Genus Zero Systems for Primitive Groups of Affine Type

by

Gehao Wang

A thesis submitted to
The University of Birmingham
for
Doctor of Philosophy

School of Mathematics
The University of Birmingham
December 2011

Abstract

Let $\mathcal{M}_g$ be the moduli space of genus g curves. A Hurwitz locus in $\mathcal{M}_g$ is a locus of points representing G-covers of fixed genus g with a given ramification type. The classification of Hurwitz loci of complex curves admitting G is by the computation of orbits of a suitable surface braid group acting on the generating tuples of G. When the genus of the curve is low, the braid orbits can be enumerated explicitly using GAP (Groups, Algorithm, Programming) computer algebra system and the BRAID package by Magaard, Shpectorov and Völklein. However, the length of the orbits dramatically increases with the size of G and genus of the curve. In order to handle larger orbits, we propose to break up the tuples into two or more shorter pieces which can be computed within reasonable time, and then recombine them together as direct products to form the braid orbits.

Acknowledgements

I would like to thank my supervisors, Dr. Kay Magaard and Prof. Sergey Shpectorov, whose encouragement, guidance and support during my research study enabled me to develop an advanced understanding on this subject, and gain more experience on the programming.

In Addition, I would like to thank my colleagues in the maths postgrad society, who provided a lot of examples and sources about LateX and file style on the website, especially Matthew Badger, whose Mphil thesis is where I first started my study on this project, and Kieran Roberts who helped me so much during my Ph.D study.

Lastly, I am heartily thankful to my family, whose emotional and financial support is the main energy that keep me going.

Contents

Chapter 1
Introduction

A Riemann surface X is a one dimensional complex manifold. Topologically, Riemann surfaces are geometric objects that look like a sphere, a torus, or several tori which have been glued together. The number of tori is called the genus, which we denote by g in this thesis. The automorphisms of a Riemann surface X are the selfmaps which preserve the complex structure. They form the group $Aut(X)$, which is a finite group provided that $g \geqslant 2$. Let G be a finite subgroup of $Aut(X)$. The orbit space X/G is also a Riemann surface, and the natural mapping $X \to X/G$ is meromorphic, in which case we say X is a G-cover. For a fixed genus g, the moduli space $\mathcal{M}_g$ is a quasi projective variety in which every point represents an isomorphism class of Riemann surfaces of genus g. A Hurwitz locus in $\mathcal{M}_g$ is a locus of points representing the surfaces of fixed genus g admitting the group G. Hurwitz loci in $\mathcal{M}_g$ can be parametrized by mapping class group orbits of Nielsen class of G. We are mostly interested in the case when X/G is the Riemann sphere $\mathbb{P}^1$. In this case the mapping class group is the braid group. In Chapter 2 we will introduce the Hurwitz space which is the moduli space of G-covers. Hurwitz spaces are used to study $\mathcal{M}_g$. For example, Hurwitz himself showed the connectedness of $\mathcal{M}_g$ by first showing that every cover admits a simple cover over the Riemann sphere, and then showing that the Hurwitz space of simple covers is connected. In our case, we will give the construction of $\mathcal{H}_r^{(A)}(G)$, which is the Hurwitz space of A-equivalence classes of G-covers of $\mathbb{P}^1$ with r branch points, where A is a fixed subgroup of $Aut(G)$. In particular, we will be looking at $\mathcal{H}_r^{in}(G)$, where in denotes the inner automorphism group $Inn(G)$. Let $\mathcal{O}_r$ be the topological space of subsets of $\mathbb{C}$ of cardinality r. The Hurwitz space $\mathcal{H}_r^{(A)}(G)$ is an unramified covering space of the base space $\mathcal{O}_r$, and the fundamental group of $\mathcal{O}_r$ is in fact the braid group B_r. So by the covering space theory, the group action of B_r on the fibers completely determines the connected components of $\mathcal{H}_r^{(A)}(G)$. Before getting to know this Hurwitz space, we will give a brief introduction to some basic facts in covering space theory. In the final two sections of Chapter 2, we will talk about the braid group action.

Definition 1.1 *Let $\sigma \in S_n$. The permutation index of σ, denoted by $Ind(\sigma)$, is the smallest number of transpositions needed in the expression of σ as a product.*

From covering space theory we know that every G-cover of the Riemann sphere corresponds to a generating tuple $(g_1, \dots, g_r)$ of G with product 1, where we use g_i to denote the group elements in G. Riemann's Existence Theorem asserts the converse, that is, for every transitive subgroup G of S_n, every generating tuple $(g_1, \dots, g_r)$ of G with product 1 corresponds to a Riemann surface X, such that the monodromy group for the covering $X \to X/G$ is G. The genus of X depends only on the tuple $(g_1, \dots, g_r)$. This is given by the Riemann-Hurwitz formula

$$2(n+g-1) = \sum_{i=1}^{r} Ind(g_i).$$

We define a **genus zero system** to be a generating tuple $(g_1, \dots, g_r)$ of a transitive group G such that $g_1 \dots g_r = 1, g_i \neq 1$ and

$$2(n-1) = \sum_{i=1}^{r} Ind(g_i).$$

The set $\overline{C} = \{C_1, \dots, C_r\}$ of conjugacy classes of G such that $g_i \in C_i$ is called the **ramification type** of the cover. Via fixing the ramification type, we can focus on the specific subset $\mathcal{H}_r^{in}(\overline{C})$ of $\mathcal{H}_r^{in}(G)$ for our study. We define the Nielsen class $\mathcal{N}(\overline{C})$ to be

$$\mathcal{N}(\overline{C}) := \{(g_1, \dots, g_r) | \quad g_i \in C_i, G = \langle g_1, \dots, g_r \rangle, \prod_{i=1}^{r} g_i = 1\}.$$

The fibers of $\mathcal{H}_r^{in}(\overline{C})$ are parametrized by the Nielsen class $\mathcal{N}(\overline{C})$. The subgroup of B_r that preserves the order of $\overline{C}$ is called the parabolic subgroup B. This means that the connected components of $\mathcal{H}_r^{in}(\overline{C})$ are parametrized by the orbits of B on the Nielsen class.

After the discussion on the Hurwitz spaces, we can see that the classification of the Hurwitz loci is equivalent to a problem of computing the braid orbits of generating tuples of G. Our computation is based on the computer algebra system GAP and the accompanying BRAID package. Via restricting the genus of the cover, we can expect a reasonable length of possible ramification types. In Chapter 3, we will first discuss some previous work done, and explain the importance of our work on the genus zero covers and affine primitive groups. Thanks to the following theorem from Neubauer [15], we only have a small number of affine primitive groups to deal with.

Theorem 1.2 *If $F^*(G)$ is elementary abelian of order p^e and $X = P^1\mathbb{C}$, then one of the following is true:*

1. *$G'' = 1$ and $1 \leqslant e \leqslant 2$*
2. *$p = 2$ and $2 \leqslant e \leqslant 8$,*
3. *$p = 3$ and $2 \leqslant e \leqslant 4$,*
4. *$p = 5$ and $2 \leqslant e \leqslant 3$,*
5. *$p = 7$ or 11 and $e = 2$.*

In order to compute all braid orbits for affine primitive groups, it is necessary to collect all the ramification type $\overline{C}$ such that the tuples of type $\overline{C}$ satisfy the Riemann-Hurwitz formula, and $\overline{C}$ contains at least one generating tuple. Based on the algorithm described in [17], we wrote a modified function **GeneratingTypes** for this computation, which is introduced in Section 5. The group $\mathrm{AGL}(8, 2)$ is too large to be dealt in our standard algorithm, which requires some extra methods. We explain some manual work done on this case in Section 6.

When the length of the tuples grows, the size of orbits increases dramatically. The restrictions of the BRAID package requires us to improve the computation both theoretically and computationally. So we propose a splitting method, which reduces the time of computing the orbits, but requires more work on putting orbits back together. In Chapter 4, we will introduce this new method. First, some background results are stated and proved in Section 1. The first stage of our new algorithm is to split the tuples into two (or more) parts. Then it computes all the parts, and match the orbits of the parts back together, thereby to enumerate all orbits. With this matching algorithm, we manage to enumerate orbits of magnitude k^2 where k is an upper bound for what our standard BRAID package can handle. We will explain in detail about how our functions work in Section 2.

We present the results from computing the genus zero systems for affine primitive groups that satisfy Theorem 1.2 (2)-(5) in the appendix. Using our algorithm and a process called translation [8], we found no genus zero system for affine primitive groups of degree 128 and 256. Strictly speaking, the matching algorithm is not needed to settle the classification of braid orbits of primitive affine genus zero systems. However, it is a good test case for our algorithm both as a debugging tool and a comparison of speed. In the final chapter, we will give some examples of using the matching algorithm to compute orbits, and compare the results from using both original BRAID package and the matching algorithm on some other cases, followed by some possible improvements in the future work.

Theorem 1.3 *The possible types of genus zero systems for the affine primitive groups that satisfy Theorem 1.2 (2)-(5) are listed in Table B.1-10.*

Chapter 2

Background knowledge

In this chapter, we cover some background knowledge to this thesis. For a Riemann surface X and a subgroup G of its automorphism group $Aut(X)$, the natural mapping $X \to X/G$, from X to the orbit space X/G, is a meromorphic function, and can be seen as a covering if we exclude some points called branch points. This important fact enables us to work on the monodromy group of the covering, which is a subgroup of S_n, from the monodromy action of the fundamental group of the base space. We first introduce the idea of the monodromy action of a covering.

2.1 Covering Spaces and Monodromy Groups

Definition 2.1 *Let R and S be two topological spaces. A* **covering** *$f : R \to S$ is a surjective map that has the following property: each point $p \in S$ has a connected neighborhood U such that the connected components of $f^{-1}(U)$ are open in R, and f maps each connected component of $f^{-1}(U)$ homeomorphically to U.*

Definition 2.2 *A* **covering space** *(or* **cover***) of S is a pair (R, f), where $f : R \to S$ is a covering.*

Definition 2.3 *We say two covers (R, f) and $(R^{'}, f^{'})$ are isomorphic if there exists a homeomorphism $\gamma : R \to R^{'}$ such that $f^{'} \circ \gamma = f$.*

Recall that for a topological space X and a fixed point $x_0 \in X$, the fundamental group $\pi_1(X, x_0)$ is the set of all homotopy classes of loops based on x_0 in X. An important application of the fundamental group in covering space theory comes from a well-known fact that, the set of isomorphism classes of connected coverings of S are in one-to-one correspondence with the subgroups of the fundamental group $\pi_1(S, p)$. The covering f induces another map from the fundamental group $\pi_1(R, b)$ of R to the fundamental group $\pi_1(S, p)$ of the base space S. The following theorem can be found in [14], page 154.

Theorem 2.4 *Let $f : R \to S$ be a covering, and $b \in R, p \in S$ with $f(b) = p$. Then, the induced map $f_* : \pi_1(R, b) \to \pi_1(S, p)$ is a monomorphism.*

Definition 2.5 *Let $f : R \to S$ be a covering and γ be a path in S. A lift $\tilde{\gamma}$ of γ in R is a path satisfying $f \circ \tilde{\gamma} = \gamma$.*

For $p \in S$, the pre-image $f^{-1}(p) \in R$ is called the **fiber** of p. We have a very useful result as follows.

Proposition 2.6 *Let γ be a path in S with initial point p. For every point $a \in f^{-1}(p)$, there exists a unique lift $\tilde{\gamma}$ with initial point a.*

The complete proof can be found in [18], page 65. This proposition also implies that, if f is a covering and S is connected, we can define a bijection between any two different fibers. So all the fibers have the same cardinality. We call this cardinality the **degree** of the covering f. It can be infinite, but we are mostly interested in the finite cases. Furthermore, the homotopy lifting gives us an action of the fundamental group $\pi_1(S, p)$ on the fiber $f^{-1}(p)$. The lift of a loop $\gamma \in \pi_1(S, p)$ with initial point $b \in f^{-1}(p)$ maps b to another point in $f^{-1}(p)$, which is the end point of its lift $\tilde{\gamma}$. This is known as the **monodromy action**. We denote the end point of $\tilde{\gamma}$ by γb.

Definition 2.7 *The monodromy action of the fundamental group $\pi_1(S,p)$ on each fiber gives a homomorphism*

$$\rho : \pi_1(S,p) \to S_n,$$

where S_n is the symmetric group and n is the degree of $f : R \to S$. The image of ρ is called the **monodromy group** *of f.*

Lemma 2.8 *Suppose the fiber $f^{-1}(p)$ is $\{b_1, b_2 \dots b_n\}$. The monodromy group G of the covering f is isomorphic to the quotient group $\pi_1(S,p)/\cap_{j=1}^{n} Stab(b_j)$, where $Stab(b_j)$ is the stabilizer of b_j, $1 \leqslant j \leqslant n$.*

Proof. From the map $\rho : \pi_1(S,p) \to S_n$ we can see that, the kernel of ρ is the elements in $\pi_1(S,p)$ such that their lifts are always loops on b_j, and hence it is $\cap_{j=1}^{n} Stab(b_j)$. $\square$

Definition 2.9 *A* **deck transformation** *of a covering $f : R \to S$ is a homeomorphism $\alpha : R \to R$, such that $f\alpha = f$. The set of all the deck transformations of f forms a group. We denote it by $Deck(f)$.*

By the definition, for each $p \in S$, the image of a point b in $f^{-1}(p)$ under any deck transformation is still in $f^{-1}(p)$. So the group $Deck(f)$ also acts on the fiber $f^{-1}(p)$. Let $\gamma \in \pi_1(S,p)$, and $\tilde{\gamma}$ be its lift with initial point b. Then the end point is γb. The initial point of the path $\alpha \circ \tilde{\gamma}$ is $\alpha(b)$, and the endpoint is $\alpha(\gamma b)$. From the monodromy action we know that the lift of γ starting from $\alpha(b)$ has end point $\gamma\alpha(b)$, and also $f\alpha \circ \tilde{\gamma} = f\tilde{\gamma} = \gamma$ by the definition of the deck transformation. So $\alpha(\gamma b) = \gamma\alpha(b)$. Hence the action of $Deck(f)$ commutes with the monodromy action of $\pi_1(S,p)$. This also implies the following result.

Corollary 2.10 *The connected components of the covering space R of S are in bijective correspondence with the orbits of $Deck(f)$ on the fiber $f^{-1}(p)$ for any $p \in S$.*

Definition 2.11 *We say $f : R \to S$ is a Galois covering, if R is connected, and $Deck(f)$ acts transitively on each fiber. We say f is finite, if the degree of f is finite.*

The Galois covering f has a unique surjective homomorphism from the fundamental group of its base space S to the deck transformation group $Deck(f)$. This is given by the following proposition in [18], page 69.

Proposition 2.12 *Let $f : R \to S$ be a Galois covering. For a fixed $b \in R$, let $p = f(b)$. Then there exists a unique surjective homomorphism*

$$\Phi_b : \pi_1(S, p) \to Deck(f),$$

such that the image of a loop class $[\gamma]$ based on $p \in S$ maps the endpoint $[\gamma]b$ of the lift to its initial point b.

Corollary 2.13 *Let $[\gamma] \in \pi_1(S, p)$, and the end point of its lift is $b^{'} \in f^{-1}(p)$. Denote the kernel of Φ_b by $\ker \Phi_b$. Then $[\gamma] \cdot \ker \Phi_b \cdot [\gamma]^{-1} = \ker \Phi_{b'}$.*

Proof. For any $[\delta] \in \ker \Phi_b$, the lift of δ is a loop based on b. So the lift of all the elements in the class $[\gamma\delta\gamma^{-1}]$ is a loop based on $b^{'}$. Hence $[\gamma] \cdot \ker \Phi_b \cdot [\gamma]^{-1} \subseteq \ker \Phi_{b'}$. Similarly, $[\gamma]^{-1} \cdot \ker \Phi_{b'} \cdot [\gamma] \subseteq \ker \Phi_b$. This completes the proof. □

If the center of the group $Deck(f)$ is trivial, then it is easy to see that $\Phi_b = \Phi_b^{'}$ if and only if $b = b^{'}$. Hence the covering $f : R \to S$ can be represented by (p, Φ_b). We will use this property to construct the Hurwitz space for our study in Section 4. Using the next lemma, we will focus on the monodromy group associated with the covering f instead of the deck transformation group for the rest of the thesis.

Lemma 2.14 *If $f : R \to S$ is a Galois covering, then $f_*\pi_1(R, b)$ is a normal subgroup of $\pi_1(S, p)$, and the monodromy group G isomorphic to $Deck(f)$.*

Proof. First, we show that

$$Deck(f) \cong Norm(f_*\pi_1(R, b))/f_*\pi_1(R, b),$$

where $Norm(f_*\pi_1(R,b))$ denotes the normalizer of $f_*\pi_1(R,b)$ in $\pi_1(S,p)$.

Let $\gamma \in Norm(f_*\pi_1(R,b))$, and the lift of γ with initial point b has endpoint γb. Then for any $\delta \in f_*\pi_1(R,b)$, the lift of $\gamma\delta\gamma^{-1}$ will fix γb. Hence we have

$$f_*\pi_1(R,\gamma b) = Stab(\gamma b) = \gamma f_*\pi_1(R,b)\gamma^{-1} = f_*\pi_1(R,b).$$

This gives us a homomorphism $Norm(f_*\pi_1(R,b)) \to Deck(f)$ with kernel $f_*\pi_1(R,b)$.

If f is a Galois covering, then $Deck(f)$ is transitive on the fiber $f^{-1}(p)$. Since the action of $Deck(f)$ and monodromy action commute, for any $\gamma \in \pi_1(S,p)$ and $\delta \in f_*\pi_1(R,b)$, the lift of $\gamma\delta\gamma^{-1}$ will have endpoint $\gamma b \in f^{-1}(p)$. Hence $Norm(f_*\pi_1(R,b)) = \pi_1(S,p)$. This shows that $Deck(f) \cong \pi_1(S,p)/f_*\pi_1(R,b) = G$. □

2.2 Riemann surfaces

In this section we define Riemann surfaces and collect some basic facts. Riemann surfaces can be defined in several ways, mainly by using topology, analytic sheaf theory and algebraic curve theory. Here we give the version of the definition via topology and analytic function theory. We start with some basic terminology.

