodic.

Now, suppose G is polycyclic by finite, with H $\lhd$ G, H polycyclic, and suppose H has a regular cross-section given by a regular expression R. As in Case 1 above, if $|G/H| = m < \infty$, $\exists\, g_1, \ldots g_m \in G$ such that

$$G = \bigcup_{i=1}^{m} g_i H,$$

and so a regular aperiodic cross-section for G is given by

$$(g_1 R) \bigcup \cdots \bigcup (g_m R).$$

3.2.7 Linear Groups

Definition: Let $\mathcal{F}$ be a field of characteristic zero. A group is a *linear group* over $\mathcal{F}$ provided it is isomorphic to a group of $k \times k$ invertible matrices over $\mathcal{F}$, for some positive integer k.

Theorem 22 *If G is a finitely generated group embeddable in $GL(n, \mathcal{F})$ then G possesses a context-sensitive Cayley Language.*

Proof: In the proof of Lemma 15, we constructed a recursive cross-section L of G. To show that L is in fact context-sensitive, it is sufficient to show that the predicate MEMBER can be decided in linear space.

The complexity of deciding membership in L, using the algorithm of Lemma 15, depends only on the complexity of the word problem for G, and on the complexity of the enumeration of $G^{\#}$ with respect to $\ll$. R. Lipton and Y. Zalcstein [45] have shown that the word problem for such a group G is solvable in logspace. We have already described a single tape Turing machine SUCCESSOR that finds the successor of a word x in $G^{\#}$, which visits at most $|x| + 1$ tape cells.

Our algorithm for deciding membership in L, on input w, does not enumerate any words past w, so the number of cells visited (not including end-of-tape markers) is at most $|w|$. Since the total space required is bounded by a linear function of the input size, we may conclude that L is context-sensitive.

3.3 The Burnside Problem.

3.3.1 A Historical Perspective.

In 1902, W. Burnside asked whether or not a finitely generated group in which every element has finite order is necessarily finite. Such groups are now called periodic groups. [28] Burnside went on to pose a less general form of this question: Is an m-generator group G in which $x^n = 1, \forall x \in G$, necessarily finite, and if so, what is its order?

Current work regarding these questions revolves around the *free Burnside group* $B(m,n)$ of rank m and exponent n, by which we mean an m-generator group subject only to the condition that the n^{th} power of every element is the identity; i.e., $B(m,n) = F/N$, where F is the free group of rank m, and N is the normal subgroup of F generated by $\{x^n : x \in F\}$. [48, page 379]

For some m,n, the answers are well-known. $B(m,1)$ is the trivial group, while $B(1,n)$ is cyclic of order n. $B(m,2)$ is easily seen to be abelian, with order 2^m. Burnside proved that $|B(m,3)| \leq 3^{2^m-1}$. In 1933, Levi and Van der Waerden strengthened this, proving that

$$|\mathrm{B(m,3)}| \leq 3^{\binom{m}{1}+\binom{m}{2}+\binom{m}{3}},$$

where $\binom{m}{k}$ is the binomial coefficient $\frac{m!}{k!(m-k)!}$.

I. Sanov proved in 1940 that all free Burnside groups of exponent 4 are finite. In 1956, Philip Hall and Graham Higman predicted the order of $B(m,6)$ if in fact $B(m,6)$ was finite; two years later, Marshal Hall, Jr. verified that $B(m,6)$ was indeed finite.

The negative answer to Burnside's problem was given in 1964, when the existence of infinite finitely generated periodic groups was demonstrated by

E. S. Golod, as a result of joint work with I. R. Shafarevitch.

In 1968, P. S. Novikov and S. I. Adian established the existence of infinite Burnside groups for infinitely many exponents. Their original result has been strengthened, and we state without proof a result due to Adian:

Theorem 23 (Adian) *The free Burnside groups $B(m,n)$ are infinite in case $m \geq 2$ and n is odd, $n \geq 665$. Moreover, the word problem for these groups is solvable.*

We remark that in 1984, M. F. Newman presented a computer-based proof that $B(m,6)$ is finite. The question of whether or not $B(2,5)$ is finite remains open.

