7171

ALGORITHMS, COMPLEXITY AND DISCRETENESS
CRITERIA IN $PSL(2, \mathbb{C})^1$

Jane GILMAN

SW9613

Institut des Hautes Etudes Scientifiques
35, route de Chartres
91440 – Bures-sur-Yvette (France)

Janvier 1996

IHES/M/96/11

Algorithms, Complexity and Discreteness Criteria in $PSL(2, \mathbf{C})$ ¹

Jane Gilman

1 Introduction

Let G be a subgroup of $PSL(2, \mathbf{C})$. The *discreteness problem for G* is the problem of determining whether or not G is discrete. In this paper we assume that all groups considered are non-elementary. If G is generated by two elements, A and B of $PSL(2, \mathbf{C})$, we have what is called the *two-generator discreteness problem*. Two-generator groups are important because by a result of Jørgensen an arbitrary subgroup of $PSL(2, \mathbf{C})$ is discrete if and only if every non-elementary two-generator subgroup is [8]. One solution to the two-generator real discreteness problem (i.e. A and B in $PSL(2, \mathbf{R})$) is a geometrically motivated algorithm which was begun in [7] and completed in [6] where the algorithm is given in three forms.

Our goal here is to compute the computational complexity of the three forms of the $PSL(2, \mathbf{R})$ discreteness algorithm that appear in [6] and of the algorithm restricted to $PSL(2, \mathbf{Q})$. While this may seem far afield from the original mathematical problem of determining discreteness, it settles the question as to in what sense the discreteness problem requires an algorithm. That is, the computational complexity of the algorithm can be used to prove that an algorithmic approach is necessary and that the geometric algorithm of [6] is the best discreteness condition that one can hope to obtain.

We also investigate the computational complexity of several other discreteness criteria including Riley's $PSL(2, \mathbf{C})$ procedure [15], which is not always an algorithm, and Jørgensen's inequality [8]. We find that the geometric two-generator $PSL(2, \mathbf{Q})$ algorithm is of linear complexity. By contrast Riley's procedure appears to be at least exponential even when restricted to the two generator $PSL(2, \mathbf{Q})$ case.

The paper is organized as follows. Sections 3 and 4 contain a summary the relevant background: an overview of the geometric discreteness algorithm, a

¹Research partially supported by NSF grants #DMS 9001881 and 9409115. The author thanks the Institut des Hautes Études Scientifique for its support. Some of the work on this paper was done while the author was a visitor there.

heuristic discussion of what an algorithm is, and a summary of computational complexity. The discussion of what an algorithm is, Turing machine algorithms, BSS machines, and the various forms of the algorithm can be found in chapter 14 of [6] in expanded form. A major step in computing the complexity of any form of the algorithm is to bound the number of generating pairs that the algorithm must consider. This is done for all forms in section 6. The main theorem here comes from careful analysis of the geometric algorithm. Special attention must be given to the elliptic order algorithm which is treated in section 5. Finally, the complexities of the various forms of the algorithm, including the Turing machine algorithm are computed in section 7.

2 Notation and Terminology

We recall some standard notation. The complex numbers, the real numbers, and the rational numbers are denoted respectively by $\mathbf{C}$, $\mathbf{R}$, and $\mathbf{Q}$. $SL(2, \mathbf{F})$ denotes the set of all two by two matrices with determinant equal to one whose entries are in $\mathbf{F}$ and $PSL(2, \mathbf{F})$ equivalence classes of all such matrices where matrices differing by $-I$ are identified, I being the identity matrix. Elements of $SL(2, \mathbf{C})$ and $PSL(2, \mathbf{C})$ both act on the complex sphere as fractional linear transformations. Since an element of $SL(2, \mathbf{C})$ and its negative induce the same transformation, we do not distinguish notationally between elements of $SL(2, \mathbf{C})$ and $PSL(2, \mathbf{C})$. Where the transformation lies will generally be clear from the context.

If Tr represents the trace of a matrix, then $|Tr A|$ is well defined whether A is thought of as lying in $SL(2, \mathbf{C})$ or $PSL(2, \mathbf{C})$. The transformations in $PSL(2, \mathbf{R})$ are classified by their geometric actions on the upper-half-plane as hyperbolic, elliptic, or parabolic. These classification can also be obtained algebraically by the absolute value of the traces. That is, A is hyperbolic, elliptic, or parabolic according as $|Tr A|$ is greater than two, less than two or equal to two.

We use $\langle A_1, \dots, A_n \rangle$ to denote the group generated by $A_1, \dots, A_n$.

If α is an algebraic number lying in a finite extension of the rationals, we let M_α denote its minimal polynomial. An *isolating interval* for α , denoted by $Isol_\alpha$, tells which root of M_α α is. It is a pair of rational numbers r and s such that $\alpha \in (r, s]$ and no other root of M_α lies in $(r, s]$. Minimal polynomials

are by definition monic polynomials (i.e. they have leading coefficient 1).

3 Preliminaries: Algorithms

We begin with an overview of the $PSL(2, \mathbf{R})$ geometric algorithm and a discussion of geometric generators.

3.1 An overview of the $PSL(2, \mathbf{R})$ geometric discreteness algorithm

If one begins with a pair of generators for a subgroup of $PSL(2, \mathbf{R})$ one of three things can happen: (i) one can determine immediately from the given pair of generators that the group is not discrete, usually by using Jørgensen's inequality; (ii) one can determine immediately that the group is discrete, usually using the Poincaré polygon theorem or (iii) one cannot determine anything directly from the given pair of generators. In case (iii) the geometric algorithm began in [7] and completed in [6] tells how to replace the given pair by a Nielsen equivalent pair and assures that after a finite number of replacements one obtains a pair where discreteness or non-discreteness can be determined immediately.

The actual procedure is more complex than the above description. The Jørgensen/Poincaré dichotomy is too simplistic. The determination of discreteness and non-discreteness uses a wide range of results beyond Jørgensen's inequality and the Poincaré polygon theorem (e.g. area inequalities). Technically the procedure determines the discreteness of a larger group, G^* , in which G sits as a subgroup of index at most two. Since a group and a subgroup of finite index are either simultaneously discrete or simultaneously non-discrete, it suffices to study the problem for the larger group. The larger group is generated by three elements of order two, either reflections or half-turns. The precise definition of G^* depends upon the geometric type of the elements A and B under consideration.

A set of generators for a discrete group is called a set of *strict geometric generators* if they are generators which satisfy the hypotheses of the Poincaré polygon theorem. A set of generators for the two-generator group are called *geometric* if either they or the corresponding generators for the three-generator group are strict geometric generators. While the algorithm

uses a number of criteria to determine discreteness, the algorithm essentially finds geometric generators when the group is discrete. That is, at the point where it stops and says *G is discrete* the geometric generators are in sight and the algorithm could easily be reworked so as to output the actual geometric generators.

3.2 What is an algorithm?

Heuristically one can regard an algorithm as a recipe for solving a problem, a recipe that is composed of simple steps and always gives the right answer. Since a recipe that does not stop does not give the right answer, the definition implies that an algorithm always comes to a stopping point. However, the definition leaves a lot of room for deciding what the allowed simple steps are.

In the case of the algorithm to determine the discreteness of a two-generator subgroup of $PSL(2, R)$ it is appropriate to distinguish between three types of algorithms. The most conceptual forms of the algorithms of [7] and [6] are called *geometric* algorithms because the simple steps are geometric in nature. They consist of such operations as finding the intersection of two hyperbolic lines, comparing the hyperbolic areas of regions in the hyperbolic plane and finding the primitive rotation corresponding to a given elliptic element as well as the four standard arithmetic operations.