Definition 2.15 *Let X be a topological space, and U be a open set of X. We define the* **chart** *to be a pair (U,h), where h is a homeomorphism $h : U \to V$, U is its domain, and V is an open set of $\mathbb{C}$. The chart is also known as the coordinate map.*

Definition 2.16 *A complex atlas on X is a collection of charts $\{(U_j,h_j) : j \in I\}$ satisfying:*

1. *$\cup_{j\in I} U_j = X$;*
2. *for any two open sets U_j and U_k with $U_j \cap U_k \neq \emptyset$, the map $h_j \circ h_k^{-1}$ is holomorphic. In this case we say the two charts h_j and h_k are compatible.*

We say that two complex atlases are equivalent if every chart of one complex atlas is compatible with every chart in the other one.

Definition 2.17 *A Hausdorff space E is a topological space such that any two distinct points of E can be separated by neighborhoods.*

Definition 2.18 *A Riemann surface X is a Hausdorff topological space associated with a complex atlas.*

Example 2.19 *The* **Riemann sphere** $\mathbb{P}^1 = \mathbb{C} \cup \{\infty\}$ *is a particular Riemann surface. We can give an atlas as follows. Consider $U_1 = \mathbb{C}, U_2 = \mathbb{C}^* \cup \{\infty\}$, where $\mathbb{C}^* = \mathbb{C} \setminus \{0\}$. We define two charts by*

$$w_1 : U_1 \to \mathbb{C}, z \to z;$$

$$w_2(z) = \begin{cases} \frac{1}{z} & \text{if } z \in C^* \\ 0 & \text{if } z = \infty \end{cases}$$

Definition 2.20 *Suppose R and S are Riemann surfaces, and $f : R \to S$ is a continuous function. We say f is analytic, if it satisfies the following. For two charts (U, z) in R and (U', z') in S with $f(U) \subset U'$, the map $z' f z^{-1} : z(U) \to z(U')$ is holomorphic.*

Riemann surfaces form a category with morphisms being the analytic functions. An analytic function f between two Riemann surfaces R and S is usually not a covering due to the existence of branch points b in S. However, without these exceptional points, we still find an honest covering between R and S. In other words, for a non-constant analytic function $f : R \to S$, f is a covering map onto its image except some points $b \in S$, where the cardinality of $f^{-1}(b)$ is strictly less than the degree of the covering map f. Now, we give the following definitions.

Definition 2.21
1. *For a holomorphic function $f : R \to \mathbb{C}$ and $p \in \mathbb{P}^1$, we say p is a pole of f if $f(z)$ approaches to infinity when z approaches to p.*

2. *We say a function $f : R \to \mathbb{C}$ is* **meromorphic** *at a point $p \in \mathbb{P}^1$, if it is holomorphic at p or p is a pole.*

3. *We say a non-constant analytic function $f : R \to \mathbb{P}^1$ is a* **meromorphic function***.*

4. *A* **branched cover** *is a pair (R, f), where $f : R \to \mathbb{P}^1$ is a meromorphic function. We say f is* **branched covering** *(or ramified covering).*

We are mainly interested in the compact Riemann surfaces as the branched covers of the Riemann sphere. The following theorem supports our assumption that the coverings we are interested in have finite degree.

Theorem 2.22 *Let Y be a compact Riemann surface, and $\phi : Y \to \mathbb{P}^1$ be a meromorphic function. Then the fiber $\phi^{-1}(p)$ over $p \in \mathbb{P}^1$ is finite.*

Proof. The zeros and poles of a meromorphic function on Y form a discrete subset of Y. Since Y is compact, the number of zeros and poles on Y is finite. Let w_1 and w_2 be the two maps as in Example 2.19. Define the meromorphic function f to be $w_1 \circ \phi - w_1$ on $\mathbb{C}$ and $w_2 \circ \phi$ on ∞. Then the fiber $\phi^{-1}(p)$ is the set of zeros when $p \neq \infty$ or the set of poles when $p = \infty$. Hence $\phi^{-1}(p)$ is finite. □

2.3 Riemann's Existence Theorem

We know that a meromorphic function f from a Riemann surface R to $\mathbb{P}^1$ is usually not a covering due to the existence of branch points. Without these points, the map f behaves like a covering map, and certainly carries out some useful properties about covering space. In this section we will first look at coverings of punctured sphere, that is, the Riemann sphere $\mathbb{P}^1$ without a finite number of points, and talk about how the elements in the fundamental group of the punctured sphere correspond to the group elements in the monodromy group G. This leads us to the Riemann's Existence Theorem.

We define a punctured disk $\mathbb{K}(r)$ in $\mathbb{C}$ to be the set $\{z \in \mathbb{C} : |z| < r\}$ for any $r > 0$.

Lemma 2.23 *The covering $f_e : \mathbb{K}(r^{\frac{1}{e}}) \to \mathbb{K}(r), z \mapsto z^e$ is Galois. And the monodromy group G is cyclic of order e, $e \in \mathbb{N}$.*

Proposition 2.24 *Let $f : E \to \mathbb{K}(r)$ be a covering with degree e and E being connected. Then f is equivalent to the covering $f_e : \mathbb{K}(r^{\frac{1}{e}}) \to \mathbb{K}(r), z \mapsto z^e$,$e \in \mathbb{N}$.*

Proof. Here we give a brief construction of a homeomorphism between E and $K(r^{\frac{1}{e}})$. The complete proof can be found in [18], page 70. It is easy to see that the degree of f_e is also e. Let $p \in \mathbb{K}(r)$ be a base point. We know that the fundamental group $\pi = \pi_1(\mathbb{K}(r), p)$ is infinite cyclic, generated by some circle based at p. By the orbit-stabilizer theorem, for any $b \in f^{-1}(p)$ and $b' \in f_e^{-1}(p)$, the stabilizers π_b and $\pi_{b'}$ have the same index e. Hence $\pi_b = \pi_{b'}$. Now we define $\alpha : E \to K(r^{\frac{1}{e}})$ as follows. Let δ be a path in E with initial point $b \in E$, and $f \circ \delta = \gamma \in \pi_1(\mathbb{K}(r), p)$. Denote lift of γ through f_e with initial point b' by δ'. The bijection α maps b to b' together with γb to $\gamma b'$. This construction guarantees that $f \circ \alpha = f_e$, and α is homeomorphic. □

Let $P = \{p_1, p_2, \ldots, p_r\}$ be a set of branch points in $\mathbb{P}^1$. We define the open sets of $\mathbb{P}^1$ to be the open discs $D(p, r)$, centered at $p \in \mathbb{C}$ with radius $r > 0$. If $p = \infty$, then we define the open disc as $D(p, r) = \{z \in \mathbb{C} : |z| > r^{-1}\} \cup \{\infty\}$. This gives us a topology on the punctured sphere $S = \mathbb{P}^1 \setminus P$. Suppose $f : R \to \mathbb{P}^1 \setminus P$ is a finite Galois covering, and G is its monodromy group. The above proposition tells us that around each branch point, the covering f behaves like $z \to z^e$. We fix p to be a branch point in P, and assume that the disc $D(p, r)$ around p does not contain any other branch points. The following two propositions can be found at Proposition 4.23 in [18].

Proposition 2.25 *Let $D^* = D(p, r) \setminus \{p\}, r > 0$. Then the group G permutes the components of $f^{-1}(D^*)$ transitively. If E is one of the components, and G_E is the stabilizer of E in H. Then G_E is cyclic.*

If $g_E \in G_E$ generates G_E, we call g_E the **canonical generator**, and denote the conjugacy class in G containing g_E by C_p labeled by the branch point p. The next proposition describes how the elements in the fundamental group of the punctured sphere correspond to the group elements in the conjugacy classes of G.

Proposition 2.26 *Suppose $b \in R$ and $q_0 = f(b)$. Let p^* be any point in D^*, and λ be a closed path based on p^*. Moreover, let δ be a path joining p^* and q_0. So $\gamma = \delta^{-1}\lambda\delta$ is a representative of elements in $\pi_1(\mathbb{P}^1 \setminus P, q_0)$. The map $\Phi_b : \pi_1(\mathbb{P}^1 \setminus P, q_0) \to G$ sends the homotopy class of γ to a group element in C_p.*

In the group $\pi_1(\mathbb{P}^1 \setminus P, q_0)$, the conjugacy class corresponding to a branch point $p \in P$ is defined in the following way. First, we choose a disc $D(p, s)$ around p with $s > 0$, such that $D(p, s)$ does not contain other branch point in P. Then we choose a path δ from q_0 to some point v on the boundary of $D(p, s)$, and denote the closed path starting from v and winding once counterclockwise along the disc $D(p, s)$ by λ. The set of loops $\delta^{-1}\lambda\delta$, when δ and v vary, forms a conjugacy class corresponding to p, . We denote such conjugacy class by Σ_p. The surjective homomorphism $\phi : \pi_1(\mathbb{P}^1 \setminus P, q_0) \to G$ is called **admissible**, if $\phi(\Sigma_p) \neq 1$. The next corollary follows directly from Proposition 2.26.

Corollary 2.27 *The surjective homomorphism $\Phi_b : \pi_1(\mathbb{P}^1 \setminus P, q_0) \to G$ sends Σ_p to C_p in G.*

Definition 2.28 *Let $f : R \to \mathbb{P}^1 \setminus P$ be a finite Galois covering. We define the* **ramification type** *$\overline{C} = \{C_{p_1}, C_{p_2}, \dots, C_{p_r}\}$ of f to be the set of non-trivial conjugacy classes in G labeled by the points in P.*

Corollary 2.29 *Let $f : R \to \mathbb{P}^1 \setminus P$ be a finite Galois covering, and h be a homeomorphism on $\mathbb{P}^1$. Then $f' = h \circ f$ is a finite Galois covering with $Deck(f') = Deck(f)$.*

Proof. Let α be an element $Deck(f)$. By the definition of deck transformation, $f \circ \alpha = f$. Then $f' \circ \alpha = h \circ f \circ \alpha = h \circ f = f'$, and vice versa. Hence $Deck(f') = Deck(f)$ □

Let $h_1 : z \to \frac{1}{z}$ and $h_2 : z \to z + z_0, z_0 \in \mathbb{C}, z_0 \neq 0$ be the two homeomorphisms on $\mathbb{P}^1$. Then $f_i = h_i \circ f : R \to \mathbb{P}^1 \setminus \{h_i(p_1), h_i(p_2), \ldots, h_i(p_{r-1}), h_i(p_r)\}$ is a finite Galois covering with $Deck(f_i) = Deck(f)$, for $i = 1, 2$. Suppose that 0 and ∞ are both in the branch point set P. Then $h_1 \circ h_2(P)$ is a branch point set not containing ∞. From now on, we do not allow ∞ to be a branch point. Instead, we can choose ∞ to be the base point of the fundamental group of $\mathbb{P}^1$. This will simplify our discussion on Galois coverings later, but is not a real restriction.

Lemma 2.30 *Let $p_1, p_2, \ldots, p_r$ be r distinct points in $\mathbb{C}$, and $X = \mathbb{P}^1 \setminus \{p_1, p_2, \ldots, p_r\}$. For $1 \leqslant i \leqslant r$, we choose a disc D_i around each p_i small enough, so that the r discs $\{D_1, D_2, \ldots, D_r\}$ are disjoint pairwise. For $1 \leqslant i \leqslant r$, we choose a fixed point $b \in \mathbb{C}$ and a point x_i on the boundary of D_i satisfying the following condition. Let λ_i be the closed path based on x_i going along the boundary of D_i, and δ_i be the ray from x_i to ∞ passing through b. The r loops $\{\gamma_1, \gamma_2, \ldots, \gamma_r\}$, where $\gamma_i = \delta_i^{-1}\lambda_i\delta_i$, are disjoint pairwise. Then $\{\gamma_1, \gamma_2, \ldots, \gamma_r\}$ generates $\pi_1(X, \infty)$.*

The set of elements $\{\gamma_1, \gamma_2, \ldots, \gamma_r\}$ forms a canonical set of generators of $\pi_1(X, \infty)$, and under the homomorphism Φ_b, it gives a set of canonical generators $\{g_1, g_2, \ldots, g_r\}$ of the monodromy group G. We also say that $\{g_1, g_2, \ldots, g_r\}$ is a **generating tuple** of G.

Definition 2.31 *Let p be a branch point in P. Let D be a punctured disc around p not including other branch points. The fiber $f^{-1}(D)$ is a set of connected components E. For any smaller punctured disc D' in D, there is a one-to-one correspondence between the components E and the components E' in the fiber $f^{-1}(D')$, given by inclusion $E' \subseteq E$. We define this inclusion to be a equivalence relation, and the equivalence classes are called the* **ideal points** *over p.*

We can also extend the Galois covering f to a branched covering from Y to $\mathbb{P}^1$, where Y is the union of R and all the **ideal points** over $\mathbb{P}^1$. In other words, we think of those ideal points as "missing centers" over the branch points. By adding them to R, we form a branched cover, namely, G-cover, over the Riemann sphere. The details can be found in [18], page 88.

Theorem 2.32 (Riemann's Existence Theorem) *Let G be a finite group, P be a finite set of elements in $\mathbb{P}^1$, $P = \{p_1, p_2, \ldots, p_r\}$. Suppose $\overline{C} = \{C_{p_1}, C_{p_2}, \ldots, C_{p_r}\}$ is a ramification type. Then there exists a G-cover of type $\overline{C}$ if and only if there exists a generating tuple $(g_1, g_2, \ldots, g_r)$ of G with $g_1 g_2 \ldots g_r = 1$, and $g_i \in C_{p_i}$ for $1 \leqslant i \leqslant r$*

The section 4.2.3 in [18] gives a complete proof to this important theorem. This theorem guarantees us that there is always a one-to-one correspondence between the equivalence classes of generating tuples $(g_1, g_2, \ldots, g_r)$ of G and equivalence classes of G-covers of type $\overline{C}$ such that the canonical generator g_i lies in C_{p_i} for $1 \leqslant i \leqslant r$. We will discuss this later.

Example 2.33 *Let $\sqrt{1 + z^{\frac{1}{3}}} = y$. Then $(y^2 - 1)^3 = z$. Let the base space be the Riemann sphere $\mathbb{P}^1$ and $z \in \mathbb{P}^1$. The branch points in $\mathbb{P}^1$ for the covering $f : (y^2 - 1)^3 \to z$ are $\{0, -1, \infty\}$. Let $z = \frac{1}{2}$ be the base point of $\mathbb{P}^1$, $\varepsilon = \frac{1}{2} + \frac{\sqrt{3}}{2} i$ and $a = 2^{-\frac{1}{3}}$. Then we have the fiber for $\frac{1}{2}$ as*

$$f^{-1}(\frac{1}{2}) = \{\pm\sqrt{1+a}, \pm\sqrt{1+\varepsilon a}, \pm\sqrt{1+\varepsilon^2 a}\}.$$

Suppose the three loops γ_0, γ_{-1} and γ_∞ generate the fundamental group of base space. The Galois group permutes the roots as

$$g_1 : \sqrt{1+a} \to \sqrt{1+\varepsilon a} \to \sqrt{1+\varepsilon^2 a} \to \sqrt{1+a} \tag{2.34}$$

$$: -\sqrt{1+a} \to -\sqrt{1+\varepsilon a} \to -\sqrt{1+\varepsilon^2 a} \to -\sqrt{1+a}. \tag{2.35}$$

$$g_2 : \sqrt{1+a} \to -\sqrt{1+a}. \tag{2.36}$$

There exist $g_1 = (1,2,3)(4,5,6), g_2 = (1,4)$ and $g_3 = (g_1 g_2)^{-1}$ in the group $G = C_2 \wr C_3$, *such that*

$$\Phi_{\frac{1}{2}}(\gamma_0) = g_1, \Phi_{\frac{1}{2}}(\gamma_{-1}) = g_2, \Phi_{\frac{1}{2}}(\gamma_\infty) = g_3.$$

The corresponding Riemann surface can be constructed by gluing 6 sheets together along the lines passing through $0, 1$ *and* -1*, according to the monodromy action.*

2.4 The Hurwitz Space $\mathcal{H}_r^{(A)}(G)$

The results in this section are mainly from [18]. From now on we assume $Z(G) = 1$, and define $Inn(G)$ to be the inner automorphism group of G. From Corollary 2.13 we know that, as the center of G is trivial, $\Phi_b = \Phi_{b'}$ if and only if $b = b'$. In fact, if there exists $h \neq 1$ in G commuting with other elements in G, then for two points b and $h(b)$ in the fiber $f^{-1}(p)$, $\Phi_b = \Phi_{h(b)}$. If $f : R \to S$ is a Galois covering, then the points in R can be represented by a pair (p, Φ_b), where $f(b) = p$.

We let $\mathcal{O}_r$ be the space of branch point sets in $\mathbb{C}$ with cardinality r, that is,

$$\mathcal{O}_r = \mathbb{C}^r \setminus \{(p_1, \ldots p_r) \in \mathbb{C}^r \quad | \quad \text{there exist} \quad i \quad \text{and} \quad j \quad \text{with} \quad p_i = p_j\}.$$

It is an open set of the complex projective space of dimension r, and, if we define the determinant to be $\prod_{i \neq j}(p_i - p_j)$, then it is the complement of the discriminant locus. For $P \in \mathcal{O}_r$, the neighborhood of P consists of all the $P' = \{p_1', p_2', \ldots, p_r'\}$ such that p_i' is in some open disc D_i in $\mathbb{C}$ containing p_i for $1 \leqslant i \leqslant r$, and all the D_i's are disjoint. This defines a topology on the space $\mathcal{O}_r$.

Now we consider a pair (P, ϕ), where $P \in \mathcal{O}_r$, and $\phi : \pi_1(\mathbb{P}^1 \setminus P, q_0) \to G$ is an admissible surjective homomorphism. By Proposition 2.12, such pair (P, ϕ) represents a G-cover. We fix a subgroup A of the automorphism group $\mathrm{Aut}(G)$, and denote $[P, \phi]_A$ to be the equivalence class $\{(P, A \circ \phi)\}$ represented by (P, ϕ). All the equivalent classes form a set denoted by $\mathcal{H}_r^{(A)}(G)$, if we vary P in $\mathcal{O}_r$. A neighborhood of $[P, \phi]_A$ is considered to be the set of all the $[P^{'}, \phi^{'}]_A$, where $P^{'}$ is another branch point set in $\mathcal{O}_r$, and $\phi^{'}$ is the composition of ϕ and the isomorphism

$$\pi_1(\mathbb{P}^1 \setminus P^{'}, \infty) \to \pi_1(\mathbb{P}^1 \setminus (D_1 \cup D_2 \cdots \cup D_r), \infty) \to \pi_1(\mathbb{P}^1 \setminus P, \infty).$$

This gives us a topology on $\mathcal{H}_r^{(A)}(G)$. If $A = Inn(G)$, then our Hurwitz space is denoted by $\mathcal{H}_r^{in}(G)$.

