POLYNOMIAL-TIME WORD PROBLEMS

SAUL SCHLEIMER

ABSTRACT. We find polynomial-time solutions to the word problem for free-by-cyclic groups, the word problem for automorphism groups of free groups, and the membership problem for the handlebody subgroup of the mapping class group. All of these results follow from observing that automorphisms of the free group strongly resemble straight line programs, which are widely studied in the theory of compressed data structures. In an effort to be self-contained we give a detailed exposition of the necessary results from computer science.

1. Introduction

Automorphisms of the free group are closely connected to two techniques in computer science: *string matching* and *compression*. The relevance of the first is obvious. The second is less clear. So, consider the fact that an automorphism of complexity n can produce, by acting on a generator, a word of size at most $\exp(n)$. Now, there are only $\exp(n)$ such automorphisms while there are $\exp(\exp(n))$ words available as output. Thus most words in the free group *cannot* be obtained in this way. Those which can are highly regular and thus susceptible to compression.

Compression techniques have already made an appearance in algorithmic topology. Word equations play a starring role in the work of Schaefer, Sedgwick, and Štefankovič [25]. One of the problems they consider, connectedness of normal curves and surfaces, is also addressed by the orbit-counting techniques of Agol, Hass, and Thurston [1]. Both results rely, directly or indirectly, on Plandowski's Algorithm [24] (Theorem 8.1 below).

The structure of the paper is as follows: Section 2 reviews *straight line programs* and also a slight variant, *composition systems*. Such programs are called *compressed words*. Section 3 is an exposition of Lohrey's Theorem [18]:

Date: March 6, 2007.

Theorem 3.5. *The word problem for compressed words in the free group is solvable in polynomial time.*

We use this theorem to answer a variety of questions; in each case the compression technique "accelerates" an obvious exponential-time algorithm.

Theorem 4.1. *For any automorphism $\Phi \in \mathrm{Aut}(F_m)$ the word problem for the free-by-cyclic group $G_\Phi = F_m \rtimes_\Phi \mathbb{Z}$ is polynomial time.*

This problem is already known to be in **NP**: Bridson and Groves [7] show that G_Φ has a quadratic isoperimetric inequality. A generalization gives:

Theorem 5.2. *The word problem for $\mathrm{Aut}(F_m)$ is polynomial time.*

This solves problem (C1) on the list maintained by Baumslag, Myasnikov, and Shpilrain [3]. In Section 6 we discuss membership problems: deciding whether or not a given word belongs to a subgroup. In particular, if V is a handlebody and $S = \partial V$ then Broaddus asks if there is a polynomial-time algorithm to decide whether or not a homeomorphism of S extends over V. We prove:

Theorem 6.4. *The membership problem for $\mathcal{MCG}(V)$ in $\mathcal{MCG}(S)$ is polynomial time.*

Sections 7 and 8 give expositions of theorems due to Hagenah [12] and Plandowski [24], upon which Lohrey's Theorem relies. As a result the computer science portions of the paper are self-contained. I have been somewhat more permissive when using techniques from combinatorial group theory or three-manifolds.

The paper ends with a brief appendix (Section A) indicating how these techniques extend to closed surface groups. In particular, we prove:

Corollary A.6. *The word problem for compressed words in a closed surface group is solvable in polynomial time.*

From this we give a new polynomial-time algorithm to solve the word problem in the mapping class group $\mathcal{MCG}(S)$.

Acknowledgements. I thank Martin Bridson and Nathan Broaddus for both inspiring seminar talks and for illuminating conversations. Without such models this paper would not exist. I also thank Ilya Kapovich for his many helpful comments.

2. Straight line programs and composition systems

Recall that if $\mathcal{L}$ is a set of characters, or an *alphabet*, then the *Kleene closure* $\mathcal{L}^*$ is the set of all *words* (finite strings of characters) in $\mathcal{L}$. If

$w \in \mathcal{L}^*$ then $|w|$ is the *length* of w: the number of characters of w counted with multiplicity. We write ϵ for the *empty string*: the word of length zero. If u and v are words we write $u = v$ if and only if u and v are identical as strings.

Fix a word w and assume $0 \leq i \leq j \leq |w|$. We take $w[i : j]$ to be the substring starting immediately before the $i + 1^{\text{th}}$ character of w and ending just after the j^{th} character. Thus $w[i : i] = \epsilon$ and, in general, $|w[i : j]| = j - i$. By convention negative indices count from the end of w. Thus:

$$w[-j : -i] = w[|w| - j : |w| - i].$$

The following abbreviations will be useful: $w[: i]$ for $w[0 : i]$, $w[i :]$ for $w[i : |w|]$, and $w[i]$ for $w[i : i + 1]$. Thus $w[0]$ is the first character of w while $w[-1]$ is the last.

If u, v are words in $\mathcal{L}^*$ then their concatenation is denoted by $u \cdot v$. This leads to the pleasant identity $w = w[: i] \cdot w[i :]$. Also, in order to *rotate* a word w exactly i characters simply form $w[i :] \cdot w[: i]$. For any word $w \in \mathcal{L}^*$ define $\operatorname{rev}(w)$ to be the *reverse* of w: so $\operatorname{rev}(w)[i] = w[-i - 1]$.

A *straight line program* $\mathbb{A} = \langle \mathcal{L}, \mathcal{A}, A_n, \mathcal{P} \rangle$ contains the following: a finite alphabet $\mathcal{L} = \{a_1, \ldots, a_m\}$ of *terminal* characters, a disjoint finite alphabet $\mathcal{A} = \{A_1, \ldots, A_n\}$ of *non-terminal* characters, a *root* terminal $A_n \in \mathcal{A}$, and a set $\mathcal{P} = \{A_i \to W_i\}$ of *production rules*. These last allow us to replace a non-terminal A_i with its *production*: a (possibly empty) word W_i in $(\mathcal{L} \cup \mathcal{A})^*$. Every non-terminal A_j appearing in W_i has index $j < i$. When indices are unimportant we shall use the non-terminal A to represent the root of $\mathbb{A}$.

To justify the term "root" define the *production tree* of a character in $\mathcal{L} \cup \mathcal{A}$ as follows: The tree for a terminal $a \in \mathcal{L}$ is a single vertex, labelled a. The tree for a non-terminal $A_i \in \mathcal{A}$ is planar, has root labelled A_i, and, attached to the root in left-to-right order, contains a copy of the production tree for every character of W_i.

If $B \in \mathcal{L} \cup \mathcal{A}$ then the *height* of B, denoted $||B||$, is the height of the production tree of B: the maximal distance from the root to a leaf. For example, terminals have height zero.

Define $w(A) = w_A$ to be the word of $\mathcal{L}^*$ that results from running the straight line program $\mathbb{A}$. That is, if A_n is the root then produce W_n, replace all non-terminals appearing with their productions, and continue doing so until the resulting word lies in $\mathcal{L}^*$. This is exactly the word appearing at the leaves of the production tree of $\mathbb{A}$. Also, define $w(A_i) = w_{A_i}$ to be the word in $\mathcal{L}^*$ produced by the non-terminal A_i.

A straight line program is in *Chomsky normal form* if every production W_i has length one or two: all of the former lie in $\mathcal{L}$ while all of the latter lie in $\mathcal{A}^*$.

Remark 2.1. Every straight line program can be placed in normal form in time polynomial in $\Sigma_i |W_i|$. This is done by introducing dummy non-terminals.

Remark 2.2. Two remarks are in order about the meaning of *polynomial time*. The variable in question is the *bit-size* of the input. The exact bit-size depends on a choice of *encoding*; for example, a straight line program in normal form, with n non-terminals, should have bit-size $O(n \log_2(n))$ The precise running-time of an algorithm also depends on the *model of computation*. It is generally true that changing encoding or model of computation transforms the bit-size or running-time by a polynomial function. Thus the claim that some problem may be solved in polynomial time is essentially independent of these choices.

Example 2.3. Here is the canonical example of a straight line program:

$$\mathbb{F} = \left\langle \begin{array}{l} \{a, b\}, \{F_i\}, F_n, \\ \{F_i \to F_{i-1} \cdot F_{i-2}\}_{i=3}^n \cup \{F_2 \to a, F_1 \to b\} \end{array} \right\rangle.$$

So $w(F_i)$ is the i^{th} Fibonacci word. For example

$$\begin{aligned} w(F_8) &= abaababaabaababaababa, \\ w(F_9) &= abaababaabaababaababaabaababaabaab. \end{aligned}$$

It follows that the length $|w(F_n)|$ grows exponentially with n.