Roughly speaking a *Turing machine* algorithm is one where the simple steps can be carried out by a computer. In particular, the input must be finite. Since it may require an infinite amount of information to specify a real number from the set of all real numbers, no Turing machine algorithm can deal with the set of all real numbers. Nevertheless, there are reasons to consider *real number* algorithms, ones where the simple steps allowed include all ordinary arithmetic operations on real numbers, comparing the size of two real numbers, taking the square root of a positive number, computing the arc cosine of a number, and determining the rationality of a real number. Our definition of the real number algorithm in [6] is a variation of the notion of a BSS machine put forward by Blum, Shub, and Smale ([2]).

The real number algorithm can be thought of as an outline or abstract of the algorithm. It describes the sequence of computational moves to be made by the geometric algorithm without taking into account how the input is given or whether the required computational operations can be implemented on a computer.

Precise definitions of all of these terms can be found in [6] where the following is proved.

Theorem: 3.1 *Let A and B be elements of $PSL(2, \mathbf{R})$ with $G = \langle A, B \rangle$ non-elementary.*

1. *There is a geometric algorithm for determining the discreteness of G .*
2. *There is a real number algorithm for determining the discreteness of G .*
3. *If A and B are two matrices in $PSL(2, \mathbf{R})$ whose entries are algebraic numbers lying in a finite extension of the rationals, then there is a Turing machine algorithm for determining whether or not G is discrete. The input to the algorithm consists of a minimal polynomial and an isolating interval for each of the eight entries in the two matrices. The output is one of two statements: G is discrete or G is not discrete.*

If we require that the entries of the matrices for the geometric or the real number algorithm actually lie in $\mathbf{Q}$ rather than $\mathbf{R}$, then we obtain two more forms of the algorithm.

Definition: 3.2 *Assume that the entries of the two matrices A and B are rational numbers. Then the geometric algorithm and the real number algorithm restricted to $\mathbf{Q}$ are known respectively as the **rational geometric algorithm** and the **rational number algorithm**.*

A recipe that does not necessarily stop is called a *procedure*.

4 Preliminaries: computational complexity

We summarize some of the the main concepts of computational complexity. The reader is referred to [1], [3] and [18] for a more formal and detailed background.

4.1 Complexity

The (time) complexity of an algorithm is the number of steps it takes the algorithm to process an input of a given size. One says that the complexity of an algorithm is $O(f(n))$ if it takes at most $c \cdot f(n)$ steps to process an input of size n where c is some real constant and f a function. If $f(n)$ can be taken to be a polynomial, the algorithm is of polynomial complexity. Generally speaking an algorithm is considered feasible if it is of polynomial time complexity although in practice many other algorithms can be usefully implemented on a computer.

It is not always clear how to count steps. For example, how many steps does it take to compute $3a^2 + 2a + 1$? At first glance it appears to be five steps. However, if we use Horner's method and compute $(3a + 2)a + 1$ it becomes four steps. Further, if one is implementing the algorithm on a machine, what actually constitutes a step may vary from machine to machine. However, if an algorithm is of polynomial complexity on one machine it will be of polynomial complexity on any other machine (although the degree of the polynomial may vary).

In order to compute the complexity of any given algorithm, one needs to state how one is measuring the size of the input and how one is counting steps.

4.2 What Complexity Tells Us

If there were a necessary and sufficient condition for discreteness that could be applied directly to every pair of generators for G , then we would call that a *closed form* discreteness condition. An algorithm is needed because not every pair of generators for a two generator discrete subgroup of $PSL(2, \mathbf{R})$ are *geometric generators*, ones to which the Poincaré polygon theorem can be directly applied. One measure of how close to a closed form condition the algorithm is is the number of generating pairs the algorithm must consider.

One measure of the size of the input is T , **the maximal initial trace**, which is defined by $T = \max \{1, |Tr A|, |Tr B|, |Tr AB|, |Tr AB^{-1}|\}$.

To simplify this exposition and for ease of comparison with other algorithms, assume for the time being that our original matrices are in $PSL(2, \mathbf{Q})$. (We will see later (theorem 5.2) that this forces elliptic elements of finite order to be of order 2 or 3.)

We will bound the number of generating pairs that the geometric $PSL(2, \mathbf{Q})$ algorithm must consider and formulate the geometric discreteness algorithm as:

Theorem: 4.1 *Assume that A and B are elements of $PSL(2, \mathbf{Q})$ with $G = \langle A, B \rangle$ non-elementary. A necessary and sufficient condition for G to be discrete is that one of at most $10T + 10$ pairs of generators which are algorithmically obtained from A and B are geometric generators.*

This theorem is an immediate corollary of theorem 6.2 of section 6.

Since changing generating pairs must be one type of step in the geometric algorithm or in any other form of the algorithm, bounding the complexity of the algorithm will always require bounding the number of pairs of generators considered. For the geometric $PSL(2, \mathbf{Q})$ algorithm it turns out that the number of operations needed to process any given pair of generators before one replaces the pair by a different generating pair is bounded by a constant independent of the geometric type of generators and independent of T . Thus theorem 4.1 has as a corollary:

Theorem: 4.2 *The complexity of the geometric algorithm restricted to $PSL(2, \mathbf{Q})$ is $O(T)$.*

The theorem says that the geometric algorithm in $PSL(2, \mathbf{Q})$ is of polynomial time complexity, in fact of linear complexity. Being of linear complexity says that the geometric algorithm is essentially the best discreteness result one could expect in the absence of a closed form condition. The way to think of the geometric $PSL(2, \mathbf{Q})$ algorithm is that it can almost always be carried out by hand.

As a basis for comparison, consider the procedure based strictly on the Jørgensen/Poincaré dichotomy. This procedure applies in a much more general setting. It is for determining whether or not a finitely generated subgroup of $PSL(2, \mathbf{C})$ is discrete, but it is not always an algorithm. It begins with a set of generators and at iteration n it tests all pairs of words of length less than or equal to n for Jørgensen's inequality and all subsets of words of length less than or equal to n for the Poincaré polyhedron theorem. When the group is geometrically finite, it is an algorithm. In particular, it is an algorithm for two-generator subgroups of $PSL(2, \mathbf{R})$. Since the n -th iteration involves considering all subsets of words of length $\leq n$ in the

generators (i.e. at least 2^{2^n} cases), the algorithm appears to be at least of double-exponential complexity even when restricted to $PSL(2, \mathbf{Q})$. Riley's $PSL(2, C)$ computer program, POINCARE [15], is a modification of the Poincaré/Jørgensen dichotomy, but it is not necessarily an algorithm. In recent work, Riley [16] shows how to make it into an algorithm in $PSL(2, R)$, but even when restricted to the two-generator $PSL(2, Q)$ case, at step n his program requires considering all words of length n making it at least exponential. We cannot be more precise because there are no known upper bounds for n , the number of iterations. The Riley procedure is not intended to be close to a closed form condition.

4.3 Measuring Input and Counting Steps

Our goal is to compute the computational complexities of the various algorithms including the three forms of the $PSL(2, \mathbf{R})$ discreteness algorithm. However the size of the input is measured or the steps counted, it is clear that to compute the complexity one needs to bound the number of pairs of generators that the algorithm must consider. Theorem 6.2 does this for the real number algorithm (the BSS machine) and the geometric algorithm and their restrictions to $\mathbf{Q}$ and for the Turing machine form of the algorithm.