Suppose an admissible ϕ_1 corresponds to the tuple $(g_1, g_2, \ldots, g_r)$. We call such tuple admissible. Let $a \in A$, then under the map $a \in A$, the tuple $(a(g_1), a(g_2), \ldots, a(g_r))$ is another admissible tuple corresponding to another cover with $\phi_2 = a \circ \phi_1$. These two covers are A-equivalent. Hence the class $[P, \phi]_A$ represents the equivalence class of G-covers with admissible tuples $(a(g_1), a(g_2), \ldots, a(g_r)), a \in A$. The next result from the Lemma 10.4 in [18] is the starting point of our investigations.

Proposition 2.37 *The map $\Psi_A : \mathcal{H}_r^{(A)}(G) \to \mathcal{O}_r$ sending $[P, \phi]_A$ to P is a covering.*

Via homotopy path lifting, the monodromy action of the fundamental group of $\mathcal{O}_r$ will completely determines the topology of the Hurwitz space $\mathcal{H}_r^{(A)}(G)$. If we fix a point P_0 in $\mathcal{O}_r$, the fiber $\Psi_A^{-1}(P_0)$ consists of all the pairs $[P_0, \phi]_A$, where ϕ is an admissible surjective homomorphism from $\pi_1(\mathbb{P}^1 \setminus P_0, \infty)$ to G sending the generator $\{[\gamma_i]\}$ of $\pi_1(\mathbb{P}^1 \setminus P_0, \infty)$ to a group element g_i in G for $1 \leqslant i \leqslant r$. By Proposition 2.26, ϕ is determined by the generating tuple $(g_1, \ldots, g_r)$ of G. We set

$$\mathcal{E}_r(G) = \{(g_1, g_2, \ldots, g_r) : G = \langle g_1, g_2, \ldots, g_r \rangle, g_1 g_2 \ldots g_r = 1, g_i \neq 1, 1 \leqslant i \leqslant r\}.$$

The group A acts on $\mathcal{E}_r(G)$ as, for $a \in A$, a sends $(g_1, \ldots, g_r)$ to $(a(g_1), a(g_2), \ldots, a(g_r))$. We denote $\mathcal{E}_r^{(A)}(G)$ to be the A-orbits on $\mathcal{E}_r(G)$. The next lemma is from Lemma 10.12 in [18].

Proposition 2.38 *There is a bijection between $\Psi_A^{-1}(P_0)$ and $\mathcal{E}_r^{(A)}(G)$ by sending $[P_0, \phi]_A$ to the A-equivalence class of $(g_1, g_2, \dots, g_r)$, where g_i's are the canonical generators for the covering $[P_0, \phi]_A$, $1 \leqslant i \leqslant r$.*

This gives us a parametrization on the fiber of Ψ_A. Each tuple t in $\mathcal{E}_r^{(A)}(G)$ corresponds to an admissible surjective homomorphism $\phi : \pi_1(\mathbb{P}^1 \setminus P, q_0) \to G$, and the Riemann's existence theorem guarantees us that any admissible tuple corresponds to a branched covering $f : R \to \mathbb{P}^1$. Hence the Hurwitz space $\mathcal{H}_r^{(A)}(G)$ is a space of A-equivalence classes of G-covers. The study of Hurwitz spaces is closely related to the Inverse Galois Problem. Our interest in the Hurwitz space derives from the next theorem given by Fried and Völklein in [5].

Theorem 2.39 *There exists a Galois extension of $\mathbb{Q}(x)$, regular over $\mathbb{Q}$, with Galois group isomorphic to G and with r branch points if and only if $\mathcal{H}_r^{in}(G)$ has a $\mathbb{Q}$-rational point. (This holds if $\mathbb{Q}$ is replaced by any field of characteristic 0).*

Furthermore, the tuple $t = (g_1, g_2, \dots, g_r)$ gives us a certain ramification type $\overline{C}$ with $g_i \in C_i, 1 \leqslant i \leqslant r$. The subset of $\mathcal{H}_r^{(A)}(G)$ that consists of all the A-equivalence classes of G-covers with a fixed ramification type $\overline{C}$ is denoted by $\mathcal{H}_r^{(A)}(\overline{C})$.

Now, we focus our study on the Hurwitz space $\mathcal{H}_r^{in}(G)$. In order to find the rational points on the Hurwitz space, it is useful to find its connected components. The Hurwitz space $\mathcal{H}_r^{in}(\overline{C})$ is a union of connected components in $\mathcal{H}_r^{in}(G)$. Via fixing the ramification type, we can concentrate on this specific subset of $\mathcal{H}_r^{in}(G)$. As described in Proposition 2.38, the fiber in $\mathcal{H}_r^{in}(\overline{C})$ corresponds to the Nielsen class defined as follows.

$$\mathcal{N}(\overline{C}) := \{(g_1, \dots, g_r)| \quad g_i \in C_i, G = \langle g_1, \dots, g_r \rangle, \prod_{i=1}^{r} g_i = 1\}.$$

We can see that $Inn(G)$ fixes $\mathcal{N}(\overline{C})$. Before getting into investigate $\mathcal{H}_r^{in}(\overline{C})$, we first note that the monodromy action of the fundamental group of $\mathcal{O}_r$ completely determines the components in the fiber of $\mathcal{H}_r^{in}(G)$ by Proposition 2.37. The section 10.1.6 in [18] proves the following lemma.

Lemma 2.40 *The fundamental group of $\mathcal{O}_r$ is in fact isomorphic to the* **Artin braid group** *B_r.*

Braid groups were defined and classified by Emil Artin in 1925. The notion of a braid element arises from its natural geometric sense. In the following section, we will talk about braids.

2.5 Braids

Definition 2.41 *Let l_1 and l_2 be two parallel straight lines in $\mathbb{R}^3$, and let $\{k_1, \dots, k_r\}$ be a set of parallel lines in $\mathbb{R}^3$ intersecting l_1 and l_2. Let a_i be the intersection point of k_i and l_1, and b_i be the intersection point of k_i and l_2 for $1 \leqslant i \leqslant r$. A strand between l_1 and l_2 is a path in $\mathbb{R}^3$ with an initial point a_i and a terminal point b_j, $1 \leqslant i, j \leqslant r$. A braid on r strands is made of the r strands between l_1 and l_2 under the following conditions:*

1. *The r strands define a bijection between the set $\{a_i\}$ and $\{b_i\}$, $1 \leqslant i \leqslant r$;*
2. *No two strands intersect or coalesce;*
3. *Two strands can form a knot in the way of moving one strand from the left side of the other strand to the right side.*

Figure 2.1 gives an example of two different braids Q_1 and Q_1^{-1}. We define Q_i to be the braid with the ith strand crossing under the $(i+1)$th strand. The inverse of Q_i, similarly, will be the braid with the $(i+1)$th strand crossing under the ith strand. Two braids Q and $Q^{'}$ on r strands can be composed by making the terminal points of one braid and the initial points of another braid equivalent. The combination of strands on

Figure 2.1: Two braid elements Q_1 and Q_1^{-1}

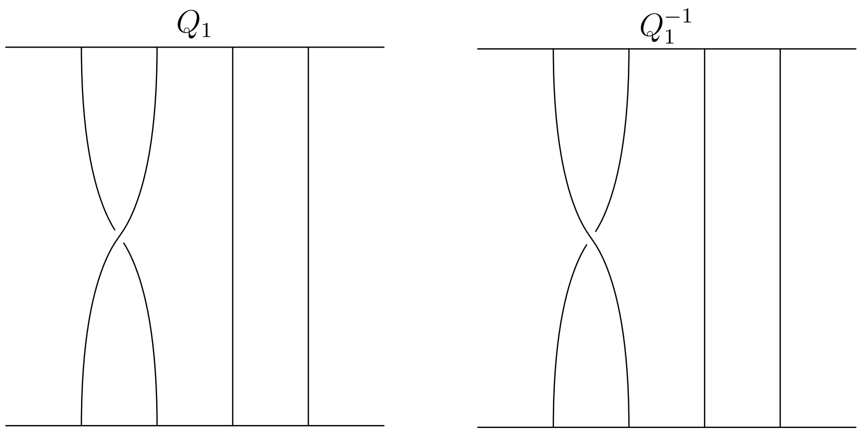


Figure 2.2: The composition of Q_1 and Q_2

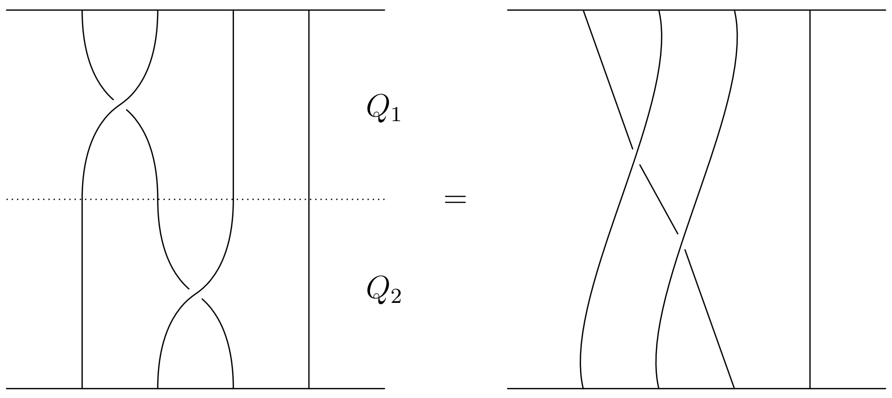
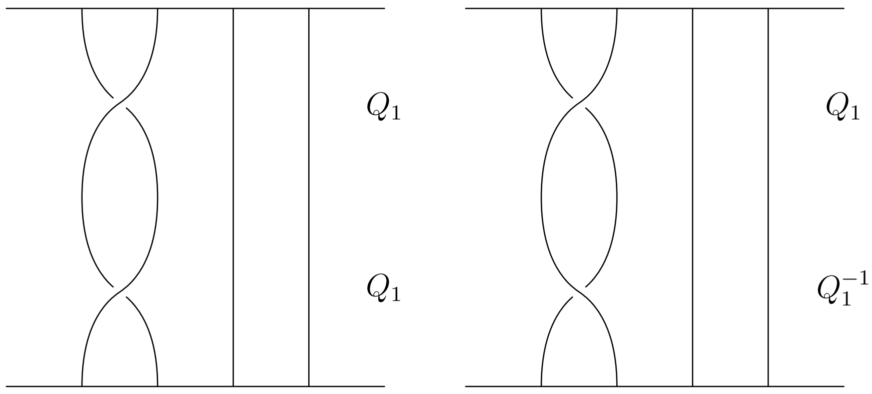


Figure 2.3: The braid elements of Q_1^2 and $Q_1 Q_1^{-1}$



Q and $Q^{'}$ makes $QQ^{'}$ into another new braid. Figure 2.2 gives an example of composing Q_1 and Q_2.

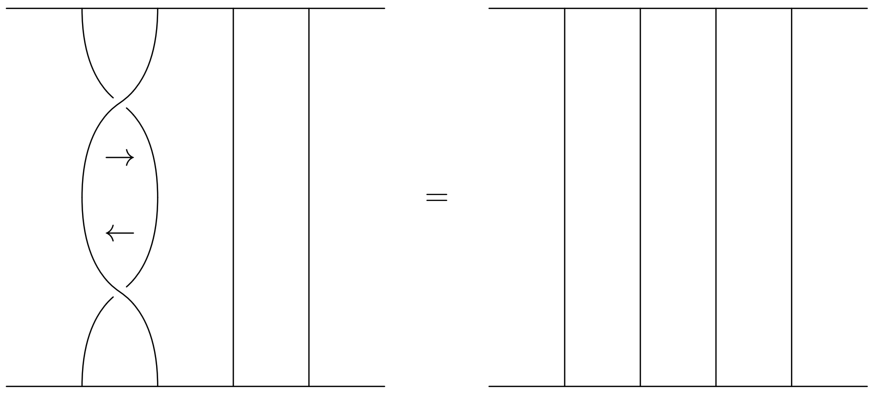
Another example is shown in Figure 2.3, which describes how to compose Q_1 with itself and its inverse Q_1^{-1}. The braid $Q_1 Q_1^{-1}$ is a trivial braid element, as the first two strands do not really form a "twist". We can simply pull one strand completely over the other one to form the identify braid which has no knots. From the braid Q_1^2 we can see that Q_i^n will always be a non-trivial braid with n knots between the ith strand and the $(i+1)$th stand, $1 \leqslant i \leqslant r-1, n \in \mathbb{Z}$.

The Reidemeister moves on knot diagrams demonstrate how two knots are equivalent up to isotopy. Each move belongs to one of the following three different types:

1. Twist or untwist a string in any direction;
2. Move one string completely over or under another string;
3. Move one string completely over or under a crossing.

For a braid diagram, the Reidemeister moves can also operate on strands between the two lines l_1 and l_2. Figure 2.4 gives an example of how to deform the braid $Q_1 Q_1^{-1}$ into the trivial braid. In fact, we have that

Figure 2.4: The second Reidemeister move on $Q_1Q_1^{-1}$



Proposition 2.42 *Every braid is invariant under the Reidemeister moves on strands.*

The braids Q_i and Q_i^{-1} for $1 \leqslant i \leqslant r-1$ are called elementary braids. Every braid on r strands is a composition of some elementary braids. Since Q_i^n is always non-trivial for any natural number n, we have the following result.

Lemma 2.43 *The set of all braids on r strands form an infinite group, called the* **braid group** *B_r.*

2.6 Braid Group and Braid action on Nielsen classes

The **braid group** B_r in the algebraic description is generated by the set of generators $\{Q_1, Q_2, \ldots, Q_{r-1}\}$ with the relations

$$Q_iQ_{i+1}Q_i = Q_{i+1}Q_iQ_{i+1};$$

$$Q_iQ_j = Q_jQ_i \quad \text{for} \quad |i-j| \geqslant 2.$$

The first equation is sometimes called the Yang-Baxter equation. The **braid action** on generating tuples of G in terms of the generators is as follows.

$$Q_i : \quad (g_1, \dots, g_i, g_{i+1}, \dots, g_r) \to (g_1, \dots, g_{i+1}, g_{i+1}^{-1} g_i g_{i+1}, \dots, g_r)$$

for $i = 1, 2, \dots, r-1$. Let $\overline{C} = \{C_1, \dots, C_r\}$ a ramification type, and the tuple $(g_1, g_2, \dots, g_r)$ is of type $\overline{C}$, that is, $C_i = \{g_i^G\}$ is the conjugacy class of G containing g_i, $1 \leqslant i \leqslant r$. Then the braid action permutes $\overline{C}$, as the homomorphism $\rho : B_r \to S_r$ sending Q_i to $(i, i+1)$. The kernel of ρ is called the pure braid group $B^{(r)}$, and it is generated by the pure braid elements as follows:

$$\begin{aligned} Q_{ij} &= Q_{j-1} \cdots Q_{i+1} Q_i^2 Q_{i+1}^{-1} \cdots Q_{j-1}^{-1} \\ &= Q_i^{-1} \cdots Q_{j-2}^{-1} Q_{j-1}^2 Q_{j-2} \cdots Q_i, \end{aligned}$$

for $1 \leqslant i < j \leqslant r$. Because of the braid group action, we can assume that the conjugacy classes in the type $\overline{C}$ are ordered in a particular way. From now on, we always assume that the same conjugacy classes in $\overline{C}$ are adjacent and form a block in $\overline{C}$. And if we mark one class in each block, these different conjugacy classes are in the same order of the character table of G. This splits $\overline{C}$ into a disjoint union of continuous blocks $\{X_1, \dots, X_s\}$ that correspond to different conjugacy classes appearing in $\overline{C}$, $1 \leqslant s \leqslant r$.

Definition 2.44 *A* **partition** P *of the set* $\overline{C}$ *is a set of blocks* $\{X_1, \dots, X_s\}$*, such that*

$$\bigcup_{i=1}^{s} X_i = \overline{C} \quad \textit{and} \quad X_i \cap X_j = \emptyset \quad \textit{for} \quad i \neq j.$$

We refer to $P = \{X_1, \dots, X_s\}$ as to the partition related to $\overline{C}$. The inverse image of the stabilizer $S_P \leqslant S_r$in B_r is called a **parabolic subgroup** of the braid group. Since the pure braid group $B^{(r)}$ always preserves P, it is contained in any parabolic subgroup.

We denote a parabolic subgroup by B in general, or B_P to indicate the partition P on $\overline{C}$.

Recall that in the previous section, the covering Ψ_A in Proposition 2.37 can restrict to a covering from a connected component which includes part of the fiber $\Psi_A^{-1}(P_0)$ to $\mathcal{O}_r$, and the connecting paths in the fiber are from lifts of elements in the fundamental group of base space. Together with the definition of B_P, we have the following proposition.

Proposition 2.45 *There is a one-to-one correspondence between B_P-orbits on Nielsen classes $\mathcal{N}(\overline{C})$ and connected components of $\mathcal{H}_r^{in}(\overline{C})$.*

The B_P-orbits are shorter than the orbits from the full braid group B_r, which gives a great advantage for our study. We will record the set of generators of the subgroup B_P. By the definition of B_P, we have the following lemma.

Lemma 2.46 *The union of the set of Q_{ij}'s such that i and j lie in the different blocks of P and the set of Q_i's such that i and $i+1$ lie in the same block is a generating set of B_P.*

Chapter 3
Investigation on Affine Primitive Groups

3.1 Genus Zero Covers

Suppose Y is a compact Riemann surface of genus g, and $f : Y \to \mathbb{P}^1$ is a meromorphic function of degree $n > 1$. From Chapter 1 we know that there exists a branch point set P with cardinality $r < \infty$ in $\mathbb{P}^1$, such that the fundamental group $\pi_1(\mathbb{P}^1 \setminus P, q_0)$ acts transitively on the fiber $f^{-1}(q_0)$, where $q_0 \in \mathbb{P}^1 \setminus P$. Let the monodromy group of f be G. We are interested in the structure of the monodromy group G when the genus of Y is less than or equal to two, and the function f is indecomposable, that is, f can not be written as a composition of two holomorphic functions with degree greater than one. This condition implies the following lemma.