3.3.2 Coset Representative Systems of Periodic Groups.

We now turn our attention to language-theoretic characterizations of these groups. We make use of a key result used to prove that a language is not context–free, proven in 1961 by Bar-Hillel, Perles, and Shamir:

Theorem 24 (The Pumping Lemma for CFL's) *Let L be a CFL. Then there exists a constant n, depending on L, such that if $z \in L$, with $|z| \geq n$, then z may be written as $z = uvwxy$, where v and x are not both empty, $|vwx| \leq n$, and $\forall i \geq 0$, $uv^iwx^iy \in L$.*

Theorem 25 *If G is a finitely generated periodic group, and H is a subgroup of G, then a coset representative system L for G/H is a CFL if and only if H has finite index in G.*

Proof: If H has finite index in G, then L is finite. A finite set of words is regular and therefore context-free. Suppose H does not have finite index in G. Then L is infinite, and since G has only a finite number of generators, L must contain words of arbitrary length. Suppose L were context-free. We could then choose a sufficiently long word $z = uvwxy$, with vx not empty, as in the Pumping Lemma. Let the orders of v and x, as elements of G, be p and q, respectively. Consider $z' = uv^{pq+1}wx^{pq+1}y$. By the Pumping Lemma, $z' \in L$. Now, z and z' are unequal as words in L, but represent the same element of G. This contradicts the assumption that L is a coset representative system for G/H. ∎

By considering Theorem 25 in the special case when H is the trivial subgroup $\{1\}$, we obtain a language-theoretic characterization of finitely generated periodic groups in terms of their possible enumerations.

Corollary 2 *A finitely generated periodic group G is infinite if and only if G does not admit a context-free enumeration.*

Corollary 3 *An infinite free Burnside group of rank $m \geq 2$ and exponent n, with n odd and $n \geq 665$ possesses a recursive, but no context-free, Cayley language.*

Proof: Let G be one of these free Burnside groups. Adian [1] has shown that G has solvable word problem, so by Lemma 15, G admits a recursive Cayley language. By corollary 2, no Cayley language of G is context-free. ∎

Bibliography

[1] S. I. Adian. "The Burnside Problem and Identities in Groups", *Ergebnisse der Mathematik und ihrer Grenzgebeite* **95**, (Springer, Berlin, 1979).

[2] A. V. Aho, J. Hopcroft and J. Ullman. *The Design and Analysis of Computer Algorithms*, (Addison-Wesley, Reading, Mass., 1974).

[3] A. V. Anisimov. "Group Languages", *Kibernetica* **4** (1971), pp. 18–24.

[4] M. Anshel. "Conjugate Powers in HNN Groups", *Proc. Amer. Math. Soc.* **54**, (1976), pp. 19–23.

[5] M. Anshel. "Decision Problems for HNN groups and Vector Addition Systems", *Math. Comp.* **30**, (1976), pp. 154–156.

[6] M. Anshel. "The Conjugacy Problem for HNN Groups and the Word Problem for Commutative Semigroups", *Proc. Amer. Math. Soc.* **54**, (1976), pp. 223–224.

[7] M. Anshel. "Vector Groups and the Equality Problem for Vector Addition Systems", *Math. Comp.* **32**, (1978), pp. 514–516.

[8] M. Anshel and P. Stebe. "The Solvability of the Conjugacy Problem for Certain HNN Groups", *Bull. Amer. Math. Soc.* **80** (2), (1974), pp. 266–269.

[9] S. Baase. *Computer Algorithms*, (Addison-Wesley, Reading, Mass., 1988).

[10] R. Book. "Confluent and Other Types of Thue Systems", *J. Assoc. Comput. Mach.*, **29**, (1982), pp. 171–182.

[11] R. Book. "Dehn's Algorithm and the Complexity of Word Problems", *Amer. Math. Monthly* **95** (10), (Dec. 1988), pp. 919–925.

[12] J. L. Britton. "The Word Problem", *Ann. of Math.*, **77**, (1963), pp. 16-32.

[13] B. Buchberger. "Ein Algorithmisches Kriterium für die Lösbarkeit eines algebraischen Gleichungssystems", *Aequationes Math.* **4** (3), (1970), pp. 374–383.

[14] B. Buchberger. "Applications of Gröbner Bases in Non–Linear Computational Geometry", in *Geometric Reasoning*, (D. Kapur and J. Mundy, Eds., MIT Press, Cambridge, Mass., 1989).