In the case of the Turing machine algorithm the size of the input is measured by D , the degree of the extension in which the eight matrix entries lie and by S , where S is a function of the maximum of the seminorms of certain minimal polynomials or representing polynomials. Roughly speaking, for the Turing machine forms of the algorithm, S plays the role that T , the maximal initial trace, plays over $\mathbf{Q}$ and $\mathbf{R}$. The precise definition of the S is given at the beginning of section 7.3.

To compute the actual complexities of the algorithms we must also take into account the number of steps it takes to process each pair of generators. For the real number algorithm and the rational number algorithm this involves calculating the number of ordinary arithmetic operations and the number of calculations of arc cosine, etc. For the Turing machine algorithm, we need to know not just the number of such operations, but also the complexities of carrying out each of them via symbolic computation. The complexities of the algorithms for the ordinary arithmetic operations in algebraic extensions are derived in [10] along with part of the computation for the complexity of the SIGN algorithm.

5 Elliptic Elements

A matrix E in $SL(2, R)$ is *elliptic* if $|Tr(E)| < 2$. If E is elliptic, let $ord(E)$ denote its order. It is the smallest integer $\bar{m}$ such that $E^{\bar{m}} = 1$. If there is no such $\bar{m}$, we set $ord(E) = \infty$. Let m be the order of the corresponding transformation (the order of the image of E in $PSL(2, \mathbf{R})$). If $\bar{m}$ is odd, $m = \bar{m}$ and if $\bar{m}$ is even, $m = \frac{\bar{m}}{2}$. We write $o(E)$ to denote the order of the image of E in $PSL(2, R)$.

For any matrix M , let $T(M) = |Tr M|$. Recall that if as a transformation E rotates by an angle of 2θ , $(T(E))^2 = 4 \cos^2 \theta$. If $2\theta = \pm 2\pi/n$ for some integer n , then E is called a (geometrically) *primitive* rotation. If E rotates by $k2\pi/n$ for some integer $k \neq \pm 1$, then k is called the *phase* of E .

If we are working over the reals, $o(E)$ can be computed from the trace of E as long as one can compute the arc cosine of a real number between -1 and 1 and determine whether a real number is rational or not.

If we are working in a finite extension of the rationals so that the entries of E lie in an extension of degree D , we proved in [6] that there is an algorithm for computing $o(E)$. The simplest form of the algorithm uses only matrix multiplication and follows from bounding the order of an elliptic element in a finite extension of the rationals by a function of the degree of the extension. Namely,

Theorem: 5.1 The elliptic order algorithm (version #2): *Let E be a matrix in $SL(2, R)$ with $|Tr E| < 2$. If the entries of E lie in a finite extension of the rationals of degree D , then E is of finite order if and only if E^m is $\pm I$ for some integer m with $m \leq 32D^2$.*

We emphasize that for the purposes of determining the order of an elliptic, the real algorithm requires the arc cosine function and being able to determine whether or not a real number is rational. The elliptic order algorithm can be applied in any finite extension of the rationals and by contrast does not require either an arc cosine function or the ability to determine rationality. It merely requires the ability to multiply matrices. It can even be applied to matrices with complex entries. In other words, there is a Turing machine algorithm for computing the order of an elliptic element in any finite extension of the rationals which only requires matrix multiplication.

One can show that if the entries of an elliptic matrix E lie in $PSL(2, \mathbf{Q})$, then there are very few possible finite orders for E . Namely,

Theorem: 5.2 *If the entries of an elliptic transformation lie in the rationals, then if $o(E)$ is finite, it is either 2 or 3. There is a rational number algorithm for computing the order of an elliptic element that uses simple multiplication and does not require the use of the arc cosine node or the rationality oracle.*

Proof: The first assertion follows the fact that if θ is a rational multiple of π with $0 \leq \theta \leq \pi$ and $\cos \theta$ is rational, then $\theta = 0, \pi/3, \pi/2, 2\pi/3$, or π . This fact can be deduced from repeated and determined applications of the rational roots theorem. (Alternately see theorem D.1 page 129 of [14].) E will rotate by 2θ .

If E is an elliptic element of $SL(2, \mathbf{Q})$, the algorithm is merely to compute E^2 and E^3 . If either of these is $\pm I$, then E is of finite order. Otherwise, E is of infinite order. *QED*

6 Bounding the number of pairs of generators

In order to bound the number of pairs of generators that each form of the algorithm considers we review the geometric algorithm.

6.1 A Summary of the Geometric Algorithm

The geometric algorithm proceeds by a cases by case analysis where the cases are determined by the geometric types of the generators. Since pairs of hyperbolics are divided into two subcases depending upon whether or not the axes intersect or are disjoint there are seven cases: hyperbolic-hyperbolic with disjoint axes, hyperbolic-parabolic, parabolic-parabolic, hyperbolic-elliptic, elliptic-hyperbolic, elliptic-parabolic, elliptic-elliptic and hyperbolic-hyperbolic with intersecting axes. It turns out that one cannot treat the first case without simultaneously treating the next five cases. Thus the first six cases are known as the *intertwining cases*. Pairs of hyperbolics with intersecting axes are the only fully self contained case. For the intertwining cases, there is a hierarchy of geometric types. That is, elliptics are *easier* than parabolics which are in turn *easier* than hyperbolics. This imposes a hierarchy on the pairs of generators in the first six cases. One begins with a pair of generators

and locates its case. When one cannot determine directly from the given pair that the group is discrete or not discrete, the algorithm either replaces the pair by an easier pair (a pair lying in an easier case) or replaces the pair by a new pair in the same case. One shows that the algorithm stops by showing that one only remains stuck in the same case a finite number of times before one must move up to an easier case. To bound the number of generating pairs that the algorithm considers, one must, therefore, only bound the number of times one returns to the same case in each of the intertwining cases and bound the number of generating pairs that the intersecting axis algorithm considers.

In what follows, we use the numbering of the steps of the Real Number Algorithm (theorem 14.4.1) given in chapter 14 of [6], but also make occassional reference to [7].

6.2 Minor Modification of the Intersecting Axes Algorithm

Before we treat the case of pairs of hyperbolics with intersecting axes, we simplify things by adding the step 2(g)* given below to the real algorithm in part 2(g) before step 2(g)(i) (see case 2 of theorem 14.4.1 of [6]).

Step 2(g)*: If $\frac{|Tr\ g|}{2} < 1.031$, G is not discrete. Stop.

Although the algorithm does not require this step in order to terminate, we add the additional step because it makes it easier to bound the number of pairs of generators one must consider.

The validity of step 2(g)* follows from:

Lemma: 6.1 *If G is discrete and A and B are hyperbolics with intersecting axes and elliptic commutator with phase $k = 2$ or 3 , then*

- $\frac{|Tr\ A|}{2} \geq \frac{\cos \pi/7}{\sin \pi/3}$ when G is a $(2, 3, n)$ triangle group.
- $\frac{|Tr\ A|}{2} \geq \frac{\cos \pi/5}{\sin \pi/4}$ when G is a $(2, 4, n)$ triangle group.

In both cases, $\cosh T_A = |Tr\ A|/2 > 1.038$ and $T_A > .03$.

Proof: Note that $T_A = d(p_1, p_2)$ is the distance between two fixed points of order two in the group G and it is half of the translation length of A .

Theorems 9.1.1, 10.1.1 and 12.1.2 of [6] imply that if (R, S) is the pair of matrices where the triangle algorithm stops, $T_R = D_0(2, 2)$ when G is a $(2, 3, n)$ triangle group, $T_R = D_0(2, 4)$ when G is a $(2, 4, n)$ triangle group, and $T_A \geq T_R$ in both cases. Here we are following the notation of [6] so that by definition (see section 4.2 of [6]) $D_0(a, b)$ is the smallest distance between any two fixed points of G of order a and b .