Lemma 3.1 *The monodromy group G is* **primitive** *if and only if every corresponding covering f is* **indecomposable***.*

In other words, if f is decomposable as $f = f_1 \circ f_2$, where the degrees of f_1 and f_2 are both greater than one, then the composition factor of G is a factor of the monodromy group of either f_1 or f_2. It is natural to consider only the indecomposable functions with their monodromy groups being primitive. Our study relates to a conjecture made by Guralnick and Thompson [11] in 1990, that the set of isomorphism types of the composition factors of G apart from A_n, $n \geqslant 5$, or cyclic group $\mathbb{Z}/p\mathbb{Z}$, p a prime, is finite, where G is the monodromy group for a G-cover of any finite genus over the Riemann sphere $\mathbb{P}^1$. Denote the set of isomorphism types of the composition factors of G by $cf(G)$. The following set $\mathcal{E}^*(g)$ is defined in their paper [11],

$$\mathcal{E}^*(g) = (\bigcup_{(Y,f)} cf(G)) \setminus \{A_n, \mathbb{Z}/p\mathbb{Z} : n \geqslant 5, \quad p \quad \text{a prime}\},$$

where $Y \in \mathcal{M}_g$, the moduli space of curves of genus g. The final case of this conjecture was established in 2001 by Frohardt and Magaard [8]. When the genus is zero, the set $\mathcal{E}^*(0)$ is distinguished in that it is contained in $\mathcal{E}^*(g)$ for all $g \in \mathbb{N}$. The proof of this conjecture also shows that it is possible to compute $\mathcal{E}^*(0)$ explicitly. Our assumption that G acts primitively on the fiber is a strong one and allows us to organize our analysis along the lines of the Aschbacher-O'Nan Scott theorem exactly as was done in the original paper of Guralnick and Thompson [11].

By the definition of two isomorphic covers, all sets of canonical generators of the fundamental group $\pi_1(\mathbb{P}^1 \setminus P, q_0)$ will lie in the same braid orbit, as the braid group acts on $\pi_1(\mathbb{P}^1 \setminus P, q_0)$ via automorphism. Also the monodromy group G acts on generating tuples via diagonal conjugation. These two action commute and preserve the equivalence relation of covers. We say two tuples are **braid equivalent** if they are equivalent under braid action and diagonal conjugation. Using the Riemann's Existence Theorem, we have

Theorem 3.2 *Two covers are equivalent if and only if their corresponding generating tuples are braid equivalent.*

The GAP package BRAID developed by Magaard, Shpectorov and Völklein [12] computes braid orbits algorithmically. In light of Lemma 3.1, we focus on finding all equivalence classes of admissible generating tuples for affine primitive groups when the genus of the cover is zero. By the Riemann-Hurwitz formula, genus zero covers correspond to the kind of tuples defined as follows.

Definition 3.3 *A genus zero system is a generating tuple* $(g_1, \dots, g_r)$ *of a transitive group* G *such that, for* $1 \leqslant i \leqslant r$, $g_1 \dots g_r = 1, g_i \neq 1$ *and*

$$2(n-1) = \sum_{i=1}^{r} Ind(g_i).$$

Note that when we consider genus zero covers Y that are not Galois, the point stabilizer H which acts irreducibly on the fiber in the monodromy action of G may not be invariant under $\mathrm{Aut}(G)$ but rather under some subgroup A in $\mathrm{Aut}(G)$. Then we will have a normal closure X of $Y = X/H$ with G being a subgroup of $A = \mathrm{Aut}(X)$. As shown in the following commutative diagram, the coverings $X \to \mathbb{P}^1$ and $X \to Y$ are both Galois, and the genus of $\mathbb{P}^1$ and Y are both zero. The Hurwitz loci for those groups G correspond to the normal closure of G-covers. The classification of these Hurwitz loci is done by computing the braid orbits of genus zero systems of G. In fact, the irreducible families of rational functions in the field $\mathbb{C}(x)$ of degree n with monodromy group G also correspond to braid orbits of genus zero systems of G.

$$\begin{tikzcd} X \arrow[d] \arrow[dr] & \\ \mathbb{P}^1 & Y \arrow[l] \end{tikzcd}$$

3.2 Previous Work

We recall the statement of the Aschbacher-O'Nan-Scott theorem from [11]. The generalized Fitting subgroup of G is denoted by $F^*(G)$.

Theorem 3.4 *Suppose G is a primitive group and H is a maximal subgroup of G. Let Q be a minimal normal subgroup of G, let L be a minimal normal subgroup of Q, and let $\Delta = \{L = L_1, L_2, \ldots, L_t\}$ be the set of G-conjugates of L. Then $G = HQ$ and precisely one of the following holds:*

(A) L is of prime order p.

(B) $F^*(G) = Q \times R$ where $Q \cong R$ and $H \cap Q = 1$.

(C1) $F^*(G) = Q$ is nonabelian, $H \cap Q = 1$.

(C2) $F^*(G) = Q$ is nonabelian, $H \cap Q \neq 1 = L \cap H$.

(C3) $F^*(G) = Q$ is nonabelian, $H \cap Q = H_1 \times \cdots \times H_t$, where $H_i = H \cap L_i \neq 1$, $1 \leqslant i \leqslant t$.

We define $P\mathcal{E}^*(g)$ to be the equivalence classes of genus g covers of the Riemann sphere such that the monodromy group is primitive with $A_n \cap G \neq A_n$ as a subgroup. In Theorem 3.4, the members of $P\mathcal{E}^*(0)$ that arise in case (C2) were determined by Aschbacher [1]. In all such examples $Q = A_5 \times A_5$. Shih [16] showed that no elements of $P\mathcal{E}^*(g)$ arise in case (B) and Guralnick and Thompson [11] showed the same in case (C1). In the case (C3) Frohardt, Guralnick, and Magaard have eastablished when L_i is of Lie type of rank one in [6], and they show that $t \leqslant 2$. Moreover, in [7], they showed that if $t = 1$, L_i is classical and L_i/H_i is a point action, then $[L_i : H_i] \leqslant 10,000$. That result, together with the results of Aschbacher, Guralnick and Magaard [2], show that if $t = 1$ and L_i is classical then $[L_i : H_i] \leqslant 10,000$.

In case (A) of Theorem 3.4, which is known as the affine case, we have that $F^*(G)$ is elementary abelian and acts regularly on the fiber. Case (A) was first considered by Guralnick and Thompson [11] and then strengthened by Neubauer [15]. This is the case that we are interested in due to its computational complexity. And thanks to Theorem

1.2 from Neubauer [15], we will have small bounds on the number of groups we need to deal with.

In this thesis, we will be looking at affine primitive groups of degree p^e, where p and e satisfy the conditions in Theorem 1.2. The groups G with $G'' = 1$ and $1 \leqslant e \leqslant 2$ are Frobenius groups and are well understood. So we will concentrate on affine groups of the following degrees:

$$\{8, 16, 32, 64, 128, 256, 9, 27, 81, 25, 125, 49, 121\}.$$

In the last part of the thesis, we state the results from the classification of genus zero systems of all those groups using our BRAID and matching algorithm.

3.3 Affine Primitive Permutation Groups

Definition 3.5 *Let H be a subgroup of G. The* **normal core** *of H in G, denoted by $Core_G(H)$, is defined as*

$$Core_G(H) = \bigcap_{s \in G} s^{-1}Hs.$$

If H is a p-subgroup then we say $Core_G(H)$ is a p-core.

In fact, the p-core is the normal core of every Sylow p-subgroup. We know that an affine primitive group always contains a normal regular subgroup. Let $(\mathbb{Z}/p\mathbb{Z})^e$ be the elementary abelian group, where p is a prime. Then the normal p-core of an affine primitive group G containing $(\mathbb{Z}/p\mathbb{Z})^e$ will have order p^e. In our study, G is an affine primitive subgroup of $\mathrm{AGL}(e, p)$.

The GAP group library contains all the primitive permutation groups of degree p^e, where p and e satisfy any of the conditions in Theorem 1.2. We can extract these groups using the function **AllPrimitiveGroups**. Once this is done, we need to choose the affine ones out of those primitive groups by checking the order of the p-core. The groups S_n

and A_n also need to be excluded. The following GAP code is an example of finding the affine primitive groups of degree 16.

```
gap> AffineGroupList:=[];
gap> GroupList:=AllPrimitiveGroups(DegreeOperation,16);;
gap> for i in [1..Length(GroupList)] do
>     g:=GroupList[i];
>     if g<>AlternatingGroup(16) and g<>SymmetricGroup(16) then
>      syl:=SylowSubgroup(g,2);
>      if Size(Core(g,syl))=16 then
>       Append(AffineGroupList,[g]);
>      fi;
>     fi;
>    od;
gap> AffineGroupList;
[ 2^4:5, 2^4:D(2*5), AGL(1, 16), (A(4) x A(4)):2, (2^4:5).4, AGL(1, 16):2,
  2^4.S(3) x S(3), 2^4.3^2:4, AGammaL(1, 16), (S(4) x S(4)):2, 2^4.PSL(4, 2),
  AGammaL(2, 4), ASL(2, 4):2, AGL(2, 4), ASL(2, 4), 2^4.S(6), 2^4.A(6),
  2^4:S(5), 2^4:A(5), 2^4.A(7) ]
```

We display the number of all the affine primitive groups of degree p^e needed in Table 3.1, and the list all those groups in the appendix.

Table 3.1: The Number of Affine Primitive Groups of Degree p^e

p	2							3			5		7	11
e	2	3	4	5	6	7	8	2	3	4	2	3	2	2
#	0	3	20	3	64	3	238	7	11	145	22	33	33	48

3.4 Possible Ramification Types

We say a ramification type $\overline{C}$ is a **generating type**, if there is at least one generating tuple of type $\overline{C}$. During this section and the following one, we describe an algorithm to compute all generating types of genus g systems for a group G.

Recall the Riemann-Hurwitz formula

$$2(n+g-1) = \sum_{i=1}^{r} Ind(g_i),$$

where n is the degree, g is the genus and $Ind(g_i)$ is the permutation index of g_i.

Lemma 3.6 *Let $\sigma \in S_n$, then*

$$Ind(\sigma) = n - Orb(\sigma),$$

where $Orb(\sigma)$ denotes the number of orbits of σ on n points.

Firstly, we need find all possible ramification types that fit the Riemann-Hurwitz formula. In the following code, we display some results from using GAP to compute those tuple types of AGL$(4,2)$ with length greater than 5. First, we find all the conjugacy class representatives of the group, then compute the permutation index for each representative. As shown below, the tuple type is a list of permutation indices such that the sum is equal to $2(n+g-1)$. Here we take g to be 0 and 1 as an example.

```
gap> g;
2^4.PSL(4, 2)
gap> CC:=List(ConjugacyClasses(g),Representative);;
gap> Ind:=List(CC{[2..25]},x->16-Length(Orbits(Group(x), [1..16])));
[ 8, 10, 14, 8, 12, 4, 12, 8, 12, 14, 12, 14, 10, 14, 6, 12, 12,
10, 12, 14, 14, 8, 12, 10 ]
gap> Filtered(RestrictedPartitions(30,Elements(Ind)),x->Length(x)>5);
[ [ 6, 4, 4, 4, 4, 4, 4 ], [ 6, 6, 6, 4, 4, 4 ], [ 8, 6, 4, 4, 4, 4 ],
[ 10, 4, 4, 4, 4, 4 ] ]
gap> Filtered(RestrictedPartitions(32,Elements(Ind)),x->Length(x)>6);
[ [ 4, 4, 4, 4, 4, 4, 4, 4 ], [ 6, 6, 4, 4, 4, 4, 4 ],
[ 8, 4, 4, 4, 4, 4, 4 ] ]
```

For those conjugacy classes with same index, we use the GAP function **UnOrdered-Tuples** to generate all the possible combinations of a fixed ramification type containing that index. For example, suppose the tuple type [8,8,8,8] is a possible genus one system. In the above code, we see that there are 4 different conjugacy classes with the same index 8. Let us use numbers 1, 2, 3 and 4 to represent them in GAP.

```
gap> UnorderedTuples([1,2,3,4],4);
[ [ 1, 1, 1, 1 ], [ 1, 1, 1, 2 ], [ 1, 1, 1, 3 ], [ 1, 1, 1, 4 ],
  [ 1, 1, 2, 2 ], [ 1, 1, 2, 3 ], [ 1, 1, 2, 4 ], [ 1, 1, 3, 3 ],
  [ 1, 1, 3, 4 ], [ 1, 1, 4, 4 ], [ 1, 2, 2, 2 ], [ 1, 2, 2, 3 ],
  [ 1, 2, 2, 4 ], [ 1, 2, 3, 3 ], [ 1, 2, 3, 4 ], [ 1, 2, 4, 4 ],
  [ 1, 3, 3, 3 ], [ 1, 3, 3, 4 ], [ 1, 3, 4, 4 ], [ 1, 4, 4, 4 ],
  [ 2, 2, 2, 2 ], [ 2, 2, 2, 3 ], [ 2, 2, 2, 4 ], [ 2, 2, 3, 3 ],
  [ 2, 2, 3, 4 ], [ 2, 2, 4, 4 ], [ 2, 3, 3, 3 ], [ 2, 3, 3, 4 ],
  [ 2, 3, 4, 4 ], [ 2, 4, 4, 4 ], [ 3, 3, 3, 3 ], [ 3, 3, 3, 4 ],
  [ 3, 3, 4, 4 ], [ 3, 4, 4, 4 ], [ 4, 4, 4, 4 ] ]
```

In this way, we can list all possible ramification types that fit the Riemann-Hurwitz formula. Secondly, as a consequence of Scott's theorem, we have the following result. Here we denote the dimension of a vector space W by $\dim W$.

Proposition 3.7 *Let G be a finite group that acts faithfully and irreducibly on the F-vector space V, and $t = (g_1, \dots, g_r)$ be a generating tuple of G with $g_1 \dots g_r = 1$. Let $[g_i, V]$ be the commutator space for $1 \leqslant i \leqslant r$. Then*

$$\sum_{i=1}^{r} \dim [g_i, V] \geqslant 2 \dim V. \tag{3.8}$$

This proposition guarantees that those ramification types which do not satisfy the formula 3.8 will not be generating types. We use the following lemma to calculate the dimension of the commutator space in GAP.

Lemma 3.9 *Let h be an element in G and $C_V(h)$ be the centralizer of h in V. Then*

$$\dim [h, V] = \dim V - \dim C_V(h).$$

For the group $\mathrm{AGL}(e,p)$, the dimension of the vector space $V = p^e$ is e. For an element $h \in G$, the order of $C_V(h)$ is p^{e_0}, where $0 \leqslant e_0 \leqslant e$. If $e_0 \neq 0$, we can use the GAP function **FactorsInt** to obtain e_0. For example, in the following code, we show how to compute the dimensions of the commutator spaces for a tuple. The tuple "t" is a generating tuple of $\mathrm{AGL}(4,2)$, and V is the elementary abelian group of order 2^4 in $\mathrm{AGL}(4,2)$. The parameter "a" at the end is the set of dimensions of commutator spaces for "t".

```
gap> g;
2^4.PSL(4, 2)
gap> NormalSubgroups(g);
[ Group(()),
  Group([ (1,2)(3,4)(5,6)(7,8)(9,10)(11,12)(13,14)(15,16), (1,5)(2,6)(3,7)(4,
        8)(9,13)(10,14)(11,15)(12,16), (1,6)(2,5)(3,8)(4,7)(9,14)(10,13)(11,
        16)(12,15), (1,9)(2,10)(3,11)(4,12)(5,13)(6,14)(7,15)(8,16),
      (1,3)(2,4)(5,7)(6,8)(9,11)(10,12)(13,15)(14,16) ]), 2^4.PSL(4, 2) ]
gap> V:=last[2];;Size(V);
16
gap> t;
[ (2,5)(3,8)(10,13)(11,16), (1,2,6)(3,8,7)(9,14,13)(11,12,16),
  (1,12,13,8)(2,14)(3,15)(4,9,16,5), (1,8,14)(2,9,4)(3,15,7)(5,12,6)(10,13,16)]
gap> a:=[];;b:=List(t,x->Size(Centralizer(V,x)));
[ 8, 4, 4, 1 ]
gap> for j in [1..Length(b)] do
   >  if b[j]<> 1 then
   >   Append(a,[4-Length(FactorsInt(b[j]))]) ;
   >  else
   >   Append(a,[4]);
   >  fi;
   > od;
gap> a;
[ 0, 2, 2, 4 ]
```

Those ramification types of $\mathrm{AGL}(e,p)$ that satisfy the Riemann-Hurwitz formula and 3.8 will contain the generating types of all the affine primitive groups of degree p^e. In order to find them, we can compute the braid orbits for each ramification type, and then verify the subgroup generated by the tuple in each orbit. What we need are the orbits that belong to affine primitive groups. For every group AGL(e, p), where p and e take the values in Theorem 1.2, we display the number of possible ramification types found at this stage in Table 3.2.

Table 3.2: The Number of Possible Ramification Types for AGL(e, p)

$ASL(3,2)$	AGL$(4,2)$	$ASL(5,2)$	AGL$(6,2)$	AGL$(7,2)$	AGL$(8,2)$	
53	230	144	355	43	132	
AGL$(2,3)$	AGL$(3,3)$	AGL$(4,3)$	AGL$(2,5)$	AGL$(3,5)$	AGL$(2,7)$	AGL$(2,11)$
38	145	351	73	121	90	7

Another option is to use the function **GeneratingTypes** which will be introduced in the next section to find all the generating types of a chosen affine primitive group. This method is mainly used to focus on one specific group and remove the ramification types that do not generate the group. In our case, there are 64 subgroups of AGL$(6, 2)$ and 355 possible ramification types. As shown in the appendix, there are only 14 groups of degree 2^6 having genus zero systems, and most of them have only two generating types. Since checking each group using our function **GeneratingTypes** does not take too much time, it seems to be unnecessary to compute orbits of all 355 ramification types.

