REU 2011: THE CONJUGACY PROBLEM IN GROUP THEORY

JENS HARLANDER (MENTOR), HANNAH LEWIS,
JONATHAN SIEGEL, CHAO XU

1. Introduction

At the center of every crypto system lies a mathematical trapdoor, that is, a computational problem that is easy to perform in one direction (encryption) but difficult to reverse (decryption). The security of the system depends on the difficulty of the reverse computation. The most common problems used are the computation of prime factorizations and the computation of discrete logarithms. These problems are not ideal for many reasons. It has not been proven that these problems do not have efficient solutions. Thus, the security on the most commonly used crypto systems rest on believe. Furthermore, effective algorithms for solving these problems on quantum computers have been found. Should such machines become a reality, the security of the present systems will be compromised. The conjugacy problem from group theory is a promising mathematical trapdoor for devising non-commutative secure crypto systems. The braid groups have been suggested as platform groups for various reasons. This choice however is also widely criticized.

The purpose of these notes is to collect various settings in which the conjugacy problems can be solved and to discuss the computational complexity of these solutions. We investigate which methods apply to the braid groups. It turns out most apply to B_3, but not to braid groups on more than three strands (automaticity being an exception, but the solution via automatic structures seems non-efficient, except perhaps for B_3).

The literature on the braid groups in vast. For the most part we restrict our attention to the braid group on three strands, which already displays a rich structure. We present complete and elementary proof for B_3, and refer to existing literature for the general setting. We hope that our exposition illuminates some of the fundamental ideas involved, without obscuring matters with technical details.

These notes record the work of three undergraduate students, conducted as part of an REU during the Summer of 2011 at Boise State University. Jens Harlander acted as mentor. Uwe Kaiser acted as remote co-mentor, via Skype from Germany, and delivered a series of lectures on braid group representations which is included as the last section. We thank the NSF for generous funding.

2. Presentations of the symmetric group and the braid group

This section needs to contain the standard presentation for the symmetric group, a proof that it indeed presents the symmetric group (normal forms). A solution for the word and conjugacy problem should be discussed, and also the complexity of the algorithms used. The standard presentation for symmetric group S_n is $S_n = \langle x_1, x_2, \dots, x_n \mid x_i^2 = 1, x_{i+1}x_ix_{i+1} = x_ix_{i+1}x_i, x_ix_j = x_jx_i\rangle$ where $|i-j| \neq 1$.

The word problem for any finite group of order n can be solved in linear time with respect to the length of the word on a finite state automata of exactly n states. Although the construction will cost $O(n^2)$ time given the multiplication table. The naive approach takes $O(l(n!)^2)$ time for S_n.

2.1. An normal form algorithm.

Let G be the group presented by

$$\langle x_1, x_2, \dots, x_n \mid x_i^2 = 1, x_{i+1}x_ix_{i+1} = x_ix_{i+1}x_i, x_ix_j = x_jx_i\rangle,$$

where $i, j \in \{1, ..., n\}$, $|i - j| \neq 1$. Let $U_k = \{1, x_k, x_k x_{k-1}, \ldots, x_k \ldots x_1\}$, $0 \leq k \leq n$. We say a word w is in normal form if $w = u_1 u_2 \ldots u_n$, where $u_k \in U_k$, $k = 1, ..., n$. Note that there are $n!$ words in normal form. The following algorithm shows that every element of G can be represented by a word in normal form:

(1) Replace each occurance of x_i^{-1} by x_i using the relations $x_1^2 = 1$.
(2) Cancel all x_i^2 in the word.
(3) Find the largest generator x_j that occurs first in w.
(4) If the next generator in w is x_i, and $|j - i| \neq 1$, then replace $x_j x_i$ with $x_i x_j$, do step 2, and repeat step 3.
(5) If step 4 cannot be applied, then we have a decreasing chain $x_j x_{j-1} \ldots x_r$. Find x_r.
(6) Apply step 4 with x_r; if that is not possible, then we have a chain of the form $x_j x_{j-1} \ldots x_{r+1} x_r x_{r+1}$. Apply the appropriate relation so that we have $x_j x_{j-1} \ldots x_r x_{r+1} x_r$ and repeat step 3.
(7) Repeat until $w = w' x_j x_{j-1} \ldots x_r$, where w' only contains generators smaller than x_j. Apply the algorithm on w'.

Let S_n be the symmetric group on $\{1, ..., n\}$. Let $\rho \colon G \to S_n$ be the group homomorphism that sends x_i to the transposition $\tau_i = (i\ i+1)$. Since the τ_i generate S_n, we see that ρ is surjective. Since every element of G can be represented by a word in normal form G has $n!$ elements. Hence ρ is a group isomorphism. We summarize our findings:

Theorem 2.1. *The presentation*

$$\langle x_1, x_2, \ldots, x_n \mid x_i^2 = 1, x_{i+1} x_i x_{i+1} = x_i x_{i+1} x_i, x_i x_j = x_j x_i \rangle,$$

where $i, j \in \{1, ..., n\}$, $|i - j| \neq 1$, defines the symmetric group S_n. The algorithm described above solves the word problem for S_n with complexity $O(l^2)$, where l is the length of the word.

Note that if there exist an algorithm A that produces the normal form $A(w)$ of a word w of length l in $O(f(l, n))$ time, then one can always construct an algorithm A' that produces the normal form in $O(lf(n^2, n))$ time. Indeed, let w be a word in the generators. We may assume it is a positive word, so $w = y_1 ... y_l$, where the y_i are generators. We produce the normal form in the following way: in the first step we find $y_1 ... A(y_l)$, in the second step we find $y_1 ... A(y_{l-1} A(y_l))$, etc. Denote this algorithm by A'. Note that A' runs the algorithm A exactly l times, and each time runs A on a word of length less than $1/2n(n+1) + 2$, because the longest word in normal form has length $1/2n(n+1) + 1$. Thus A' produces the normal form in $O(lf(n^2, n))$ time. If we apply this method to the algorithm above, which runs in $O(l^2)$ time, we obtain one that runs in $O(ln^4)$, time.

Theorem 2.2. *There exists an algorithm that produces the normal form in S_n in $O(ln^4)$ time.*

2.2. **Algorithms for solving the word and conjugacy problem.** Given the representation of S_n as the set of cyclic permutations, there exist a linear algorithm in both n and l.

Theorem 2.3. *The following is a $O(n + l)$ algorithm for the word problem.*

(1) *Produce an list of number $a_1, \ldots, a_n$, such that $a_i = i$.*
(2) *Read the word letter by letter. If one encounters x_i, swap the value of a_i and a_{i+1}.*
(3) *If $a_i = i$ for all i, then the word is trivial, else it's not.*

Proof. The algorithm takes $O(n + l)$ time is obvious. The correctness needs to be justified.

x_i can be represented as the transposition $(i\ i+1)$. Define $(n\ n+1) = (n\ 1)$.

Represent a element of the group as a permutation π in the 2 line notation. wlog, assume $\pi(j) = i$ and $\pi(k) = i + 1$.

$$\begin{pmatrix} 1 & \cdots & j & \cdots & k & \cdots & n \\ \pi(1) & \cdots & i & \cdots & i+1 & \cdots & \pi(n) \end{pmatrix} (i\ i+1) = \begin{pmatrix} 1 & \cdots & j & \cdots & k & \cdots & n \\ \pi(1) & \cdots & i+1 & \cdots & i & \cdots & \pi(n) \end{pmatrix}$$

If we call j the index of i if $\pi(j) = i$, then each transposition is a swap of indices.

The value of a_i in the algorithm stores the index of i. The list represent the identity iff $a_i = i$ for all i.

This proves the the correctness of the algorithm.

□

The conjugacy problem can be solved similarly. It is known that two elements are conjugate if and only if they have the same number of the same cycle types. The algorithm also runs in $O(n+l)$ time.

(1) Use the algorithm in the word problem, find the final permutation of each word. Let it be $a_1, \ldots, a_n$ and $b_1, \ldots, b_n$.
(2) Construct a list $\{L\}$ of length n. $L_i = 0$ initially for all i. In this algorithm, $L_i = 1$ if and only if we already found a i in some cycle.
(3) Construct a list $\{N\}$ of length n. $N_i = 0$ initially for all i. In the end of the algorithm, N_i will hold the difference of the times a cycle with length i occurs between w and w'.
(4) Set $i \leftarrow 1$.
(5) If $L_i = 1$, then $i \leftarrow i + 1$ and repeat this step, else find the cycle in w that contains i. For every element j visited in the cycle, mark them visited by setting $L_j \leftarrow 1$. Let the length of the cycle be c, then $N_c \leftarrow N_c + 1$.
(6) Set $\{L\}$ to be all 0, $i \leftarrow 1$.
(7) Do step (5), but this time, use w', and set $N_c \leftarrow N_c - 1$ if a cycle of length c is found.
(8) If $N_i = 0$ for all i, then $w \sim w'$.

The first step cost $O(n + l)$. To find a cycle, start from some element i, then find the a_ith element, and then find the a_{a_i}th element, ..., until i is reached again. The time to search for a cycle depend on the cycle length. If the length is c, then the time complexity is $O(c)$. The sum of cycle length is n, therefore step (5) and (7) cost $O(n)$ time. It's trivial to show the other steps are also $O(n)$. The first step dominates the complexity, and the algorithm runs in $O(n + l)$ time.

2.3. **Word problem for braid group.** The braid group on n strands B_n is defined by the presentation $B_n = \langle \sigma_1, \sigma_2, \ldots, \sigma_{n-1} | \sigma_i\sigma_{i+1}\sigma_i = \sigma_{i+1}\sigma_i\sigma_{i+1} \ , \ \sigma_i\sigma_j = \sigma_j\sigma_i \rangle$, where $|i - j| \neq 1$.

Artin first solved the word problem for the braid groups using a representation[reference]. There exist a faithful homomorphism $^* : B_n \to \mathrm{Aut}(F_n)$.

If $B_n \to \mathrm{Aut}(F_n)$ is faithful and it sends w to w^*, then w is 1 iff $w^*(a) = a$ for every generator a of the free group F_n.

Artin demonstrated a explicit map. Defined as

$$\sigma_i^*(t_j) = \begin{cases} t_j & \text{if } j \neq i, i+1 \\ t_{i+1} & \text{if } j = i \\ t_{i+1}t_it_{i+1}^{-1} & \text{if } j = i+1 \end{cases}$$

Where t_i is generators of F_n, and σ_i are the generators of the braid group. The inverse can easily be found. Using a recursive definition, let any word of the form $(\sigma_i w)^*$ be $\sigma_i^* \circ w^*$. The algorithm is simply apply the automorphisms to every t_i, and see if all of them is the identity.

An application of σ_i^* can potentially triple the length of the word. The complexity of the algorithm is $O(3^l)$.

Consider the special case B_3. If we define a_n, b_n, c_n to be the number of t_1, t_2, t_3 (include it's inverses) in $((\sigma_2\sigma_1^{-1})^n)^*(t_1)$, ignoring the possibility of cancellation. We have the following recurrence relation.

$$a_{n+1} = 2a_n + b_n$$
$$b_{n+1} = c_n$$
$$c_{n+1} = a_n + 2c_n$$

Solving the recurrence relation, we find $a_n + b_n + c_n = 2F_{2n+1} - 1$, therefore $|((\sigma_2\sigma_1^{-1})^n)^*(t_1)| = 2F_{2n+1} - 1$, where F_n is the nth Fibonacci number.

It's easy to show there is no cancelation at any step. Thus it is exponential even for B_3.

3. Amalgamated products

3.1. **Different Presentations for B_3.** The three string braid group has standard representation $< \sigma_1, \sigma_2 | \sigma_1\sigma_2\sigma_1 = \sigma_2\sigma_1\sigma_2 >$. However we can create an equivalent presentation which is easier to work with by making the substitution $a = \sigma_1\sigma_2\sigma_1$ and $b = \sigma_1\sigma_2$. Clearly we then also have $\sigma_1 = b^{-1}a$ and $\sigma_2 = ab^{-1}$. This yeilds the presentation $B_3 =< a, b | a^2 = b^3 >$. We can now use this presentation and the fact that $b^3 = a^2$ commutes with every other element to give a simple algorithm for the word problem.

1. Write the word in the braid group with respect to the generators a, b.
2. Reduce freely, i.e cancel all aa^{-1} and bb^{-1}.
3. Rewrite all a^2 as b^3 and a^{-2} as b^{-3}.
4. Commute all b^3 and b^{-3} to the left end of the word and repeat step 2.
5. If the resulting word is empty then the original word was the identity.

Before solving the conjugacy problem in B_3 I will indicate some other results that we have obtained. First I will need to define the amalgamated product as most of our original results had to do with the properties of the amalgamated product especially with regard to the word and conjugacy problems.

Definition 3.1. *Let G and H be groups. Then the free product of G and H, denoted by $G * H$ is the group of reduced strings whose characters are elements of $G \cup H$. A string is reduced using multiplication in either of the groups G or H, that is g_1g_2 is replaced by the single element $g_1 * g_2$ if both $g_1 \in G$ and $g_2 \in G$. The group operation is concatenation.*

Definition 3.2. *Let G, H, and K be groups and let $\phi : K \to G$ and $\nu : K \to H$ be homomorphisms. Then the amalgamated product of G and H is $G * H / << \phi(k)(\nu(k))^{-1} >>$. So the amalgamated product is the free product factored over the relations $\phi(k) = \nu(k)$ for all $k \in K$.*

Now we prove a fundamental theorem involving amalgamated products.

Theorem 3.1. *Let G, H, K we groups and let $\phi : K \to G$ and $\nu : K \to H$ be injective homomorphisms. Then the amalgamated free product $G * H$ with additional relations $\phi(k) = \nu(k)$ contains a subgroup isomorphic to K.*

Proof. Let $N =<< \phi(k)(\nu(k))^{-1} >>$ and consider the mapping ϕ' into the amalgamated product which maps $k \to \phi(k)N$. Here $\phi(k)N$ is the coset containing $\phi(k)$. If $\phi(k_1) \neq \phi(k_2)$, then $\phi(k_1)N \neq \phi(k_2)N$. This is true since equality here would imply that $\phi(k_1) \in \phi(k_2)N$. However, this is impossible since each of the reduced elements of N except for the identity must contain at least one character from G and at least one character from H since N is generated by such product pairs. Thus no element of $\phi(k_2)N$ reduces to an element of G except for $\phi(k_2)$ itself and thus $\phi(k_1) \notin \phi(k_2)N$. Thus we have proved that ϕ' is injective since if $k_1 \neq k_2$ we have $\phi(k_1) \neq \phi(k_2)$ since ϕ is injective and thus by the previous logic $\phi'(k_1) = \phi(k_1)N \neq \phi(k_2)N = \phi'(k_2)$. Hence the range of ϕ' is a subgroup of the amalgamated product which is isomorphic to K as required. □

Definition 3.3. *Let x be a reduced word in an amalgamated product of two groups. Define the length of x, denoted $l(x)$ to be the number of characters in x. So if $x = h_1k_2h_3$, then the length of x would be 3. Notice that the characters must be coming alternatingly from both groups for the word to be reduced.*

Now we come to our first theorem involving amalgamated products and the word problem.