Composition systems are a more flexible version of straight line programs (in normal form), introduced in [11]. We present a slight generalization, due to [18]: If $A, B, C \in \mathcal{A}$, then productions of the form $A \to B[i : j] \cdot C[k : l]$ are allowed. Here $B[i : j]$ is a *truncated* non-terminal. Truncated non-terminals only appear on the right hand side of productions, never on the left. A truncated non-terminal $B[i : j]$ is well formed if the indices satisfy $0 \leq i \leq j \leq |w_B|$. Define $w(B[i : j]) = w_B[i : j]$. Repeated truncation behaves quite simply: $(B[i : j])[k : l] = B[i + k : i + l]$.

Example 2.4. Consider the straight line program in normal form

$$\mathbb{A} = \left\langle \begin{array}{l} \{a, b\}, \{A_i\}, A_n, \\ \{A_i \to A_{i-1} \cdot A_{i-1}\}_{i=4}^n \cup \{A_3 \to A_2 \cdot A_1, A_2 \to a, A_1 \to b\} \end{array} \right\rangle.$$

Of course, $w(A_i) = (ab)^{2^{i-3}}$ for $i \geq 3$. Truncate a little to obtain:

$$\mathbb{B} = \left\langle \begin{array}{c} \{a, b\}, \{B_i\}, B_n, \{B_i \to B_{i-1}[1:] \cdot B_{i-1}[1:]\}_{i=5}^{n} \cup \\ \cup \{B_4 \to B_3 \cdot B_3, B_3 \to B_2 \cdot B_1, B_2 \to a, B_1 \to b\} \end{array} \right\rangle.$$

Now the output appears to be more interesting: for example

$$w(B_8) = bababbabbbababbabbababbabbbababbab.$$

For future use, we record:

Lemma 2.5. *If $\mathbb{A}$ is a composition system of height $||A||$ then $|w_A| \leq 2^{||A||}$.* □

Lemma 2.6. *There is a polynomial-time algorithm that, given a composition system $\mathbb{A}$, computes $|w_B|$ for all $B \in \mathcal{A}$.* □

Lemma 2.7. *There is a polynomial-time algorithm that, given a composition system $\mathbb{A}$ and an integer i, computes the character $w_A[i]$.* □

Remark 2.8. In particular, suppose we are given i, j with $0 \leq i \leq j \leq |w_A|$. Then we may compute the word $w_A[i : j]$ in time polynomial in $j - i$ and in the size of $\mathbb{A}$.

It is perhaps surprising that the expressive power of composition systems and straight line programs are nearly the same. In Hagenah's thesis [12, Chapter 8] we find:

Theorem 7.1. *There is a polynomial-time algorithm that, given a composition system $\mathbb{A}$, finds a straight line program $\mathbb{X}$ with $w_X = w_A$.*

For the convenience of the reader, the algorithm and a proof of correctness are presented in Section 7. A more subtle result is due to Plandowski [24]:

Theorem 8.1. *There is a polynomial-time algorithm that, given straight line programs $\mathbb{A}$ and $\mathbb{X}$ in normal form, decides whether or not $w_A = w_X$.*

In an attempt to be self-contained a proof appears in Section 8. Gasieniec, Karpinski, Plandowski, and Rytter [11] strengthen Theorem 8.1 as follows:

Theorem 2.9. *There is a polynomial-time algorithm that, given composition systems $\mathbb{A}$ and $\mathbb{X}$, computes the largest integer $k \geq 0$ where $w_A[: k] = w_X[: k]$.*

Proof. This is an application of Theorems 7.1 and 8.1 and of binary search. Suppose that A and X are the root non-terminals for $\mathbb{A}$ and $\mathbb{X}$, respectively. Theorems 7.1 and 8.1 result in a polynomial-time algorithm that, given any number i with $0 \leq i \leq \min\{|w_A|, |w_X|\}$, decides if $w_A[: i] = w_X[: i]$. Call this algorithm the *prefix checker.*

Let $j_0 = \min\{|w_A|, |w_X|\}$. If the prefix checker accepts j_0 then take $k = j_0$ and we are done. Otherwise, let $i_0 = 0$ and proceed as follows. We are given numbers i_n, j_n with

- $i_0 \leq i_n < j_n \leq j_0$ and
- $w_A[: i_n] = w_X[: i_n]$ and $w_A[: j_n] \neq w_X[: j_n]$

If $i_n + 1 = j_n$ then set $k = i_n$ and we are done. Otherwise, let i' be greatest integer less than $(i_n + j_n)/2$. Run the prefix checker with input i'. If i' is accepted then let $i_{n+1} = i'$ and let $j_{n+1} = j_n$. If i' is rejected then let $i_{n+1} = i_n$ and let $j_{n+1} = i'$.

Finally, notice that that the prefix checker is called at most $O(\log_2(j_0))$ times. This completes the proof. □

3. Lohrey's Algorithm

We now turn our attention to the free group. Let $\mathcal{L}_m = \{a_i, \overline{a}_i\}_{i=1}^m$. Let $\overline{\cdot}\colon \mathcal{L}_m \to \mathcal{L}_m$ be the obvious involution. Given a word $w(a_i) \in \mathcal{L}_m^*$ define $\overline{w}$ to be $\operatorname{rev}(w(\overline{a}_i))$.

Compressed word is the umbrella term for a straight line program or composition system which produces a word in $\mathcal{L}_m^*$. The involution given above extends to compressed words; constructions like $\overline{B}$ are allowed on the right hand side of productions. If $A \to B[i : j] \cdot C[k : l]$ then $\overline{A} \to \overline{C}[-l : -k] \cdot \overline{B}[-j : -i]$. Finally, if $A \to a$ then $\overline{A} \to \overline{a}$. See Example 3.2 below for an illustration.

Lemma 3.1. *There is a polynomial-time algorithm that, given a compressed word $\mathbb{A}$, computes a new compressed word $\overline{\mathbb{A}}$ with $w(\overline{A}) = \overline{w(A)}$.* □

Now suppose that $\{a_1, \ldots, a_m\}$ generate the free group F_m and that $\overline{a}_i$ is the inverse of a_i. Recall that a word in the free group is *freely reduced* if it has no subwords of the form $a_i\overline{a}_i$ or $\overline{a}_i a_i$ for any $i \in \{1, \ldots m\}$. A word is *cyclically reduced* if all of its rotations are freely reduced.

Example 3.2. To simplify notation inside this example, let $\mathcal{L}_2 = \{a, b, \overline{a}, \overline{b}\}$. Form a straight line program

$$\mathbb{A} = \left\langle \begin{array}{l} \mathcal{L}_2, \{A_i, B_i\}_{i=0}^n, A_n, \\ \{A_{k+1} \to B_k, B_{k+1} \to B_k A_k \overline{B}_k\}_{k=1}^{n-1} \cup \{A_0 \to a, B_0 \to b\} \end{array} \right\rangle.$$

Thus

$$w(A_5) = ba\overline{b}bb\overline{a}\overline{b}ba\overline{b}ba\overline{b}\overline{b}b\overline{a}\overline{b}ba\overline{b}bb\overline{a}\overline{b}ba\overline{b}bb\overline{a}\overline{b}b\overline{a}\overline{b}ba\overline{b}\overline{b}b\overline{a}\overline{b}.$$

Note the close relation with $\varphi^5(a)$ where $\varphi\colon F_2 \to F_2$ is the automorphism $a \mapsto b, b \mapsto ba\overline{b}$. Notice also that $w(A_5)$ has both free and cyclic reductions.

Using the generalization of Plandowski's work (given as Theorem 2.9 above), Lohrey [18] has proven:

Theorem 3.3. *There is a polynomial-time algorithm that, given a straight line program $\mathbb{A}$ in normal form, finds a composition system $\mathbb{X}$ where w_X is the free reduction of w_A.*

Proof. Induct on n. Suppose $\mathbb{A} = \langle \mathcal{L}_m, \mathcal{A}, A_n, \mathcal{P} \rangle$ is the given straight line program. Now define a composition system $\mathbb{X} = \langle \mathcal{L}_m, \mathcal{X}, X_n, \mathcal{Q} \rangle$. For every non-terminal A_i of height one in $\mathcal{A}$ place X_i in $\mathcal{X}$ and add the production $X_i \to w(A_i)$ to $\mathcal{Q}$.

Now, if $\text{height}(A_n) = 1$ then there is nothing more to prove. So assume that $\text{height}(A_n) \geq 2$. By induction assume that X_i, for $i < n$, lies in $\mathcal{X}$ and assume that the corresponding production lies in $\mathcal{Q}$. Thus $w(X_i)$ is freely reduced for all $i < n$. Now place X_n in $\mathcal{X}$ and consider what X_n will produce.