3.5 Generating Types

For a group G, the function **GeneratingTypes** will first compute all the possible ramification types that satisfy the Riemann-Hurwitz formula and 3.8, then put them in the final routine to check if there exists a generating tuple of G. The computational method of the final routine is from [17], whose function is mainly for counting the number of genus g systems using the character table of G. However, computing the character table of a large group in GAP may cause an issue. In our case, we only want to check the existence of genus g system of G before running our BRAID program and the matching algorithm. The character table of G is not necessary at this stage. We give a brief description as follows. Suppose we have a ramification type $\overline{C} = \{C_1, \ldots, C_r\}$, and G has n distinct conjugacy classes with class representatives $\{c_1, \ldots, c_n\}$. We create a list L with n sublists $L = \{L_1, \ldots, L_n\}$ such that L_i corresponds to C_i for $1 \leqslant i \leqslant n$. Each L_i contains all the possible subgroups of G generated by tuples $(g_1, \ldots, g_k)$ up to conjugation by the centralizer of c_i in G, such that $g_1 \ldots g_k = c_i$, and $g_j \in C_j$ for $1 \leqslant j \leqslant k, 1 < k \leqslant r - 1$. Starting from $g_1 = c_1$, we keep adding generator $g_k \in C_k$ in every cycle of the routine to the tuple $(g_1, \ldots)$ until we reach to $(g_1, \ldots, g_{r-1})$, and construct the subgroup generated by $(g_1, \ldots, g_k)$. If it is a new subgroup, we add it to the list L_i. After the last cycle of adding g_{r-1} finishes, if the sublist corresponding to the inverse of C_r contains the group G, then $\overline{C}$ is a generating type.

The function **GeneratingTypes** will compute the generating types for an affine primitive group with a fixed genus using this algorithm. It takes three inputs: the group, the degree of the action and the genus. It is in the form

GeneratingTypes(group,degree,genus).

When **GeneratingTypes** terminates, it will save all the generating types in a file together with the name of the group and its conjugacy class representatives. Using this function, we determined all the generating types for affine primitive groups of degree p^e in Theorem 1.2. In particular, we see that there are no genus zero systems in degree 128 and none of degree 256 apart from possibly $G = \mathrm{AGL}(8, 2)$. Since the group $\mathrm{AGL}(8, 2)$ is too large to be dealt in our functions, we will use some other methods introduced in the next section to find generating types of this group.

As an example in Table 3.3, we display the results from computing all the generating types for the affine primitive groups of degree 16 at genera 0 and 1 with our function, including the number of possible ramification types that satisfy the Riemann-Hurwitz formula and 3.8. We can see that when the genus increases, the number of generating types for these groups will increase.

Table 3.3: Affine Primitive Groups of Degree 16 at Genera 0 and 1

Group	number of possible ramification types		number of generating types	
	genus 0	genus 1	genus 0	genus 1
$2^4 : 5$	0	30	0	6
$2^4 : D(2 * 5)$	15	18	9	9
$\mathrm{AGL}(1, 16)$	12	38	0	0
$(A_4 \times A_4) : 2$	83	104	4	6
$(2^4 : 5).4$	15	22	2	6
$\mathrm{AGL}(1, 16) : 2$	42	65	0	4
$2^4 : S_3 \times S_3$	195	188	5	9
$2^4.3^2 : 4$	29	44	3	8
$A\Gamma L(1, 16)$	34	46	0	2
$(S_4 \times S_4) : 2$	167	300	12	32
$2^4.PSL(4, 2)$	230	459	72	282
$A\Gamma L(2, 4)$	128	178	4	12
$\mathrm{ASL}(2, 4) : 2$	57	46	6	8
$\mathrm{AGL}(2, 4)$	131	211	0	21
$\mathrm{ASL}(2, 4)$	19	12	0	3
$2^4.S_6$	250	429	6	56
$2^4.A_6$	34	48	8	16
$2^4 : S_5$	54	111	13	62
$2^4 : A_5$	22	34	13	21
$2^4.A_7$	47	73	14	42

3.6 The Group $\mathrm{AGL}(8, 2)$

Theorem 1.2 gives us a complete list of affine primitive groups for our study. The largest group we will deal with is $\mathrm{AGL}(8, 2)$, which has order $1, 369, 104, 324, 918, 194, 995, 200$. This group is too large to be put in some of our current functions for computing. The GAP system is unable to compute its character table, but we can still have its conjugacy classes within several seconds. In order to check whether or not a ramification type is a generating type for this special group, we will have to use other methods manually instead of the standard routines.

Since the degree of $\mathrm{AGL}(8, 2)$ is 2^8, its conjugacy class representatives will have relatively large permutation index. While restricting the genus to 0, we expect to see most of the ramification types with short length. In fact, as shown in the GAP code below, we can see that most of the genus zero systems for this case will be 3-tuples. Usually, for tuples of length 3, our standard BRAID program is not the best choice for such cases, and there is a much easier way to find generating tuples instead of applying braid action on random tuples.

```
gap> g;
AGL(8, 2)
gap> CC:=List(ConjugacyClasses(g),Representative);;
gap> Ind:=List(CC{[2..Length(CC)]},x->256-Length(Orbits(Group(x),[1..256])));
gap> Elements(Ind);
[ 64, 96, 112, 120, 128, 144, 152, 160, 168, 170, 172, 176, 180, 184, 186,
  190, 192, 196, 200, 202, 204, 206, 208, 210, 212, 216, 218, 220, 224, 226,
  228, 232, 234, 236, 238, 240, 242, 244, 246, 248, 250, 252, 254 ]
gap> Sort(Ind);
gap> Length(Ind);
474
gap> Ind{[1..40]};
[ 64, 96, 112, 120, 128, 128, 128, 128, 128, 128, 144, 152, 160, 160, 160,
  160, 160, 160, 168, 168, 168, 170, 172, 172, 176, 176, 176, 176, 180, 184,
  184, 184, 186, 190, 192, 192, 192, 192, 192, 192 ]
```

Let (C_1, C_2, C_3) be a ramification type of length 3 with class representatives c_1, c_2 and c_3. Since we consider the generating tuples up to the conjugation of G, we can always fix one of the entries to be the class representative. Without loss of generality, we fix the first entry to be c_1.

Lemma 3.10 *Suppose (C_1, C_2, C_3) is a generating type. Let $C_G(c_i)$ be the centralizer of c_i in G, $i = 1, 2$. Then the generating tuple of this type up to conjugation in G is in the form $(c_1, c_2^k, (c_1c_2^k)^{-1})$, where k is in the double coset of $C_G(c_2)$ and $Norm(C_G(c_1), \langle c_1 \rangle)$ in G, and $(c_1c_2^k)^{-1} \in C_3$.*

Proof. Let (c_1, g_2, g_3) be a generating tuple of ramification type (C_1, C_2, C_3) with $g_3 = (c_1g_2)^{-1}$. Since it generates G, we have $\langle c_1, g_2 \rangle = G$. There exists $k_1 \in Norm(C_G(c_1), \langle c_1 \rangle)$, such that $\langle c_1, g_2^{k_1}\rangle = \langle c_1, g_2\rangle = G$. Furthermore, we can find $k_2 \in C_G(c_2)$ and $k' \in G$, such that $g_2 = c_2^{k_2 k} = c_2^{k_2 k' k_1}$. Hence there exists k in the double coset of $C_G(c_2)$ and $Norm(C_G(c_1), \langle c_1\rangle)$ in G such that $\langle c_1, c_2^k\rangle = G$. This proves the lemma. $\square$

There are a total of 132 ramification types of genus zero systems for AGL$(8, 2)$ that satisfy both Riemann-Hurwitz formula and 3.8, all of which are 3-tuples. Considering the size of this group, it will still be unrealistic to go through each type using Lemma 3.10. However, we can probably further reduce the number of the types by translating the genus zero system to a genus g system, where $g \geqslant 0$, using Lemma 3.2 in [8]. Here, we give a simple version for 3-tuples as follows.

Lemma 3.11 *Let (g_1, g_2, g_3) be a generating tuple of G with $g_1g_2g_3 = 1$. Suppose the element g_2 has order n. Then, the tuple*

$$t = (g_1, g_2g_1g_2^{-1}, g_2^2g_1(g_2^2)^{-1}, \ldots, g_2^{n-1}g_1g_2, g_3^n)$$

generates a normal subgroup N of G, and its product is also 1.

Changing g_2 to g_3 or reverse g_1 and g_2 will lead us to similar results as the above lemma. The group AGL$(8, 2)$ contains only one non-trivial normal subgroup, which is the elementary abelian group $V = (\mathbb{Z}/2\mathbb{Z})^8$. Hence if g_1 is not in V, then the tuple t is also a generating tuple which belongs to a genus g system for some $g \geqslant 0$. Our idea is to translate a 3-tuple into a longer one by removing one conjugacy class representative g_i, such that the dimension of the commutator space $[g_i, V]$ is relatively large. In this way, if the tuple t does not satisfy the formula 3.8, then the original 3-tuple (g_1, g_2, g_3) will not be a generating tuple of G. Here we give an example about how to apply this method to find some 3-tuples that do not generate. In the following code, g_1 lies in the 13th conjugacy class of AGL$(8, 2)$. The element g_2 and g_3 lie in the 92nd and 25th conjugacy class respectively.

```
gap> g;
AGL(8, 2)
gap> CC:=List(ConjugacyClasses(g),Representative);;
gap> List([CC[13],CC[92],CC[25]],x->Order(x));
[ 2, 3, 6 ]
gap> n:=NormalSubgroups(g)[2];
<permutation group of size 256 with 65 generators>
gap> List([CC[13],CC[92],CC[25]],x->Size(Centralizer(n,x)));
[ 64, 1, 4 ]
gap> List([6,0,2],x->8-x);
[ 2, 8, 6 ]
```

From the above results, we can see that if we translate the tuple (g_1, g_2, g_3) into (g_1, g_1, g_1, g_3^3), where the dimension of $[g_1, V]$ is 2, the sum of the dimensions of commutator spaces for this 4-tuple will always be less than 16, which means that this tuple can not generate the whole group. With the help of Lemma 3.11, we successfully checked all the 132 possible ramification types for AGL$(8, 2)$, and obtained no generating types.

Now, using our computational methods introduced in Section 3.3,3.4 and 3.5, we have known all the affine primitive groups and their generating types for genus zero systems. In order to determine the braid orbits, we will use the BRAID package for our computation. However, this does not always work so well due to the fact that some generating type will have a very large structure constant. In the next chapter, we introduce another method for dealing with those large cases.

Chapter 4

The Matching Algorithm

4.1 Some Background Results

From this section on, we will talk about a matching algorithm for computing braid orbits of B on the Nielsen class. Let $t = (g_1, \ldots, g_r)$ be a generating tuple of G with ramification type $\overline{C}$. Computing the braid orbit O_t of t directly is very straightforward, that is, we keep applying the generators of the braid group on t until we get no new tuples. When the length of the tuple gets longer, the size of its braid orbit increases dramatically. The structure constant formula tells us that after adding one more entry to the tuple, the size will be multiplied approximately by the size of the conjugacy class containing that extra generator. So computing braid orbits of a long tuple directly takes considerable time and effort. In order to handle larger Nielsen class, we propose to break up the class into smaller classes under the action of a subgroup of B, then combine them together to form full orbits. As a start, we discuss some results about this process of combination of two parts in the Nielsen class. The first part includes elements $t_1 = (g_1, \ldots, g_k)$, and the second one is $t_2 = (g_{k+1}, \ldots, g_r)$.

We choose a number k such that $1 \leqslant k \leqslant r-1$. Let B_L and B_R be the two subgroups of B_P such that

$$B_L = \langle Q_1, \dots, Q_{k-1} \rangle \cap B_P \quad \text{and} \quad B_R = \langle Q_{k+1}, \dots, Q_{r-1} \rangle \cap B_P.$$

Our standard BRAID package computes the orbits for the direct product of braid action and G-conjugation. Let $\mathcal{L}$ be the set of all tuples $(g_1, g_2, \dots, g_k)$ with $g_i \in C_i$ for $1 \leqslant i \leqslant k$. Similarly, let $\mathcal{R}$ be the set of all tuples $(g_{k+1}, \dots, g_r)$ with $g_i \in C_i$ for $k+1 \leqslant i \leqslant r$. We define **head** to be the orbit of $B_L \times G$ on $\mathcal{L}$, and **tail** to be the orbit of $B_R \times G$ on $\mathcal{R}$. It is easy to see that B_L and B_R intersect trivially and commute. Therefore $B_L \times B_R$ is a subgroup of B_P.

We define **nodes** to be the orbits of $B_L \times B_R \times G$ on the Nielsen class $\mathcal{N}(\overline{C})$. Let $x = (g_1 \dots g_k)^{-1}$ and $y = (g_{k+1} \dots g_r)^{-1}$. Our approach is to construct all possible nodes and then decide which nodes lie in the same orbits for $B \times G$, thus identify all the orbits of $B \times G$, and hence all the components of the Hurwitz space. Let us note the following property of the action of B_L and B_R.

Lemma 4.1 *The action of $B_L \times B_R$ on the tuples $(g_1, \dots, g_r)$ does not change x and y.*

Taking this into account, we have that

Lemma 4.2 *Each node in $\mathcal{N}(\overline{C})$ corresponds to a conjugacy class C_x with representative $x \in G$.*

In view of this lemma, the conjugacy class C_x is an invariant of every $B_L \times G$ orbit on $\mathcal{L}$. Similarly, the class C_y is an invariant of every $B_R \times G$ orbit on $\mathcal{R}$. Here we say that the orbit of B_L on the tuples $(g_1, \dots, g_k)$ is a B_L-orbit, and the orbit of B_R on the tuples $(g_{k+1}, \dots, g_r)$ is a B_R-orbit. Our first task is to identify all the nodes in $\mathcal{N}(\overline{C})$ with direct products of B_L-orbits and B_R-orbits with the following conditions:

1. $y = x^{-1}$;

2. In the direct product $J_1 \times J_2$, where J_1 is a B_L-orbit and J_2 is a B_R-orbit, any tuple $t_1 = (g_1, \ldots, g_k)$ in J_1 and $t_2 = (g_{k+1}, \ldots, g_r)$ in J_2 form a generating tuple $(g_1, \ldots, g_r)$ of G.

Such a tuple (t_1, t_2) in $J_1 \times J_2$ is called a **matching tuple**, and the pair (J_1, J_2) that satisfies the above two conditions is called a **matching pair**.

Lemma 4.3 *A node in $\mathcal{N}(\overline{C})$ is a union of G-orbits of a matching pair.*

Proof. Since B_L and B_R intersect trivially, a matching pair (J_1, J_2) is in fact an orbit of $B_L \times B_R$ on Nielsen class. Hence a node is the set of G-orbits on (J_1, J_2). □

Definition 4.4 *Let O_t be an orbit of B containing a tuple t. The normalizer N_t of O_t is the set of elements h in G, such that for any $t' \in O_t$, $h^{-1}t'h \in O_t$.*

$$N_t = \{h \in G | h^{-1} O_t h = O_t\}.$$

Lemma 4.5 *From the definition above, we have that N_t is a subgroup of G, and*

$$N_t = \{h \in G | h^{-1} t h \in O_t\}.$$

Proof. Suppose $h_1, h_2 \in N_t$, then $O_t^{h_1 h_2} = (O_t^{h_1})^{h_2} = O_t^{h_2} = O_t$. So N_t is a subgroup of G. Also, we know that the conjugation and braid action commute. For $t' \in O_t$, there exists a braid element Q sending t to t'. Then if $h^{-1}t'h \in O_t$, $h^{-1}t'h = h^{-1}(tQ)h = (t^h)Q$. Hence $t^h \in O_t$. This proves the lemma. □

Let $\mathcal{H}$ be a head, then it is a union of B_L-orbits. Let J_1 be an B_L-orbit in $\mathcal{H}$. Note that the action of B_L does not change x. Suppose the product of tuples in J_1 is x_0^{-1}. We choose x_0 to be the class representative in C_x. Let K be the normalizer of J_1. It is easy to see that K is a subgroup of $C_G(x_0)$. By Definition 4.4, we have

$$\mathcal{H} = \bigcup_{s \in G} J_1^s,$$

and the number of B_L-orbits is $[G : K]$. We define the x_0-trace of $\mathcal{H}$ to be the subset $\{(g_1, \ldots, g_k) \in \mathcal{H} : g_1 \ldots g_k = x_0\}$.

Lemma 4.6 *The x_0-trace of $\mathcal{H}$, denoted by $Tr(x_0)$, is an orbit for the group $B_L \times C_G(x_0)$, where $C_G(x_0)$ denotes the centralizer of x_0 in G.*

Proof. Suppose $t = (g_1, \ldots, g_k)$ with product x_0. For any tuple $t' \in \mathcal{H}$, there exists a braid element $Q \in B_L$ and $h \in G$, such that $t' = (t^h)Q = (g_1^h, \ldots, g_k^h)Q$. If $t' \in Tr(x_0)$, then $(g_1 \ldots g_k)^h = x_0^h = x_0$, which means that $h \in C_G(x_0)$. Hence the orbit $Tr(x_0)$ is under the action of $B_L \times C_G(x_0)$. □

In fact, the x_0-trace identifies $\mathcal{H}$ as the head $\mathcal{H}$ is a disjoint union of G-orbits of x_0-trace. Each trace is also a union of B_L-orbits, that is,

$$Tr(x_0) = \bigcup_{s \in C_G(x_0)} J_1^s,$$

and the number of B_L-orbits is $[C_G(x_0) : K]$. Similarly, we can define the y_0-trace for the tails $\mathcal{T}$ containing an B_R-orbit J_2 and have the same result. Choosing $y_0 = x_0^{-1}$, by Lemma 4.6, we have the following result.

Proposition 4.7 *The nodes in $\mathcal{H} \times \mathcal{T}$ bijectively correspond to the orbits of $B_L \times B_R \times C_G(x_0)$ in $Tr(x_0) \times Tr(x_0^{-1})$, where the tuples are generating tuples.*

Proof. Let (J_1, J_2) be a matching pair, and the normalizer of J_1 is the subgroup K in $C_G(x_0)$. By Lemma 4.3, we let J be the node with $J = (J_1, J_2)^G$. By our assumption, (J_1, J_2) is a matching pair in $Tr(x_0) \times Tr(x_0^{-1})$. Hence $(J_1, J_2)^{C_G x_0}$ is an orbit of $B_L \times$

$B_R \times C_G(x_0)$, which is also a subset of J. This inclusion gives us a bijection between nodes in $\mathcal{H} \times \mathcal{T}$ and orbits of $B_L \times B_R \times C_G(x_0)$ in $Tr(x_0) \times Tr(x_0^{-1})$. $\square$

Suppose t is a tuple in J and H_t is the subgroup of G generated by t. It is easy to see that the centralizer of H_t in G is a subgroup of N_t. Furthermore, we have the following proposition.