Theorem 3.2. *Let H and K be groups and let G be their amalgamated product over some subgroup L. If the word problem is algorithmically solvable in both H and K and the membership problem is solvable for L, then the word problem in solvable for G.*

In the course of the proof we will give an algorithm for solving the word problem in G.

Proof. Denote the image of L in G by L^*. Define an element in G to be reduced if it cannot be shortened using the relations which hold in H or K. Let $x = h_1k_1h_2k_2...$ be a reduced element in G. Notice that x may end in either a character in H or a character in K. Define the number of characters in x to be the length of x and denote it $l(x)$. For example, if $x = h_1k_2h_3k_4h_5$, then $l(x) = 5$. Now apply the following algorithm:
1. If $l(x) = 0$, then $x = 1$, else find the first character in x which is in L^* (if it exists) and switch it to the corresponding character in the opposite group. Now reduce x using the relations which hold in H and K. Notice that this will always decrease $l(x)$.
2. If at any point $l(x) > 0$ and no character of x is in L^*, then $x \neq 1$.
To complete the proof we need to show that if $x = h_1k_2h_3...$ is a reduced element of G such that none of it's characters are in L^*, then x is not the identity. We will show that under these assumptions the length of x is minimal in the sense that any other word representing x cannot have shorter length.
Note that we cannot shorten x using any of the relations in H or K since x is reduced and that we cannot shorten x by rewriting a character in L^* by the assumption that none of the characters are in L^*. Hence the only operation we could perform on x which would potentially shorten it is adding an aa^{-1} somewhere in x and then reducing in some different way. Assume we have $h_rk_{r+1}aa^{-1}h_{r+2}$. Now assume that $a \notin L^*$. If $k_{r+1}a \in L^*$ and $a^{-1}h_{r+2} \notin L^*$ or vice versa, then by rewriting $k_{r+1}a$ or $a^{-1}h_{r+2}$ we see that the length of the word stays the same since one of the terms switches subgroups but the a or a^{-1} associated to the other term is in the original subgroup of the one that switched and is not in L^* and thus the length stays the same. If neither of the preceding terms is in L^* then we can only reduce using the relations in H and K and thus the aa^{-1} cancel and the length of the word also stays the same. Notice that it is not possible for both of the preceding terms to be in L^* since that would imply that $k_{r+1}h_{r+2} \in L^*$ which is absurd since then $k_{r+1} \in H$ and $h_{r+2} \in K$ and hence both are in L^* contrary to our assumption. Now assume that $a \in L^*$. Then the preceding case where neither of the terms is in L^* must be the case since L^* forms a subgroup of G and by assumption both k_{r+1} and h_{r+2} are not in L^*. Now assume we have lengthened the word by $h_rk_{r+1}aa^{-1}k_{r+2}h_{r+3}$. Then by a similar argument we have that the length of x stays the if $a \in L^*$ and if $a \notin L^*$. This covers all possible cases and completes the proof. □

Before stating the next lemma we introduce another definition.

Definition 3.4. *A word x in an amalgamated product is cyclically reduced if its length cannot be shortened by cyclically permuting or cancelling. This implies in particular that the first and last characters in x must come from opposite factors in the amalgamated product.*

Clearly if the word problem is solvable in each of the factor groups and the membership problem is solvable for the subgroup over which is being amalgamated, then an algorithm very similar to the one described in theorem 1.2 will easily cyclically reduce a word. Thus in our construction of an algorithm for solving the conjugacy problem we assume that the words being analyzed are already cyclically reduced.

Lemma 3.3. *Let H and K be groups and let G be their amalgamated product over some subgroup L. So we have $\phi : L \to H$ and $\nu : L \to K$ where both ϕ and ν are injective and we are amalgamating over the image of ϕ and ν. Then G contains a subgroup L^* isomorphic to L and if x and y are two cyclically reduced words which are conjugate, then $l(x) = l(y)$ and there exists a cyclic permutation of y say y^* such that x and y^* are conjuagate by an element c such that $l(c) = 1$. Moreover, if $l(x) = l(y) > 1$, then $c \in L^*$. Here $l(x)$ denotes the length defined in the proof of the previous theorem.*

Proof. Consider both of the cyclically reduced words $x = h_1k_1...h_nk_n$ and $y = h_1^*k_1^*...h_q^*k_q^*$. I claim that $n = r$. This is true because assume without loss of generality that $q < n$. Then since we are assuming that y is conjugate to x this means that we can conjugate x by some word and shorten it. However, this is equivalent to shortening the word by cyclically cancelling which we are assuming has already been done in the reduction process. Thus $n = r$. Notice in particular that the length of this reduced word depends only upon the conjugacy class of x. Also we can always make the first and last characters in x and y come from opposite groups since we are allowed to rewrite words cyclically. Now let j be the reduced word in G such that $x = jyj^{-1}$. If j has length greater than 1, then since the length of x is equal to the length of y and of course equal to the length of jyj^{-1} we must have that $l(jyj^{-1}) = l(y)$. This implies that there must be cancellation at one end of the word y. I claim that if we take the part which was cancelled and cyclically permute it to the other side, then we can shorten the length of j until $l(j) = 1$. To prove this assume that $l(j) > 1$ and then we have $j_1j_2...j_rh_1^*k_1^*...h_q^*k_q^*j_r^{-1}...j_1^{-1} = x$. Assume without loss of generality that $j_r \in H$. We can do this since if $j_r \notin H$ then $j_r \in K$ and we have cancellation at the other end in which case the proof is analogous. As we have lengthened the end of the word by r we must have shortened the beginning of the word by r and so the $j_1...j_r$ at the beginning must have completely cancelled. In particular this implies that $j_rh_1^* \in L^*$. Thus $j_{r-1}j_rh_1^* \in K$. Now consider the cyclically permuted word $y^* = k_1^*h_2^*...h_r^*k_r^*h_1^*$ and the $j^* = j_1j_2...j_{r-2}(j_{r-1}j_rh_1^*)$. It is easy to verify that $j^*y^*j^{*^{-1}} = x$. Also we have that since $j_{r-1}j_rh_1^* \in K$, $l(j^*) = l(j) - 1$. We can now repeat this process as long as $l(j) > 1$ (since we require j_{r-1} in the construction of j^*). Thus we have that some cyclic permutation of y, y^* satisfies $jy^*j^{-1} = x$ with $l(j) = 1$. Thus $j \in H$ or $j \in K$ and to complete the proof we need to show that if $l(x) = l(y) > 1$ then in fact $j \in H$ and $j \in K$ for this will imply that $j \in L^*$. To show this note that since x and y are cyclically reduced and of length greater than 1, we have that the first and last characters of x come from opposite groups and likewise for y. Now consider jyj^{-1}. This word must have the same length as y since it is equal to x. But that means that j and j^{-1} do not increase the length of the word and must thus reduce at both ends of y. Since the first and last characters of y come from opposite groups we must thus have that j is in both groups and hence $j \in L^*$ which completes the proof. □

The previous lemma allows us to solve the conjugacy in an amalgamated product over a central subgroup.

Theorem 3.4. *Let G be the amalgamated product of H and K over a subgroup of H and K isomorphic to L. Thus we have maps $\pi : L \to H$ and $\nu : L \to K$. Assume that H and K have solvable conjugacy problem. Also assume that the subgroup L^* of G isomorphic to L has solvable membership problem. Then if L^* is in the center of G, the conjugacy problem in G is solvable.*

Proof. Let x and y be cyclically reduced words in G. If either x or y have length 1, then the problem reduces to the conjugacy problem in one of the subgroups. So assume that x and y both have length greater than 1. By the previous theorem, they must have the same length if they are conjuagate and moreover some cyclic permutation of y must be a conjugate of x by and element in L^*. But since L^* is central, this means that some cyclic permutation of y must be equal to x. Thus the conjugacy problem in G has been reduced to the word problem in this case and is solvable by theorem 1.2. □

The very simple algorithm described in the preceding proof can be applied to B_3 since $< a, b|a^2 = b^3 >$ is the amalgamation of two infinite cyclic groups over the central subgroup $a^2 = b^3$. This theorem can also be used to solve the conjugacy problem in any group of the form $a, b|a^k = b^n >$ which can in a sense be seen as generalizations of B_3 since if $k = 2$ and n is odd this is equivalent to $< \sigma_1, \sigma_2|(\sigma_1\sigma_2)^{\frac{n-1}{2}}\sigma_1 = \sigma_2(\sigma_1\sigma_2)^{\frac{n-1}{2}} >$.
Using lemma 1.3 we can also show that the conjugacy problem is solvable in the amalgamated product of two free groups over a certain type of cyclic subgroup.

Definition 3.5. *A cyclic subgroup of a free group F is called maximal if it is not contained in any other cyclic subgroup of F.*

Notice that this is equivalent to the generator g not being a power of any other element in the group which means that all of the cyclic permutations of g are distinct.

Theorem 3.5. *Let H and K be two free groups on any number of generators. Let G be their amalgamated product over a maximal cyclic subgroup generated by g. Then the conjugacy problem is solvable in G.*

Proof. We will construct an algorithm which solves the conjugacy problem in G. Again consider two cyclically reduced word x and y. Let $x = h_1k_1h_2k_2...h_nk_n$ and $y = h_1^*k_1^*...h_n^*k_n^*$. By lemma 1.3 we know that the length of both reduced words must be the same if they are conjugate. Additionally we know that there exists a cyclic permutation of y which is conjugate to x by a element in $< g >$ say g^r. I claim that given any cyclic permutation of y there are only finitely many powers of g which could possibly conjugate x to the given cyclic permutation of y. To prove this let $y^* = h_1^*k_1^*...h_n^*k_n^*$ be a cyclic permuation of y. Notice that in order to be conjugate to x by a power of g we must have that the first and last characters in y^* come from opposite factor groups. This is true since conjugation by an element in $< g >$ doesn't change the first or last character of any reduced word. Now if x is conjugate to y^* by an element of $< g >$ then we have $h_1k_1...h_nk_ng^rk_n^{*^{-1}}...h_1^{*^{-1}}g^{-r} = 1$. In order for this to cancel $k_ng^rk_n^{*^{-1}} \in < g >$. In fact I claim that since $k_n \notin < g >$ and $k^{*^{-1}} \notin < g >$ since y^* and x are cyclically reduced we must have $k_ng^rk_n^{*^{-1}} = 1$. To prove this assume to the contrary that $k_ng^rk_n^{*^{-1}} = g^j$ with $j \neq 0$. Then we have $k_ng^{r_1}g^jg^{r_2}k_n^{*^{-1}} = g^j$ for some $r_1 + j + r_2 = r$ such that g^{r_1} is precisely cancelled by k_n and g^{r_2} is precisely cancelled by $k_n^{*^{-1}}$. This is always true since the only reductions we have in the free group are free reductions and it is not possible for $g^{'}g^kg^* = g^j$ where $g^{'}$ and g^* are pieces of the word g. The reason this last statement is true is because g is not equal to any of its cyclic permutations. However, this implies that k_n and $k_n^{*^{-1}}$ are in $< g >$ which we are assuming isn't the case. The reason why it is still possible for $k_ng^rk_n^{*^{-1}} = 1$ is that one of the occurences of the generator in g^r is partially cancelled by k_n and partially cancelled by $k^{*^{-1}}$. For example $g = cd$, $k_n = c^{-1}$, and $k_n^{*^{-1}} = d^{-1}$ would be such a situation. If any powers of g are left after cancellation this is not possible hence the claim.
Now define the length of an element x in the free group K denoted by $l_K(x)$ to be the number of generators in the reduced word defining x. Then since $k_ng^rk_n^{*^{-1}} = 1$ and all three of these factors are reduced words (this is true since x and y^* are reduced and a power of g is always reduced if g is written in reduced form which we can always assume to be true), and the only reductions possible in a free group are free cancellations we must have that $l_K(g^r) \leq l_K(k_n) + l_K(k_n^{*^{-1}})$. If not, then g^r could not be fully cancelled away. As we now have an upper bound on $l_K(g^r)$ and there are only finitely many elements in $< g >$ satisfying this upper bound we have completed the proof. The algorithm runs through all cyclic permutations of y and checks for each one whether any of the finitely many possible g^r conjuagates that specific permutation to x, thus reducing the conjugacy problem to the word problem in G, which is solvable by theorem 1.2. □

Now I will introduce another type of construction used in combinatorial group theory and see how the word and conjugacy problem behave under this construction, the HNN Extension.

Definition 3.6. *Given a group G and two subgroups of H and K of G which are isomorphic. The HNN Extension of G is then the group defined by adding an additional generator, t to G, such that $tHt^{-1} = K$. If G is given by $< G|R >$ and H and K are finitely generated, then the HNN Extension will be given by $< G, t|R, tx_it^{-1} = y_i >$ where x_i are the generators of H and y_i are the corresponding generators of K.*

The first result concerns the word problem in an HNN Extension.