Suppose that $A_n \to A_i \cdot A_j$. Build $\overline{X}_i$ using the algorithm of Lemma 3.1. Apply the algorithm of Theorem 2.9 to find the largest k so that $w(\overline{X}_i)[: k] = w(X_j)[: k]$. Add the production

$$X_n \to X_i[: -k] \cdot X_j[k :]$$

to $\mathcal{Q}$. The word $w(X_n)$ is now freely reduced. □

Example 3.4. We continue Example 3.2. Given $\mathbb{A}$ as above Lohrey's algorithm produces the following composition system:

$$\mathbb{X} = \left\langle \begin{array}{l} \mathcal{L}_2, \{X_i, Y_i\}, X_n, \{X_{i+1} \to Y_i, Y_{i+1} \to Y_i[: 2] \cdot \overline{Y}_i\}_{i=1}^{n-1} \cup \\ \cup \{X_1 \to Y_0, Y_1 \to Y_0 X_0 \overline{Y}_0, X_0 \to a, Y_0 \to b\} \end{array} \right\rangle .$$

For example, $w(X_5) = baba\bar{b}\bar{a}\bar{b}$. Deduce that $w(X_{k+2}) = w(Y_{k+1}) = ba \cdot \overline{w(Y_k)}$ for $k \geq 1$.

There is an important corollary of Theorem 3.3:

Theorem 3.5 (Lohrey [18])**.** *The word problem for compressed words in the free group is solvable in polynomial time.* □

We notice two more consequences.

Corollary 3.6. *There is a polynomial-time algorithm that, given a compressed word $\mathbb{A}$, finds a compressed word $\mathbb{X}$ where w_X is cyclic reduction of w_A. The algorithm also gives the compressed conjugating word.*

Proof. Following Theorems 7.1 and 3.3 assume that w_A is freely reduced. Using the algorithm of Lemma 3.1 produce the compressed word $\overline{\mathbb{A}}$. Now apply the generalization of Plandowski's algorithm (Theorem 2.9) to find the largest k so that $w_A[: k] = w_{\overline{A}}[: k]$. It follows that the composition system $A' \to A[: k]$ produces the conjugating word. Also, the composition system $X \to A[k : -k]$ produces the cyclic reduction of w_A, as promised. □

The second consequence is more subtle:

Theorem 3.7. *The conjugacy problem for compressed words in the free group is solvable in polynomial time. The algorithm also computes the compressed conjugating word.*

We only sketch the proof, as the theorem is not used in the sequel.

Proof sketch of Theorem 3.7. Suppose that $\mathbb{A}$ and $\mathbb{X}$ are the given compressed words. Using Corollary 3.6 assume that $\mathbb{A}$ and $\mathbb{X}$ produce cyclically reduced words. Using Lemma 2.6 check that $|w_A|$ and $|w_X|$ are equal.

Let $\mathbb{W}$ be the compressed word with root production $W \to X \cdot X$. That is $w_W = w_X \cdot w_X$. Thus, to prove that w_A and w_X are conjugate it suffices to prove that w_A appears as a subword of w_W. But this is a special case of the *fully compressed pattern matching problem* which can be solved in polynomial time. See, for example, the work of Karpinski, Rytter, and Shinohara [16], of Gasieniec *et al* [11], or of Miyazaki, Shinohara, and Takeda [22]. □

4. Free-by-cyclic groups

For background in group theory the reader should consult Lyndon and Schupp's book [20]. Recall the definition of $\mathrm{Aut}(F_m)$: the group of all automorphisms of the free group F_m. Fix $\Phi \in \mathrm{Aut}(F_m)$. The *free-by-cyclic* group G_Φ is presented by:

$$\langle a_i, t \mid t a_i \bar{t} = \Phi(a_i), i \in \{1, \ldots, m\} \rangle.$$

The goal of this section is to prove:

Theorem 4.1. *The word problem for G_Φ is polynomial time.*

Proof. Let $\mathcal{L}_m = \{a_i, \bar{a}_i\}$ and $\mathcal{M} = \mathcal{L}_m \cup \{t, \bar{t}\}$. Let $\mathcal{A} = \{A_{i,p} \mid i \in \{1, \ldots m\}, p \in \mathbb{N}\}$. Fix $\Phi \in \mathrm{Aut}(F_m)$ by assuming that the words $u_i(a_1, \ldots, a_m) = \Phi(a_i)$ are given as input.

Define production rules as follows:

$$\begin{aligned} A_{i,0} &\to a_i \\ A_{i,p} &\to u_i(A_{1,p-1}, \ldots, A_{m,p-1}), \quad p \geq 1 \end{aligned}$$

Suppose now that W is a word in $\mathcal{M}^*$. The length of W determines the size of the given word problem. Now rewrite W in stages: First freely reduce. Next replace every a_i and $\bar{a}_i$ appearing by $A_{i,0}$ and by

$\overline{A}_{i,0}$, respectively. Now move all occurrences of t to the right, and of $\overline{t}$ to the left, rewriting as follows:

$$\begin{array}{rcl} t \cdot A_{i,p} & \rightsquigarrow & A_{i,p+1} \cdot t \\ t \cdot \overline{A}_{i,p} & \rightsquigarrow & \overline{A}_{i,p+1} \cdot t \\ A_{i,p} \cdot \overline{t} & \rightsquigarrow & \overline{t} \cdot A_{i,p+1} \\ \overline{A}_{i,p} \cdot \overline{t} & \rightsquigarrow & \overline{t} \cdot \overline{A}_{i,p+1} \\ t \cdot \overline{t} & \rightsquigarrow & \epsilon \end{array}$$

The result is a word in $\{A_{i,p}, \overline{A}_{i,p}\}^*$, possibly with powers t^k and $\overline{t}^l$ appearing at the end and beginning. Let $W'(A_{i,p})$ be this word, omitting the leading and trailing powers of t. Construct a straight line program with a root non-terminal $A \to W'(A_{i,p})$. Notice that W is trivial in G_Φ if and only if the word w_A freely reduces in F_m and the powers satisfy $k = l$. (This is a simple form of Britton's Lemma. See page 181 of [20].)

However the latter occurs if and only if the total exponent of t in W is zero. The former is exactly solved by applying Lohrey's Algorithm (Theorem 3.3). □

Remark 4.2. The statement of Theorem 4.1 may be generalized to *ascending* HNN extensions: Φ is assumed to be an injection instead of an automorphism. The proof is identical. The question for HNN extensions in general appears to be more delicate.

5. The automorphism group of a free group

We now examine the automorphism group in greater detail. Recall that the automorphism group $\mathrm{Aut}(F_m)$ is finitely generated by the *Nielsen generators* (see Chapter 1.4 of [20]):

(1) $\alpha_i \in \mathrm{Aut}(F_m)$ so that $\alpha_i|\mathcal{L}_m$ interchanges a_i and $\overline{a}_i$, fixing all other elements of $\mathcal{L}_m$.
(2) $\beta_{ij} \in \mathrm{Aut}(F_m)$, with $i \neq j$, has $\beta_{ij}(a_i) = a_i a_j$, and $\beta_{ij}(a_k) = a_k$ for all $k \neq i$.

Remark 5.1. Choosing a different generating set alters running times by at most a multiplicative constant. The choice above simplifies the proof below.

Theorem 5.2. *The word problem for* $\mathrm{Aut}(F_m)$ *is polynomial time.*

Proof. Suppose that $\Phi = \varphi_1 \dots \varphi_n$ is a word in the Nielsen generators. It suffices to check that $\Phi(a_i)$ freely reduces to a_i, for all i.

To do this, define a straight line program: Let $\{A_{i,p}\}$ be the set of non-terminals with $i \in \{1, \ldots m\}$ and $p \in \{0, 1, \ldots n\}$. Create the following production rules:

$$\begin{aligned} A_{i,0} &\to a_i \\ A_{i,p} &\to \varphi_p(A_{i,p-1}), \quad p \geq 1 \end{aligned}$$

If $\varphi_p = \alpha_i$ is of the first kind then $\varphi_p(A_{i,p})$ equals $\overline{A}_{i,p-1}$ while $\varphi_p(A_{j,p})$ equals $A_{j,p-1}$, for $j \neq i$. If $\varphi_p = \beta_{ij}$ is of the second kind then $\varphi_p(A_{i,p})$ equals $A_{i,p-1} \cdot A_{j,p-1}$, and so on.

Now apply the algorithm of Theorem 3.3 to rewrite this straight line program so that all outputs are freely reduced. If the resulting composition system has $|w(A_{i,n})| \geq 2$ for any i then the automorphism Φ is nontrivial. If $|w(A_{i,n})| = 1$ for all i then, using the algorithm of Lemma 2.7, check if $w(A_{i,n}) = a_i$. If this is the case for all i then Φ is the identity element of $\mathrm{Aut}(F_m)$. □

Remark 5.3. In our analysis of the word problem for free-by-cyclic groups we could accept as input both the word W in $\mathcal{M}^*$ and automorphism Φ given as a word in the Nielsen generators. Now we need not precompute the words u_i. Instead we find, as in the proof of Theorem 5.2, straight line programs producing these words. The running time of Theorem 4.1 then becomes polynomial in the two inputs W and Φ.