Proposition 4.8 *Let $t_1 = (g_1, g_2, \ldots, g_k)$ and $x = (g_1 \ldots g_k)^{-1}$. Suppose J_1 is the orbit of t_1 under the action of B_L. Then the cyclic group $\langle x \rangle$ is a subgroup of the normalizer of J_1.*

Proof. We prove this by induction. For the tuple (g_1, g_2), and $x_0 = (g_1 g_2)^{-1}$, the braid element $Q_1^{-1} Q_1^{-1} \in L$ sends (g_1, g_2) to $(g_1^{x_0}, g_2^{x_0})$. Hence it is true for a tuple of length 2. Assume that for $t_1 = (g_1, g_2, \ldots, g_{k-1})$, there exists a braid element Q' in B_L mapping t_1 to $t_1^y = (g_1^y, g_2^y, \ldots, g_{k-1}^y)$, where $y = (g_1 \ldots g_{k-1})^{-1}$. Let $x = g_k^{-1} y$, then

$$\begin{aligned}
Q' &: (g_1, \ldots, g_{k-1}, g_k) \to (g_1^y, \ldots, g_{k-1}^y, g_k) \\
Q_{k-1}^{-1} \ldots Q_2^{-1} Q_1^{-1} &: (g_1^y, \ldots, g_{k-1}^y, g_k) \to (g_k^y, g_1^y, \ldots, g_{k-1}^y) \\
Q_1^{-1} Q_2^{-1} \ldots Q_{k-1}^{-1} &: (g_k^y, g_1^y, \ldots, g_{k-1}^y) \to (g_1^x, \ldots, g_{k-1}^x, g_k^y).
\end{aligned}$$

Since $g_k^y = g_k^{g_k^{-1} y} = g_k^x$, we have shown that t_1^x is in J_1. This proves the proposition. $\square$

For an orbit J_2 under the action of B_R, we have the similar results. Recall the definition of the double coset in a group G.

Definition 4.9 *Let K_1 and K_2 be two subgroups of group G. A* **double coset** *of (K_1, K_2) in G is an equivalence class of elements under the equivalence relation defined by $x \sim y$ if there exist $k \in K_1$ and $l \in K_2$, such that $kxl = y$.*

By Proposition 4.7, we only need to identify the nodes in the orbits of $B_L \times B_R \times C_G(x_0)$ on $Tr(x_0) \times Tr(x_0^{-1})$ in order to compute all the nodes in the Nielsen class. The next proposition gives a precise description of how these nodes look like.

Proposition 4.10 *The nodes in the orbits of $B_L \times B_R \times C_G(x_0)$ on $Tr(x_0) \times Tr(x_0^{-1})$ correspond to the double coset representatives $\{r_1, \dots, r_m\}$ of N_1 and N_2 in $C_G(x_0)$, where N_1 is the normalizer of a B_L-orbit J_1, and N_2 is the normalizer of a B_R-orbit J_2, such that $(J_1^{r_i}, J_2)$ is a matching pair, $1 \leqslant i \leqslant m$.*

Proof. Suppose $t = (g_1, \dots, g_r)$ is a generating tuple, where $g_1 \dots g_k = x_0$, and J is the node in $Tr(x_0) \times Tr(x_0^{-1})$ for the group $B_L \times B_R \times C_G(x_0)$ containing t. Then, there exists a tuple t_1 in an B_L-orbit J_1 of $Tr(x_0)$, t_2 in an B_R-orbit J_2 of $Tr(x_0^{-1})$, and h_i in $C_G(x_0)$, $i = 1, 2$, such that $t_1^{h_1} = (g_1, \dots, g_k), t_2^{h_2} = (g_{k+1}, \dots, g_r)$. By Lemma 4.5, N_1 and N_2 are subgroups of $C_G(x_0)$. Hence there exists a double coset $N_1 s N_2$ in $C_G(x_0)$ containing $h_1 h_2^{-1}$. Since $t^{h_2^{-1}} \in J$, we have a bijection $J_1^{h_1 h_2^{-1}} \times J_2 \to s$. Since $t^{h_2^{-1}}$ is also a generating tuple, we have that (J_1^s, J_2) is a matching pair. □

This proposition implies that in order to identify all the nodes, we first need to record J_1 and J_2 together with their corresponding normalizers, and compute the list of double cosets. Then, we search the matching pairs to obtain nodes.

Lemma 4.11 *Let the tuples obtained from the matching pair (J_1, J_2) be of type C_x, that is, the product of the tuple in J_2 lies in C_x. Define C_0 to be the class of the identity in G. Suppose we have a generating tuple t' of type C_0, in which the two subgroups generated by J_1 and J_2 do not centralize each other. Then t' is braid equivalent to a generating tuple of type C_x, $x \neq 1$.*

Proof. Since $Q_1 \dots Q_{r-1}$ is a cyclic permutation of any tuple, without loss of generality, let the generating tuple $t' = (g_1, \dots, g_r)$, where $g_1 \dots g_k = 1$, and assume $g_m g_n \neq g_n g_m$

for $1 \leqslant m \leqslant k, k < n \leqslant r$. Then we have

$$\begin{aligned} Q_m^{-1} \dots Q_{k-1}^{-1}: \quad & (g_1, \dots, g_k) \to t_1 = (g_1, \dots, g_{m-1}, g_{m+1}^{g_m^{-1}}, \dots, g_k^{g_m^{-1}}, g_m); \\ Q_{n-1} \dots Q_{k+1}: \quad & (g_{k+1}, \dots, g_r) \to t_2 = (g_n, g_{k+1}^{g_n}, \dots, g_{n-1}^{g_n}, g_{n+1}, \dots, g_r). \end{aligned}$$

Hence $t^{'}$ is braid equivalent to (t_1, t_2) which is a tuple of type C_0. Since $g_m g_n \neq g_n g_m$, the tuple $(t_1, t_2)Q_k^2$ is of type C_x for some $x \neq 1$, which can be sent to a tuple t of ramification type $\overline{C}$ by the braid element $Q_{k-1} \dots Q_m Q_{k+1} \dots Q_{n-1}$. The tuple t is of type C_x. This means that there exists a pure braid element in B_P mapping $t^{'}$ of type C_0 to a tuple t of type C_x, where $x \neq 1$. $\square$

The next lemma follows immediately from Lemma 4.11.

Lemma 4.12 *Let $t = (g_1, \dots, g_k, g_{k+1}, \dots, g_r)$ be a tuple of type C_{x_0} such that the two subgroups generated by $(g_1, \dots, g_k)$ and $(g_{k+1}, \dots, g_r)$ do not centralize each other. Let O_t be the orbit of $B \times C_G(x_0)$ containing t. Denote the set of nodes J contained in O_t from the action of $B_L \times B_R \times C_G(x_0)$ by $\{J\}$. If there exists a tuple $t^{'} \in O_t$ of type C_0, then $t^{'}$ lies in a known node in $\{J\}$.*

In view of this lemma, we can disregard those matching tuples of type C_0 during the computation in order to minimize the number of matching pairs needed to compute.

4.2 The Functions for The Matching Algorithm

In this section we introduce the functions of our matching algorithm. Suppose $t = (t_1, t_2)$, and the ramification types of t_1 and t_2 are $\overline{C}_1$ and $\overline{C}_2$ respectively. Let C_x be the conjugacy class of G containing x. First, we need to compute all the orbits of ramification type $(\overline{C}_1, C_x)$ and $(C_{x^{-1}}, \overline{C}_2)$ separately, and then store them in files.

Lemma 4.13 *The map $f : G^r \to G^r$ defined by*

$$f : (g_1, \ldots, g_r) \to (g_r^{-1}, \ldots, g_1^{-1})$$

induces a permutation on braid orbits.

Proof. It is easy to see that $f \circ Q_i^{-1} = Q_{r-i} \circ f$ for $1 \leqslant i \leqslant r-1$. Hence the lemma holds. □

The above lemma tells us that the methods of computing the left half and right half are essentially the same, so here we only discuss the computation of the right half. This is done by our function **GenerateOrbits**. It is in the form

GenerateOrbits(n,$\overline{C}_2$,partition,G),

where n is any number for labeling the output files, $\overline{C}_2$ is the ramification type which is a list of integers representing the index of the conjugacy classes, and "partition" is a list of integers representing the length of each block in the partition of $\overline{C}_2$. The function will first calculate the character table of G, and then use the given ramification type $\overline{C}_2$ to call **AllBraidOrbits** from the BRAID package for computing orbits of type $(C_{x_0^{-1}}, \overline{C}_2)$ using the action of the group $B_R \times G$, where x_0 ranges over all the non-identity class representatives. When this computation finishes, the orbits will be represented by their corresponding x_0-traces, and saved in the directories labeled as "mOrbitsForClass$i$", where i represents the index of the conjugacy class of x. If $\overline{C}_1 = \overline{C}_2$, we only need to compute this once. Otherwise, by Lemma 4.13 we will choose the proper ramification type and a different number m to call the function. When this is done, we will have tuples in the form (x_0, s_1) from the heads, and (x_0, s_2) from the tails. Here s_2 is a tuple with ramification type $\overline{C}_2$, but the ramification type of s_1 is in fact the reversed $\overline{C}_1$ and every entry being replaced by its inverse. In this way we guarantee the product one condition on the them.

The next step is to use **AllMatchingPairs** to start matching, and collect the matching pairs that are nodes from the action of $B_L \times G \times C_G(x_0)$. The function

$$\text{AllMatchingPairs(case},m,n)$$

takes three inputs, with "case" being any string that is used to label outputs. The numbers m and n must be the same as what we used in **GenerateOrbits**, meaning that we match the orbits labeled by m to those labeled by n. The matching process has two stages. At stage one, for each conjugacy class, it will generate a matching list. The list is formed by orbits in each conjugacy class together with the corresponding normalizers. In fact, our function uses Lemma 4.5 to compute the normalizer of each orbit. After this, it will start matching immediately within the list in stage two. For each pair of orbits J_{s_i}, $i = 1, 2$, **AllMatchingPairs** will list all the double coset representatives h of N_{s_i} in the $C_G(x_0)$, and then check if $(J_{s_1}^h, J_{s_2})$ forms a matching pair before storing them in the file indexed by the position of C_{x_0} in the character table of G. Also, when the function finishes, it will show the total number of matching pairs on the screen. Therefore, using Proposition 4.7, Proposition 4.10 and Lemma 4.11, for every non-identity conjugacy class representative x_0, the computation of all the matching pairs of type C_{x_0} gives us all the nodes in $\mathcal{N}(\overline{C})$. The user can also use this function to determine whether there exist generating tuples of the ramification type $\overline{C}$ for the group G using the half orbits obtained. If it shows "The total number of vertices is 0" at the end, then it means that there is no generating tuple.

Definition 4.14 *Let K be the graph whose vertices are the nodes in $\mathcal{N}(\overline{C})$. We say two vertices v and v' in K have an edge between them if there exists a matching tuple t in v and t' in v', such that t and t' are braid equivalent.*

Proposition 4.15 *The number of braid orbits of ramification type $\overline{C}$ is equal to the number of connected components in K.*

Once we have all of the nodes needed, we can start connecting them into components. To do this we first call the routine

AllNeighbors(case,m,n,a,b).

This function will find a couple of neighbors for each vertex. That is, from each node, we first pick a random tuple, and apply pure braid elements Q_{ij} to generate new tuples, where $1 \leqslant i \leqslant k-1, k+1 \leqslant j \leqslant r-1$. In other words, we apply the pure braid elements that go across the splitting line, and then collect the neighbors of the tuple that lie in different nodes, which gives a braid equivalence among them. Each vertex is given by a coordinate (u, v), where u represents the position of the conjugacy class C_{x_0} and v is a number representing the position of the node in the list of matching pairs of type C_{x_0}. For example, if there are a total 100 matching pairs of type C_{x_0}, where the conjugacy class C_{x_0} is the 4th class in the character table of G, then the first matching pair will be assigned as coordinate [4,1], and the last one will be [4,100]. This coordinate system simplifies the connecting process in the final stage.

Finding neighbors for one vertex takes approximately 1 to 5 seconds, depending on the number of orbits and length of the tuple. If it seems to take a long time to finish computing all vertices within one job, the user can do a parallel computation for this process. We can divide them into several intervals, and put them in different jobs. For example, suppose that we have 10,000 vertices in total, and we want to use 10 GAP programs to run this simultaneously with each one computing 1,000 vertices. All we need to do is to divide $[1, 10000]$ into $\{[1, 1000], [1001, 2000], \dots\}$, and assign each GAP to run "AllNeighbors" with a being the first integer and b being the second integer of the interval. If not, we just assign $a = 1$ and b to be the total number of vertices. This method guarantees that we will not struggle on the time issue on this stage. After checking neighbors for every vertex, the function will save the vertices together with corresponding known neighbors in a list, which is one of the inputs needed for the final step, to connect the vertices with neighbors.

We run **ConnectVertices** to start connecting. Using the coordinate system, this function will join those vertices that share the same neighbors together, and then save the list of components in a file labeled by the other input. Each component consists of nodes that are braid equivalent. There is a small chance that **AllNeighbors** will only find loops for some isolated vertices at first try due to the randomness of choosing matching tuples. Based on the data we already have, the probability is below 1% depending on the group G and complexity of the computation. The user can run the subroutine **FindNeighbors** to re-check these vertices manually, or using BRAID program to test on generating tuples contained in these vertices, until they are convinced that no more isolated vertices can be joined to the known components, or there are no vertices left.

Note that this algorithm is not suitably designed to verify the number of braid orbits of certain ramification type when there are more than one orbit. The approach of our matching algorithm loses some braid relation in the computation in order to obtain all the generating tuples successfully. The connecting process is a replacement for recovering those braid relation which greatly relies on the random choice of tuples. For a relatively large Nielsen class $\mathcal{N}(\overline{C})$, if there is only one braid orbit, then it is very likely to connect all vertices together within reasonable time using this algorithm. However, for those large and difficult cases that may contain more than one orbit, our algorithm can not determine the exact number of full braid orbits, but gives an upper bound. The next chapter is devoted to discussing the use of this algorithm for the generating types with large structure constant we have found.

Chapter 5

Results

5.1 Applications

In this section, we first use the example of AGL$(4, 2)$ to describe the process of computing the nodes and the number of braid orbits using our matching algorithm. Recall the GAP codes we have shown in Section 4 of Chapter 3. The function **AllMatchingPairs** shows that there is no generating tuple for the tuple type [6,4,4,4,4,4,4]. The involution of index six is a product of two involutions of index 4. Thus the fact that there exists no genus zero system of type [6,4,4,4,4,4,4] implies that there does not exit a genus 1 system of [4,4,4,4,4,4,4,4] either. The tuple type [6,6,6,4,4,4] corresponds to the ramification type $\overline{C}$=(2B,2B,2B,2D,2D,2D) represented by using the labels from the character table of AGL$(4, 2)$ in GAP. The number represents the order of the elements in the conjugacy class, and the letter separates different classes with the same order. The structure constant for $\overline{C}$ is $21, 267, 671, 040$. This case is too large to be dealt with our standard BRAID package at the moment. However, after splitting $\overline{C}$ from the middle into (2B,2B,2B) and (2D,2D,2D), we will be looking at B_L-orbits and B_R-orbits of tuples of length 3, which can be easily computed within reasonable time. This case do have generating tuples, which means that there exist genus one systems for the case [6,6,4,4,4,4,4]. We state the results from computing the half orbits of these two cases in Table 5.1.

Table 5.1: Results from the function **GenerateOrbits**

half	# orbits	time spent	type with the most orbits
(2B,2B,2B)	155	2 mins	(2B,2B,2B,2B)
(2D,2D,2D)	619	57 mins	(4B,2D,2D,2D)
(2B,2B,2B,2B)	3076	$>$ 48 hours	(4D,2B,2B,2B,2B)
(2B,2D,2D)	316	28 mins	(4B,2B,2D,2D)

The next step is to match the orbits obtained and form matching pairs. Recall that we define the matching tuples(pairs) to be of type C_x if the product of the first half tuple is contained in the conjugacy class C_x. There are 24 non-identity conjugacy classes for $\mathrm{AGL}(4,2)$. Table 5.2 gives the number of nodes for each case, followed by the type that contains the most nodes, the least nodes and the number of types that have no nodes. The reason that we do not have any nodes for certain types is either there are no tuples generating the group, or there are no B_L-orbits(or B_R-orbits).

Table 5.2: Results from the function **AllMatchingPairs**

case	# total pairs	most pairs	least pairs	# types with no pairs
(2B,2B,2B,2D,2D,2D)	903	3A	4E	11
(2B,2B,2B,2B,2D,2D,2D)	4076	2B	15A,15B	2

As shown in Table 5.2, the first case with structure constant over 21 billion is now a connecting process of only 903 vertices. We run **AllNeighbors** to generate enough neighbors for each vertex, which takes only 10 to 12 minutes, and the function **ConnectVertices** successfully connects all vertices into one component. For the second case, we connected 4062 vertices into one component at first try, with 14 vertices left. After re-computing neighbors for those isolated vertices with several tries, we connected them into the previous component. This also proves that there should be only one component for the first case.

Via lifting invariant, Fried [4] showed that for the alternating group A_n with non-zero

Table 5.3: Results from BRAID and The Matching Algorithm

	total time spent	1st orbit	2nd orbit	file size
BRAID	4 hours	160,020 tuples	181,440 tuples	101 MB
Matching	5 mins	104 vertices	99 vertices	2.74 MB

genus, there are exactly two braid orbits for the ramification type that consists of 3-cycles. In Table 5.3, we compare the computations using the standard BRAID package and the program for matching pairs on the case $\mathcal{H}(A_5, 0, C_1, ..., C_7)$, where C_i is the conjugacy class of 3-cycle $(1, 2, 3)$, $1 \leqslant i \leqslant 7$. The structure constant for this type is $21, 347, 340$. The BRAID program spent 4 hours to find the two orbits in the Nielsen class, and then another 1 hours and 40 minutes to verify this. The first orbit file that contains over 100,000 admissible tuples generated by BRAID has size over 48MB already. Using the matching algorithm, the number of components can be evaluated faster, and the files will not require a large amount of space, which makes them easy to access.

The final results of genus zero systems for all the affine primitive groups of degree p^e are shown in Appendix B. Note that the cases of groups of degree 8 have been done by Matthew Badger in his Mphil thesis [3] in 2008.