Theorem 3.6. *Let G be a group in which the word problem is solvable. Then if H and K are isomorphic subgroups of G whose membership problem is solvable, then the word problem in the HNN Extension of G by H and K is solvable.*

Proof. We again construct a simple algorithm to solve the word problem. Call G_t the HNN Extension in question and let x be a word in G_t. Then if x contains no occurrences of t the problem is equivalent to the word problem in G. If x does contain occurrences of t write $x = g_1t^{i_1}g_2t^{i_2}...g_nt^{i_n}g_{n+1}$ where the $i_k = \pm 1$ and the g_k are possibly empty words in G. If $x = 1$ then it must be true that the number of t's and t^{-1}'s are the same since otherwise we couldn't cancel all of them. Find the first k such that i_k and i_{k+1} have opposite signs. Consider the segment $t^{i_k}g_{k+1}t^{i_{k+1}}$. If g_{k+1} is in H or K such that the previous segment reduces via the conjugation relation, reduce it to an element of the opposite subgroup. If not, x is not the identity. Repeat the preceding steps until all of the occurences of t have been removed if possible, if not, then x is not the identity. Once all occurrences of t have been removed the word problem has been reduced to that of the word problem in G. To complete the proof we need to show that it is not possible to add occurrences of t and then cancel differently to ultimately shorten the word. However this is clear since if we take $t^{i_{k-1}}g_kt^{i_k}$ and add a tt^{-1} to obtain $t^{i_{k-1}}g_{k_1}tt^{-1}g_{k_2}t^{i_k}$ then either only one of $t^{i_{k-1}}g_{k_1}t$ or $t^{-1}g_{k_2}t^{i_k}$ can be reduced (they can't both be reduced since then the original expression can also be reduced which we are assuming isn't the case) or neither can reduced. In either case the number of occurrences of t does not decrease. Hence if the algorithm described above cannot reduce x to the identity then x in fact cannot be reduced to the identity at all. □

The conjugacy problem in an HNN Extension is more difficult, however we have following, which theorem is very similar to the previous analog in the case of an amalgamated product. First I will introduce another definition.

Definition 3.7. *Let G be a group and let G_t be the HNN Extension of this group with respect to the subgroups H and K and the free variable t. Let x be a word in G_t and let $x = g_1t^{e_1}g_2t^{e_2}...g_nt^{e_n}$. Then define the t-structure of x to be the sequence $e_1, e_2, ..., e_n$.*

Theorem 3.7. *Let G be a group and let G_t be the HNN Extension of this group with respect to the subgroups H and K and the free variable t. Now let x and y be two cyclically reduced words in G_t. Here cyclically reduced means reduced via the process described in the previous proof with the additional ability to cyclically permute the word. Then if x and y contain occurences of t, there must be a cyclic permutation of y which is equal to x. In particular the t-structure of this cyclic permutation of y must be identical to t-structure of x.*

Proof. Let x and y be cyclically reduced words in G_t and assume that $x = g_1t^{e_1}...g_nt^{e_n}$ and $y = g_1^*t^{e_1^*}...g_k^*t^{e_k^*}$. We may assume that both words begin in an element of G and end in a power of t since we are allowed to cyclically reduce and permute. I claim that if x and y are conjugate then $n = k$. This is true since both words are fully cyclically reduced and hence neither of them can be shortened by cyclically permuting and calcelling. This implies in particular that neither of them can be shortened by conjugation with any element in the group. Let c be the word which conjugates y to x and consider $x = cyc^{-1} = c_1c_2c_3...c_jg_1^*t^{i_1}...g_k^*t^{e_k}c_j^{-1}...c_1^{-1}$ where $c_i = t^{q_i}$ or $c_i \in G$ and the c_i alternate through these possibilities. Then since x has the same length as y we must have cancellation at one of these two ends. Assume without loss of generality that we have cancellation

at the $c_{j-1}c_j g_1^* t^{i_1}$ section. Now we have two possibilities. Assume that $c_j = t^{q_j}$. Then the only possibility for cancellation is if $c_j g_1^* t^{i_1}$ reduces by the HNN Extension relations. Then we can move $g_1^* t^{i_1}$ to then end of y and replace c by $cg_1^* t^{i_1}$ which shortens c since $c_{j-1}c_j g_1^* t^{i_1}$ reduces to a single element as $c_{j-1} \in G$. Now if $c_j \in G$ then either $c_j g_1^* = 1$ in which case we can move g_1^* to the end of the word and replace c by cg_1^* which shortens c since $c_j g_1^* = 1$ or $c_{j-1}c_j g_1^* t^{i_1}$ reduces according to the HNN Extension relations and we can move $g_1^* t^{i_1}$ to the end of y and replace c by $cg_1^* t^{i_1}$ which shortens c because $c_{j-1}c_j g_1^* t^{i_1}$ reduces to a single element. The case where cancellation occurs at the other end of y is analogous. This shows that by cyclically permuting y we can always decrease the length of c until it has length one and in particular until it doesn't contain any t's. Then since the only reductions involving only one element in G are the calcellations $c_j g_1^* = 1$. But this implies that by moving g_1^* to the end of y we have a cyclic permutation of y that is equal to x. The statement about the t-structures follows from this equality. □

4. Linear time algorithms for B_3

A set of algorithms that solves both the word and conjugacy problem in B_3 in linear time.

Algorithm B: Input word w, return w in normal form.

(1) Replace a^{-1} with $a^{-2}a$, b^{-1} with $b^{-3}b^2$
(2) Initialize counter $k \leftarrow 0$, initialize a empty word l
(3) Read one character c, $l \leftarrow lc$.
   (a) If $l = l'a^2$ then $k \leftarrow k+1$, $l \leftarrow l'$
   (b) If $l = l'a^{-2}$ then $k \leftarrow k-1$, $l \leftarrow l'$
   (c) If $l = l'b^3$ then $k \leftarrow k+1$, $l \leftarrow l'$
   (d) If $l = l'b^{-3}$ then $k \leftarrow k-1$, $l \leftarrow l'$

   Repeat this step if there are more characters to read. If not, return $b^{3k}l$.

Example of Algorithm B: $ab^{-1}aa^{-1}ababba$. | shows where the reading pointer is. It is shown only when there is a event. At step 3, where the reading pointer is also indicate everything before it is in the list l.

$$\begin{aligned}
ab^{-1}aa^{-1}ababba &= a(b^{-1})|aa^{-1}ababba \\
&= ab^{-1}b^{-1}b^{-1}bb|aa^{-1}ababba \\
&= ab^{-1}b^{-1}b^{-1}bba(a^{-1})|ababba \\
&= ab^{-1}b^{-1}b^{-1}bbaa^{-1}a^{-1}a|ababba \\
&= a(b^{-1}b^{-1}b^{-1})|bbaa^{-1}a^{-1}aababba \\
&= b^{-3}\cdot a|bbaa^{-1}a^{-1}aababba \\
&= b^{-3}\cdot abba(a^{-1}a^{-1})|aababba \\
&= b^{-6}\cdot abba|aababba \\
&= b^{-6}\cdot abb(aa)|ababba \\
&= b^{-3}\cdot abb|ababba \\
&= b^{-3}\cdot abbababba
\end{aligned}$$

Algorithm C: Return normalized positive w in cyclic normal form.

(1) $c \leftarrow 0$
(2) If w is of the form $aw'a$, $bbw'b$ or $bw'bb$, then $w \leftarrow w'$, $c \leftarrow c+1$ and repeat this step. Else go to step (3).
(3) Return $b^{3c}w$.

Example of Algorithm C: *abbababba*

$$\begin{aligned} abbababba &= (a)bbabab(a) \\ &= b^3 \cdot bbabab \\ &= b^3 \cdot (bb)abab(b) \\ &= b^6 \cdot abab \end{aligned}$$

Algorithm A: Return true if $w \sim w'$, false otherwise.

(1) Replace $\sigma_1, \sigma_2, \sigma_1^{-1}, \sigma_2^{-1}$ with $b^{-1}a, ab^{-1}, a^{-1}b, ba^{-1}$.
(2) Use Algorithm B, put w and w' in normal form $b^{3n}x$ and $b^{3m}y$ respectively.
(3) Use Algorthm C, put x and y in cyclic normal form: $b^{3n'}x'$ and $b^{3m'}y'$ respectively.
(4) If $n + n' \neq m + m'$, then return false, else go to next step.
(5) If y' is a cyclic permutation of x', then return true, else return false.

If the input of Algorithm C is a word of length n, then step 2 in algorithm C runs at most $n/2$ times, each time it only need to check a constant set of characters. Algorithm C therefore runs in $O(n)$ time.

If the input of Algorithm B is a word of length n, step 1 runs at most n times, each time it increase the word length by at most 4. It will result a new word of length at most $4n$. Step 2 will run at most $4n$ times, each time it only do a constant amount of work. Therefore Algorithm B also runs in $O(n)$ time.

Algorithm B and C runs only once, each is a linear algorithm. The last step take can be done by checking if y' is a substring of $x'x'$. Substring searching algorithm is well known to be $O(n+m)$[cite KMP], where n and m are the length of two strings. Overall, Algorithm A takes $O(n)$ time.

5. The pure braid group

The pure braid group is a subgroup of the braid group: $P_n < B_n$. By definition any braid β defines a permutation $i \to \beta_i(1)$. This is a well-defined element of the symmetric group S_n. The kernel of the homomorphism $B_n \to S_n$ is P_n the pure braid group on n strands.

$$P_n \to B_n \to S_n$$

A pure braid is defined as a braid where the beginning points and ending points are not permuted. In other words, no matter how the strands are intertwined, each one ends up in the same position that it started in.

In P_n, if one forgets the n^{th} strand of an n braid, it results in an $n-1$ braid. The resulting map:

$$f : P_n \to P_{n-1}$$

is a well defined homomorphism for P_n. The kernel of f, denoted by $\ker f$, is the free group which has $n-1$ generators.

$$\ker f = f(\alpha_1, ..., \alpha_{n-1})$$

For example in P_3:

Because we also know that P_{n-1} is a subgroup of P_n it can be written as a semi-direct product

$$P_n = f(\alpha_1, ..., \alpha_{n-1}) \rtimes P_{n-1}$$

We started with this presentation for P_3:

$$P_3 =< \alpha_1, \alpha_2, g_1 | g_1^{-1}\alpha_1 g_1 = w_1,\ g_1^{-1}\alpha_2 g_1 = w_2 >,$$

where g_1 is a generator in P_2. We want to determine how P_{n-1} acts on the the free group f. Denote the generators P_{n-1} by $g_1, ..., g_k$. To determine the action of P_n on the free group we conjugate the elements of f with the generators in P_{n-1}.

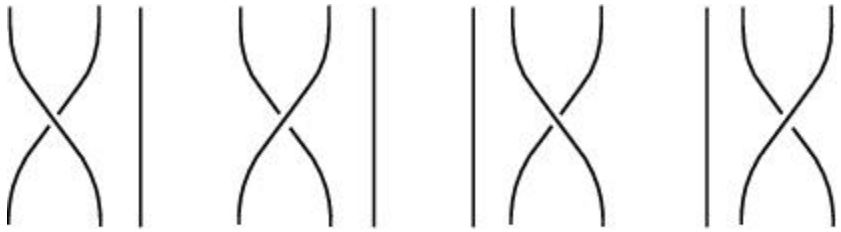
$$w = g_i \alpha_j g_i^{-1}$$

Because P_2 only has one generator we only need to consider two words to determine a presentation for P_3

$w_1 : \alpha_1^{-1}\alpha_2^{-1}\alpha_1\alpha_2\alpha_1 = g_1\alpha_1 g_1^{-1}$

$w_2 : \alpha_1^{-1}\alpha_2\alpha_1 = g_1\alpha_2 g_1^{-1}$, where g_1 is the generator of P_2

Where: $\alpha_1 = \sigma_2^2$, $\alpha_2 = \sigma_2\sigma_1^2\sigma_2^{-1}$, $g_1 = \sigma_1^2$, $\Delta = \sigma_2\sigma_1\sigma_2 = \sigma_1\sigma_2\sigma_1$, $\sigma_1\Delta = \Delta\sigma_2$, $\sigma_2\Delta = \Delta\sigma_1$



$$\begin{gathered}
w_1 : \alpha_1^{-1}\alpha_2^{-1}\alpha_1\alpha_2\alpha_1 = g_1\alpha_1 g_1^{-1} \\
\alpha_1\alpha_2\alpha_1 g_1 = \alpha_2\alpha_1 g_1\alpha_1 \\
\sigma_2\sigma_2\sigma_2\sigma_1\sigma_1\sigma_2^{-1}\sigma_2\sigma_2\sigma_1\sigma_1 = \sigma_2\sigma_1\sigma_1\sigma_2^{-1}\sigma_2\sigma_2\sigma_1\sigma_1\sigma_2\sigma_2 \\
\sigma_2\sigma_2\sigma_1(\sigma_1\sigma_2\sigma_1)\sigma_1 = \sigma_1(\sigma_1\sigma_2\sigma_1)\sigma_1\sigma_2\sigma_2 \\
\sigma_2\sigma_2\sigma_1\Delta\sigma_1 = \sigma_1\Delta\sigma_1\sigma_2\sigma_2 \\
\Delta\sigma_1(\sigma_1\sigma_2\sigma_1) = \Delta(\sigma_2\sigma_1\sigma_2)\sigma_2 \\
\Delta\sigma_1\Delta = \Delta\Delta\sigma_2 \\
\Delta^2\sigma_2 = \Delta^2\sigma_2
\end{gathered}$$

$$\begin{gathered}
w_2 : \alpha_1^{-1}\alpha_2\alpha_1 = g_1\alpha_2 g_1^{-1} \\
\alpha_2\alpha_1 g_1 = \alpha_1 g_1\alpha_2 \\
\sigma_2\sigma_1\sigma_1\sigma_2^{-1}\sigma_2\sigma_2\sigma_1\sigma_1 = \sigma_2\sigma_2\sigma_1\sigma_1\sigma_2\sigma_1\sigma_1\sigma_2^{-1} \\
\sigma_2\sigma_1(\sigma_1\sigma_2\sigma_1)\sigma_1 = \sigma_2\sigma_2\sigma_1(\sigma_1\sigma_2\sigma_1)\sigma_1\sigma_2^{-1} \\
\sigma_2\sigma_1\Delta\sigma_1 = \sigma_2\sigma_2\sigma_1\Delta\sigma_1\sigma_2^{-1} \\
\Delta(\sigma_1\sigma_2\sigma_1) = \Delta\sigma_1(\sigma_1\sigma_2\sigma_1)\sigma_2^{-1} \\
\Delta\Delta = \Delta\sigma_1\Delta\sigma_2^{-1} \\
\Delta^2 = \Delta\Delta\sigma_2\sigma_2^{-1} \\
\Delta^2 = \Delta^2
\end{gathered}$$

5.1. **Normal Form.** The normal form the word representing the pure braid group has any combination of reduced α_1's and α_2's first, and all the g_1's are at the end of the word.