Note that solving the word problem for a group also solves the word problem for subgroups. Of course, there are many beautiful subgroups of $\mathrm{Aut}(F_m)$. As just a single example consider the *braid group*, B_m: Let $\mathbb{D}_m$ be a disk with m points removed from the interior. Then B_m is the group of homeomorphisms of $\mathbb{D}_m$ that fix the boundary pointwise, modulo boundary and puncture fixing isotopies. Choosing a basepoint on the boundary makes B_n act on $\pi_1(\mathbb{D}_m) \cong F_m$ and so embeds B_n into $\mathrm{Aut}(F_m)$. A simple corollary to Theorem 5.2 is the well-known:

Corollary 5.4 ([4], [17], [10], [5], [13])**.** *The word problem for B_m is polynomial time.* □

6. Membership problems

We now turn our attention to *membership problems*, also called *generalized word problems*. Suppose that H is a subgroup of a finitely presented group G. We seek an algorithm that, given a word W written in the generators of G, decides if W represents an element of H. Note that if H is normal in G then such an algorithm also solves the word problem in the quotient G/H.

Recall that $\mathrm{Inn}(F_m)$ is the normal subgroup of $\mathrm{Aut}(F_m)$ that contains Φ_U for all words $U \in \mathcal{L}_m^*$: $\Phi_U(V) = UV\overline{U}$. The quotient is the *outer automorphism* group, $\mathrm{Out}(F_m)$.

Theorem 6.1. *The word problem for* $\mathrm{Out}(F_m)$ *is polynomial time.*

Proof. To see this, note that the membership problem for $\mathrm{Inn}(F_m)$, inside of $\mathrm{Aut}(F_n)$, is solved by the construction given in the proof of Theorem 5.2 and by Corollary 3.6. □

Another interesting membership problem is that of the braid group (or more generally, *mapping class groups* of punctured surfaces) inside of $\mathrm{Aut}(F_m)$. In order to avoid multiplying examples we only discuss the problem for the braid group: it suffices to check that the boundary word is fixed and all punctures are preserved, up to conjugacy. (See [21, Theorem N6], for example.) Corollary 3.6 can check the latter and the former is dealt with by Lohrey's Algorithm (Theorem 3.3).

Here is another kind of membership problem, proposed by Nathan Broaddus: Let $\mathcal{MCG}(S)$ denote the mapping class group of the closed connected orientable genus g surface, S. Consider a handlebody V with $\partial V = S$. Then $\mathcal{MCG}(V)$ naturally includes in $\mathcal{MCG}(S)$.

Remark 6.2. Recall the fundamental fact that $\Phi \in \mathcal{MCG}(S)$ lies in the subgroup $\mathcal{MCG}(V)$ if and only if Φ preserves the set of *meridians*: the set of curves in S that bound disks in V. In fact a "weaker" condition is equivalent: let $\mathbb{D}$ be a collection of g disjoint disks in V so that $V \setminus \mathbb{D}$ is a three-ball. Then $\Phi \in \mathcal{MCG}(V)$ if and only if the curves $\Phi(\partial\mathbb{D})$ bound disks in V.

Fix a point $x \in S$ and let $\{a_1, \ldots, a_g, b_1, \ldots, b_g\}$ be the standard set of generators of $\pi_1(S, x)$. We arrange matters so that all of the b_i are meridians. See Figure 1.

Choose also a standard set $\{\tau\}$ of Dehn twist generators for $\mathcal{MCG}(S)$. In fact we will over-specify these twists: for each Dehn twist in the generating set choose a twisting curve $\{\alpha\}$ that avoids a small neighborhood of the basepoint $x \in S$. See Figure 2.

Pick finally a point y in this small neighborhood of x, avoiding all the loops a_i and b_i. It follows that $\pi_1(S \setminus \{y\}, x) \cong F_{2g}$ is freely generated by the a_i and b_i loops. Also, the Dehn twists give free group automorphisms. (See [21, Theorem N10].) Broaddus tells us the useful:

Lemma 6.3. *Fix* $W \in \{a_i, b_i, \overline{a}_i, \overline{b}_i\}^*$ *so that* W *is homotopic in* S *to a simple closed curve* ω. *The following two conditions are equivalent:*

- ω *is a meridian.*
- *After deleting all* b_i*'s and* $\overline{b}_i$*'s from* W *the resulting word in* $\{a_i, \overline{a}_i\}^*$ *freely reduces to the empty word.*

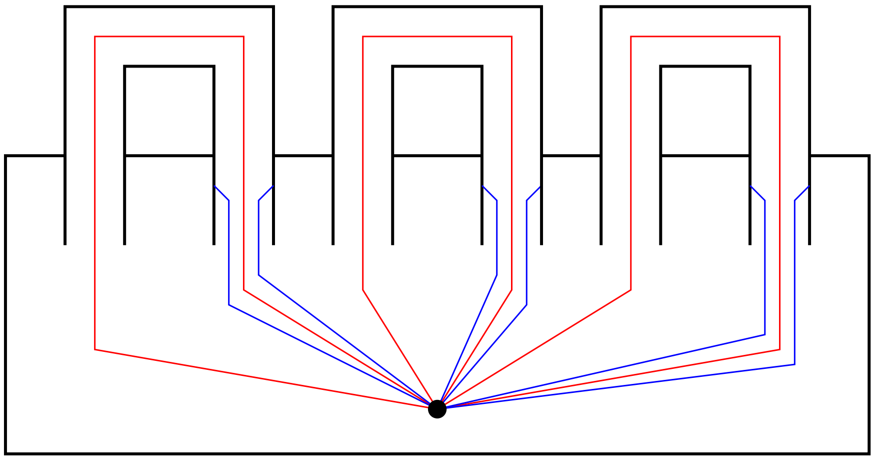

FIGURE 1. Generators for the fundamental group of S.

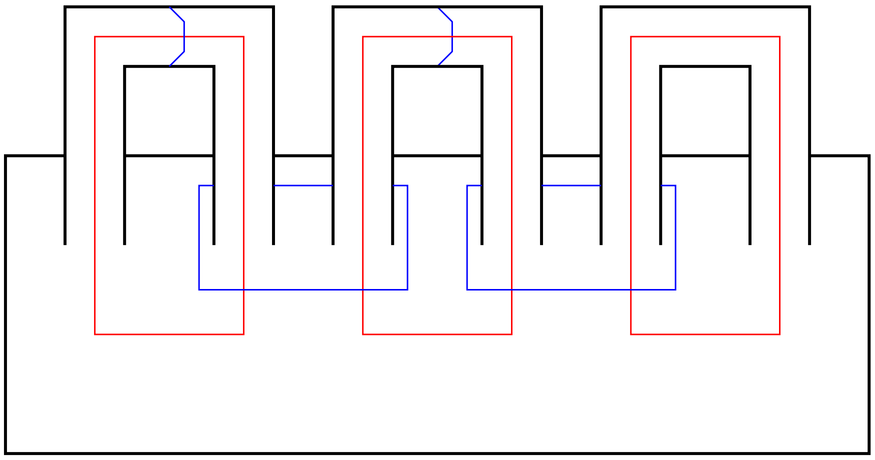

FIGURE 2. Generators for the mapping class group.

Proof. Consider the map $\pi_1(S, x) \to \pi_1(V, x) \cong F_g$ induced by inclusion. As in Figure 1 all of the b_i lie in the kernel while the images of the a_i freely generate $\pi_1(V)$. Accordingly, identify $\pi_1(V)$ and the free group $\langle a_i | \rangle$.

Suppose now that the first condition holds: ω is a meridian. Then W lies in the kernel. Since the b_i normally generate the kernel the second condition follows.

Suppose instead that the second condition holds. Since W is in the kernel, deduce that ω bounds a singular disk in V. By the celebrated Loop Theorem [14] ω is a meridian. $\square$

We are now prepared to prove:

Theorem 6.4. *The membership problem for $\mathcal{MCG}(V)$ in $\mathcal{MCG}(S)$ is polynomial time.*

Proof. As is well-known [15] the $2g+1$ Dehn twists shown in Figure 2 generate the mapping class group, $\mathcal{MCG}(S)$. So fix a word $\Phi = \tau_1 \ldots \tau_n$, written in terms of these twists and their inverses. By Remark 6.2 it is enough to check that $\Phi(b_i)$, thought of as a word in the free group generated by the a_i and the b_i, satisfies the second condition of Lemma 6.3. This can be done directly, but $\Phi(b_i)$ might have length exponential in n.