If G is the $(2, 3, n)$ triangle groups, apply proposition 4.4.1 of [6] to see that $\cosh D_0(2, 2) = \cosh^2 D_0(2, 3) - \sinh^2 D_0(2, 3) \cdot \cos(2\pi/3) \geq \cosh^2 D_0(2, 3) \geq \cosh D_0(2, 3)$. From fact 6.2.1 of [6] $\cosh D_0(2, 3) = \frac{\cos \pi/n}{\sin \pi/3}$ and $\cosh D_0(2, 4) = \frac{\cos \pi/n}{\sin \pi/4}$. In a $(2, 3, n)$ triangle group, $n \geq 7$ and in a $(2, 4, n)$ triangle group, $n \geq 5$. Use a trig table to see that $\cos \pi/7 > .9001$ and $\cos \pi/5 > .808$. Compute that $\cosh T_A > 1.038$ in the first case and $\cosh T_A > 1.41$ in the second. Finally use the fact that if $M = \cosh T_A$, $T_A = \ln(M + \sqrt{M^2 - 1}) \geq \ln(M)$ to verify that if $\cosh T_A > 1.038$, $T_A > .03$. *QED*

From now on when we refer to the algorithm, we mean the modified algorithm with this additional step.

6.3 Computing the bounds

Recall that T , the maximal initial trace, was defined in 4.2. Let d , the *initial elliptic order*, be defined as follows: it is the order of the first elliptic element of finite order that the real number algorithm encounters when the algorithm encounters such an element and it is 1 if the algorithm never encounters an elliptic element or if the order of the first elliptic element is infinite.

In the case of the Turing machine algorithm it is assumed that the entries in the two matrices are algebraic numbers each lying in a finite extension of the rationals. In this case the input consists of $\alpha_1, \dots, \alpha_8$, the eight entries in the two matrices A and B . Each entry is specified by giving its minimal polynomial together with an isolating interval. There is a primitive element γ so that $Q(\alpha_1, \dots, \alpha_8) = Q(\gamma)$ is a simple extension. We let D be the degree of γ . D is bounded by the product of the degrees of the eight minimal polynomials.

Theorem: 6.2 1. *The rational number algorithm, $PSL(2, \mathbf{Q})$: The number of pairs of generators that the rational algorithm considers is bounded by a linear polynomial in T , $P(T) = 10T + 10$.*

2. *The real number algorithm, $PSL(2, \mathbf{R})$: The number of pairs of generators that the real number algorithm considers is bounded by a polynomial of degree two in T and degree one in d , $P(T, d) = 170T^2 + d + 14$.*
3. *The Turing machine algorithm, $PSL(2, \mathbf{Q}(\alpha_1, \dots, \alpha_8)) = PSL(2, Q(\gamma))$: The number of pairs of generators that the TM algorithm considers is bounded by a polynomial of degree two in T and in D , $P(T, D) = 170T^2 + 32D^2 + 14$.*

Proof: The proof consist of two main pieces. First we establish a bound for the number of generating pairs that the real number algorithm considers. The bound depends upon T and another input measure, α , and applies to all of the relevant forms of the algorithm. The second step is to translate α to an appropriate measure for each of the various forms of the algorithm. For the intertwining algorithm we bound the number of times one has to return to the same geometric case before one moves up to an *easier* geometric case. In terms of this notation of chapter 14 of [6], it suffices to bound the number of times one returns to $8(a)$ from $8(o)$ and $8(n)$, to $6(a)$ from $6(k)$, to $4(a)$ from $4(k)$, to $3(a)$ from $3(i)$, and to step $2(g)$ from $2(i)$. Note that when a step returns to step #1, it goes through step #1 for renormalization and then returns to a higher step. (E.g., $3(g)$ passes through 1 and goes to 4; $3(h)$ passes through 1 and goes to 6; $4(i)$ via 1 to 7; $4(j)$ via 1 to 5; $5(e)$ via 1 to 7; $6(j)$ via 1 to 8; and $7(g)$ via 1 to 8.) Also we never return to step 5 from step 5 or to step 7 from step 7.

In what follows we return to [7] and use the computations found there. Let $[]$ denote the greatest integer function. In [7] it was pointed out that the number of times one remained trapped the case of pairs of hyperbolics with disjoint axes could be bounded, but it was (erroneously) stated that it was too difficult to bound the other cases. In fact, many of the bounds for the other cases can already be found in [7].

We procede with a step by step analysis.

Step 3(i): (Same as case I of [7].) When we replace the pair (g, h) by the pair (g, gh) , we have $Tr\ h - Tr\ g \geq \frac{(\sqrt{2}-1)^2}{\sqrt{2}}$. Thus as pointed out in [7] p. 24 we return to step 3(a) at most $\lceil \frac{Tr\ h - 2}{\frac{(\sqrt{2}-1)^2}{\sqrt{2}}} \rceil$. Thus we return to step 3(a) at most $[9(Tr\ h)]$ times.

Step 4(k): (Case II of [7].) We replace the pair (g, h) by the pair (g, gh) where $Tr\ (gh) = Tr\ h + b(g)b(h)$ and $|b(g)b(h)| \geq 1$. Since $b(h)$ is negative

and $b(g)$ is positive, we return to step 4(a) at most $[Tr\ h - 2]$ times. (When we are in this case $Tr\ h > 2$). This is linear in $Tr\ h$.

Step 6(k): (Case IV (p. 28) of [7].) We return to this step at most $\alpha - 1$ times where α is the order of g .

Step 8(n) and 8(o): (Case VI of [7])

Each time we return to this step we cut the area of the prospective fundamental region in half. The region is a double triangular region. We stop once the area goes below $\pi/42$ for then the group cannot be discrete. The initial area of the triangular region is $\pi(1 - \alpha_1 - \alpha_2 - \alpha_3)$ where the α_i are the angles of the triangular region. Since the area of this initial triangular region is at most π , after seven steps the area is at most $\frac{\pi}{2^7}$ (which is less than $\frac{\pi}{84}$).

To summarize for the intertwining cases, we let α be the order of the first elliptic element encountered in case 6. The total number of pairs of generators the algorithm considers is bounded by the sum of the bounds on cases 3, 4, 5, 6, 7 and 8, which is $([9T] + 1) + ([T] + 1) + 1 + (\alpha - 1 + 1) + 1 + 7 \leq [10T] + \alpha + 12$.

Next we turn to the case of pairs of hyperbolics with intersecting axes, case 2 of [6].

Step(2): Bounding the number of times we return to step 2(g) from step 2(i) is equivalent to bounding the number of pairs of generators that the triangle algorithm of chapter 2 of [6] considers. To do this, we need to bound the number of steps we move along an axis and the number of times we turn a corner.

We saw in section 2.6 of [6] that we move along the axis of A at most m steps where m is the smallest integer such that $m \cdot T_A \geq [p_1, Q]_d$. Now $[p_1, Q]_d \leq \max\{T_B, T_{BA^{-1}}\}$. Thus when moving along the axis of A_i it suffices to bound the smallest integer m_i such that $m_i \geq \lceil \frac{\max\{T_{B_i}, T_{B_i A_i^{-1}}\}}{T_{A_i}} \rceil$. We want to find a bound that works for all i . Note that $T_{C_i} \geq \max\{T_{B_i}, T_{B_i A_i^{-1}}\}$ and $T_{C_1} \geq T_{C_i}$. Further for each i we have $T_{A_i} > .03$. Thus we can choose $m < 33.34T_{C_1}$. m is now bounded by a linear function of T_{C_1} which in turn is a logarithmic function of $|Tr\ C_1|$. Namely, $T_{C_1} = \ln(M + \sqrt{M^2 - 1})$ where $M = \cosh T_{C_1} = |Tr\ C_1|/2 \leq T/2$. Use the fact that for any $X > 1$, $\ln X \leq X/(2.7)$ to conclude that we move along any axis at most $33.34T/(2.7)$ steps. Thus at most $13T$ steps.