5.2 Future Work

The computational method introduced in this thesis is aimed to solve the restrictions of the BRAID package when dealing with long Nielsen classes. At the first stage, after breaking up tuples with large structure constants into shorter ones, the computation of their orbits will not be a big issue. Every braid orbit has the property, that the tuples contained in an orbit generate the same subgroup of G. This guarantees that the recombination of shorter classes of tuples into generating orbits only needs to check one tuple from each orbit. This process usually will not cause any limiting issue in our timeframe after having all the shorter classes needed.

However, since every short class should be handled properly by the BRAID package, more than two pieces will need to be computed separately when the tuples are twice longer than the limitation of BRAID. This may give a large amount of short classes, and therefore increases the number of possible combinations of classes before the process of checking if they are generating. We believe that it is possible to find such a balance between computing longer tuples and computing more shorter ones in the future.

In Chapter 4, the process of connecting vertices is based on a graph with every vertex having known neighbors. There is no unique way of finding neighbors for each vertex, and we may keep having loops after several attempts when there are potential neighbors. Suppose we choose a random matching tuple $t = (g_1, \dots, g_k, |g_{k+1}, \dots, g_r)$ which is split into two halves as before. The braid elements that send t to a new tuple with the same ramification type which is more likely to lie in different matching pairs are the pure braid element Q_{ij} with $1 \leqslant i \leqslant k, k+1 \leqslant j \leqslant r$. It is not clear that how trivial the tuple is and how many neighbors we need for the connecting. If most entries of t commute each other, then this may add unnecessary computation during the whole process. Also, computing more neighbors for each vertex may take considerable time, whilst less neighbors may lose some edge connections needed when connecting the graph.

There are several things that can be considered if we want to reduce the number of vertices in the graph. For example, first, the subgroup $B_L \times B_R$ of B_P is relatively a small group we can use for the replacement. One may think of using another group $B_L^{'} \supset B_L$ and $B_R^{'} \supset B_R$ without influencing the braid equivalence between the matching tuples in the orbits of $B_L^{'} \times B_R^{'}$. That is, for the tuple $t = (t_1, t_2)$, instead of using the group B_L on $t_1 = (g_1, \dots, g_k)$ and B_R on $t_2 = (g_{k+1}, \dots, g_r)$, we consider the group $B_L^{'}$ acting on $(g_1, \dots, g_k, x)$ and $B_R^{'}$ on $(x^{-1}, g_{k+1}, \dots, g_r)$, where $x = (g_1 \dots g_k)^{-1}$. Then we join the tuples with certain condition such that the resulting tuple lies in the orbit of t under the action of B_P. This requires more work on the conditions, which is unclear at the moment.

However we explain some work done for this attempt in the following.

Let $\overline{C}_1 = \{C_1, \dots, C_k\}$ and $\overline{C}_2 = \{C_{k+1}, \dots, C_r\}$ with the partitions being P_1 and P_2 respectively, where $1 \leqslant k < r, P = P_1 \cup P_2$. Let B_L be the subgroup of B_P that only acts on $\overline{C}_1$ and B_R be the subgroup of B_P that only acts on $\overline{C}_2$ as before. Suppose $x = g_1 \dots g_k$. We add the extra element x^{-1} to t_1 so that the product of the new tuple $(t_1, x^{-1}) = (g_1, \dots, g_k, x^{-1})$ is still 1. The parabolic subgroup that fixes both P_1 and the position of the last entry is generated by B_L and the pure braid elements $Q_{i,k+1}, 1 \leqslant i \leqslant k$. We have

$$Q_{i,k+1} : (g_1, \dots, g_k, x^{-1}) \to (g_1, \dots, g_{i-1}, H_1, g_{i+1}, \dots, g_k, a_i^{-1} x^{-1} a_i),$$

where $H_1 = g_i^{g_{i+1} \cdots g_k \cdots g_{i+1}^{-1}}$, and

$$a_i = g_1 \dots g_{i-1} g_i g_{i-1}^{-1} \dots g_1^{-1}.$$

Similarly, for $(x, t_2) = (x, g_{k+1}, \dots, g_r)$, using the pure braid element Q_{1j}, $2 \leqslant j \leqslant n-k+1$, we have

$$Q_{1j} : (x, g_{k+1}, \dots, g_r) \to (b_j^{-1} x b_j, g_{k+1}, \dots, g_{k+j-2}, H_2, g_{k+j}, \dots, g_r),$$

where $H_2 = g_{k+j-1}^{g_{k+j-1}^{-1} \cdots g_{k+1} \cdots g_{k+j-1}}$, and

$$b_j = g_{k+1} \dots g_{k+j-2} g_{k+j-1} g_{k+j-2}^{-1} \dots g_{k+1}^{-1}.$$

We want to compute the orbits of (t_1, x^{-1}) and (x, t_2) separately, and then join the tuples together to form a generating tuple. However, those pure braid elements above may change the product of the half tuple. Let the group generated by the tuple t_1 be H_1. If h

is an element in the centralizer of x^{-1} in H_1, it is not always true that the tuple (t_1^h, t_2) is in O_t. For example, in Table 5.2, the first case (2B,5A,15B) has only one braid orbit up to conjugation of AGL$(4,2)$ on the tuple. Let t_1 be the tuple of ramification type (2B) , and t_2 be of ramification type (5A,15B). In the following GAP code, the tuple t displayed is the generating tuple of ramification type (2B,5A,15B). We can see that for a non-identity element h in the centralizer of x^{-1}, the tuple (t_1^h, t_2) is clearly not braid equivalent to t.

```
gap> g;
2^4.PSL(4, 2)
gap> t;
[ (2,5)(3,8)(10,13)(11,16), (1,4,10,9,7)(2,14,16,12,13)(3,8,15,6,11),
  (1,7,9,13,12,11,6,15,3,16,14,5,2,10,4) ]
gap> Group(t)=g;
true
gap> Centralizer(g,Group([t[2],t[3]]));
Group(())
gap> Centralizer(g,Inverse(t[1]));
<permutation group with 9 generators>
```

On the other hand, we define the following conjugation maps on any tuple t:

$$f_{a_i} : t \to a_i t a_i^{-1} \quad f_{b_j} : t \to b_j t b_j^{-1},$$

and let $Q^{'}_{i,k+1} = Q_{i,k+1} \circ f_{a_i}, Q^{'}_{1j} = Q_{1j} \circ f_{b_j}$. Regardless of the extra element x^{-1} and x, we consider action of the following two groups

$$B^{'}_L = \langle \{Q^{'}_{1k}, \ldots, Q^{'}_{k-1,k}\}, B_L \rangle,$$

$$B^{'}_R = \langle \{Q^{'}_{12}, \ldots, Q^{'}_{1,n-k+1,}\}, B_R \rangle$$

on half tuples of ramification type $\overline{C}_1$ and $\overline{C}_2$ respectively. We have the next lemma.

Lemma 5.1 *For $1 \leqslant i \leqslant k, 2 \leqslant j \leqslant n-k+1,$, we have $Q^{'}_{i,k+1} \in B_L$, and $Q^{'}_{1j} \in B_R$. Hence $B^{'}_L = B_L, B^{'}_R = B_R$.*

Proof. Here we only show that $Q'_{i,k+1} \in B_L$. The deduction of the right half is entirely similar. We already know from above that

$$Q'_{i,k+1} : t_1 \to (g_1^{a_i^{-1}}, \dots, g_{i-1}^{a_i^{-1}}, H_1^{a_i^{-1}}, g_{i+1}^{a_i^{-1}}, \dots, g_k^{a_i^{-1}}),$$

where $a_i = g_1 \dots g_{i-1} g_i g_{i-1}^{-1} \dots g_1^{-1}$. Define "$\sim$" to be the braid equivalence relation.

$$\begin{aligned}
t_1 &\sim (g_i^{g_{i-1}^{-1} \dots g_1^{-1}}, g_1, \dots, g_{i-1}, g_{i+1}, \dots, g_k) && (Q_{i-1}^{-1} \dots Q_1^{-1}) \\
&\sim (g_1^{a_i^{-1}}, \dots, g_{i-1}^{a_i^{-1}}, g_{i+1}^{a_i^{-1}}, \dots, g_k^{a_i^{-1}}, g_i^{a_i}) && (Q_1^{-1} \dots Q_{k-1}^{-1}) \\
&\sim (g_1^{a_i^{-1}}, \dots, g_{i-1}^{a_i^{-1}}, H_1^{a_i^{-1}}, g_{i+1}^{a_i^{-1}}, \dots, g_k^{a_i^{-1}}) && (Q_{k-1}^{-1} \dots Q_i^{-1}) \\
&= t_1 Q'_{i,k+1}
\end{aligned}$$

Hence

$$Q'_{ik} = Q_{i-1}^{-1} \dots Q_1^{-1} Q_1^{-1} \dots Q_{k-1}^{-1} Q_{k-1}^{-1} \dots Q_i^{-1}.$$

When $i = k+1$, $Q'_{ik} = Q_1^{-1} \dots Q_{k-1}^{-1} Q_{k-1}^{-1} \dots Q_i^{-1}$. Since Q'_{ik} fixes the positions of every entry of the tuple, it fixes the partition of $\overline{C}_1$, which means that $Q'_{i,k+1} \in B_L$. $\square$

Furthermore, from Proposition 4.8, we can make a conjecture that whether or not the conjugation of the subgroup $\langle x \rangle$ on tuples $(g_1, \dots, g_k)$ is the only one that we can get from the action of B_L. If not, then, by determining the conjugation subgroup, we can save some time when finding the normalizer for each shorter orbit.

Chapter A

Affine Primitive Permutation Groups

Table A.1: Affine Primitive Groups of Degree 8, 16, 32, 64 and 128

Degree 8
$\mathrm{AGL}(1,8), A\Gamma L(1,8), \mathrm{ASL}(3,2)$
Degree 16
$2^4:5, 2^4:D(2*5), \mathrm{AGL}(1,16), (A_4\times A_4):2, (2^4:5).4, \mathrm{AGL}(1,16):2,$ $2^4.S_3\times S_3, 2^4.3^2:4, A\Gamma L(1,16), (S_4\times S_4):2, 2^4.\mathrm{PSL}(4,2),$ $A\Gamma L(2,4), \mathrm{ASL}(2,4):2, \mathrm{AGL}(2,4), \mathrm{ASL}(2,4), 2^4.S_6, 2^4.A_6,$ $2^4:S_5, 2^4:A_5, 2^4.A_7$
Degree 32
$\mathrm{AGL}(1,32), A\Gamma L(1,32), \mathrm{ASL}(5,2)$
Degree 64
$2^6:9, 2^6:D_14, 2^6:D_18, 2^6:21, 2^6:9:3, 2^6:3^2:3, 2^6:7:6,$ $2^6:(7\times S_3), 2^6:3^2:S_3, 2^6:3^2:S_3, 2^6:9:6, \mathrm{AGL}(1,2^6),$ $2^6:(3\times 7:3), 2^6:7:9, 2^6:(3\wr A_3), 2^6:(7\times D_14),$ $2^6:3^2:D_12, 2^6:(3^2:3):4, 2^6:(S_3\times 7:3), 2^6:(7\times D_18),$ $2^6:(3\wr S_3), 2^6:(3\wr A_3):2, 2^6:(3\wr A_3):2,$ $2^6:7:9:3, 2^6:(3^2:3):Q_8, 2^6:(3^2:3):8, 2^6:(3^2:3):D_8, 2^6:7:7:6,$ $2^6:7^2:S_3, 2^6:3^3:D_12, 2^6:3^3:A_4, 2^6:7:9:6, 2^6:(3^2:3):SD_16,$ $2^6:(GL(2,2)\wr A_3), 2^6:3^3:S_4, 2^6:3^3:S_4,$ $2^6:(3^2:3):Q_8:3, 2^6:7^2:(3\times S_3), 2^6:(GL(2,2)\wr S_3),$ $2^6:(3^2:3):Q_8:S_3, \mathrm{AGL}(6,2), 2^6:(\mathrm{GL}(3,2)\wr 2), A\Gamma L(3,4),$ $A\Sigma L(3,4), \mathrm{AGL}(3,4), \mathrm{ASL}(3,4), 2^6:3.S_6, 2^6:3.A_6,$ $A\Gamma L(2,8), A\Sigma L(2,8), \mathrm{AGL}(2,8), 2^6:\mathrm{PSL}(2,8), 2^6:(6\times \mathrm{GL}(3,2)),$ $2^6:(3\times GL(3,2)), 2^6:Sp(6,2), 2^6:GO-(6,2), 2^6:O-(6,2), 2^6:S_8,$ $2^6:A_8, 2^6:S_7, 2^6:A_7, 2^6:\Sigma U(3,3), 2^6:SU(3,3),$ $2^6:\mathrm{GL}(3,2)$
Degree 128
$\mathrm{AGL}(1,2^7), A\Gamma L(1,2^7), \mathrm{AGL}(7,2)$

Table A.2: Affine Primitive Groups of Degree 256 Part 1

Degree 256
$2^8 : (\mathrm{GL}(2,2) \wr S_4), 2^8 : (GL(2,2) \wr A_4),$
$2^8 : (\mathrm{GL}(2,2) \wr D_8), 2^8 : (GL(2,2) \wr 2^2),$
$2^8 : (\mathrm{GL}(2,2) \wr 4), 2^8 : (3 \times (3^3 : 2^2)), 2^8 : 3^4 : 2^3, 2^8 : 3^4 : (2 \times 4),$
$2^8 : 3^4 : D_8, 2^8 : 3^4 : Q_8, 2^8 : (2 \times D_8), 2^8 : 3^4 : D_8.2, 2^8 : 3^4 : (2 \times D_8) : 2,$
$2^8 : 3^4 : (2^2 : 4) : 2, 2^8 : (3 \times (3^3 : A_4)), 2^8 : 3^4 : A_4 : 2, 2^8 : 3^4 : Q_8 : 2,$
$2^8 : 3^4 : Q_8 : A_4, 2^8 : (3 \times 3^3 : S_4), 2^8 : 3^4 : 4 : S_3,$
$2^8 : 3^3 : S_4 : S_3, 2^8 : 3^4 : Q_8 : S_3, 2^8 : 3^4 : 2^3 : S_4,$
$2^8 : 3^4 : Q_8 : S_4, 2^8 : (3 \times (3^3 : D_8)), 2^8 : 3^4 : D_8, 2^8 : 3^4 : (2 \times D_8),$
$2^8 : 3^4 : (2^2 : 4), 2^8 : 3^4 : D_16, 2^8 : 3^4 : SD_16, 2^8 : 3^4 : (2 \times D_8) : 2,$
$2^8 : 3^4 : D_16 : 2, 2^8 : 3^4 : SA_16 : 2, 2^8 : 3^4 : (2^2 : 4) : 2,$
$2^8 : (\mathrm{GL}(2,2) \wr 2^2), 2^8 : (\mathrm{GL}(2,2) \wr 4), 2^8 : 3^4 : SA_16 : 2^2,$
$2^8 : 3^4 : (2 \times D_8) : 4, 2^8 : (3 \times (3^3 : 4)), 2^8 : 3^4 : (2 \times 4), 2^8 : 3^4 : 8,$
$2^8 : 3^4 : (2^2 : 4), 2^8 : 3^4 : SA_16, 2^8 : 3^4 : (2^2 : 4) : 2, 2^8 : 3^4 : SA_16 : 2,$
$2^8 : (3 \times (3^2 : 2^2)), 2^8 : 3^3 : 2^2, 2^8 : 3^3 : 2^3, 2^8 : 3^2 : Q_8, 2^8 : 3^3 : 2^2 : 3,$
$2^8 : 3^3 : 2^2 : 3, 2^8 : 3^3 : A_4 : 2, 2^8 : (3^2 : Q_8) : 3, 2^8 : 3^3 : S_4,$
$2^8 : 3^3 : S_4, 2^8 : 3^3 : S_4, 2^8 : 3^4 : S_4 : 2, 2^8 : 3^2 : Q_8 : S_3,$
$2^8 : (3^2 : 2) : (2^2 \times 3), 2^8 : 3^3 : D_8, 2^8 : 3^3 : D_8, 2^8 : 3^3 : D_8,$
$2^8 : (D_6 \times (3^2 : D_8)), 2^8 : 3^2 : SD_16, 2^8 : (3 \times (3^2 : 4)), 2^8 : 3^3 : 4,$
$2^8 : 3^3 : (2 \times 4), 2^8 : 3^2 : 8, 2^8 : (3^2 \times 5^2) : SA_16 : 2,$
$2^8 : (3 \times (3 \times 5^2) : D_8), 2^8 : (3^2 \times 5^2) : D_8.2, 2^8 : (3^2 \times 5^2) : SA_16,$
$2^8 : 5^2 : (SA_16 : 2), 2^8 : 5^2 : D_8, 2^8 : (3 \times (5^2 : D_8)), 2^8 : (3 \times 5^2) : D_8,$
$2^8 : 5^2 : (Q_8 : 2), 2^8 : (3 \times 5^2) : (D_8.2), 2^8 : 5^2 : SA_16, 2^8 : (15 \times D_30),$
$2^8 : (3 \times (3 \times 5^2) : 2^2), 2^8 : (3 \times (3 \times 5^2) : 4), 2^8 : (5 \times D_10),$
$2^8 : 5^2 : 2^2, 2^8 : 5^2 : 4, 2^8 : (15 \times D_10), 2^8 : (3 \times (5^2 : 2^2)),$
$2^8 : (3 \times (5^2 : 4)), 2^8 : (5 \times D_30), 2^8 : (3 \times 5^2) : 2^2, 2^8 : (3 \times 5^2) : 4,$
$2^8 : (3^2 \times 5^2) : (2 \times 4), 2^8 : (3^2 \times 5^2) : D_8, 2^8 : (3^2 \times 5^2) : Q_8,$
$2^8 : 5^2 : (2 \times 4), 2^8 : 5^2 : D_8, 2^8 : 5^2 : Q_8, 2^8 : (3 \times 5^2) : (2 \times 4),$
$2^8 : (3 \times 5^2) : D_8, 2^8 : (3 \times 5^2) : (2 \times 4), 2^8 : (3 \times 5^2) : Q_8,$
$2^8 : (3^2 \times 5^2) : 8, 2^8 : 5^2 : 8, 2^8 : (15 \times S_3), 2^8 : (5 \times 3^2) : 2,$
$2^8 : (3 \times D_6 \times D_10), 2^8 : (3 \times (15 : 4)), 2^8 : (3 \times (5 : 4)), 2^8 : (5 \times S_3),$
$2^8 : D_30, 2^8 : 15 : 2^2, 2^8 : (S_3 \times (15 : 4)), 2^8 : (D_6 \times (5 : 4)),$
$A\Gamma L(1, 2^8), 2^8 : (3 \times (85 : 4)), 2^8 : (15 \times D_34), AGL(1, 2^8),$
$2^8 : (5 \times D_34), 2^8 : 85 : 4, 2^8 : 85 : 8, 2^8 : 85, 2^8 : 51, 2^8 : (3 \times D_34),$
$2^8 : D_34, 2^8 : (3 \times (17 : 4)), 2^8 : 17 : 4, 2^8 : 51 : 8, 2^8 : 17 : 8, 2^8 : 17$