Example 1 (Normal Form)**.** $\alpha_1\alpha_2^2\alpha_1^{-1}\alpha_2\alpha_1 g_1^3$

5.2. **Word Problem Algorithm.** Move all the g_1's to the right using the following relations:

$$g_1\alpha_1 = \alpha_1^{-1}\alpha_2^{-1}\alpha_1\alpha_2\alpha_1(g_1)$$
$$g_1 alpha_1^{-1} = alpha_1^{-1} alpha_2^{-1}\alpha_1^{-1}\alpha_2\alpha_1(g_1)$$
$$g_1\alpha_2 = alpha_1^{-1}\alpha_2\alpha_1(g_1)$$
$$g_1\alpha_2^{-1} = \alpha_1^{-1}\alpha_2^{-1}\alpha_1(g_1)$$
$$g_1^{-1}\alpha_1 = \alpha_2\alpha_1\alpha_2^{-1}(g_1^{-1})$$
$$g_1^{-1}\alpha_1^{-1} = \alpha_2\alpha_1^{-1}\alpha_2^{-1}(g_1^{-1})$$
$$g_1^{-1}\alpha_2 = \alpha_2\alpha_1\alpha_2\alpha_1^{-1}\alpha_2^{-1}(g_1^{-1})$$
$$g_1^{-1}\alpha_2^{-1} = \alpha_2\alpha_1\alpha_2^{-1}\alpha_1^{-1}\alpha_2^{-1}(g_1^{-1})$$

If the result is the empty word, then the braid is trivial.

5.3. **New Presentation.** For ease of notation $\alpha_1 = x$, $\alpha_2 = y$, *and* $g_1 = z$

$$< x, y, c | xc = cx, yc = cy >$$

where $c = z^{-1}x^{-1}y^{-}1$. We can write this as an amalgamated product of two groups with c=d at the center.

$$< x, c | xc = cx > *_{c=d} < y, d | yd = dy >$$

5.4. **Word Problem.** This gives a new solution to the word problem and a solution to the conjugacy problem in P_3. The normal form can be written by simply moving all the c's to the right using the above relations. After putting the word in normal form and reducing using free reductions ($xx^{-1} = 1$, $yy^{-1} = 1$, $cc^{-1} = 1$), if the result is the empty word then the braid is trivial.

5.5. **Conjugacy Problem.** Because we can write the presentation of P_3 we can say that there exists a word $g = a_1b_1a_2b_2a_3b_3...$ and another word $g' = a_{1'}b_{1'}a_{2'}b_{2'}a_{3'}b_{3'}$, where $g, g' \in G, a \in A, and\, b \in B$
note: Both g and g' are cyclically reduced.

$$g \text{ is conjugate to } g' \Leftrightarrow g' = c\bar{g}c^{-1},$$
where $\bar{g}$ is a cyclic permutation of $g \in G$ and $c \in C$

6. Basse-Serre Theory

Basse-Serre Theory is the theory of group actions on trees. We have been able to use this theory to derive the same presentation for P_3 that was indicated in the previous section. The first part of this section will be concerned with introducing group actions on trees. To this end, let us introduce some definitions.

Definition 6.1. *A graph G is a pair of sets $V(G)$ and $E(G)$ called verticies and edges and two functions $i : E(G) \to V(G)$ and $j : E(G) \to V(G)$. This allows us to view an edge $e \in E(G)$ as an oriented arrow pointing from $i(e)$ to $j(e)$.*

The graphs which we will be interested in are unoriented graphs which have the property that for every edge $e \in E(G)$ there exists an edge $e^* \in E(G)$ such that $i(e) = j(e^*)$ and $j(e) = i(e^*)$. Thus given any edge there exists another edge pointing in the opposite direction.

Definition 6.2. *A path in a graph is a sequence of verticies $v_1, ..., v_n$ such for each $i \in [1, , , n-1]$ there exists an edge $e \in E(G)$ such that $i(e) = v_i$ and $j(e) = v_{i+1}$. Alternatively a path can be identified by the edges it traverses.*

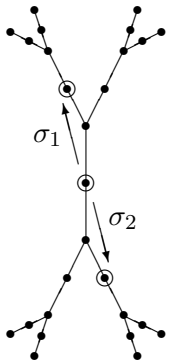
A path such that $v_1 = v_n$ is called a loop.

Definition 6.3. *A tree is a graph such that every loop contains a pair of adjacent edges e and e^* such that $i(e) = j(e^*)$ and $j(e) = i(e^*)$.*

Definition 6.4. *An automorphism of a tree is a pair of bijections $\phi : V(G) \to V(G)$ and $\nu : E(G) \to E(G)$ which preserve adjacency. That means that $\phi(j(e)) = j(\nu(e))$ and $\phi(i(e)) = i(\nu(e))$ for all edges $e \in E(G)$.*

Definition 6.5. *A group action on a tree is a homomorphism from the groups into the group of automorphisms of the tree. This action is said to be without inversion if it never maps and edge onto an edge which points in the opposite direction.*

As an example consider the action of B_3 on the following tree.



Notice that we only need to specify the actions of each of the generators in order to specify the group action. I leave it as an excercise to the reader that both of the depicted automorphisms satisfy the same relations as the generators of B_3 and thus that this is actually a homomorphism. Now we will need some simple lemma's concerning automorphisms of trees.

Lemma 6.1. *Let T be a tree and let ϕ be an automorphism without inversion on T. Then if ϕ fixes no vertex, T must be infinite and there must be an infinite path through T such that ϕ maps that path to itself and preserves the order of the verticies along that path. In a sense, the entire tree is moved along that infinite path.*

Proof. Pick a vertex v such that v and $\phi(v)$ have the shortest possible path to each other. This is always possible since given any two verticies in the tree there is a finite path connecting them. Since ϕ has no fixed verticies, the path between v and $\phi(v)$ has length greater than 0. Also, since T is a tree, this path is unique. Let $p_k(v)$ be the path from $\phi^{k-1}(v)$ to $\phi^k(v)$. Let p be the infinite path $p = \cup_{k\in\mathbb{Z}} p_k(v)$. I claim that this infinite path is mapped onto itself by ϕ. To prove this pick any vertex v^* on p and assume that $phi(v^*) \notin p$. Now notice that v^* must lie between $\phi^{k-1}(v)$ and $\phi^k(v)$ for some integer k and that each of these two points gets mapped back onto p but the definition of p. Now there is a path from $\phi^{k-1}(v)$ to $\phi^k(v)$ passing through v^* since v^* lies on p. This is also by definition of p the shortest path from $\phi^{k-1}(v)$ to $\phi^k(v)$, has the same length as the path from $\phi^k(v)$ to $\phi^{k+1}(v)$, since ϕ is an automorphism. Now consider the image of this path. Since ϕ is an automorphism the length of the image of this path is the same as the length of the original path. Also the assumption that $\phi(v^*) \notin p$ implies that this image does not lie completely on p. However, this means we have two distinct shortest possible paths between $\phi^k(v)$ and $\phi^{k+1}(v)$, namely the image of the path from $\phi^{k-1}(v)$ to $\phi^k(v)$ and the path from $\phi^k(v)$ to $\phi^{k+1}(v)$ defined by p. These both have the shortest possible length by the definition of p and the fact that ϕ is an automorphism and must be distinct since one lies completely on p and the other doesn't. However, this contradicts the asumption that T is a tree. Now I will prove that p doesn't backtrack. Namely that p does not contain consecutive edges which are reverses of each other. Again assume to the contrary that p does contain such a pair of edges. Then we can remove all such edges to obtain another path which is "shorter" and which is also mapped onto itself by ϕ. However, let v^* be a vertex on this new path. Notice that v^* now defines this new path. Also since the new path is a

subpath of the old path we have that the length of the path from v^* to $\phi(v^*)$ is strictly less than the length of the path from v to $\phi(v)$ which contradicts the minimality of this length. □

We have now shown that an automorphism which which fixes no verticies must fix an infinite path through the tree. If the automorphism acts without inversion then this tree must be in a sense translated some distance along this path. The action of the generators of B_3 on the tree that was previously presented have precisely this property. This also shows that the only trees which have automorphisms that don't fix any vertex are infinite trees.

Lemma 6.2. *Let α and β be two automorphisms without inversion of a tree T which both have fixed verticies but don't have any fixed verticies in common. Then the composition $\nu = \alpha\beta$ doesn't fix any verticies and by the previous lemma moves the entire tree along some infinite line.*

Proof. Let $x \in fix(\alpha)$ and $y \in fix(\beta)$ be two elements fixed by α and β respectively such that their path distance is minimal. Consider the path of minimal length from x to y. Call this path p. Notice that $\alpha(p)$ and p intersect only in x and $\beta(p)$ and p intersect only in y. This is true since more points of intersection would contradict the minimality of the distance between x and y as one of the two maps would fix points between x and y. Consider the path $P = p \cup \beta^{-1}(p)$. Now P is a path from $\nu^{-1}(x)$ to x with no backtracking. Now I claim that $\nu(P) \cap P = x$. This is true since $\nu(\beta^{-1}(p)) = \alpha(p)$ and $\alpha(p) \cap p = x$ and this implies since T is a tree and thus there can be no loops that $\alpha(p) \cap P = x$. Also since $\beta(p) \cap p = y$ we have $\nu(p) \cap \alpha(p) = \alpha(y)$. Thus it follows that $P \cup \nu(p)$ forms a connected path. Since we have a tree and there are no loops it follows that $(\nu(P) = \nu(p) \cup \alpha(p)) \cap (p \cup \beta^{-1}(p)) = x$ since $x = p \cap \alpha(p)$ and any other points of intersection would create a loop. Now consider ther infinite path $\cup_{k\in\mathbb{Z}}\nu^k(P)$. This path is mapped onto itself by ν and ν fixes no points on the path. Hence ν must shift the tree along this infinite path and thus has no fixed points as desired. □

Now I will introduce the concept of a graph of groups.

Definition 6.6. *A graph of groups is a graph where the elements of the vertex and edge sets are groups and a set of homomorphisms $\phi_e : e \to i(e)$ and $\nu_e : e \to j(e)$ for all $e \in E(G)$.*

Essentially a graph of groups is just a graph where we have a group associated to every vertex and every edge. We also have maps from the group associated to each edge into the groups associated to the adjacent verticies. Now we explain how to construct what is called the fundamental group of a connected graph of groups. To construct the fundamental group $\pi_1(Gr)$ of a connected graph of groups we must:
1. Find a spanning tree. The fact that our graph of groups is connected means such a spanning tree is guaranteed to exist.
2. Form the amalgamated product of all of the vertex groups in the spanning tree where the amalgamation is over the edge groups between corresponding vertex groups and the amalgamation mapping is the one specified in the definition of the graph.
3. For each edge not in the spanning tree form an HNN Extension which conjugates the image of the edge group in it's $i(e)$ vertex the the image of the edge group in it's $j(e)$ vertex.
For example the fundamental groups of a loop at a single vertex is just an HNN Extension and the fundamental group of two verticies connected by a single edge is an amalgamated product. Now we can state the theorem of Bass-Serre.

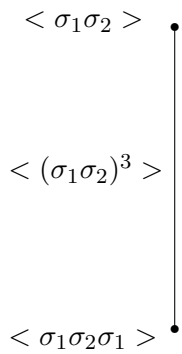
Theorem 6.3. *Let G be a group acting without inversion on a tree. Define the fundamental domain of the action to be the graph of groups whose underlying graph structure are equivalence classes of verticies and edges under the action and whose associated groups are the stabilizers of an embedding of this quotient graph in the original tree. Then G is isomorphic to the fundamental group or this graph or groups.*

The first step in proving this theorem is to prove that the fundamental domain is connected. This follows since we can embed the fundamental domain in the original graph by associating to each equivalence class a representative in the original tree. Given any two equivalence classes of verticies there exists a path in the original tree connecting two of their representatives. If these two representatives are chosen as close together as possible then each of the edges and verticies on the shortest path between them must be in its own equivalence class. This is true since every group element corresponds to an automorphism which can not collapse the path between these two verticies and thus can not map any of the edges or verticies in this path onto other edges or verticies in the path. This shows that we can always embed the fundamental domain in the original tree such that it is connected. Hence the fundamental domain is connected. Now we can proceed to prove the general theorem.

Proof. The idea behind this result is that the group obtained as an amalgamated product of the vertex groups of the spanning tree is generated by all of the automorphisms which have fixed points and the HNN Extensions added afterward represent the elements without fixed points which are not the product of elements which have fixed points. The spanning tree of the fundamental domain contains the stabilizers of some representative of every equivalence class of verticies. Thus the normal closure of the vertex groups in the fundamental domain contains all elements in the group which have fixed verticies. This is clear since we can conjugate any stabilizer of one representative with the element taking that representative to any other vertex in the equivalence class to obtain a corresponding stabilizer of that other vertex. Now consider the stabilizers of the edges. Since the group acts without inversion, any stabilizer of the edges must stabilize both verticies. Thus the two subgroups of the stabilizers of the verticies which fix the edge between them fix both verticies and thus these subgroups actually correspond to the same elements in the original group. Thus we must amalgamate over them. This gives us all of the relations which hold in the subgroup generated by elements which fix a vertex. Now consider the edges which are not a part of the spanning tree. These occur whenever we look at an embedding of the fundamental domain and there is an edge which has one end not pointing to a vertex in the embedding. This implies that there exists an automorphism which takes some other point in the embedding and maps it onto the free end of that edge but which maps no other edge in the embedding onto that edge. This automorphism can have no fixed verticies and thus it has infinite order. To see why it has no fixed verticies assume it did. Let v be one of its fixed verticies and x be the vertex that it moves to the free end of the edge e in question. Let p be the path from v to x. If p contains the edge e then in contains the image of x which is a contradiction since v is fixed and so the path p whould be mapped to a subpath of itself. Now assume that p does not contain e. Then if the final edge of p containing x is not in the embedding of the fundamental domain, then since the embedding of fundamental domain is connected and the image of x is not in the fundamental domain, the image of the final edge is not in the fundamental domain and thus can't be e contrary to assumption. Likewise, if the final edge is in the embedding of the fundamental domain then an edge in this embedding gets mapped to e which is again assumed not to be true. Hence these automorphisms have no fixed veritices. Also, they cannot be written as a product of automorphisms with fixed points in the fundamental domain. This is true since the embedding of the fundamental domain is connected and so if the path from one of its verticies to a vertex outside of the domain contains a final edge e, then so does the path from any vertex in the embedding to the outside vertex. This implies along with the previous argument that the edge connecting x to the domain would have to be mapped to the edge e as it is inside of the domain and since being the final edge of a path to a vertex outside of the fundamental domain doesn't depend on the initial vertex in the domain and so the image of the edge connecting x to the domain would always be the edge connecting x to the domain and hence would finally have to be mapped to e. Since no edge in the fundamental domain is getting mapped to e under this mapping this proves that we have a completely new generator of infinite order. This generator also conjugates the image of the stabilizer of the edge in each of its vertex groups

onto each other since it moves one vertex in the embedding to its other possible position where it shares an edge with the other adjacent vertex and so they share the stabilizer of the original edge which is now between them in the embedding. Thus we have an HNN Extension as described by the fundamental group of the fundamental domain. □

The preceding proof is not rigorous and should only give an idea as to why this theorem is true. For a more rigorous treatment consult the literature.
These techniques have allowed us the derive the presentation $B_3 =< a, b|a^2 = b^3 >$ and the presentation $P_3 =< a, b, c|ac = ca, bc = cb >$. Also using the mapping $\phi : B_4 \to B_3$ defined by $\phi(\sigma_1) = \sigma_1$, $\phi(\sigma_2) = \sigma_2$, and $\phi(\sigma_3) = \sigma_1$, we can have B_4 act on a tree. However, the resulting decomposition as an amalgamated product involves factors which are not finitely generated. This is a result of the fact that the kernel of the mapping from B_4 into B_3 is not finitely generated. For example using the action of B_3 on the tree previously described we obtain the following fundamental domain.