[15] N. Chomsky. "Context-free Grammars and Pushdown Storage", *Quarterly Prog. Rept. No 65*, (MIT Res. Lab Elect., Cambridge, Mass.), pp. 187–194.

[16] N. Chomsky and G. Miller. "Finite State Languages", *Information and Control*, **1**, pp. 91–112.

[17] T. Chou and G. Collins. "Algorithms for the Solution of Systems of Linear Diophantine Equations", *SIAM J. Comput.* **11** (4), (Nov. 1982).

[18] D. Cohen. *Combinatorial Group Theory: A Topological Approach*, (Cambridge University Press, Cambridge, 1989).

[19] B. Domanski and M. Anshel. "The Complexity of Dehn's Algorithm for Word Problems in Groups", *J. Algorithms* **6**, (1985), pp. 543–549.

[20] T. Dubé. "The Structure of Polynomial Ideals and Gröbner Bases", *SIAM J. Comput.* **19** (4), (Aug. 1990), pp. 750–773.

[21] M. J. Dunwoody. "The Accessibility of Finitely Presented Groups", *Invent. Math.* **81**, (1985), pp. 449–457.

[22] J. Evey "Application of Pushdown Store Machines", *Proc. 1963 Fall Joint Computer Conference*, pp. 215–227, (AFIPS Press, Montvale, New Jersey).

[23] M. Frumkin. "Polynomial Time Algorithms in the Theory of Linear Diophantine Equations", in *Fundamentals of Computation Theory*, Lecure Notes in Comp. Sci. **56**, M. Karpinski, Ed. (Springer, New York, 1977) pp. 386–392.

[24] M. Garey and D. S. Johnson. *Computers and Intractibility; A Guide to the Theory of NP-completeness*, (W. H. Freeman and Company, San Fransisco, 1979).

[25] J. von zur Gathen and M. Sieveking. "Weitere zum Erfullungsprobleme polynomial äquivalente kombinatorische Aufgaben", in *Lecture Notes in Computer Science* **43**, (Springer, Berlin, 1976).

[26] P. Gianni, B. Trager and G. Zacharias. "Gröbner Bases and Primary Decomposition of Polynomial Ideals", *preprint, IBM T.J. Watson Laboratories* (Yorktown Heights, N.Y., 1986).

[27] R. H. Gilman. "Groups with a Rational Cross-Section", (in *Combinatorial Group Theory and Topology*, S.M. Geister and J.R. Stallings, Princeton University Press, Princeton, New Jersey, 1987).

[28] N. Gupta. "On Groups in which Every Element has Finite Order", *Amer. Math. Monthly* **96** (4) (1989), pp. 297–308.

[29] R. H. Haring-Smith. "Groups and Simple Languages", *Trans. Amer. Math. Soc.* **79** (1), (1980), pp. 223–224.

[30] G. Higman, B. H. Neumann, and H. Neumann. "Embedding Theorems for Groups", *J. London Math. Soc*, **26**, pp. 247–254.

[31] D. Hoffman. "The Computer-Aided Discovery of New Embedded Minimal Surfaces", *The Mathematical Intelligencer* **9** (3), (Springer, New York, 1987).

[32] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation* (Addison–Wesley, New York, 1979).

[33] R. D. Hurwitz. "On Cyclic Subgroups and the Conjugacy Problem", *Proc. Amer. Math. Soc.* **279** (1), (1983), pp. 337–356.

[34] D.T. Huynh. "A Superexponential Lower Bound for Gröbner Bases and Church–Rosser Commutative Thue Systems" *Information and Control*, **68**, (1986), pp. 196–206.

[35] C. Iliopoulos, "Worst-Case Complexity Bounds on Algorithms for Computing the Canonical Structure of Infinite Abelian Groups and Solving Systems of Linear Diophantine Equations", *SIAM J. Comput.* **18** (4), (Aug. 1979).

[36] N. Immerman. "Nondeterministic Space is Closed Under Complementation", *SIAM J. Comput.*, **17** (5), (1988), pp. 935–938.

[37] N. Jacobson. *Basic Algebra I*, (W. H. Freeman and Company, San Fransisco, 1974).