We bound the number of times we turn a corner as follows: When we

turn a corner the shortest side becomes the longest side. That is, given triangle T_i , $T_i = (A_i, B_i, C_i)$ there is a $j > i$ such that $T_j = (A_j, B_j, C_j)$ and $C_j \leq A_i$ ([6] lemma 2.7.1). It suffices to bound $\cosh C_i - \cosh C_j$ from below. At each step where the algorithm does not stop we have $\cosh C_i \geq \cosh A_i \cosh B_i \geq (\cosh A_i)^2$. The first inequality sign is from equation 2.3 of [6]. The second inequality sign is from the fact that $A_i \leq B_i$. Since $C_j \leq A_i$, $-\cosh C_j \geq -\cosh A_i$, whence $\cosh C_i - \cosh C_j \geq (\cosh A_i)^2 - \cosh A_i$. Since $\cosh A_i \geq 1.038$, $(\cosh A_i)^2 - \cosh A_i = (\cosh A_i)(\cosh A_i - 1) > .039$. Thus we will turn a corner at most $26 \cdot \cosh C_1$ times.

Multiplying the number of times one moves along the axis of A before one turns a corner and the number of times one turns a corner yields at worst a polynomial function of $|Tr C_1|$ or T . Namely since $\cosh C_1 \leq T/2$, we treat at most $13T \cdot 13T$ pairs of generators.

We have thus established a bound for the number of generating pairs the real algorithm must consider. This bound is $[10T] + \alpha - 1 + 12 + [(13T)(13T)] + 1$ where α is the order of the elliptic element that the algorithm encounters if it must pass through step 6. This bound only depends upon T and α and applies to all forms of the algorithm. If one traces through the algorithm from all possible starting points, the α of case 6 is the order of the first elliptic element of finite order that the algorithm encounters except if the first elliptic element is encountered in case 8, in which case the actual order of the elliptic does not matter. If the algorithm encounters an elliptic element of infinite order, it stops.

To obtain the result over $PSL(2, \mathbf{Q})$ we observe that by 5.2 the order of an elliptic element in $PSL(2, \mathbf{Q})$ is either 2 or 3. This fact simplifies the algorithm and replaces α of case 6 by some constant. It also affects the count for case 8. In case 8 we never obtain a $(2, 3, n)$ triangle group with $n \geq 5$ or a $(2, 4, n)$ triangle group with $n \geq 4$. Thus if the phase of the commutator, (in the notation of [6]) $k(\gamma) = 2$ or 3 , G is not discrete. Thus we replace the T^2 term arising in case 8 by a constant. To obtain a finer upper bound on the number of generators, we can trace all possible routes the algorithm takes starting with step 1 and conclude that under any path the number of generators is less than $10T + 10$.

For the Turing machine algorithm, the orders of the elliptic elements are by theorem 5.1 at most $32D^2$. Thus we replace α by $32D^2$. Since $T > 1$, T^2 dominates T and we can eliminate the linear term in the estimate by raising the coefficient of the T^2 term. *QED*

6.4 The Length function

We can use the proof theorem 6.2 to compute how the length of the words in the generators grow. Let $Length(T, \mathcal{D})$ denote the maximal length as a word in A and B of any pair of matrices the algorithm considers where $\mathcal{D} = \mathcal{D}$ for the Turing machine algorithm and $\mathcal{D} = \lceil$ for the real number algorithm.

Lemma: 6.3 *If at step j we begin with initial matrices $(A_{I,j}, B_{I,j})$ and leave step j with exit matrices $(A_{E,j}, B_{E,j})$, then the length l_j of the exit matrices as words in the initial matrices are as follows:*

$$\begin{aligned} l_1 &= 1 \\ l_2 &= 2^{(13T)^2} \\ l_3 &= 2^{9T} \\ l_4 &= T \\ l_5 &= 2 \\ l_6 &= 34D^2 \text{ or } d \\ l_7 &= 34D^2 \text{ or } d \\ l_8 &= (34D^2 + 1)^7 \text{ or } d^7. \end{aligned}$$

Proof: Trace through the details of how each step is repeated. For those cases where the initial matrices are not both hyperbolic, observe that the length of the words in the initial generators at step n of repeating case j , is linear in n . Whence for these j , $l_j =$ the number of times we repeat case j . In the case of pairs of hyperbolics, the way in which we repeat the cases allows the possibility that the length grows as a Fibonacci sequence. For example, a possible sequence might be $(A, B) \rightarrow (B, AB^{-1}) \rightarrow (AB^{-1}, BAB^{-1}) \rightarrow (BAB^{-1}, BAB^{-1}AB^{-1}) \rightarrow \dots$. That is, if $W(n)$ represents the maximum possible length of the word in the generators at step n , $W(n+1) = W(n) + W(n-1)$. (whence $2^n \geq W(n) \geq 2^{n-1}$). This gives in the intersecting axes case $l_2 = 2^{(13T)^2}$ and in the disjoint axes case $l_3 = 2^{9T}$. *QED*

As a corollary we have:

Corollary: 6.4 *$Length(T, \mathcal{D})$ is at most $2^{9T\delta_{dh}} 2T(34\mathcal{D}^2 + 1)^9 + \delta_{ih} 2^{(13T)^2}$ where $\delta_{dh} = 1$ if the initial generators are pairs of hyperbolics with disjoint axes and 0 otherwise and $\delta_{ih} = 1$ if the initial generators are pairs of hyperbolics with intersecting axes and 0 otherwise. Here T is the maximal initial trace and $\mathcal{D}$ is either the degree of the extension or the maximal initial elliptic order.*

Proof: In addition to analysing how many times one is locked into any given type of step, one can also trace through the algorithm to see which sequences of types of steps actually can occur. For example, (omitting the occurrences of 1) the route $3 - 4 - 7 - 8$, is a possible route, but the route $3 - 4 - 5 - 6 - 7 - 8$ never occurs. This sequence notation means that any given type of step j will be repeated l_j times before the algorithm moves to the next type of step in the sequence. In particular, if we have hyperbolics with intersecting axes, the route 2 is the only possible route. For hyperbolics with disjoint axes, the route will always begin with 3 but not all possible sequences following 3 necessarily occur. If one of the two matrices is not hyperbolic, then the route must begin with a number higher than 3. The lengths multiply. That is, if we enter step j with words of length l in the original A and B , then we exit step j with words of length $l \cdot l_j$ in the original A and B . *QED*

7 Complexities

In this section we use the bounds on the number of generating pairs obtained in section 6 to finish computing the complexities of the various forms of the algorithm. Since all forms of the algorithm use Jørgensen's inequality and an elliptic order algorithm, the complexity of both are implicit in our results. We keep explicit track of the complexity of implementing Jørgensen's inequality. This is not done because its complexity is of interest in its own right. Rather, since we think of Jørgensen's inequality as a simple straightforward condition, it is informative to compare its complexity under various forms with the complexity of the similar form of the full algorithm.

We begin with a few words on Jørgensen's inequality.