This fundamental domain leads to the presentation $< a, b|a^2 = b^3 >$. Additionally when restiction the aforementioned action to the subgroup P_3 we obtain the presentation $< a, b, c|ac = ca, bc = cb >$.

7. Garside's work on braid groups

Garside was the first person to solve the word and conjugacy problem in B_n. The solution was developed on a normal form of the braids.

7.1. **Garside Normal Form.**

Definition 7.1 (The Positive Braid Monoid)**.** *The positive braid monoid of n strands is denoted as B_n^+. Such that $B_n^+ \subseteq B_n$ and B_n^+ contains only word of the form $\sigma_{a_1} \dots \sigma_{a_l}$.*

Definition 7.2.
(1) *$|w|$ is the length, or the number of generators required to write the word w, where $w \in B_n^+$. It is well-defined because the relations in B_n^+ doesn't change $|w|$.*
(2) *$abc = d$, where $a, b, c, d \in B_n^+$. We say a, b, c are factors of d. a is a left divisor, or a prefix. c is a right divisor, or a suffix.*
(3) *$a \wedge b = c$, if $cw = a$ and $cv = b$, such that $|c|$ is maximized, where $a, b, c, w, v \in B_n^+$. c is called the greatest common divisor of a and b.*

Definition 7.3 (The Fundamental Braid)**.** *The fundamental braid, or half-twist, on n strands*

$$\Delta_n = (\sigma_1 \dots \sigma_{n-1})(\sigma_1 \dots \sigma_{n-2}) \dots (\sigma_1\sigma_2)\sigma_1$$

.

One uses $\Delta = \Delta_n$ if n is obvious from the context.

Two half-twists make a full-twist. After a full-twist, every strand is at it's original position. It explains the relation: $\sigma_i\Delta = \Delta\sigma_{n-i}$.

The relation can be extend to an automorphism. Define automorphism R, such that $R(\sigma_i) = \sigma_{n-i}$. R's action on a word has a simple characterization.

$$R(\sigma_{a_1}^{b_1} \dots \sigma_{a_k}^{b_k}) = \sigma_{n-a_1}^{b_1} \dots \sigma_{n-a_k}^{b_k}$$

$w\Delta = \Delta R(w)$ is true for all $w \in B_n^+$.

Theorem 7.1. *There exist a $x_i, y_i \in B_n^+$, such that $\sigma_i^{-1} = x_i\Delta^{-1} = \Delta^{-1}y_i$ for all i.*

Proof.

$$\begin{aligned}\Delta^{-1} &= \sigma_1^{-1}(\sigma_2^{-1}\sigma_1^{-1}) \dots (\sigma_{n-1}^{-1} \dots \sigma_1^{-1}) \\ &= \sigma_1^{-1}(\sigma_2^{-1}\sigma_1^{-1}) \dots (\sigma_i^{-1} \dots \sigma_1^{-1}) \dots (\sigma_{n-1}^{-1} \dots \sigma_1^{-1}) \\ &= \sigma_1^{-1}(\sigma_2^{-1}\sigma_1^{-1}) \dots (\sigma_i^{-1})(\sigma_{i-1}^{-1} \dots \sigma_1^{-1}) \dots (\sigma_{n-1}^{-1} \dots \sigma_1^{-1})\end{aligned}$$

$$\begin{aligned}(\sigma_1 \dots \sigma_{i-1}) \dots (\sigma_1\sigma_2)\sigma_1\Delta^{-1}(\sigma_1 \dots \sigma_{n-1}) \dots (\sigma_1 \dots \sigma_{i+1})(\sigma_1 \dots \sigma_{i-1}) &= \sigma_i^{-1} \\ \Delta^{-1}R[(\sigma_1 \dots \sigma_{i-1}) \dots (\sigma_1\sigma_2)\sigma_1](\sigma_1 \dots \sigma_{n-1}) \dots (\sigma_1 \dots \sigma_{i+1})(\sigma_1 \dots \sigma_{i-1}) &= \sigma_i^{-1} \\ (\sigma_1 \dots \sigma_{i-1}) \dots (\sigma_1\sigma_2)\sigma_1 R[(\sigma_1 \dots \sigma_{n-1}) \dots (\sigma_1 \dots \sigma_{i+1})(\sigma_1 \dots \sigma_{i-1})]\Delta^{-1} &= \sigma_i^{-1}\end{aligned}$$

□

Every negative generator can be replaced with a Δ^{-1} and some positive generators. Potentially increase the positive part's length by $|\Delta| = n!$ for each replacement of the negative generator.

Definition 7.4 (The Garside Normal Form)**.** *The Garside normal form of a word $w \in B_n$ is $w = \Delta^m p$. $p \in B_n^+$, and p contains no factors of Δ.*

Definition 7.5. *w has the Garside normal form $\Delta^m p$, then we define the following functions.*

1. *The infimum, or the delta exponent, of w,* $\inf(w) = m$
2. *The index of w, $\lambda(w) = |p| + m|\Delta|$.*

The index of w is also $h - g$, where h, g are the number of positive and negative generators respectively. No matter how w is presented, the index is a constant. This is obvious because all braid relations doesn't change the index. λ can be interpreted as a homomorphism, such that $\lambda(\sigma_i^{-1}) = -1$,$\lambda(\sigma_i) = 1$ $\lambda(ab) = \lambda(a) + \lambda(b)$.

Theorem 7.2 (Uniqueness of Garside Normal Form)**.** *The Garside normal form $w = \Delta^m p$ is unique up to the element $p \in B_n^+$.*

Proof. If $w \in B_n^+$, find the largest m, such that $\Delta^m \leq w$, then $w = \Delta^m p$, both m and p are unique.

If $w \in B_n$, first rewrite every σ_i^{-1} in w by $x_i\Delta^{-1}$, move all the Δ to the left, and we will have $w = \Delta^{-j}p'$, where p' is a positive word. Then p' must have a normal form, let it be $\Delta^k p$. The Garside normal form of $w = \Delta^{k-j}p$, It's unique since k, j and p are unique. □

The proof give us an algorithm to compute the Garside normal form.

7.1.1. *Solution to the word problem.* To solve the word problem with Garside normal form, follow the step in the proof. Let $l = |w|$. Rewrite negative generators cost $O(ln!)$ time. Factor out Δ divisors in p can take exponential time: use the relations to generate every equivalent presentation of p, and check if it contain Δ as a subword. If so, move Δ to the left and repeate the process until there is no more Δ.

The word is the identity if and only if the Garside normal form is Δ^0.

7.1.2. *Linear time solution to the word problem in B_3.* In B_3, the only relation is $\sigma_1\sigma_2\sigma_1 = \sigma_2\sigma_1\sigma_2$. This is special enough such that there exist a linear time algorithm.

(1) Rewrite all generator with negative exponents into ones with positive exponents and a Δ^{-1}.
(2) Move every Δ^{-1} to the beginning of the word: First locate the last Δ^{-1}, and move it forward using the relations $\Delta^{2n+1}\sigma_i = \sigma_{3-i}\Delta^{2n+1}$ and $\Delta^{2n}\sigma_i = \sigma_i\Delta^{2n}$. Pick up other Δ^{-1}'s in between.
(3) Move every Δ factor in the word to the beginning of the word: Read from the end of the word, move it forward using Δ relations, pick up other Δ on the way. Check if moving Δ generates another Δ. B_3 don't have $\sigma_i\sigma_j = \sigma_j\sigma_i$ relation, therefore if a new Δ is created from moving Δ's around, it is a local event. To be precise, during this process, $x\Delta^m y = x\Delta^{m+1}y'$ if and only if the first 3 generator of y is Δ.

The algorithm walks though the word three times. Each time the algorithm reads a character, it does a constant number of operations. The amount of time is bounded by the maximum length reached by the word. It is at most $3n$. The algorithm runs in linear time.

A demonstration on a positive word.

$$\begin{aligned}
\sigma_2\sigma_1\sigma_1\sigma_2\sigma_1\sigma_1\sigma_1\sigma_1 &= \sigma_2\sigma_1\sigma_1\sigma_2\sigma_1(\sigma_1\sigma_1\sigma_1)\\
&= \sigma_2\sigma_1\sigma_1\sigma_2(\sigma_1\sigma_1\sigma_1)\sigma_1\\
&= \sigma_2\sigma_1\sigma_1(\sigma_2\sigma_1\sigma_1)\sigma_1\sigma_1\\
&= \sigma_2\sigma_1(\sigma_1\sigma_2\sigma_1)\sigma_1\sigma_1\sigma_1\\
&= \sigma_2\sigma_1\Delta\sigma_1\sigma_1\sigma_1\\
&= \sigma_2\sigma_1\Delta(\sigma_1\sigma_1\sigma_1)\\
&= \sigma_2\Delta(\sigma_2\sigma_1\sigma_1)\sigma_1\\
&= \Delta(\sigma_1\sigma_2\sigma_1)\sigma_1\sigma_1\\
&= \Delta^2\sigma_1\sigma_1\\
&= \Delta^2(\sigma_1\sigma_1\cdot 1)\\
&= \Delta^2\sigma_1^2
\end{aligned}$$

7.1.3. *Thurston left greedy form.* There are many ways to express the positive braid p in the Garside normal form. There is no simple way to check if a word contains Δ. In the linear algorithm that put B_3 in Garside normal form, we used a special property that is only true in B_3: If Δ is a factor of w, then Δ is a subword of w.

In B_4, this breaks down. There exist counterexample with length 7. There is no word that equals Δ that contains the subword $\sigma_3\sigma_3$ in B_4, but $\sigma_3\sigma_2\sigma_3\sigma_3\sigma_1\sigma_2\sigma_3 = \Delta\sigma_2$.

In other words, the existance of Δ in a word is not a local property, one need to generate a few equivalent words. Depend on the braid's length, there are exponential many ways to express a positive braid. Is it possible to check only a polynomial number of words?

Thurston[make a reference] in [year] refined the Garside normal form with simple elements and showed only a quadratic number of words need to be checked. Garside already used simple elements in his paper to solve the conjugy problem, but didn't apply it to the word problem.

Definition 7.6 (Simple elements)**.** *A positive braid p is simple if $p \in \mathbb{D} = \{x | x \leq \Delta\}$.*

Lemma 7.1. *If $cd = \Delta$ and c is simple, then d is simple.*

Proof. $dR(c) = c^{-1}cdR(c) = c^{-1}\Delta R(c) = \Delta R(c^{-1})R(c) = \Delta$ □

Note $\mathbb{D}$ generates the braid group, because the lemma shows $\sigma_i \leq \Delta$ for all i. The only simple element that contains all σ_i as a prefix is Δ.

Definition 7.7. *If $w = \Delta^m p$ is in Garside normal form, then the Thurston left greedy form is*

$$w = \Delta^m a_1 \dots a_k$$

where $a_1 = p \wedge \Delta$, $a_i = (a_{i-1}^{-1} \dots a_1^{-1})p \wedge \Delta$.

The new form can be found in quadratic time because there exist an automatic structure using Thurston left greedy form. Automatic structure will be addressed in the next section.

Definition 7.8. *The starting set and finishing set of $w \in B_n^+$, denoted as $S(w)$ and $F(w)$ respectively.*

(1) *$\sigma_i \in S(w)$ if and only if $\sigma_i w' = w$, where $w' \in B_n^+$.*
(2) *$\sigma_i \in F(w)$ if and only if $w'\sigma_i = w$, where $w' \in B_n^+$.*

The following theorems we will state but does not provide a proof:

Theorem 7.3. *$w = \Delta^m a_1 \dots a_k$ is in left greedy form if and only if $S(a_{i+1}) \subseteq F(a_i)$.*

Theorem 7.4. *$F(w) = S(w) = \{\sigma_1, \dots, \sigma_{n-1}\}$ if and only if $w = \Delta p$ for some positive word p.*

7.2. **Summit sets and Garside's solution of the conjugacy problem.** Let $[w]$ be the conjugacy class of w. in other words, $w' \in [w]$ if and only if $w = c^{-1}w'c$ for some $c \in B_n$. One write $w \sim w'$ if $w' \in [w]$

$[w]$ is not finite in general. Search through the entire set is impossible. However, the solution is similar to the word problem, it's also about finding a "normal form" for $[w]$. There are special subsets of $[w]$, such that there exist an algorithm with input w', where the algorithm output the subset if and only if $w' \in [w]$.

Garside introduced a set with such property — the summit set.

Let $\inf[w] = \sup\{\inf(x) | x \in [w]\}$, the maximal delta exponent in the conjugacy class.

Definition 7.9 (The Summit Set)**.**

$$SS(w) = \{x | \inf(x) = \inf[w], x \in [w]\}$$

$SS(w)$ is called the summit set of w.

Theorem 7.5. *For $w, w' \in B_n$, $w \sim w'$ if and only if $SS(w) = SS(w')$*

Proof. $SS(w) = SS(w') \implies w \sim w'$

$[w]$ and $[w']$ are equivalence classes, if there exist one element in common, it shows $[w] = [w']$. Given that $SS(w)$ can't be empty, $[w] = [w']$ and $w \sim w'$.