Instead, for each i encode $\Phi(b_i)$ as a compressed word, say $\mathbb{A}(i)$. It is a triviality to remove all b_i and $\bar{b}_i$ appearing in $w(A(i))$: for every non-terminal B with $B \to b_i$ replace the production by $B \to \epsilon$. Call the new compressed word $\mathbb{A}'(i)$.

Now run Lohrey's Algorithm (Theorem 3.3) on $\mathbb{A}'(i)$. The mapping class Φ lies in $\mathcal{MCG}(V)$ if and only if $w(A'(i))$ freely reduces to the empty string, for all i. $\square$

Remark 6.5. Suppose that W is another handlebody of genus g with $\partial W = S$ so that $V \cup_S W$ is the three-sphere. Here the a_i loops bound disks in W. Define the *Goeritz group* $\mathcal{G}$ to be the intersection $\mathcal{MCG}(V) \cap \mathcal{MCG}(W)$, thought of as subgroups of $\mathcal{MCG}(S)$. (See [26], [2], and [8] for discussion of the Goeritz group in genus two.) Applying Theorem 6.4 twice gives a polynomial-time algorithm for the membership problem of $\mathcal{G}$ in $\mathcal{MCG}(S)$.

7. Hagenah's Algorithm

We now discuss the computer science underpinnings of the discussion above. To begin, in Hagenah's thesis [12, Chapter 8] we find:

Theorem 7.1. *There is a polynomial-time algorithm that, given a composition system $\mathbb{A}$, finds a straight line program $\mathbb{X}$ with $w_X = w_A$.*

The exposition of this result in [12] is wonderfully clear. I present a proof only to make this paper self-contained.

Proof of Theorem 7.1. Fixing notation, suppose that $\mathbb{A} = \langle \mathcal{L}, \mathcal{A}, A, \mathcal{P} \rangle$. Note that all productions in $\mathcal{P}$ are either of the form $B \to C[i:j] \cdot D[k:l]$ or of the form $B \to a[i:j]$.

Build the straight line program $\mathbb{X} = \langle \mathcal{L}, \mathcal{X}, X, \mathcal{Q} \rangle$ from the bottom up. The set $\mathcal{X}$ will contain *plain* non-terminals, one for each non-terminal of $\mathcal{A}$, and *decorated* non-terminals, each associated to some plain non-terminal. Proceed as follows: for every non-terminal $B \in \mathcal{A}$

of height one we add a plain non-terminal Y to $\mathcal{X}$. Suppose B produces $a[i : j]$. Then

- if $j = i + 1$ add $Y \to a$ to $\mathcal{Q}$ and
- if $j = i$ add $Y \to \epsilon$ to $\mathcal{Q}$.

Let A be the root non-terminal of $\mathcal{A}$. Assume via induction that for every other non-terminal $B \in \mathcal{A}$ a plain non-terminal Y has been added to $\mathcal{X}$, so that $w_Y = w_B$. We now describe the decorated non-terminals that may also, by induction, appear in $\mathcal{X}$. Fix any plain non-terminal Y in $\mathcal{X}$. Then $Y^{[i:j]}$ is a *decorated* non-terminal. There are various cases:

- If $0 < i < j < |w_Y|$ then $Y^{[i:j]}$ is a *subword* non-terminal.
- If $0 < j < |w_Y|$ then $Y^{[:j]}$ is a *prefix* non-terminal.
- If $0 < i < |w_Y|$ then $Y^{[i:]}$ is a *suffix* non-terminal.
- $Y^{[:]} = Y$ is the plain non-terminal.
- $Y^{[i:i]} = \epsilon$ is the empty word.

Repeated decoration behaves as expected: $(Y^{[i:j]})^{[l:k]} = Y^{[i+k:i+l]}$. The production rules for decorated non-terminals are given below.

Suppose now that the root has production $A \to B[i : j] \cdot C[k : l]$. Suppose that Y and Z are plain non-terminals in $\mathcal{X}$ corresponding to the non-terminals B and C. Add a plain non-terminal X to $\mathcal{X}$ corresponding to A. Add the non-terminals $Y^{[i:j]}$ and $Z^{[k:l]}$ to $\mathcal{X}$. Add the production rule $X \to Y^{[i:j]} \cdot Z^{[k:l]}$ to $\mathcal{Q}$.

Production rules are needed for every new decorated non-terminal, $Y^{[i:j]}$, created by the addition of the plain X. Suppose that the plain non-terminal Y produces $U \cdot V$. (Here it is possible that U and V are themselves decorated non-terminals.) There are several cases and subcases: Suppose first that $Y^{[i:j]}$ is a subword non-terminal.

Subcase	Add to $\mathcal{X}$	Add to $\mathcal{Q}$
$\lvert w_U\rvert \le i$	$V^{[i-\lvert w_U\rvert:j-\lvert w_U\rvert]}$	$Y^{[i:j]} \to V^{[i-\lvert w_U\rvert:j-\lvert w_U\rvert]}$
$i < \lvert w_U\rvert < j$	$U^{[i:]}, V^{[:j-\lvert w_U\rvert]}$	$Y^{[i:j]} \to U^{[i:]} \cdot V^{[:j-\lvert w_U\rvert]}$
$j \le \lvert w_U\rvert$	$U^{[i:j]}$	$Y^{[i:j]} \to U^{[i:j]}$

Suppose now that $Y^{[:j]}$ is a prefix non-terminal.

Subcase	Add to $\mathcal{X}$	Add to $\mathcal{Q}$
$\lvert w_U\rvert < j$	$V^{[:j-\lvert w_U\rvert]}$	$Y^{[:j]} \to U \cdot V^{[:j-\lvert w_U\rvert]}$
$j \le \lvert w_U\rvert$	$U^{[:j]}$	$Y^{[:j]} \to U^{[:j]}$

Suppose now that $Y^{[i:]}$ is a suffix non-terminal.

Subcase	Add to $\mathcal{X}$	Add to $\mathcal{Q}$
$\lvert w_U \rvert \le i$	$V^{[i-\lvert w_U \rvert:]}$	$Y^{[i:]} \to V^{[i-\lvert w_U \rvert:]}$
$i < \lvert w_U \rvert$	$U^{[i:]}$	$Y^{[i:]} \to U^{[i:]} \cdot V$

Notice that creating the plain non-terminal X causes at most two decorated non-terminals to be created, both of lesser height. Every subword non-terminal in turn creates at most one subword non-terminal or at most one prefix and at most one suffix non-terminal. Again, these have lesser height. Finally, any prefix (suffix) non-terminal causes at most one prefix (suffix) non-terminal to be created. As usual, the height decreases.

Suppose that $n = ||A||$. It follows that the creation of the plain non-terminal X adds at most $1 + 2(2n) = 1 + 4n$ new decorated non-terminals to $\mathcal{X}$. Thus the total number of non-terminals in $\mathcal{X}$, at the end of the construction, is

$$n + \sum_{B \in \mathcal{A}} (1 + 4||B||) \le n + \sum_{i=1}^{n} (1 + 4i) = 2n^2 + 4n.$$

This completes both the description of the algorithm and its proof of correctness. $\square$

8. Plandowski's Algorithm

The final piece of the puzzle is:

Theorem 8.1 (Plandowski [24])**.** *There is a polynomial-time algorithm that, given straight line programs $\mathbb{A}$ and $\mathbb{X}$ in normal form, decides whether or not $w_A = w_X$.*

A proof, essentially following [24], is provided for the convenience of the reader.

Proof of Theorem 8.1. Let $\mathbb{A} = \langle \mathcal{L}, \mathcal{A}, A, \mathcal{P} \rangle$ and $\mathbb{X} = \langle \mathcal{L}, \mathcal{X}, X, \mathcal{Q} \rangle$. Note that we assume, as we may, that $\mathbb{A}$ and $\mathbb{X}$ have the same terminal alphabet. Making a copy of $\mathbb{X}$ if necessary, assume that $\mathcal{A} \cap \mathcal{X} = \emptyset$. Finally assume that $|w_A| = |w_X|$ and $|\mathcal{A}| = m \ge n = |\mathcal{X}|$.

We begin with the following definition: a triple (B, Y, i) is an *assertion* if:

- $B \in \mathcal{A} \cup \mathcal{L}$ and $Y \in \mathcal{X} \cup \mathcal{L}$.
- $0 \le i < |w_B|$.

If $0 \le i$ and $|w_B| \le i + |w_Y|$ then (B, Y, i) is a *overlap* assertion. If $0 < i$ and $i + |w_Y| < |w_B|$, and then (B, Y, i) is a *subword* assertion. Assertions of the form (Y, B, i), are defined similarly. We do not allow a pair of non-terminals from the same program to appear in a single assertion.