[38] M. Jantzen. "On a Special Monoid with a Single Defining Relation", *Theor. Comput. Sci.* **16**, (1981), pp. 61–73.

[39] M. Jantzen. *Confluent String Rewriting*, (Springer, Berlin, 1988).

[40] R. Kannan and A. Bachem, "Polynomial Algorithms for Computing the Smith and Hermite Normal Forms of an Integer Matrix", *SIAM J. Comput.* 8 (4), (Nov. 1979).

[41] S. Kleene. "Representation of Events in Nerve Nets and Finite Automata" *Automata Studies*, pp. 3–42, (Princeton University Press, Princeton, New Jersey, 1956.)

[42] D. Knuth, J. Morris, V. Pratt. "Fast Pattern Matching in Strings", *SIAM J. Comput*, **6** (2), (1977), pp. 323–350.

[43] S. Y. Kuroda. "Classes of Languages and Linear Bounded Automata", *Information and Control*, **7** (2), (1964), pp. 207–223.

[44] P. Le Chenadec. "A Catalogue of Complete Group Presentations", *J. Symbolic Computation* **2** (1986), pp. 363–381.

[45] R. Lipton and Y. Zalcstein. "Word Problems Solvable in Logspace", *J. Assoc. Comput. Mach.*, **24** (3), (1977), pp. 522–526.

[46] R. Lyndon and P. Schupp. *Combinatorial Group Theory*, (Springer, Berlin, 1977).

[47] Z. Manna. *Mathematical Theory of Computation*, (McGraw-Hill, New York, 1974).

[48] W. Magnus, A. Karrass, D. Solitar. *Combinatorial Group Theory*, (Second Edition, Dover, New York, 1976).

[49] E. Mayr and A. Meyer. "The Complexity of the Word Problems for Commutative Semigroups and Polynomial Ideals", *Adv. in Math.*, **46**, (1982), pp. 305–329.

[50] C. F. Miller III. *On Group-theoretic Decision Problems and their Classification*, Ann. of Math. Studies, **68**, (Princeton University Press, Princeton, New Jersey, 1971.)

[51] D. Muller and P. Schupp. "Groups, the Theory of Ends, and Context-Free Languages", *J. Comput. System Sci.*, **26**, (1983), pp. 295–310.

[52] B. Nash. "Reachability Problems in Vector Addition Systems", *Amer. Math. Monthly* **80**, (1973), pp. 292–295.

[53] M.H.A. Newman. "On Theories with a Combinatorial Definition of Equivalence",*Annals of Math.* **43**, (1942) pp. 223–243.

[54] A. Oettinger. "Automatic Syntactic Analysis and the PushdownStore", *Proc. Symposia in Applied Math.* **12**, (American Mathematical Society, Providence, Rhode Island, 1961).

[55] C. Pomerance. "Factoring", in *Proceedings of Symposia in Applied Math., (Cryptology and Computational Number Theory*, C. Pomerance, Ed.), **42**, (American Mathematical Society, 1990).

[56] F. S. Roberts. *Applied Combinatorics*, (Prentice-Hall, New Jersey, 1984).

[57] M. Schutzenberger. "On Context-free Languages and Pushdown Automata", *Information and Control* **6** (3), pp. 246– 264.

[58] H. Simmons. "The Word and Torsion Problems for Commutative Thue Systems", in *Word Problems II* (S. I. Adian, W.W.Boone, and G. Higman, Eds.), (North-Holland, New York, 1980), pp. 395-400.

[59] C. Tretkoff. "Complexity, Combinatorial Group Theory and the Language of Paluators", *Theor. Comp. Sci.*, **56**, (1988), pp. 253–275.

[60] W. Trinks. "Über B. Buchbergers Verfahrnen, Systeme algebraischer Gleichungen zu Lösen.", *J. Number Theory*, **10** (4), (1978), pp. 475–488.

[61] B. Wehrfritz. "Infinite Linear Groups", *Ergebnisse der Mathematik und ihrer Grenzgebeite*, **76**, Springer, Berlin (1973).

[62] F. Winkler, B. Buchberger, F. Lichtenberger, and H. Rolletscheck. Algorithm 628: An Algorithm for Constructing Canonical Bases of Polynomial Ideals", *ACM Trans. on Math. Software* **11** (1), (1985), pp. 66–78.