$w \sim w' \implies SS(w) = SS(w')$

$w \sim w'$ shows $[w] = [w']$, by the definition of summit set. $SS(w) = \{x | \inf x = \inf[w], x \in [w]\} = \{x | \inf x = \inf[w'], x \in [w']\} = SS(w')$. □

Theorem 7.6. *The summit set exists and is finite.*

Proof. The index of any element $w' \in [w]$ is the same, because

$$\begin{aligned} \lambda(w) &= \lambda(c^{-1}wc) \\ &= -\lambda(c) + \lambda(w') + \lambda(c) \\ &= \lambda(w') \end{aligned}$$

The index formula restricts $\inf[w]$.

$$\inf[w] = \frac{\lambda(w) - |p|}{|\Delta|}$$

$$\inf[w] \leq \frac{\lambda(w)}{|\Delta|}$$

Since $\inf[w]$ is bounded above, the summit set exists.

$|p| = \lambda(w) - |\Delta| \inf[w]$. The size is bounded by the amount of words with such length. This shows $|SS(w)| \leq n^{\lambda(w)-|\Delta|\inf[w]}$. The summit set is finite. □

Theorem 7.7. *$w \sim w'$, $w, w' \in B_n$, then there exist a $c \in B_n^+$, such that $w = c^{-1}w'c$.*

Proof. $w = (\Delta^m p)^{-1} w' (\Delta^m p)$.

(1) If m is even: $w = p^{-1}w'p$ by cancel the Δ's.
(2) If $m > 0$ and is odd: $w = (\Delta p)^{-1} w' (\Delta p)$ by cancel $m-1$ Δ's.
(3) If $m < 0$ and is odd:

$$\begin{aligned} w &= (\Delta^{-1}p)^{-1} w' (\Delta^{-1} p) \\ w &= p^{-1}\Delta w' (\Delta^{-1} p) \\ \Delta^{-2} w \Delta^2 &= \Delta^{-2} p^{-1} \Delta w' (\Delta^{-1} p) \Delta^2 \\ w &= p^{-1}\Delta^{-1} w' (\Delta p) \\ w &= (\Delta p)^{-1} w' (\Delta p) \end{aligned}$$

c in the theorem is either Δp or p, both are positive words. □

The theorem allows one to ignore B_n and only search inside B_n^+. Note the positive words are generated by the set of simple elements. The summit set can be found though conjugation of simple elements.

Theorem 7.8. *$w \in B_n$ and $x \in B_n^+$. For $a \in SS(w)$, if $x^{-1}ax \in SS(w)$, then $c^{-1}ac \in SS(w)$, where $c = x \wedge \Delta$.*

The proof is difficult. First we have to show two lemmas.

Lemma 7.2. *If x, p, q is positive, and $x^{-1}\Delta^m p x = \Delta^{m+r} q$, then px contains Δ^r*

Proof.

$$\begin{aligned} x^{-1}\Delta^m p x &= \Delta^{m+r} q \\ p x &= \Delta^{-m}\Delta^{m+r} q \\ p x &= \Delta^r R^{m+r}(x) q \end{aligned}$$

□

Lemma 7.3. *If $w = \Delta p$ for $p \in B_n^+$, then for any $xy = w$, for every σ_i, either $x = a\sigma_i$ or $y = \sigma_i b$, where $a, b \in B_n^+$*

Proof. Geometrically, the condition charactrize the set of paths S from 1 to w on the Cayley graph. At any vertex in S, the in edge and out edge contains every σ_i.

Proof by induction on $|x|$. It's true when $|x| = 1$. Assume it is true for all x where $|x| = k$. Fix a x, let $|x'| = k+1$, $w = x'y'$ and $x' = x\sigma_i$ for some i.

For every σ_j, it is either a out edge of x or a in edge of x.

(1) If it is an in edge, and $|j - i| \geq 2$. $x = a\sigma_j$ for some positive word a. $x' = a\sigma_j\sigma_i = a\sigma_i\sigma_j = x\sigma_j$.
(2) If it is an in edge, and $|j - i| = 1$, $x = a\sigma_j$, then $a\sigma_j\sigma_i\sigma_j = a\sigma_i\sigma_j\sigma_i = x'\sigma_i$. $x' = x\sigma_j$.
(3) If it is an out edge, and $|j - i| \geq 2$, we express $w = x\sigma_j\sigma_i b = x\sigma_i\sigma_j b$, then this shows $y' = \sigma_j b$.
(4) If it is an out edge, and $|j - i| = 1$, $w = x\sigma_j\sigma_i\sigma_j b = x\sigma_i\sigma_j\sigma_i b$. $y' = \sigma_j\sigma_i b$.

□

We have the tools to finally prove theorem 7.8.

Proof. Let $x = cy$ and $x^{-1}ax = \Delta^m p$ where $y \in B_n^+$. There exist simple element d such that $cd = \Delta$ Assume x doesn't contain a Δ factor. (If there is, it will be cancelled in the conjugation).

$$\begin{aligned} x^{-1}ax &= \Delta^m q \\ y^{-1}c^{-1}acy &= \Delta^m q \\ y^{-1}c^{-1}cd\Delta^{-1}acy &= \Delta^m q \\ y^{-1}d\Delta^{-1}\Delta^m pcy &= \Delta^m q \\ y^{-1}d\Delta^{m-1}pcy &= \Delta^m q \\ y^{-1}\Delta^{m-1}R^{m-1}(d)pcy &= \Delta^m q \end{aligned}$$

Let $c' = R^{m-1}(d)$. Lemma 7.2 shows $\inf c'pcy = 1$. Lemma 7.3 shows the set $F(c'pc) \cup S(y)$ contains every generator. $S(y) \subseteq F(c'pc)$,if not, $c'pcy = c'pc\sigma_k y'$, and $x \wedge y = c\sigma_k$ for $\sigma_k \in S(y)$ and $\sigma_k \notin F(c)$. It's a contradiction because $c\sigma_k \neq c$. This shows $F(c'pc) \cup S(y) = F(c'pc)$. Only words contain Δ has finishing set containing all generator. Therefore $c'pc = \Delta q$ for some positive word q. $\inf \Delta^{m-1}c'pc = \inf c^{-1}ac = m$. Proves $c^{-1}ac \in SS(w)$. □

The theorem is true for more than the summit set, replace $SS(w)$ with $\{x | x \in [w], \inf x = m\}$, the above is still true.

7.3. **Complexity of the conjugacy problem.** The theorems implie a algorithm for solving the conjugacy problem.

Conjugacy Algorithm: Given w, w', return if $w \sim w'$ is true.

(1) Input w and w', present them in Garside normal form.
(2) Find $SS(w)$ and $SS(w')$.
(3) Test if $SS(w) = SS(w')$

Summit Set Algorithm: Given w, find $SS(w)$

(1) conjugate the input w with every simple element.
(2) Pick the set of produced elements, such that the Δ exponent is the largest.
(3) For each one of those elements, go though (1) again.
(4) If the process doesn't produce any more elements with larger Δ exponent, the summit set has been found.

The complexity greatly depend on the size of $SS(w)$. It is known $SS(w)$ could be exponential in l and n. In the algorithm of finding $SS(w)$, even step (1) takes $O(n!)$ operations, because there are $n!$ simple elements. For each one of the newly produced word, another $O(n!)$ conjugation has to be applied. To test if $SS(w) = SS(w')$, it have to solve a word problem, which is $O(l^2)$. The algorithm is at least $\Omega(kn!l^2)$, and likely to be much lager. A very loose upper bound would be $O((n!)^t l^2)$, where t is the amount of elements in $\{x | x \in [w], \inf x \geq \inf w\}$.

Garside's solution have inspired many optimizations, [names] have worked on restricted summit set to a even smaller sets, super summit set, later ultra summit set. Conjugation was improved by cycling elements. Simple elements was replaced by minimal simple elements. With all these improvements, the state of art algrithms is proven to run in $O(kl^2n^4)$ time, where k is the size of the ultra summit set, a refined subset of the summit set.

The size of ultra summit set is a open problem. However, k is recently proven to be at least exponential in n[Reference].

8. Automatic groups

8.1. **Regular languages.**

Definition 8.1 (String)**.** *A string over Σ is a finite sequence of elements in Σ. Σ is called the alphabet. Elements in Σ is called a character or a letter.*

A is usually used in place of Σ. Strings are similar to words in groups. In fact, words of a group G can be defined as strings with A equal to some generating set of G. String and word will be used interchangeably in this section. Similarly, character is the same as generator in this section.

Due to the similarity, we don't write strings as $a_1, a_2, \ldots, a_n$, but $a_1a_2 \ldots a_n$, where $a_i \in A$.

A set of string is known as a *language*. A empty set is therefore a empty language.

Definition 8.2 (Concatenation)**.** *The concatenation of string s_1 and s_2, written as s_1s_2, defined as $s_1s_2 = a_1 \ldots a_nb_1 \ldots b_m$ if $s_1 = a_1 \ldots a_n$ and $s_2 = b_1 \ldots b_m$, where $a_i, b_i \in A$.*

Definition 8.3 (Kleene Star)**.** *Kleene Star is a unary operator that takes set of strings A to the closure of finite sequence concatenation over A, written as A^*.*

A string s over A is therefore $s \in A^*$.

Definition 8.4 (Finite state automaton(FSA))**.** *A finite state automaton, or a finite state machine, is a 5-tuple $(\Sigma, S, s_0, \delta, F)$. Where σ is the alphabet, S is the set of states, $s_0 \in S$ is the starting state, $\delta : S \times \Sigma \to S$ is the state-transition function, $F \subseteq S$ the set of accept state.*

A FSA is used to accept or reject a string. The machine start at state s_0 and start reading the string from the left character by character. At state s, it read a character c, then the automaton move t the state $\delta(s, c)$. At the end of the automation, where there is no more characters in the string, if the automaton is at state $s \in F$, we say the automaton accepts the string, else we say the automaton rejects the string.

A FSA is called minimal, if no FSA that accept the same language uses less states. In this section, we assume all FSA are minimal.

Definition 8.5 (Regular Language)**.** *A language is regular if it is accepted by a FSA.*

We use $L(M)$ to denote the language accepted by FSA M. $|M|$ is the number of states in M.

We state some well known property of regular language and FSA without proof.

Theorem 8.1. *Regular language is closed under intersection, union, compliment, concatenation and kleene star.*

The concatenation of two languages L_1 and L_2, is $L_1L_2 = \{s_1s_2 | s_1 \in L_1, s_2 \in L_2\}$

Theorem 8.2. *There exist an algorithm to check if a FSA M accept only the empty language in $O(|M|^2)$ time*

A regular expression is a expression that defines a languages though concatnation, keleen star and union of strings.

Theorem 8.3. *If a regular expression matches language L, then there exist a FSA M, such that $L(M) = L$.*

8.2. **Background on automatic groups.**

Regular languages are easy, i.e. almost all natural problems involving regular languages can be solved easily, and most are in polynomial time.

Relating regular language with groups leads to the study of automatic groups. Automatic groups are groups with the following properties.

(1) There is a homomorphic and surjective map from some regular language to the group. This shows that, in linear time, one can read the string and decide if it represent an element of the group.

(2) Using the same map, there exist a FSA that can check if two words represent the same element in the group, or differ by one generator in the right.

Definition 8.6. *The alphabet A is a semigroup generator for G, if there exist a surjective homomorphism $\pi : A^* \to G$.*

The tuple (w, w'), where $w = a_1 \dots a_n$,$w' = b_1 \dots b_m$ and $a_i, b_i \in A$, is the string

$$(a_1, b_1) \dots (a_n, b_n)(\$, b_{n+1}) \dots (\$, b_m)$$

if $n < m$, and

$$(a_1, b_1) \dots (a_m, b_m)(a_{m+1}, \$) \dots (a_n, \$)$$

otherwise. \$ is the padding character so one can always make two words have the same length. It can be thought as the identity.

Formally, the automatic group are groups with automatic structures.

Definition 8.7. (A, L) *is the* automatic structure *of G, if the alphabet A is the semigroup generator of G, and if the following are true.*

(1) *There exist a* word acceptor*. The word acceptor W is an FSA on A, such that $L = L(W)$ and $\pi : L(W) \to G$ is surjective.*
(2) *There exist a FSA M_x on $(A, A) = \{(x, y)|x, y \in A\}$ for every generator $x \in A$, such that M_x accept words (w, w') if and only if $\pi(wx) = \pi(w')$. M_ϵ accept words (w, w'), such that $\pi(w) = \pi(w')$. M_x is called a* multiplier automaton*, M_ϵ is called the* equality recognizer*.*

Given $\pi : L \to G$ is a surjection, where L is regular. L has the (right) fellow traveler property if for any two word $w, w' \in L$, such that $wx = w'$ for some generator x, the distance between $\pi(w), \pi(w')$ at time t is on is at most K apart on the Cayley graph. K is called the fellow traveler constant.

The distance between w, w', $d(w, w')$ is defined as the length of the shortest path from $\pi(w)$ to $\pi(w')$ in the Cayley graph.

w at time t, denoted as $w(t)$, is the prefix of w with length t. If $t \geq |w|$, $w(t) = w$. In some contexts, $w(t) = w\$^{t-|w|}$ if $t > |w|$.

Theorem 8.4. *If regular language L has a surjection to G, then G has a automatic structure with word acceptor L if and only if L has the fellow traveller property.*

Proof. For any 2 strings w, w' differs by the generator x. M_x is part of some automatic structure with s states. At time t, let there be u and v, such that they are the shortest words where $(w(t)u, w'(t)v)$ is accepted by M_x. u and v must exist, since (w, w') are accepted by M_x. u, v can't have length longer than the amount of states in M_x, and will result a bound on the distance $d(w(t), w'(t))$. $\sup_{t \in N} d(w(t), w'(t)) \leq 2s$.

Given the fellow traveler property, one can construct M_x. Assume the fellow traveler constnat is K. Then the FSA M_x's state need to keep track only the difference between the word $w(t), w'(t)$. If it's larger than K, then move to the reject state. There are at most $2|A|^K$ ways they are different, each one can be encoded as a state.

M_ϵ can be constructed similarly. □

8.2.1. *Solution to the word problem.* The word problem for automatic group can be solved in $O(n^2)$ time, where n is the length of the word.

Find $w \in L(W)$, such that $\pi(w) = g$ for the element $g \in G$. Then run $(w, 1)$ with M_ϵ.