An overlap assertion (B, Y, i) is *satisfied* if and only if $w_B[i :] = w_Y[: |w_B| - i]$. Likewise, a subword assertion is satisfied if and only if $w_B[i : i + |w_Y|] = w_Y$. As a bit of terminology a set of assertions, Γ, is satisfied if and only if every assertion $\gamma \in \Gamma$ is. In point of fact, the algorithm checks satisfiability of (B, Y, i) when and only when both B and Y are terminal characters and $i = 0$.

Our algorithm transforms a set of assertions Γ_k into another such set, Γ_{k+1}. Beginning with $\Gamma_0 = \{(A, X, 0)\}$ the following properties are maintained:

(a) Γ_{k+1} is satisfied if and only if Γ_k is satisfied.
(b) At most $m + n - k$ elements of $\mathcal{A} \cup \mathcal{X}$ are mentioned in Γ_k.
(c) For all k, $|\Gamma_k|$ is bounded by $(k + 1)4mn(m + n)$.

There are two ways to produce a new assertions from old, *splitting* and *compacting.*

Splitting. Fix Γ, a set of assertions. Suppose that, of all non-terminals from $\mathcal{A}$ and $\mathcal{X}$ appearing in Γ, the non-terminal $B \in \mathcal{A}$ has maximal length. Fix $\gamma \in \Gamma$. We must define $\text{Split}(\gamma, B)$ and then $\text{Split}(\Gamma, B)$.

There are several cases to consider. If B does not appear in γ then $\text{Split}(\gamma, B) = \{\gamma\}$. Now suppose that $\gamma = (B, Y, i)$ or $\gamma = (Y, B, i)$. Note that γ is either an overlap or subword assertion and that we have assumed $|w_B| \geq |w_Y|$. Suppose that $B \to C \cdot D$. Now consider subcases. If $\gamma = (B, Y, i)$ is an overlap assertion then:

Subcase	$\text{Split}(\gamma, B)$	Type
$i < \|w_C\|$	(C, Y, i)	overlap
	$(Y, D, \|w_c\| - i)$	either
$\|w_C\| \leq i$	$(D, Y, i - \|w_c\|)$	overlap

The table should be read as follows: When $i < |w_C|$ then $\text{Split}(\gamma, B)$ contains two assertions: either a pair of overlaps or one of each type. When $|w_C| \leq i$ the set $\text{Split}(\gamma, B)$ contains a single overlap assertion. Suppose now that $\gamma = (B, Y, i)$ is a subword assertion:

Subcase	Split(γ, B)	Type
$i + \lvert w_Y\rvert < \lvert w_C\rvert$	(C, Y, i)	subword
$i + \lvert w_Y\rvert = \lvert w_C\rvert$	(C, Y, i)	overlap
$i < \lvert w_C\rvert < i + \lvert w_Y\rvert$	(C, Y, i)	overlap
	$(Y, D, \lvert w_c\rvert - i)$	overlap
$i = \lvert w_C\rvert$	$(Y, D, 0)$	overlap
$\lvert w_C\rvert < i$	$(D, Y, i - \lvert w_C\rvert)$	subword

Suppose now that $\gamma = (Y, B, i)$ is an overlap assertion. As usual assume that $|w_B| \geq |w_Y|$:

Subcase	Split(γ, B)	Type
$\lvert w_Y\rvert \leq i + \lvert w_c\rvert$	(Y, C, i)	overlap
$i + \lvert w_C\rvert < \lvert w_Y\rvert$	(Y, C, i)	subword
	$(Y, D, i + \lvert w_c\rvert)$	overlap

Finally (Y, B, i) cannot be a subword assertion because $|w_B| \geq |w_Y|$. This finishes the definition of Split(γ, B). Define

$$\text{Split}(\Gamma, B) = \cup_{\gamma \in \Gamma} \text{Split}(\gamma, B).$$

Immediate from the definitions is:

Claim 8.2. A set of assertions Γ is satisfied if and only if Split(Γ, B) is satisfied. $\square$

Define now $o(\Gamma)$ to be the number of overlap assertions in Γ. Similarly we take $s(\Gamma)$ to be the number of subword assertions of Γ. So $|\Gamma| = o(\Gamma) + s(\Gamma)$. From the tables above deduce:

Claim 8.3. Suppose that Γ is a set of assertions. Then

$$\begin{aligned} o(\text{Split}(\Gamma, P)) &\leq o(\Gamma) + 2s(\Gamma) \\ s(\text{Split}(\Gamma, P)) &\leq o(\Gamma) + s(\Gamma) \end{aligned}$$

when $P \in \mathcal{A} \cup \mathcal{X}$ is a non-terminal of maximal length in Γ. $\square$

Compact. Now for the definition of Compact(Γ). Note that, if u is a word, then $p \in \mathbb{N}$ is a *period* of u if

- $1 \leq p \leq |u| - 1$ and
- $u[i] = u[i + p]$ for all $0 \leq i \leq |u| - 1 - p$.

An immediate consequence of the definition is:

Claim 8.4. Suppose that $\gamma = (B, Y, i)$ and $\gamma' = (B, Y, j)$ are overlap assertions with $i < j$. Then γ and γ' are satisfied if and only if γ is satisfied and $j - i$ is a period of the word $w_B[i :]$. $\square$

We now give a restricted version of the famous Periodicity Lemma [19]:

Lemma 8.5. *If p and q are periods of u, where $p + q \leq |u|$, then $\gcd(p, q)$ is also a period of u.* □

The following claim is the engine in the proof of correctness of Plandowski's Algorithm:

Claim 8.6. Suppose that $\gamma = (B, Y, i)$, $\gamma' = (B, Y, j)$, and $\gamma'' = (B, Y, k)$ are overlap assertions with $i < j < k$ and $j - i + k - i \leq |w_B| - i$. Then γ, γ', and γ'' are satisfied if and only if γ is satisfied and $\gcd(j - i, k - i)$ is a period of $w_B[i:]$.

Proof. This follows from two applications of Claim 8.4 and from the Periodicity Lemma 8.5. □

Equivalently γ, γ' and γ'' are satisfied if and only if γ and $\delta = (B, Y, i + \gcd(j - i, k - i))$ are satisfied. This leads directly to the definition of SimpleCompact: given $\{\gamma, \gamma', \gamma''\}$ as in the hypothesis of Claim 8.6 define SimpleCompact$(\{\gamma, \gamma', \gamma''\}) = \{\gamma, \delta\}$, with δ as above.

Now, for any set of assertions Γ define Compact(Γ) as follows: Order the sets $\mathcal{A}$ and $\mathcal{X}$ of non-terminals. Let $\Gamma^0 = \Gamma$. For all j we may assume that the set Γ^j is ordered lexicographically. Scan through Γ^j to find the first consecutive triple of assertions that satisfies the requirements of Claim 8.6. Apply SimpleCompact to this triple to obtain Γ^{j+1} and notice that $|\Gamma^{j+1}| = |\Gamma^j| - 1$. Finally, if no such triples exist then set Compact$(\Gamma) = \Gamma^j$.

It follows that a single Compact operation involves calling the subroutine SimpleCompact at most $O(|\Gamma|)$ times. We also record the fact:

Claim 8.7. Γ is satisfied if and only if Compact(Γ) is satisfied. □

Now to define Γ_{k+1} in terms of Γ_k. Suppose that $P \in \mathcal{A} \cup \mathcal{X}$ is a non-terminal appearing in Γ_k maximizing the length of $|w_P|$. Then take

$$\Gamma_{k+1} = \text{Compact}(\text{Split}(\Gamma_k, P)).$$

Note that property (a) above is guaranteed by Claims 8.2 and 8.7 while property (b) is provided by the fact that every non-terminal is split for at most one value of k. We must now bound the size of Γ_k.

Fix attention on any pair of non-terminals $B \in \mathcal{A}$ and $Y \in \mathcal{X}$. Let $\{(B, Y, i_j)\}_{j=1}^N$ be the overlap assertions of Γ_k that mention B and Y in that order and indexed so that $i_j < i_{j+1}$. As Γ_k is compact it follows from Claim 8.6 that

$$|w_B| - i_j < i_{j+2} - i_j + i_{j+1} - i_j.$$

Since $i_{j+1} < i_{j+2}$ it follows that

$$\frac{1}{2}(|w_B| - i_j) < i_{j+2} - i_j.$$

Deduce that $N \leq 2\log_2(|w_B|) + 1 \leq 2||B|| + 1$, with the last inequality following from Lemma 2.5.

Claim 8.8. With Γ_{k+1} as given:

$$\begin{aligned} o(\Gamma_{k+1}) &\leq mn(2m+1) + nm(2n+1) \leq 4mn(m+n) \\ s(\Gamma_{k+1}) &\leq 4mn(m+n) + s(\Gamma_k) \leq k(4mn(m+n)). \end{aligned}$$

This verifies property (c) above. □

This completes both the description of the algorithm and its proof of correctness. □

Appendix A. On surfaces

The discussion above gives a satisfactory picture of the behavior of compressed words in the free group. One immediately asks for a similar treatment of hyperbolic groups in general. However the situation there appears to require a new idea.