7.1 The Complexity of Jørgensen's Inequality

Jørgensen's inequality says that if G is discrete, then whenever A and B in G generate a non-elementary subgroup, $|Tr^2 A - 4| + |Tr[A, B] - 2| \geq 1$. Here $[A, B]$ denotes the commutator of A and B .

Whatever form of the algorithm we are considering, when we apply the Jørgensen test, we apply it to one pair of generators. To treat a pair of generators one must compute $[A, B]$ and its trace, the square of the trace of

A , do two subtractions, take two absolute values and perform one addition. Multiplying two matrices requires sixteen multiplications of matrix entries and four additions. The commutator of A and B can be computed by computing three matrix multiplications. Thus the total number of arithmetic steps is bounded by some constant.

When we think of testing the inequality as a geometric or real number procedure, we will only need to take into account this finite number of arithmetic steps. However, when we think of it as a Turing machine procedure where the entries of the two matrices lie in a finite extension of the rationals, we will need to take into account the complexity of multiplying in algebraic extensions and other symbolic computation operations such as a SIGN algorithm. Thus we will see that while the Jørgensen complexity is $O(1)$ for the geometric and real number algorithms (see theorem 7.1), it becomes $O(D^8(L(S_0))^2)$ for the Turing machine algorithm. (The measures of input S_0 and D are defined below.)

7.2 The Complexities of the Geometric and Real Number Algorithms

For the rational number algorithm and the rational geometric algorithm we measure the size of the input by T , the maximal initial trace. For the real number algorithm and the geometric algorithm we measure the size of the input by T and d , the initial elliptic order.

We obtain

- Theorem: 7.1**
1. $PSL(2, \mathbf{Q})$: The complexity of the rational number algorithm and of the rational geometric algorithm is $O(T)$.
 2. $PSL(2, \mathbf{R})$: The complexity of the real number algorithm and of the geometric algorithm is $O(T^2 + d)$.
 3. Jørgensen: The complexity of implementing the Jørgensen test as a real number algorithm, a geometric algorithm, a rational number algorithm or a rational geometric algorithm is $O(1)$.

Proof: To compute the complexity of the geometric algorithm we need to bound the number of steps it takes to process an input of given size. We have already bounded the number of pairs of generators that the algorithm

considers, by $P(T) = 10T + 10$ in the rational case and by $P(T, d) = 170T^2 + d + 14$ in the real case. We go through the intertwining geometric algorithm as given in [7] and the geometric intersecting axes algorithm as given in [6] step by step and compute the number of geometric and simple arithmetic steps it takes to process a single pairs of generators lying in any one of the seven cases. That is, we count the number of times we find intersections of hyperbolic lines, compute areas, as well as carry out standard arithmetic operations. For each possible type of generators this count is finite, a constant that does not depend upon T or d .

The real number algorithm in [6] comes from translating the simple geometric steps into purely computational steps. We compute in the same manner as for the geometric algorithm but instead of counting the geometric operations, we count the number of standard arithmetic operations (e.g. additions, multiplications, subtractions and divisions) and the number of times we use the various nodes (e.g. the arc cosine node, the less-than or equal node) and the rationality oracle. Again for each type of pair of generators that must be treated the number of simple steps required to process the pair is a finite constant independent of T or d . *QED*

7.3 The Complexity of the Turing Machine Algorithm

We measure a polynomial by its degree and its seminorm. If $P(x) = p_t x^t + \dots + p_1 x + p_0$ is a polynomial with rational coefficients with $p_i = \frac{r_i}{s_i}$ where s_i and r_i are relatively prime integers, the *seminorm of P* , is denoted by $SN(P)$, and is defined by $SN(P) = |r_t| + \dots + |r_0| + |s_t| \dots |s_0|$.

For the Turing machine algorithm the size of the input is measured by D , the degree of the extension, and by the maximal seminorm of certain polynomials. Basically to compute the complexity, we need to take three things into account: the total number of steps including the number of times one is locked into a given one of the seven cases of the algorithm, the complexity of implementing each simple step, and whether and how each step increases the seminorm. We bound the complexity by multiplying the number of steps by the worst complexity of implementing any given simple step using the highest possible seminorm one can possibly obtain after all of the steps are implemented.

We begin with some definitions.

7.3.1 Additional Terminology

$L(SN(P))$, the length of the seminorm of P , is defined by $L(SN(P)) = \lfloor \log_\beta |SN(P)| \rfloor + 1$ where β is the basis of classical integer arithmetic and is the order of the largest integer fitting a single word. (See page 286 of [12] for details.) For our purposes the important property of L is that it behaves like a logarithmic function in that $L(ab) \leq L(a) + L(b)$ and $L(a^b) \leq bL(a)$ for any a and b .

We observe that if $P(x)$ and $Q(x)$ are polynomials with rational coefficients, then $SN(PQ) \leq SN(P)SN(Q)$ and $SN(P + Q) \leq SN(P)SN(Q)$. However, if P and Q have the same set of denominators (i.e. the denominator of coefficient of x^i in $P(x)$ is the same as that of x^i in $Q(x)$), then $SN(P + Q) \leq SN(P) + SN(Q)$.

If $\mathbf{Q}(\gamma)$ is a simple extension of the rationals and α is an element of $\mathbf{Q}(\gamma)$, then the *representing polynomial* for α is the rational polynomial in γ , $R_{\alpha,\gamma}$ with $\alpha = R_{\alpha,\gamma}(\gamma)$.

7.3.2 Two Forms of the Turing Machine Algorithm

We distinguish between two Turing machine algorithms:

Definition: 7.2 TM1 denotes the **first Turing machine algorithm**. We assume that the eight entries of the two matrices, $\alpha_1, \dots, \alpha_8$, lie in a finite simple extension of the rationals, $\mathbf{Q}(\gamma)$, of degree D . The input to the algorithm consists of the minimal polynomial for γ , its isolating interval, and the representing polynomials for the eight entries, namely, M_γ , $Isol_\gamma$, and $R_{\alpha_i,\gamma}$, $i = 1, \dots, 8$.

Definition: 7.3 TM2 denotes the **second Turing machine algorithm**. We assume that the eight entries of the two matrices, $\alpha_1, \dots, \alpha_8$, are algebraic over $\mathbf{Q}$ and are given by their minimal polynomials and their isolating intervals. The input to the algorithm consists of $\{M_{\alpha_i}, Isol_{\alpha_i}, i = 1, \dots, 8\}$.

The Turing Machine algorithm of [6] is TM2. There is an algorithm that inputs the input to TM2 and outputs the input to TM1. This algorithm involves seven applications of the algorithm SIMPLE of [10]. If we use S8 to

denote the seven applications (adjoining the eight roots), then the algorithm TM2 is the same as following S8 by TM1.

For the sake of brevity we compute the complexity of TM1. The complexity of TM2 and several other variations of Turing machine algorithms will appear in another paper.

7.3.3 The Complexity of TM1

Definition: 7.4 We define the **maximal initial seminorm** to be the maximal seminorm of the primitive element γ and the eight representing polynomials and denote it by S_0 . We also require that $S_0 \geq 2$. That is, $S_0 = \max\{SN(M_\gamma), SN(R_{\alpha_1, \gamma}), \dots, SN(R_{\alpha_8, \gamma}), 2\}$.

We normalize our input data as follows. Let N be the smallest positive integer such that $|\frac{\gamma}{N}| < 1$. Since $Q(\gamma) = Q(\frac{\gamma}{N})$, we can replace M_γ by $M_{\frac{\gamma}{N}}$ and the R_{α_i} by the representing polynomials for the α_i with respect to $\frac{\gamma}{N}$.