To find $w \in L(W)$, one have to express $g = \pi(a_1 a_2 \dots a_n)$, where a_i are generators. Use M_{a_1} find a_1, M_{a_2} find $a_1 a_2$, ..., M_{a_n} find $a_1 \dots a_n$. Given the string $s(i) = a_1 \dots a_i$, find the string $s(i+1)$ in by finding an accepted state on $M_{a_{i+1}}$, such that that state must be reached by $(s(i), y)$ for some y.

This process is linear. Define a sequence of sets $S_1, \ldots, S_{i+1}$. Such that S_j consist of the states reachable by $(s(j), x)$ for any x in FSA $M_{a_{i+1}}$. Each set is smaller than the set of states in $M_{a_{i+1}}$, which is a constant. From set S_j, S_{j+1} can be found in constant time.

To find y, pick any element in S_1, and find a transition to go to a element in S_2. Repeat the process until reach S_{i+1}, keep track of all the path taken. The path contains the string $(s(j), y)$.

From $s(i)$ to $s(i+1)$, it takes $O(i)$ time. The cost for finding g is therefore $O(n^2)$. Running $(w, 1)$ on M_ϵ is linear. The word problem is therefore solved in $O(n^2)$ time.

8.2.2. *Bi-automatic groups.* A group is bi-automatic if it is automatic, and also have another set of automata M'_x, $x \in A$, such that it accepts (w, w') if and only if $\pi(xw) = \pi(w')$. We can define a similar fellow traveler property for the left side.

Bi-automatic groups have solvable conjugacy problem. To check if g, g' are conjugates. create the language $L(g, g') = \{(w, w')|g\pi(w) = \pi(w')g'\}$, and note if there exist $(w, w) \in L(g, g')$, then $g = \pi(w)g\pi(w)^{-1}$. This is done by checking if $L(g, g') \cap \{(w, w)|w \in L\}$ is empty.

Automatic group also have the extended version of fellow traveler property.

Definition 8.8. *The language L has right [left] fellow traveler property if for word $wg = w'$ [$gw = w'$] , $\sup_{t\in\mathbb{N}} d(wg(t), w'(t)) \leq K|g|$ [$\sup_{t\in\mathbb{N}} d(gw(t), w'(t)) \leq K|g|$] for some constant K.*

This property also implies the existance of a FSA to check if w, w' differs a word g.

$L(g, g')$ is regular because bi-automatic imposes fellow traveler constant $\delta = K \max(|g|, |g'|)$. The set of $\{(w, w)\}$ regular. The intersection is regular. Emptyness of a regular language is decidable, thus the conjugacy problem is solvable.

Complexity wise, this is not a good solution. Construct FSA W, such that $L(W) = L(g, g')$. W must store the difference between the words, which is at most $2|A|^\delta$ possibilities. W' is W with all transition with (a, b) direct to a state that can't reach accept, where $a \neq b$. $L(W') = L(g, g') \cap \{(w, w)|w \in A^*\}$. The algorithm that check if $L(W')$ is empty for a FSA W' runs in $O(n^2)$ time, where n is the amount of states. If l is the length of the longer of 2 words, the conjugacy problem is solved in $O(|A|^{2Kl})$ time. Space complexity is the same, as the entire FSA has to be stored in memory.

8.3. **Application to the braid group.** Thurston proved the braid groups are bi-automatic with his left greedy form.[reference] The fellow traveler constant for B_n is $|\Delta| = \frac{(n-1)n}{2}$, therefore the algorithm cost $O((2n-2)^{l(n-1)n})$ time.

The rest of the section include a particular elementry proof that B_3 is automatic.

B_3 is particularly nice to analyze due to the uniqueness of the Garside normal form.

Theorem 8.5. *The Garside normal form is unique in B_3, and is in the form $\Delta^m \sigma_{a_1}^{e_1} \ldots \sigma_{a_k}^{e_k}$, such that $a_i \neq a_{i+1}$, $e_1, e_k \geq 1$ and $e_2, \ldots, e_{k-1} \geq 2$.*

Proof. For any word $w \in B_n$, the Garside normal form $\Delta^m p$ is unique up to elements p in B_n^+ and p does not have a factor of Δ. Now we show that there is only one way to write p.

Every p must be in this form. If for some $k > i > 1$, $e_i = 1$, then $a_{i-1}^{e_{i-1}} a_i^{e_i} a_{i+1}^{e_{i+1}} = a_{i-1}^{e_{i-1}-1} \Delta a_{i+1}^{e_{i+1}-1}$, which is a contradiction because p doesn't have a factor of Δ.

Every word in that form is also in B_n^+, and without Δ factor.

It's impossible to rewrite any word in this form with the relations $\sigma_1\sigma_2\sigma_1 = \sigma_2\sigma_1\sigma_2$. Therefore this form is unique. □

Theorem 8.6. *The Garside normal form of B_3 is regular.*

Proof. Let $A = \{\Delta, \Delta^{-1}, \sigma_1, \sigma_2\}$, then A generates G. The Garside normal form is a regular lanuage over A because they exist an explicit regular expression R that matches it.

$R = (\Delta^* \cup (\Delta^{-1*}))(X \cup X')$ where $X = \sigma_1^*(\sigma_2\sigma_2\sigma_2^*\sigma_1\sigma_1\sigma_1^*)^*((\sigma_2\sigma_2\sigma_2^*\sigma_1^*) \cup \sigma_2^*))$ and X' is the same except the σ_2 and σ_1 are switched. □

Let W be a FSA generated by the regular expression R.

M_ϵ exists. M_ϵ accepts all language of the form $\{(w,w)|w \in L(W)\}$. It have the same structure as W, but replacing every transition x with (x,x).

Instead of construct M_x directly, we only need to prove the fellow traveler property is true. This can be done by checking it is true for every M_x independently.

Note that every generator has a constant distance from another generator.

Consider x case by case:

(1) Generator Δ: The words differ in 1 Δ are $\Delta^m p$ and $\Delta^{m+1}R(p)$. Assume m is non-negative. At time $|m|+1$, one encounters $\Delta^m\sigma_{a_1}$ and Δ^{m+1}, which has a distance of 2. At time $|m|+2$, $\Delta^m\sigma_{a_1}\sigma_{a_2}$ and $\Delta^{m+1}\sigma_{3-a_1} = \Delta^m\sigma_{a_1}\Delta$, which also has distance 2. By induction, the distance is 2 till the end, where one has $\Delta^m\sigma_{a_1}\dots\sigma_{a_k}$ and $\Delta^m\sigma_{a_1}\dots\sigma_{a_{k-1}}\Delta$. With one more step, it decrease the distance to 1. When m is negative, it's similar.

(2) Generator σ_1: There are two cases

Case 1: If the word end with σ_1, or end with σ_2^b, where $b \geq 2$, then the two words are exactly the same until the end. It implies the fellow traveler property.

Case 2: If the word end with a single σ_2, then the two words are $\Delta^m p\sigma_1^k\sigma_2$ and $\Delta^m p\sigma_1^k\sigma_2\sigma_1$. The second word in normal form is $\Delta^{m+1}R(p\sigma_1^{k-1})$. Note the first word is $\Delta^m p\sigma_1^{k-1}\cdot\sigma_1\sigma_2$. We can use the fact that adding the Δ have fellow traveler property to prove this also have that property. In fact, the two words at any time are at most 3 apart.

The cases for Δ^{-1} and σ_2 can be proved similarly

The fellow traveler constant is 3. It implies B_3 is automatic.

B_3 is also biautomatic, to see that, one can check the fellow traveler property for each generator when multiplied on the left. The proof are similar to the one above.

Given that we found the fellow traveler constant to be 3, the time complexity of the conjugacy problem is $O(4^{2Kl}) = O(2^{12l})$.

9. Small cancellation theory

9.1. **Dehn Diagrams.** This section introduces the geometric tool of Dehn Diagrams. Intuitively, given a word in the generators of a group which represents the identity, a Dehn Diagram for that word is a geometric representation of the process of reducing the word to the identity.

Definition 9.1. *A Dehn Diagram is a finite set of verticies and edges which form a connected graph. To each edge there is associated an element from a group G, via a function $\phi : E(D) \to G$, such that the following hold:*
1. Given any edge $e \in E(D)$, $\phi(e) = \phi^{-1}(e^*)$ where e^* denotes the edge pointing in the opposite direction as e.
2. Given any closed loop consisting of the edges $e_1, ..., e_n$, $\phi(e_1)...\phi(e_n) = 1$.

If the reader is familiar with Cayley Graphs, then it is clear that any connected subgraph of a Cayley graph forms a diagram. In fact if we consider a word in the generators of G which is trivial and thus forms a loop in the Cayley graph, and we take the interior and the boundary of this loop, we will obtain a Dehn Diagram which describes the process of reducing the given word to the identity.

Definition 9.2. *A Dehn Diagram is said to represent a word $w = 1 \in G$, if the boundary of the Diagram, when read either clockwise or counterclockwise is precisely that word.*

Now let $G =< X|R >$ be a presentation for a group. We call a Dehn Diagram for a word $w = 1 \in G$ reduced if the boundary of every region inside the Diagram represents a cyclic permutation of a generator. This means that the boundary of no region inside the Diagram freely reduces to the trivial element.

Theorem 9.1. *Let $G =< X|R >$ be a group. If a word w in the generators and their inverses is trivial in G then there exists a reduced Dehn Diagram representing the given word.*

Proof. Construct a Cayley graph for G and consider the word w which forms a loop in the Cayley graph. The interior and boundary of this loop for a reduced Dehn Diagram for w. This simply follows from the fact that in the Cayley graph of a group the boundary of a region can never freely reduce to the identity, since a word which freely reduces in a Cayley graph completely backtracks andso doesn't bound any region. □

As an example, in a free group all reduced Dehn Diagrams are lines since no closed loop can bound a region. The word is read by reading all the way along the line and then returning back to the starting point. It is clear that such a word will cancel freely. In the group $\mathbb{Z} \times \mathbb{Z}$ the reduced Dehn Diagrams are bounded regions of the integer lattice in $\mathbb{R}^2$. In this case each square in the lattice represents the defining relator $aba^{-1}b^{-1} = 1$.

9.2. **Small Cancellation Theory.** The idea of small cancellation theory is to use combinatorial properties of the relators of a group to make geometric statements about its Dehn Diagrams. These geometric statements can be used to show for example that Dehn's algorithm solves the word problem in a group G.

Definition 9.3. *Let $G =< X|R >$ be a presentation for a group in which every relator is freely cyclically reduced and let R^* denote the set of all cyclic permutations of relators and their inverses. Define a piece to be a subword of an element in R^* which can be cancelled in a product of two words r_1r_2 such that $r_1, r_2 \in R^*$ and such that $r_1 \neq r_2^{-1}$.*

Definition 9.4. *Let $G =< X|R >$ be a presentation for a group in which every relator is freely cyclically reduced and let R^* denote the set of all cyclic permutations of relators and their inverses. The presentation is said to satisfy the $C'(\lambda)$ small cancellation condition if given any two elements $r_1, r_2 \in R^*$ which are not inverses, the total length of the piece of r_1 which cancels in the product r_1r_2 is less that λ times the length of r_1.*

Definition 9.5. *Let $G =< X|R >$ be defined as above. The the presentation satisfies the $C(p)$ small cancellation condition if every relator is the product of at least p pieces.*

Notice that $C'(\lambda)$ implies $C(p)$ for $\lambda < \frac{1}{p-1}$. The geometric interpretation of the $C(p)$ condition is that every interior region in any reduced Dehn Diagram has at least p sides.

Definition 9.6. *Let $G =< X|R >$ be the presentation of a group. Let R^* denote the set of cyclic permutations of the elements of R and their inverses. The the presentation satisfies the $T(k)$ small cancellation condition if given any sequence of terms $r_1, r_2, ..., r_l$ such that $r_i \in R^*$, no two consecutive terms are inverses and the first and last terms are not inverses and $l \geq 3$, if $l < k$ then at least one of the cyclic products r_ir_{i+1} or r_lr_1 must be reduced without cancellation in the free group. This means that if all of these cyclic products have cancellation, then $l \geq k$.*

The geometric interpretation of this condition is that every interior vertex in any Dehn Diagram must have degree at least k.
Dehn's algorithm will be important in what follows. Dehn's algorithm for solving the word problem in a group freely reduces the word and then checks whether or not a subword of one of the cyclic permutations of that reduced word contains more than half of a defining relator. If this is the case, then that piece of the relator is replaced by the inverse of the rest of the relator which shortens the length of the word. This process is repeated until either the word is trivial or does not contain more than half of one of the defining relators. If the latter case occurs, then Dehn's algorithm returns that the word is nontrivial. Dehn's algorithm solves the word problem if every word which is trivial must contain more than half of one of the defining relators.

Theorem 9.2. *Let $G =< X|R >$ be a presentation for a group and assume that this presentation satisfies the $C^{'}(\lambda)$ small cancellation condition for $\lambda \leq \frac{1}{6}$. Then any word which is trivial in G must contain more than half of one of the relators. In particular, Dehn's algorithm will solve the word problem for these groups.*

Proof. The proof uses a geometric invariant called the Euler characteristic. The Euler characteristic of a Diagram or polyhedron is $F-E+V$ where F is the number of faces or regions, E is the number of edges, and V is the number of verticies. It is a theorem in geometry that the Euler characteristic of a diagram is 1. Now consider the reduced Dehn diagram of a word which is trivial in G. In particular consider the contribution of the interior regions to the Euler characteristic. Since every vertex has degree at least three since the diagram is reduced and every region has six sides we count that for each edge we have $\frac{2}{6} = \frac{1}{3}$ of a region (each region has six edges and each edge borders two regions since we are considering the interior) and $\frac{2}{3}$ of a vertex (each vertex borders three edges and each edge connects two verticies). Summing this over the interior of the diagram we count that the total contribution from the interior to the Euler characteristic is less than 0 since the regions which are on the boundary were not counted in this sum but some of their edges were. Thus in order to obtain 1 we must have that the contribution of the boundary regions is positive. This means in particular that the number of sides of some region must be strictly less that six. Since the boundary regions contain one edge which borders no other region, we see that even five sides is too many for a boundary region to give a positive contribution. Thus we have that at least one of the boundary regions of the diagram must have at most 4 sides. Three of these sides are interior edges and thus can consist of less that $\frac{1}{6}$ of the relator corresponding to that region. Thus the one side which is on the boundary and is thus a part of one of the cyclic permutations of the word must contain more than half of the relator corresponding to the given boundary region. This completes the proof. □

Groups satisfying the above property are called sixth groups.