Instead we briefly describe *well-tempered paths*: a beautiful geodesic language for closed surface groups. Our discussion is meant to be more inspiring than exhaustive: many details are omitted. For simplicity, we restrict ourselves to the closed, orientable, connected genus two surface.

Let D be the regular decagon in the hyperbolic plane with angles $2\pi/5$. Label the boundary of D with the word $abcdeabcde$, read counter-clockwise. The first five edges are oriented counter-clockwise while the last five are oriented clockwise. Let $\mathcal{L} = \{a, b, c, d, e, \overline{a}, \overline{b}, \overline{c}, \overline{d}, \overline{e}\}$. See Figure 3.

The decagon and its labelling extends to a tiling $\mathcal{D}$ of the hyperbolic plane. Notice that a path in the one-skeleton determines a word in $\mathcal{L}^*$. Let $\mathcal{M} \subset \mathcal{L}^*$ be the subset that can be realized in this way. Conversely, a word of $\mathcal{M}$ determines a unique path in the tiling, up to the action of $\pi_1(S)$. When it cannot cause confusion we treat paths and words interchangeably.

The following words of length five are called *bad turns*:

$$edcba,\ \overline{b}\overline{a}edc,\ \overline{d}\overline{c}\overline{b}\overline{a}e,\ a\overline{e}\overline{d}\overline{c}\overline{b},\ cba\overline{e}\overline{d},$$

$$\overline{a}\overline{b}\overline{c}\overline{d}\overline{e},\ \overline{c}\overline{d}\overline{e}ab,\ \overline{e}abcd,\ bcde\overline{a},\ de\overline{a}\overline{b}\overline{c}.$$

A path is *well-tempered* if the corresponding word is freely reduced and contains no bad turn. The intent, and hence the name, is that these

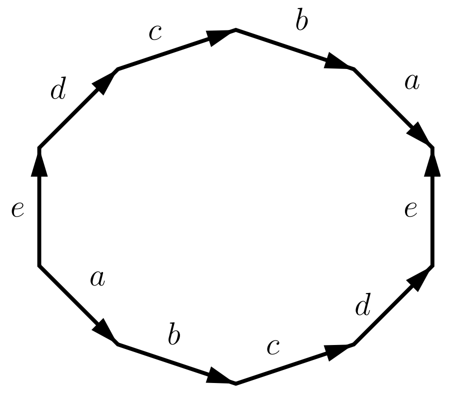

FIGURE 3. A labelled regular hyperbolic decagon.

paths want to "turn right" as often as they "turn left." This is possible because 10/2 is odd. See Figure 4 for a picture of the good turns.

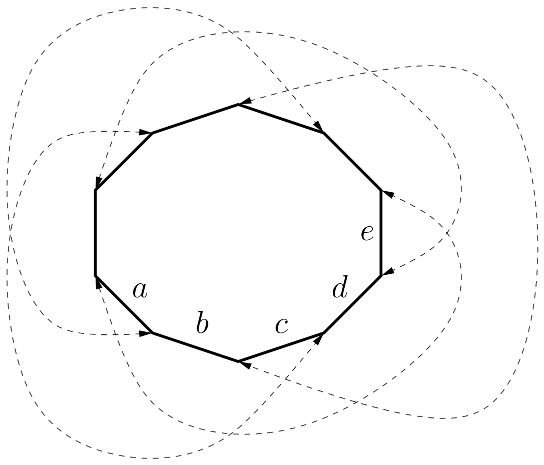

FIGURE 4. All of the good turns are shown.

Notice that well-tempered paths are well-behaved:

Theorem A.1. *For any ordered pair of vertices in the tiling $\mathcal{D}$ there is a unique well-tempered path connecting one to the other. Well-tempered paths are geodesic. Subpaths are again well-tempered as are inverses. Finally, well-tempered paths are* locally detectable*: to verify the property it suffices to check all subpaths of length five.*

Remark A.2. Well-tempered words are similar to *short-lex* words in hyperbolic groups (see [10]). For example, both form regular geodesic languages which satisfy uniqueness and which are closed under taking subwords. However, short-lex seems satisfy the additional properties of local detection and inverse closed only when the group is free.

Here is a sketch of the proof of Theorem A.1. Suppose that α and β are paths and the final vertex of α is the initial vertex of β. Define $\alpha \cdot \beta$ to be their concatenation. If α and β are well-tempered and $\alpha \cdot \beta$ is not then the free reduction (or bad turns) must overlap the point of concatenation.

We may straighten $\alpha{\cdot}\beta$ (rel endpoints) until it becomes well-tempered. There are four stages:

(1) Free reduction.
(2) Sweeping across two sides of at most one corridor.
(3) Sweeping across at most three pieces.
(4) Sweeping across one side of at most two corridors.

Before elaborating on these we briefly give definitions: A *piece* is a path δ of length two to ten where all edges of δ are on the boundary of a single decagon of the tiling. Now, fix a decagon $D = D_0$ and suppose that D_i is the image of D_0 under the i^{th} power of a fixed side pairing transformation of D_0. The union $C = \cup_{i=0}^{k-1} D_i$ is called a *corridor*. The two edges in ∂C corresponding to the transformation are the *ends* of C. The other two components of ∂C are the *sides* of C. Note that sides of corridors always have period four. See Figure 5.

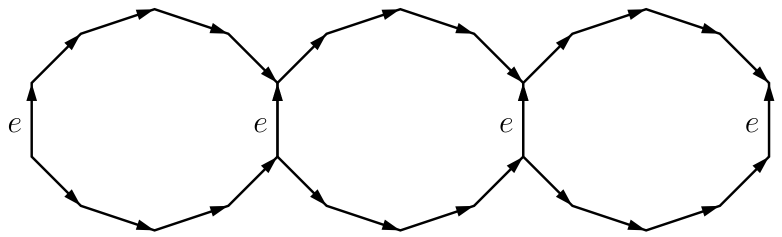

Figure 5. An e-corridor of length three.

Suppose again that α and β are well-tempered and $\gamma_0 = \alpha \cdot \beta$ is the concatenation. We can now flesh out the stages required to make γ_0 well-tempered, assuming it is not already. First freely reduce, if possible, to produce γ_1. If γ_1 contains two sides and one end of a corridor C then sweep γ_1 across C to obtain γ_2. This deals with all pieces of length nine and ten.

Next sweep γ_2 across at most three pieces of lengths between five and eight to form γ_3. The pieces of length five are necessarily bad turns. The proof that there are at most three such is a lengthy but straight-forward combinatorial argument.

If γ_3 contains an end and a side of a corridor C, forming a bad turn, then sweep γ_3 across C. This occurs at most twice. Call the resulting curve, which must be well-tempered, γ. These four stages simply move $\alpha \cdot \beta$ through the thin triangle bounded by α, β, and γ. See Figure 6.

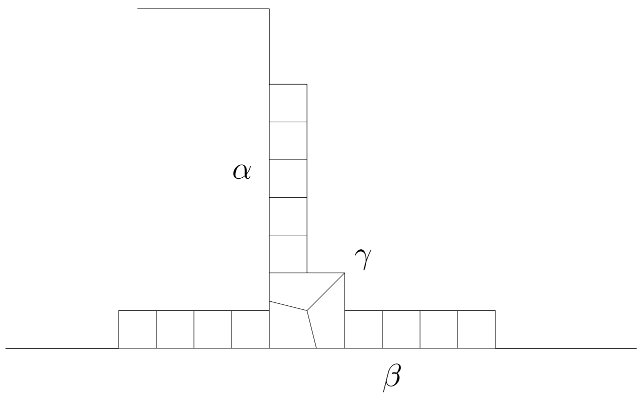

FIGURE 6. A cartoon of a thin triangle in the decagon tiling.

Recall that $\mathcal{M}$ is the set of words corresponding to paths in the one-skeleton of the tiling $\mathcal{D}$. For any compressed word $\mathbb{A}$ in $\mathcal{M}$ let γ_A be the corresponding path. Again γ_A is only defined up to the action of $\pi_1(S)$. The above discussion leads both to a proof of Theorem A.1 and of:

Theorem A.3. *There is a polynomial-time algorithm that, given a compressed word $\mathbb{A}$ in $\mathcal{M}$, finds a compressed word $\mathbb{X}$ where*

- *the path γ_X is well-tempered and*
- *γ_A and γ_X are homotopic rel endpoints.*

Proof sketch. Suppose that the root of $\mathbb{A}$ has production $A \to B \cdot C$. By induction assume that γ_B and γ_C are well-tempered. The first stage is a straight-forward application of Plandowski's Algorithm (Theorem 2.9). The second and fourth stages require both Plandowski's Algorithm and the fact that words of period four are highly compressible. To deal with the third simply examine a constant sized suffix of $w(B'')$ and a constant sized prefix of $w(C'')$, where B'' and C'' are the compressed words output by the second stage. $\square$

Remark A.4. There is a subtlety hidden in this proof sketch – the compressed word $\mathbb{X}$ produced may have bit-size larger than that of $\mathbb{A}$. Since the proof is inductive the growth must be carefully controlled, in part using Hagenah's Algorithm (Theorem 7.1)

Remark A.5. It may be possible to prove versions of Theorems A.1 and A.3 using short-lex paths. However the number of combinatorial possibilities appears to greatly increase. Also, I do not know how to control the growth in size indicated in Remark A.4 when using short-lex paths. If this could be done then the entire discussion should apply to general word hyperbolic groups.