We trace through the algorithm and observe that the only step that permanently increases the seminorm is changing the pair of generating matrices. The entries of the new generators are sums of products of words in the α_i . All other steps that change the seminorm do so temporarily and by a bounded amount. We let S_M denote the largest seminorm of any element of $Q(\gamma)$ that the algorithm encounters.

Lemma: 7.5 Assume that the entries of the matrices lie in a finite simple extension $\mathbf{Q}(\gamma)$ of degree D . Let the minimal polynomial for γ , M_γ , and the representing polynomials $R_{\alpha_i, \gamma}$ be a normalized set of input data. Let S_M be the maximum seminorm that the algorithm encounters. Let T be the maximal initial trace of the two matrices. Then the complexity of TM1 is

$$O(D^8(L(S_M))^2) \cdot P(T, D)$$

where $P(T, D) = 32D^2 + 170T^2 + 14$.

The complexity of Jørgensen's inequality similarly implemented is

$$O(D^8(L(S_M))^2).$$

Proof: The complexity is bounded by multiplying the number of steps by the dominant time of any step evaluated for S_M . Tracing through the algorithm

one sees that the algorithm actually only uses the arithmetic operations of addition, subtraction and multiplication in $Q(\gamma)$. (Division is never needed because when inverting a two-by-two matrix of determinant one, one only needs to compute the negatives of the entries.) The dominant time here is the time for multiplication which is $O(D^3(L(S))^2)$ (see [10] page 174). Here D is the degree of $Q(\gamma)$ and S is the maximum of the seminorms of the elements being multiplied and of $SN(M_\gamma)$. The other operation that one must use is the SIGN algorithm (p. 175 of [10]). Finding the sign of $B(\gamma)$ in $Q(\gamma)$ involves three steps: (1) computing the largest square free divisor $\bar{B}$ of $B^* \in \mathbf{Z}[x]$ where $b \in Q - \{0\}$ and $\frac{1}{b}B^* = B$, (2) counting the zeros of $\bar{B}$ in a given interval and (3) performing a certain number, $O(DL(S))$, of interval bisections. We note that $|b| \leq S^D$ as b can be taken to be the least common multiple of the denominators of the coefficients in $B(x)$. We use the algorithm for square free decomposition of a primitive polynomial (p. 98 of [9]). The dominant step here is the GCD algorithm (p. 84 of [5]) which yields a maximum time of $O(D^6 + D^4(L(S))^2)$ for the square free factorization. We then use the modified Uspensky algorithm to compute the number of positive roots of $\bar{B}(x)$ and $\bar{B}(-x)$ (p. 90 of [5]). This has maximum computing time $O(D^6L(S(\bar{B})^2))$ (p. 93 of [5]). Since $SN(\bar{B}) \leq SN(B^*) \leq S^{D+1}$, we obtain $O(D^8L(S)^2)$ for the maximum time.

The number of pairs of generators the algorithm encounters is bounded by $P(T, D)$ where P and D are as in theorem 6.2. The proof of theorem 7.1 shows that the number of computational steps one uses to treat a set of generators within a given one of the seven cases is bounded and is independent of D and T . Thus that the total number of computational steps is $O(P(T, D))$.

To implement Jørgensen's inequality, one must compute $[A, B]$, the commutator of A and B , its trace and the square of the trace of A or B . This will involve multiplying the initial α 's together at least sixteen times and adding them a total of 4 times. Thus at worst one must multiply with a seminorm of $((S_0)^{16})^4$ yielding a complexity of $O(D^3(L((S_0)^{16})^4)^2)$ which is the same as $O(D^3(L(S_0))^2)$. We also need to test one inequality which is the same thing as one SIGN calculation which is $O(D^8(L(S))^2)$. For Jørgensen's inequality, we can take $S_M = S_0$. *QED*

We next seek to replace T and S_M by functions of D and S_0 . Replacing T is straight forward.

Lemma: 7.6 $T \leq 4(S_0 D)^2$.

Proof: If M is a bound on the absolute values of the eight entries in the two matrices, A and B , and T is the maximal initial trace, then it is easy to see that $T \leq 4M^2$. But $|\alpha_i| = |R_{\alpha_i, \gamma}(\gamma)| \leq SN(R_{\alpha_i, \gamma})|1 + \gamma + \dots + \gamma^{D-1}|$. Since $|\gamma| \leq 1$, the lemma follows. *QED*

7.3.4 How the algorithm increases the seminorm

To compute S_M we first observe that the only type of operation that permanently increases the seminorm is the replacement of a pair of generators by a new pair. $\text{Length}(T, D)$ of section 6.4 gives the maximal length of a word in the matrices A and B that the algorithm encounters.

Let $P(x)$ be a polynomial in $\mathbf{Q}[x]$ and let P_0 denote P reduced modulo $M_\gamma(x)$. We modify the division and remainder algorithm over $\mathbf{Z}[x]$ (page 133 of [11]) to apply to P and M_γ and compute that if m is the degree of P , then $L(SN(P_0)) = O((2m + 1)L(S))$ where S is the maximum of $SN(P)$ and $SN(M_\gamma)$.

In particular, if P is a word of length at most t in the initial eight entries, then $m = (D - 1)t$ and $S(P) \leq S^t$, which gives

$$L(SN(P_0)) = O((2t(D - 1) + 1)L(S^t)) = O(t(2t(D - 1) + 1)L(S)).$$

In particular if $(S_t)_0$ denotes the maximal seminorm of any word of length t in the eight initial entries we have after reduction modulo M_γ , we have

$$L((S_{\text{Length}(T, D)})_0) = O((2(D - 1)\text{Length}(T, D) + 1)\text{Length}(T, D)L(S)).$$

Now when we work with the matrix entries, if the longest word in the original matrices is $\text{Length}(T, D)$, then each matrix entry is the sum of $2^{\text{Length}(T, D)}$ words of length $\text{Length}(T, D)$.

For any polynomial P we let $\text{lcm}(P)$ be the least common multiple of the denominators of its coefficients and we let LCM denote the least common multiple of $\text{lcm}(R_{\alpha_1, \gamma}), \dots, \text{lcm}(R_{\alpha_8, \gamma})$ and $\text{lcm}(M_\gamma)$. We note that $\text{lcm}(R_{\alpha_i, \gamma}) \leq (S_0)^D$ and thus $LCM \leq ((S_0)^D)^9 = (S_0)^{9D}$. If for a polynomial P , $\bar{P}$ denotes P written without the coefficients reduced to lowest terms, that is written so that each term has denominator LCM , then we let $\overline{SN}(P)$ be the sum of the absolute values of the unreduced numerators and denominators of $\bar{P}$. Extending this convention, we may assume

that when P is a word of length t in the original eight polynomials, then $\bar{P}$ has unreduced coefficients each with denominator $(LCM)^t$. Then we have $SN(P) \leq \overline{SN}(P)$ and in particular, whenever P and Q are words of the same length, $SN(P + Q) \leq \overline{SN}(P) + \overline{SN}(Q)$. We conclude that if we replace S_0 by $(S_0)^{9D}$, then $L(SN(P_0)) \leq L(\overline{SN}(P_0)) = O((2t(D-1) + 1)L((S_0)^{9D})^t)$. The seminorm of the sum of 2^t terms each of length t , is at most 2^t times the maximum seminorm of word of length t . Since $S_0 \geq 2$, we can conclude that

$$L(S_M) \leq O(L(((S_0)^{9D})^{(2(D-1)Length(T,D)+1)Length(T,D)})).$$

This allows us to conclude that

Corollary: 7.7 *The complexity of TM1 is at worst*

$$O(D^8(L(S_0))^2 \cdot [D(2(D-1)Length(T, D) + 1)Length(T, D)]^2 \cdot P(T, D)).$$

here

$$P(T, D) = 170T^2 + 32D^2 + 14 \quad \text{and} \\ Length(T, D) = 2^{\delta_{dh}9T} 2T(34D^2 + 1)^9 + \delta_{ih}2^{(13T)^2}$$

where

$\delta_{dh} = 1$ when the initial generators are a pair of hyperbolics with disjoint axes and 0 otherwise, and

$\delta_{ih} = 1$ when the initial generators are a pair of hyperbolics with intersecting axes and 0 otherwise.