Theorem 9.3. *Let $G =< X|R >$ be a presentation for a group and assume that this presentation satisfies the $C^{'}(\lambda)$ small cancellation condition for $\lambda \leq \frac{1}{4}$ and additionally satisfy the $T(4)$ condition. Then any word which is trivial in G must contain more than half of one of the relators. In particular, Dehn's algorithm will solve the word problem for these groups.*

Proof. As in the proof of the last statement we use the geometric fact that the Euler characteristic of a diagram must be 1. Again we consider only interior regions of the diagram and count that for each edge we have $\frac{2}{4} = \frac{1}{2}$ of a region (each interior region has at least four sides and each edge is the boundary of two regions) and $\frac{2}{4} = \frac{1}{2}$ verticies (each interior vertex has degree at least four and each edge connects to verticies). Summing this we obtain again that the total contribution of the interior regions to the Euler characteristic is at most 0 since the regions which are on the boundary were not counted in this sum but some of their edges were. Thus we must have that the contribution of the boundary regions is positive and thus that some boundary region has at most 3 sides. But the two of those sides which are on the interior contain less than half of the defining relator corresponding to that region by the $C^{'}$ condition. Hence the side of that region which is on the boundary must consist of more than half of the defining relator. □

If we replace the $C^{'}(\lambda)$ condition by an strong enough $C(k)$ condition so that we obtain the same restriction on the number of sides of any interior region of a diagram, we can obtain similar results in the same way. Thus the following theorems have analogous proofs.

Theorem 9.4. *Let $G =< X|R >$ be a presentation for a group and assume that this presentation satisfies the $C(k)$ small cancellation condition for $k \geq 6$. Then any word which is trivial in G must contain a subword which is the product of three pieces from a relator.*

Theorem 9.5. *Let $G =< X|R >$ be a presentation for a group and assume that this presentation satisfies the $C(k)$ small cancellation condition for $k \geq 4$ and additionally satisfy the $T(4)$ condition. Then any word which is trivial in G must contain a subword which is the product of two pieces from a relator.*

The last theorem applies to B_3 and can be used to produce a modified Dehn alogithm to solve the word problem. The idea of the algorithm is to find all subwords of the given word and it's cyclic permutations which are the product of two pieces from a relator. We may then rewrite the word by replacing any of these subwords by the inverse of the rest of the relator. The only problem is that these rewrittings do not shorten the word. Thus we have to try all possible such rewrittings until we find one which actually shortens the word by allowing us to get rid of three pieces instead of just two. We continue this process until we either end up with the identity or none of the possible rewrittings lead to a shortening of the word. This algoithm is considerably more inefficient than Dehn's algorithm since we must try all possible rewrittings at every step. This algorithm is also guarateed to work since if none of the possible rewrittings lead to cancellation of more that two pieces then the word can not be further shortened by any of the relations. Of course in order to use this algorithm we must first show that B_3 satisfies the conditions of the theorem.

Theorem 9.6. *B_3 satisfies the $C(4)$ and $T(4)$ small cancellation conditions.*

Proof. To prove this we use the presentation $B_3 =< a, b|a^2b^{-3} >$. Notice that we can cancel b^{-2} in any product which doesn't completely cancel, for example $a^2b^{-3}b^2a^{-2}b$ leads to b^{-2} calcelling. However, we can never cancel more that one a in any such product, since cancelling both a's would mean that only b's would be left over in either word and these b's must necessarily have opposite powers and thus the entire product cancells. Finally, ab^{-1} can also never cancel without the entire product cancelling since then both words must be of the form $b^{-2}a^2b^{-1}ba^2b^2$ which completely cancels. Thus the pieces of this relator are a,a,b^{-1},b^{-2}. Hence this presentation satisfies the $C(4)$ condition. To prove the $T(4)$ condition consider a sequence of 3 relators r_1, r_2, r_3. Assume that there is cancellation between r_1 and r_2 and assume that without loss of generality that r_1 contains a's and b^{-1}'s. Then r_2 must contain b's and a^{-1}'s and in order for there to be cancellation between r_2 and r_3 we must have that r_3 again contains a's and b^{-1}'s. But then the product r_3r_1 can not have any cancellation since both contain the a's and b^{-1}'s. Thus this presentation satisfies $T(4)$. □

This modified Dehn algorithm is considerably less efficient that the algorithms which we have previously developed using the given presentation for B_3. Thus we didn't further study this modified Dehn algorithm.

9.3. **Conjugacy Problem in Small Cancellation Groups.** It is possible to solve the conjugacy problem in groups which satisfy the aforementioned small cancellation conditions. This is done by introducing a variant of the Dehn diagram, namely an annular diagram.

Definition 9.7. *An annular diagram connecting two words in the generators of a group is a Dehn Diagram which has the shape of an annulus. Namely there is an outer boundary and an inner boundary. This is to say that the diagram is not completely filled in or has a hole in the middle.*

The idea is that two words are conjugate if and only if they are the outer and inner boundaries of an annular diagram.

Theorem 9.7. *Let $G =< X|R >$ be the presentation of a group. Let w_1 and w_2 be words in the generators of G. Then w_1 and w_2 are conjugate iff there exists an annular diagram such that w_1 is the outer boundary and w_2 is the inner boundary.*

Proof. Assume that there exists such an annular diagram. Choose any vertex on the exterior and any vertex on the interior boundary. Choose a shortest path connecting these verticies and let a denote the word associated to that path. Such a path must exists since the diagram is connected.

Now cut the diagram open along this path to obtain a Dehn diagram whose boundary is $w_1 a w_2^{-1} a^{-1}$. Hence by the previous section we must have $w_1 a w_2^{-1} a^{-1} = 1$ and hence w_1 and w_2 are conjugate. Now assume that $w_1 = a w_2 a^{-1}$. Then create a Dehn diagram for the word $w_1 a w_2^{-1} a^{-1}$ and undo the process of the previous argument. That is glue the a and a^{-1} sections of the diagram together to create an annular diagram. This completes the proof. □

Now we come to the main theorem.

Theorem 9.8. *Let $G =< X|R >$ be a group presentation which satisfies either the $C'(\frac{1}{6})$ or the $C'(4)$ and $T(4)$ small cancellation conditions. Assume that w_1 and w_2 are two words which are completely reduced using Dehn's algorithm. That is w_1 and w_2 do not contain more than half of any relator. Then if w_1 and w_2 are conjugate, then two of their cyclic permutations w_1^* and w_2^* must be conjugate by an element which is the product of at most 3 pieces of one of the relators. In particular, if the presentation in finite, then the group has solvable conjugacy problem.*

Proof. The idea of the proof is to use the assumption that w_1 and w_2 are both completely reduced using Dehn's algorithm and the small cancellation conditions to show that any annular diagram with w_1 on the outer boundary and w_2 on the inner boundary has to be in a certain sense thin. Essentially this means that there are no interior regions for such an annular diagram. It is again a geometric fact that the Euler characteristic of such an annular diagram is 0. Now since the outer edges are reduced by Dehn's algorithm they must have at least 5 sides for the $C'(\frac{1}{6})$ condition or 4 sides for the $C'(\frac{1}{4})$ condition (the edge on the boundary must make up less that half of the relator in either case). Thus since one of their edges is on the boundary and all others are on the interior, and two verticies are on the boundary and all others are on the interior, a simple calculation shows that the total contribution of the outer regions to the Euler characteristic must be 0 because the $C'(\frac{1}{6})$ or $C'(\frac{1}{4})$ conditions are satisfies for all interior edges and verticies. But if the diagram had any regions which were completely on the interior, then the $C'(\frac{1}{6})$ condition would imply that these regions have at least 7 sides and their verticies had degree at least 3 which would mean that their contribution to the Euler characteristic would be negative, a contradicion. If the $C'(\frac{1}{4})$ and $T(4)$ conditions were satisfied then any interior region would have at least 5 sides and all of its verticies would have degree at least 4. Thus the contribution of any interior region to the Euler characteristic in this case would also be negative, a contradiction. Hence if either of the small cancellation conditions are satisfied, then such an annular diagram cannot have any interior regions. However, it is then easy to see that the shortest path from the inner to the outer boundary must have length at most three. Hence two cyclic permutations of w_1 and w_2 respectively must be conjugate by an element which is the proudct of at most three pieces of the relators. If the presentation is finite, solubility of the conjugacy problem now results from the fact that the number of possible conjugators is finite. □

10. Conclusions

Algorithm	B_3		B_n	
	Time	Space	Time	Space
Word problem with Artin Combing	$O(3^l)$	$O(3^l)$	$O(n3^l)$	$O(n3^l)$
Word problem with unique Graside normal form	$O(l)$	$O(l)$	-	-
Word Problem with Thurston left normal form	$O(l^2)$	$O(l)$	$O(l^2 n \log n)$	$O(ln)$
Word Problem with $a^2 = b^3$ presentation	$O(l)$	$O(l)$	-	-
Conjugacy Problem with $a^2 = b^3$ presentation	$O(l)$	$O(l)$	-	-
Conjugacy Problem with automatic groups	$O(2^{12l})$	$O(2^{12l})$	$O((2n-2)^{l(n-1)n})$	$O((2n-2)^{l(n-1)n})$
Conjugacy Problem with Ultra Summit Sets	$O(kl^2)$	$O(kl^2)$	$O(kl^2n^4)$	$O(kl^2n^4)$

k is the size of the ultra summit set depending on the input. l is the length of the longer word. n is the number of strands.

10.1. **Automorphism of free group.** There exist a faithful homomorphism $^* : B_n \to \mathrm{Aut}(F_n)$.

If $B_n \to \mathrm{Aut}(F_n)$ is faithful and it sends w to w^*, then w is I iff $w^*(a) = a$ for every generator a of the free group F_n.

There is one explicate map. Defined as

$$\sigma_i^*(t_j) = \begin{cases} t_j & \text{if } j \neq i, i+1 \\ t_{i+1} & \text{if } j = i \\ t_{i+1} t_i t_{i+1}^{-1} & \text{if } j = i+1 \end{cases}$$

Where t_i is generators of F_n, and σ_i are the generators of the braid group. The inverse can easily be found,

Using a recursive definition. Let any word of the form $(\sigma_i w)^*$ be $\sigma_i^* \circ w^*$. Test if $w^*(t_i) = t_i$ for every generator in F_n is applying a list of σ_i to elements in F_n.

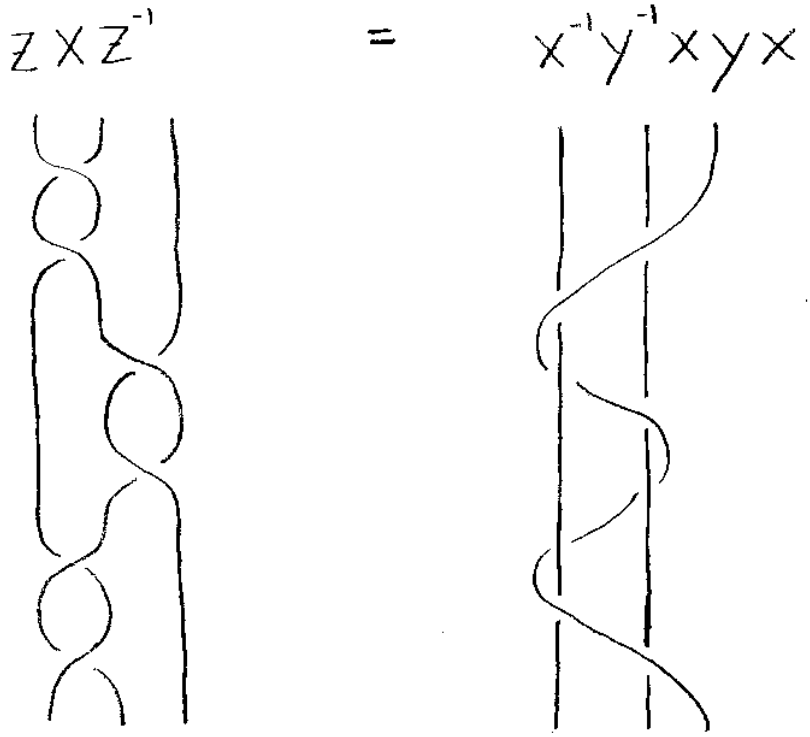
An application of σ_i^* can potentially triple the length of the word. The complexity of the algorithm is $O(3^l)$.

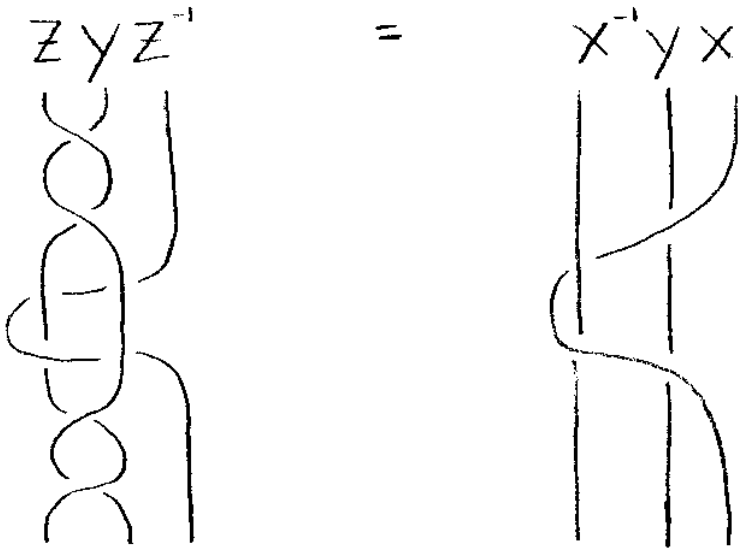
Consider the special case B_3. If we define a_n, b_n, c_n to be the number of t_1, t_2, t_3 (include it's inverses) in $((\sigma_2 \sigma_1^{-1})^n)^*(t_1)$, ignoring the possibility of cancellation. We have the following recurrence relation.

$$\begin{aligned} a_{n+1} &= 2a_n + b_n \\ b_{n+1} &= c_n \\ c_{n+1} &= a_n + 2c_n \end{aligned}$$

Solving the recurrence relation, we find $a_n + b_n + c_n = 2F_{2n+1} - 1$, therefore $|((\sigma_2 \sigma_1^{-1})^n)^*(t_1)| = 2F_{2n+1} - 1$, where F_n is the nth Fibonacci number.

It's easy to show there is no cancelation at any step. Thus it is exponential even for B_3.


zxz^-1
=
x^-1y^-1xyx


zyz^-1
=
x^-1yx