From Theorem A.3 deduce:

Corollary A.6. *The word problem for compressed words in $\pi_1(S)$ is solvable in polynomial time. This gives a solution to the word problem in* $\mathrm{Aut}(\pi_1(S))$. $\square$

The compressed conjugacy problem follows from a careful reading of Epstein and Holt's paper [9]. In a few places their subroutines, acting on words, must be altered to act on compressed words. In particular a solution to the fully compressed matching problem (see the proof of Theorem 3.7) replaces the Knuth-Morris-Pratt algorithm for checking if two words are cyclic conjugates and for computing roots. Also, as noted above, the language of well-tempered paths is regular; this is used in their proof to find certain bounds. Thus:

Theorem A.7. *The conjugacy problem for compressed words in $\pi_1(S)$ is solvable in polynomial time.* $\square$

Since $\pi_1(S)$ has no torsion the simple version of Bridson and Howie's algorithm [6], adapted to compressed words, now solves the membership problem for the inner automorphism group $\mathrm{Inn}(\pi_1(S))$. Finally, recall Nielsen's Theorem (see [21, page 175]): the mapping class group $\mathcal{MCG}(S)$ is isomorphic to the outer automorphism group of $\pi_1(S)$. So, similar to the proof of Theorem 6.1, well-tempered paths give:

Theorem A.8 ([23], [13])**.** *The word problem in $\mathcal{MCG}(S)$ is solvable in polynomial time.* $\square$

References

[1] Ian Agol, Joel Hass, and William Thurston. 3-manifold knot genus is NP-complete. In *Proceedings of the Thirty-Fourth Annual ACM Symposium on Theory of Computing*, pages 761–766 (electronic), New York, 2002. ACM. arXiv:math.GT/0205057.

[2] Erol Akbas. A presentation for the automorphisms of the 3-sphere that preserve a genus two Heegaard splitting. arXiv:math.GT/0504519.

[3] Gilbert Baumslag, Alexei G. Myasnikov, and Vladimir Shpilrain. Open problems in combinatorial group theory. Second edition. In *Combinatorial and geometric group theory (New York, 2000/Hoboken, NJ, 2001)*, volume 296 of *Contemp. Math.*, pages 1–38. Amer. Math. Soc., Providence, RI, 2002. http://www.sci.ccny.cuny.edu/∼shpil/gworld/problems/oproblems.html.

[4] Stephen J. Bigelow. Braid groups are linear. *J. Amer. Math. Soc.*, 14(2):471–486 (electronic), 2001. http://www.ams.org/.

[5] Joan Birman, Ki Hyoung Ko, and Sang Jin Lee. A new approach to the word and conjugacy problems in the braid groups. *Adv. Math.*, 139(2):322–353, 1998. http://www.math.columbia.edu/~jb/papers.html.

[6] Martin R. Bridson and James Howie. Conjugacy of finite subsets in hyperbolic groups. http://www.ma.ic.ac.uk/~mbrids/papers/bhowie/.

[7] M.R. Bridson and D. Groves. The quadratic isoperimetric inequality for mapping tori of free group automorphisms II: The general case. arXiv:math.GR/0610332.

[8] Sangbum Cho. Homeomorphisms of the 3-sphere that preserve a Heegaard splitting of genus two. arXiv:math.GT/0611767.

[9] David Epstein and Derek Holt. The linearity of the conjugacy problem in word-hyperbolic groups. 2005. http://www.maths.warwick.ac.uk/~dfh/download/papers/.

[10] David B. A. Epstein, James W. Cannon, Derek F. Holt, Silvio V. F. Levy, Michael S. Paterson, and William P. Thurston. *Word processing in groups*. Jones and Bartlett Publishers, Boston, MA, 1992.

[11] Leszek Gasieniec, Marek Karpinski, Wojciech Plandowski, and Wojciech Rytter. Efficient algorithms for Lempel-Ziv encoding. 1097:392–403, 1996. http://citeseer.ist.psu.edu/22169.html.

[12] Christian Hagenah. Gleichungen mit regulären Randbedingungen über freien Gruppen. Dissertation, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, August 2000. http://elib.uni-stuttgart.de/opus/volltexte/2000/673/.

[13] Hessam Hamidi-Tehrani. On complexity of the word problem in braid groups and mapping class groups. *Topology Appl.*, 105(3):237–259, 2000. http://www.math.columbia.edu/~hessam/.

[14] John Hempel. 3-*Manifolds*. Princeton University Press, Princeton, N. J., 1976. Ann. of Math. Studies, No. 86.

[15] Stephen P. Humphries. Generators for the mapping class group. In *Topology of low-dimensional manifolds (Proc. Second Sussex Conf., Chelwood Gate, 1977)*, volume 722 of *Lecture Notes in Math.*, pages 44–47. Springer, Berlin, 1979.

[16] Marek Karpinski, Wojciech Rytter, and Ayumi Shinohara. Pattern-matching for strings with short descriptions. In *Combinatorial pattern matching (Espoo, 1995)*, volume 937 of *Lecture Notes in Comput. Sci.*, pages 205–214. Springer, Berlin, 1995.

[17] Daan Krammer. Braid groups are linear. *Ann. of Math. (2)*, 155(1):131–156, 2002. http://www.maths.warwick.ac.uk/~daan/.

[18] Markus Lohrey. Word problems on compressed words. In *Automata, languages and programming*, volume 3142 of *Lecture Notes in Comput. Sci.*, pages 906–918. Springer, Berlin, 2004. http://www.informatik.uni-stuttgart.de/fmi/ti/personen/Lohrey/.

[19] M. Lothaire. *Combinatorics on words*, volume 17 of *Encyclopedia of Mathematics and its Applications*. Addison-Wesley Publishing Co., Reading, Mass., 1983. A collective work by Dominique Perrin, Jean Berstel, Christian Choffrut, Robert Cori, Dominique Foata, Jean Eric Pin, Guiseppe Pirillo, Christophe

Reutenauer, Marcel-P. Schützenberger, Jacques Sakarovitch and Imre Simon, With a foreword by Roger Lyndon, Edited and with a preface by Perrin.

[20] Roger C. Lyndon and Paul E. Schupp. *Combinatorial group theory.* Springer-Verlag, Berlin, 1977. Ergebnisse der Mathematik und ihrer Grenzgebiete, Band 89.

[21] Wilhelm Magnus, Abraham Karrass, and Donald Solitar. *Combinatorial group theory: Presentations of groups in terms of generators and relations.* Interscience Publishers [John Wiley & Sons, Inc.], New York-London-Sydney, 1966.

[22] Masamichi Miyazaki, Ayumi Shinohara, and Masayuki Takeda. An improved pattern matching algorithm for strings in terms of straight-line programs. *J. Discrete Algorithms (Oxf.)*, 1(1):187–204, 2000. http://www.shino.ecei.tohoku.ac.jp/∼ayumi/publications.html.

[23] Lee Mosher. Mapping class groups are automatic. *Ann. of Math. (2)*, 142(2):303–384, 1995.

[24] Wojciech Plandowski. Testing equivalence of morphisms on context-free languages. In *Algorithms—ESA '94 (Utrecht)*, volume 855 of *Lecture Notes in Comput. Sci.*, pages 460–470. Springer, Berlin, 1994.

[25] Marcus Schaefer, Eric Sedgwick, and Daniel Štefankovič. Algorithms for normal curves and surfaces. In *Computing and combinatorics*, volume 2387 of *Lecture Notes in Comput. Sci.*, pages 370–380. Springer, Berlin, 2002. http://www.cs.rochester.edu/∼stefanko/.

[26] Martin Scharlemann. Automorphisms of the 3-sphere that preserve a genus two Heegaard splitting. UCSB Math 2003-14. arXiv:math.GT/0307231.

Department of Mathematics, Rutgers University, Piscataway, New Jersey 08854

E-mail address: saulsch@math.rutgers.edu

URL: http://www.math.rutgers.edu/∼saulsch