Proof: This follows from lemma 7.5 and corollary 6.4. *QED*

Theorem: 7.8 *Assume that the entries of the matrices lie in a finite simple extension $\mathbf{Q}(\gamma)$ of degree D . Let the minimal polynomial for γ , M_γ , and the representing polynomials $R_{\alpha_i, \gamma}$ be a normalized set of input data. Let S_0 be the maximum initial seminorm and D the degree of M_γ . The complexity of TM1 is at worst*

$$O(D^{14}(S_0)^2(L(S_0))^2 \cdot [2^{\delta_{dh}144(S_0D)^2}(S_0)^8 D^{80} + \delta_{ih}2^{10,816 \cdot (S_0D)^4}])$$

where

$\delta_{dh} = 1$ when the initial generators are a pair of hyperbolics with disjoint axes and 0 otherwise, and

$\delta_{ih} = 1$ when the initial generators are a pair of hyperbolics with intersecting axes and 0 otherwise.

For Jørgensen's inequality similarly implemented the complexity is

$$O(D^8(L(S_0))^2).$$

Proof: Use lemma 7.6 to write T in terms of S_0 and D , substitute the formulae for $P(T, D)$ and $L(T, D)$, replace $D(2(D-1)Length(T, D)+1)(Length(T, D))^2$ by $D(D[Length(T, D)]^2)$ to obtain:

$$O(D^8(L(S_0))^2 \cdot D^4 \cdot [(2^{\delta_{dh} 9(4S_0 D)^2} (S_0 D)^2 (34D^2 + 1)^9 + \delta_{ih} 2^{(169 \cdot (16(S_0 D)^4)})]^4 \cdot (170 \cdot 4(S_0 D)^2 + 32D^2)).$$

We use the facts that $S_0 \geq 2$, $D \geq 1$, and $O((a+b)^4) = O(a^4 + b^4)$ as long as $a \geq 1$ and $b \geq 1$. *QED*

If we begin with a pair of matrices one of which is not hyperbolic, then the Turning machine algorithm is polynomial in S_0 and D . If the matrices are hyperbolic with disjoint axes, then the algorithm is exponential in S_0 and D with the exponential term of the form $2^{144(DS_0)^2}$. If both matrices are hyperbolic with intersecting axes, then the algorithm is exponential in S_0 and D with the exponential term of the form $2^{10,816(DS_0)^4}$.

Thus in moving from the real number algorithm to this Turing machine implementation, Jørgensen's inequality goes from constant complexity to polynomial complexity and the Gilman-Maskit algorithm goes from polynomial complexity to exponential complexity. Ideally one would like to obtain polynomial complexity for *TM1*. It is possible that the exponential growth of the lengths of the words one considers in the two initial generators can be avoided either by finding a better algorithm for pairs of hyperbolics or a better analysis of the existing algorithm. A better analysis of the relationship between the maximal initial trace, T , and the seminorm might also improve the estimate. Since trace minimizing does not necessarily imply seminorm minimizing, our complexity estimates do not use the full force of the fact that this is a trace minimizing algorithm. This fact is essential to the algorithm. It is what forces the algorithm to stop.

Acknowledgements

A number of people have helped me in the preparation of this manuscript, especially in learning the necessary theory of real number algorithms and symbolic computation. I want to thank Mike Shub who spent time explaining BSS machines, nodes, and oracles to me and Perry Susskind with whom I had some useful conversations. I also want to thank Professor George Collins, Professor R. Loos, Dr. Georg Hagel and Professor Lars Langemyr with whom I had helpful e-mail conversations about the complexity of certain symbolic computations.

I also owe thanks to Gerhard Rosenberger who first introduced me to the two generator discreteness problem. The idea of a trace minimizing algorithm is due to him (see [17] and the references given there). Our geometric algorithm builds upon his algebraic algorithm and Matelski's geometric treatment [13].

References

- [1] A. Aho, J. Hopcroft, and J. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley Publishing Co., (1974).
- [2] L. Blum, M. Shub, and S. Smale, On a Theory of Computation and Complexity over the Real Numbers: NP-Completeness, Recursive Functions and Universal Machines, *Bulletin of the AMS* **21(1)**, (1989), 1-46.
- [3] A. Borodin and L. Munro, The Computational Complexity of Algebraic and Numeric Problems, Theory of Computation Series **1**, Elsevier Pub. Co., NY (1975).
- [4] G.E. Collins, Computer Algebra of Polynomials and Rational Functions, *Amer Math. Mon.* **80/7**, (1973) 725-755.
- [5] Collins, G.E. and Loos, R. Real Zeros of Polynomials in *Computer Algebra: symbolic and algebraic computation*, edited by B. Buchburger, G.E. Collins, and R.Loos, Springer-Verlag, NY, (1983).
- [6] J. Gilman, Two-generator discrete subgroups of $PSL(2, R)$, *Memoirs of the AMS*, Volume 117, No. 561, (1995).

- [7] J. Gilman and B. Maskit, An algorithm for two-generator discrete groups, *Mich. Math. J.*, **38** (1), (1991), 13-32.
- [8] T. Jorgensen, On discrete groups of Mobius transformations, *Amer. J. Math* **98** (1976), 739-749.
- [9] Kaltofen, E, Factorization of Polynomials, in *Computer Algebra: symbolic and algebraic computation*, edited by B. Buchberger, G.E. Collins, and R.Loos, Springer-Verlag, NY, (1983).
- [10] R. Loos, Computing in Algebraic Extensions, in *Computer Algebra: symbolic and algebraic computation*, Ed. by B. Buchberger, C.E. Collins, and R. Loos, Springer-Verlag 1982.
- [11] R. Loos, Generalized Polynomial Remainder Sequences, in *Computer Algebra: symbolic and algebraic computation*, Ed. by B. Buchberger, C.E. Collins, and R. Loos. Springer-Verlag 1982.
- [12] Computing Rational zeros of integral polynomials by p-adic expansion, *SIAM. J. Comput.*, **12** (no.2), (1983), 286-293.
- [13] J.P. Matelski, The classification of discrete 2-generator subgroups of $PSL(2, \mathbb{R})$ *Israel J. Math.* **42** (1982), 309-317.
- [14] Ivan Niven, Numbers: Rational and Irrational, Mathematical Association of America, (1961).
- [15] R. Riley, Applications of a Computer Implementation of Poincare's theorem on Fundamental Polyhedra *Math. Computation* **40** (1983), 633-646.
- [16] R. Riley, In preparation.
- [17] G. Rosenberger, All generating pairs of all two-generator Fuchsian groups, *Arch. Math.* **46** (1986), 198-204.
- [18] S. Winograd, *Arithmetic Complexity of Computations*, CBMS-NSF series in Applied Math, **33**, Philadelphia, (1980).

Mathematics Department, Rutgers University, Newark, NJ 07102
e-mail: gilman@andromeda.rutgers.edu