Table A.3: Affine Primitive Groups of Degree 256 Part 2

$2^8 : L(3,2), 2^8 : (3 \times A_5), 2^8 : (5 \times A_5), 2^8 : GL(3,2) : 2,$
$2^8 : \mathit{\Gamma} L(2,4), 2^8 : (3 \times S_5, 2^8 : (S_3 \times A_5),$
$2^8 : (S_3 \times A_5), 2^8 : (3 \times S_5), 2^8 : PSL(2,8), 2^8 : (D_10 \times A_5),$
$2^8 : (S_3 \times S_5), 2^8 : (S_3 \times S_5), 2^8 : PGL(2,9), 2^8 : M_10,$
$2^8 : (15 \times A_5), 2^8 : (3 \times A_6), 2^8 : (GL(2,4) \times S_3),$
$2^8 : (3 \times \mathit{\Gamma} L(2,4)), 2^8 : (D_10 \times A_5).2, 2^8 : P\mathit{\Gamma} L(2,9),$
$2^8 : PSL(2,8) : 3, 2^8 : (3 \times D_10 \times A_5), 2^8 : (S_3 \times A_6),$
$2^8 : (3 \times A_6) : 2, 2^8 : (3 \times S_6), 2^8 : (S_3 \times \mathit{\Gamma} L(2,4)),$
$2^8 : PSL(2,17), 2^8 : (15 \times GL(2,4)).2, 2^8 : A_5^2, 2^8 : PSL(2,16),$
$2^8 : PSL(2,16), 2^8 : (S_3 \times S_6), 2^8 : Sym(7),$
$2^8 : (A_5 \wr S_2), 2^8 : (A_5 \wr S_2), 2^8 : A_5^2 : 2,$
$2^8 : (A_5 \wr S_2), 2^8 : (A_5 \wr S_2), 2^8 : (3 \times Alt(7)),$
$2^8 : PSL(2,16) : 2, 2^8 : PSL(2,16) : 2, 2^8 : (3 \times A_5^2),$
$2^8 : (3 \times PSL(2,16)), 2^8 : (3 \times PSL(2,16)), 2^8 : A_5^2 : 2^2,$
$2^8 : A_5^2 : 2^2, 2^8 : A_5^2 : 4, 2^8 : A_5^2 : 2^2, 2^8 : A_5^2 : 4,$
$2^8 : (S_3 \times Alt(7)), ASigmaL(2,16), ASigmaL(2,16), 2^8 : (5 \times PSL(2,16),$
$2^8 : (3 \times (A_5 \wr S_2)), 2^8 : (3 \times A_5^2) : 2,$
$2^8 : (3 \times A_5^2) : 2, 2^8 : (3 \times (A_5 \wr S_2), 2^8 : (3 \times A_5^2) : 2,$
$2^8 : (3 \times (PSL(2,16) : 2)), 2^8 : (3 \times PSL(2,16) : 2), 2^8 : SU(4,2),$
$2^8 : (S_5 \wr S_2), 2^8 : (S_5 \wr S_2), 2^8 : S_8,$
$2^8 : (5 \times PSL(2,16)) : 2, 2^8 : (3 \times (A_5 \wr S_2)) : 2,$
$2^8 : (3 \times (A_5 \wr S_2)) : 2, 2^8 : (3 \times SL(2,16)) : 4,$
$2^8 : (3 \times SL(2,16)) : 4, 2^8 : SU(4,2) : 2, 2^8 : (3 \times A_8), AGL(2,16),$
$2^8 : (3 \times (3 \times A_5^2) : 2), 2^8 : GU(4,2), 2^8 : (5 \times PSL(2,16)) : 4,$
$2^8 : (S_3 \times A_8), 2^8 : GL(2,16) : 2, 2^8 : (GL(2,4) \wr S_2) : 2,$
$2^8 : GL(2,4)^2 : 2^2, 2^8 : GU(4,2) : 2, 2^8 : A_9, 2^8 : A_9, A\mathit{\Gamma} L(2,16),$
$2^8 : (A_6 \wr S_2), 2^8 : (\mathit{\Gamma} L(2,4) \wr S_2), 2^8 : S_9,$
$2^8 : A_6^2 : 4, 2^8 : A_6^2 : 2^2, 2^8 : Sp(4,4), 2^8 : (S_6 \wr S_2),$
$2^8 : Sp(6,2), 2^8 : A_10, 2^8 : \mathit{\Gamma} Sp(4,4), 2^8 : (3 \times Sp(4,4)),$
$2^8 : S_10, 2^8 : (3 \times Sp(4,4)) : 2, 2^8 : (A_7 \wr S_2),$
$2^8 : O+(8,2), 2^8 : O-(8,2), 2^8 : SO+(8,2), 2^8 : SO-(8,2),$
$2^8 : (GL(4,2) \wr S_2), ASL(4,4), A\Sigma L(4,4), AGL(4,4),$
$A\mathit{\Gamma} L(4,4), 2^8 : Sp(8,2), AGL(8,2)$

Table A.4: Affine Primitive Groups of Degree 9, 27 and 81

Degree 9
$3^2 : 4, 3^2 : D(2 * 4), 3^2 : Q(8) = M(9), 3^2 : 8 = AGL(1,9), A\mathit{\Gamma}L(1,9),$ $3^2 : (2'A(4)), AGL(2,3)$
Degree 27
$[3^3.A(4), 3^3 : 13, 3^3(A(4) \times 2), 3^3.2.A(4), 3^3.S(4), AGL(1,27),$ $3^3.13.3, 3^3(S(4) \times 2), A\mathit{\Gamma}L(1,27), ASL(3,3), AGL(3,3)]$
Degree 81
$3^4 : 5, 3^4 : D_10, 3^4 : 10, 3^4 : D_16, 3^4 : SA_16, 3^4 : Q_8 : 2, 3^4 : 16, 3^4 : SD_16,$ $3^4 : D_20, 3^4 : 5 : 4, 3^4 : 20, 3^4 : 5 : 4, 3^4 : SA_16 : 2, 3^4 : 2^2 : 4 : 2, 3^4 : SA_16 : 2,$ $3^4 : 2^3 : 2^2, 3^4 : D_16 : 2, 3^4 : 2^{(2+2+1)}, 3^4 : Q_16 : 2, 3^4 : 2^{(2+1+2)},$ $3^4 : D_16 : 2, 3^4 : (2 \times Q_8) : 2, 3^4 : SA_16 : 2, 3^4 : SA_32, 3^4 : 40, 3^4 : (2 \times 5 : 4),$ $3^4 : 5 : 8, 3^4 : (4 \times D_10), 3^4 : 5 : 8, 3^4 : Q_8.S_3, 3^4 : (Q_8 : 3) : 2,$ $3^4 : (GL(1,3) \wr 4), 3^4 : 4^2 : 4, 3^4 : Q_8 : D_8, 3^4 : 2^3 : D_8, 3^4 : 8.D_8,$ $3^4 : 2^{(2+3+1)}, 3^4 : D_16 : 4, 3^4 : D_16 : 4, 3^4 : 2^{(2+2+2)}, 3^4 : 16 : 4, 3^4 : D_16 : 4,$ $3^4 : (SA_16 : 2) : 2, 3^4 : 5 : 16, 3^4 : (8 \times D_10), 3^4 : 20 : 4, 3^4 : 80, 3^4 : 5 : SA_16,$ $3^4 : 2^3 : A_4, 3^4 : (Q_8 : 2) : S_3, 3^4 : Q_8.S_3 : 2, 3^4 : (2 \times Q_8) : 6,$ $3^4 : Q_8.S_3 : 2, 3^4 : (SA_16 : 2) : 3, 3^4 : (Q_8 : 3) : 2^2, 3^4 : Q_8.S_3 : 2,$ $3^4 : (Q_8 : 3) : 4, 3^4 : (GL(1,3) \wr D_4), 3^4 : Q_16 : D_8, 3^4 : (4 \times 8) : 4,$ $3^4 : 2^{(2+3+1+1)}, 3^4 : 8.D_8 : 2, 3^4 : 2^{(2+3+1+1)}, 3^4 : 2^{(3+4)} : 4, 3^4 : D_16 : Q_8,$ $3^4 : 2^{(2+3+1+1)}, 3^4 : D_16 : 8, 3^4 : 40 : 4, 3^4 : 5 : 2^{(2+1+2)}, 3^4 : 16 : D_10,$ $3^4 : ((2 \times Q_8) : 2) : 5, 3^4 : (GL(1,3) \wr A_4), 3^4 : Q_8 : S_4,$ $3^4 : 2^3 : S_4, 3^4 : (2 \times Q_8) : A_4, 3^4 : 2^(3 + 2) : S_3,$ $3^4 : (Q_8.S_3 : 2) : 2, 3^4 : (SA_16 : 2) : 6, 3^4 : (SA_16 : 2) : 6, 3^4 : Q_8.S_3 : 4,$ $3^4 : Q_8.S_3 : 2^2, 3^4 : 2^{(2+3+1+2)}, 3^4 : (GL(1,3) \wr D_4) : 2,$ $3^4 : Q_8^2 : 4, 3^4 : 8^2 : 4, 3^4 : 8^2 : 4, 3^4 : (8.D_8 : 2) : 2, 3^4 : SL(2,3) : A_4,$ $3^4 : 16 : 5 : 4, 3^4 : ((2 \times Q_8) : 2) : D_10, 3^4 : (GL(1,3) \wr S_4),$ $3^4 : Q_8^2 : 6, 3^4 : Q_8^2 : S_3, 3^4 : GL(2,3) : D_8, 3^4 : 8^2 : D_8,$ $3^4 : SL(2,3) : S_4, 3^4 : (2^3 : A_4) : S_3, 3^4 : GL(2,3) : (3 \times S_3),$ $3^4 : ((2 \times Q_8) : 2) : 5 : 4, 3^4 : Q_8^2 : D_12, 3^4 : (SL(2,3) \wr 2),$ $3^4 : GL(2,3) : S_4, 3^4 : (2^3 : A_4) : S_3, 3^4 : (2^3 : 2^2) : (3^2 : 4),$ $3^4 : Q_8^2 : 3^2 : 4, 3^4 : Q_8^2 : S_3^2, 3^4 : (2^3 : 2^2) : 3^2 : D_8,$ $3^4 : (GL(2,3) \wr 2), AGL(4,3), ASL(4,3), 3^4 : 2.A_6, 3^4 : 4.A_6,$ $3^4 : 2.S_6, 3^4 : 8.A_6, 3^4 : SL(2,9) : 2^2, 3^4 : 4.A_6.2,$ $3^4 : 2.A_6 : D_8, AGL(2,9), 3^4 : 2.A_6 : Q_8, A\mathit{\Gamma}L(2,9), 3^4 : 2.A_5,$ $3^4 : 4.A_5, 3^4 : 2.A_5 : 2, 3^4 : 2.A_5.2, 3^5 : 4.S_5, 3^4 : 8.A_5,$ $3^4 : 4.S_5, 3^4 : 8.S_5, 3^4 : 2^{(1+4)}.A_5 : 2, 3^4 : 2^{(1+4)}.A_5,$ $3^4 : Sp(4,3) : 2, 3^4 : Sp(4,3), 3^4 : A_6, 3^4 : (2 \times A_6), 3^4 : S_6,$ $3^4 : A_6.2, 3^4 : 2.PGL(2,9), 3^4 : (2 \times S_6), 3^4 : (2 \times A_6.2),$ $3^4 : 2.P\mathit{\Gamma}L(2,9), 3^4 : A_5, 3^4 : S_5, 3^4 : S_5, 3^4 : 2.A_5,$ $3^4 : (2 \times S_5)$

Table A.5: Affine Primitive Groups of Degree 25, 125, 48, and 121

Degree 25:
$5^2 : 3, 5^2 : S(3), 5^2 : 6, 5^2 : Q(8), 5^2 : D(2*4), 5^2 : 8, 5^2 : D(2*6), 5^2 : Q(12),$ $5^2 : 12, 5^2 : 8 : 2, 5^2 : D(2*4) : 2, 5^2 : 3 : 8, 5^2 : 4 \times D(2*3), \mathrm{AGL}(1,25),$ $5^2 : (Q(8) : 3), 5^2 : O + (2,5), A\mathit{\Gamma} L(1,25), 5^2 : ((Q(8) : 3)'2),$ $5^2 : ((Q(8) : 3)'4), \mathrm{ASL}(2,5), \mathrm{ASL}(2,5) : 2, \mathrm{AGL}(2,5)$
Degree 125:
$5^3 : A_4, 5^3 : S_4, 5^3 : S_4, 5^3 : (2 \wr A_3), 5^3 : 31,$ $5^3 : (4 \times A_4), 5^3 : 2^2 : Q_12, 5^3 : 4^2 : 3, 5^3 : (2 \times S_4), 5^3 : 62,$ $5^3 : 31 : 3, 5^3 : (4 \times S_4), 5^3 : (2 \times 4^2 : 3), 5^3 : 4^2 : S_3,$ $5^3 : 4^2 : S_3, \mathrm{AGL}(1,5^3), 5^3 : (2 \times 91 : 3), 5^3 : 4^2 : Q_12,$ $5^3 : (4 \wr A_3), 5^3 : (2 \times 4^2 : S_3), A\mathit{\Gamma} L(1,5^3),$ $5^3 : (\mathrm{GL}(1,5) \wr S_3), \mathrm{ASL}(3,5), 5^3 : (\mathrm{SL}(3,5) : 2), \mathrm{AGL}(3,5),$ $5^3 : A_5, 5^3 : S_5, 5^3 : S_5, 5^3 : (2 \times A_5), 5^3 : (2 \times S_5),$ $5^3 : (4 \times A_5), 5^3 : (2 : S_5), 5^3 : (4 \times S_5)$
Degree 49:
$7^2 : 4, 7^2 : S(3), 7^2 : D(2*4), 7^2 : Q(8), 7^2 : 8, 7^2 : Q(12), 7^2 : D(2*6),$ $7^2 : 12, 7^2 : D(2*8), 7^2 : 16, 7^2 : Q(16), 7^2 : 3 \times D(2*3), 7^2 : 3 : D(2*4),$ $7^2 : 3 \times D(2*4), 7^2 : 24, 7^2 : 3 \times Q(8), 7^2 : Q(8) : 3, 7^2 : Q(8) : 3, 7^2 : Q(16) : 2,$ $7^2 : 3 \times Q(12), 7^2 : 3 \times D(2*6), 7^2 : (3 \times Q(16)), \mathrm{AGL}(1,49) = 7^2 : 48,$ $7^2 : 3 \times D(2*8), 7^2 : (Q(8)'D(2*3)), (\mathrm{AGL}(1,7) \times \mathrm{AGL}(1,7)) : 2,$ $7^2 : 3 \times (Q(8) : 3), A\mathit{\Gamma} L(1,49), 7^2 : ((Q(8)'D(2*3)) \times 3), \mathrm{ASL}(2,7),$ $\mathrm{ASL}(2,7) : 2, \mathrm{ASL}(2,7) : 3, \mathrm{AGL}(2,7)$
Degree 121:
$11^2 : 3, 11^2 : 4, 11^2 : 6, 11^2 : D_6, 11^2 : D_8, 11^2 : 8, 11^2 : Q_8, 11^2 : D_10,$ $11^2 : 12, 11^2 : Q_12, 11^2 : D_12, 11^2 : 15, 11^2 : SD_16, 11^2 : (2 \times D_10),$ $11^2 : (5 : 4), 11^2 : 20, 11^2 : D_24, 11^2 : 24, 11^2 : Q_24, 11^2 : (Q_8 : 3), 11^2 : 30,$ $11^2 : (5 \times D_6), 11^2 : (5 \times D_8), 11^2 : (5 : D_8), 11^2 : (5 \times Q_8), 11^2 : 40,$ $11^2 : SD_48, 11^2 : (Q_8 : D_6), 11^2 : (5 \times D_10), 11^2 : (5 \times D_12),$ $11^2 : (5 \times Q_12), 11^2 : 60, 11^2 : (5 \times SD_16), 11^2 : (10 \times D_10),$ $11^2 : (5 \times (5 : 4)), 11^2 : (5 \times D_24), 11^2 : 120, 11^2 : (5 \times Q_12),$ $11^2 : (5 \times (Q_8 : C_3)), 11^2 : (10 \times D_20), 11^2 : (10 \times D_24),$ $11^2 : (5 \times (Q_8 : D_6)), \mathrm{ASL}(2,11), 11^2 : (SL(2,11) : 2), 11^2 : (5 \times \mathrm{SL}(2,11)),$ $\mathrm{AGL}(2,11), 11^2 : (2.A_5), 11^2 : (5 \times 2.A_5)$

Chapter B

Genus Zero Systems

Table B.1: Genus Zero Systems for Affine Primitive Groups of Degree 8

group	ramification type	# of orbits	largest orbit	ramification type	# of orbits	largest orbit
$A\Gamma L(1,8)$	(3B,3B,6B)	2	1	(3A,3B,7B)	1	1
	(3A,3B,7A)	1	1	(3A,3A,6A)	2	1
$ASL(3,2)$	(4B,3A,7B)	2	1	(4B,3A,7A)	2	1
	(4B,3A,6A)	4	1	(4B,3A,4C)	2	1
	(4B,4B,7B)	1	1	(4B,4B,7A)	1	1
	(4B,4B,6A)	2	1	(4B,4B,4C)	4	1
	(2C,4B,7B)	1	1	(2C,4B,7A)	1	1
	(2B,7B,7B)	1	1	(2B,7A,7A)	1	1
	(2B,6A,7B)	1	1	(2B,6A,7A)	1	1
	(2B,3A,3A,3A)	1	120	(2B,4C,7B)	1	1
	(2B,4C,7A)	1	1	(2B,4B,3A,3A)	1	84
	(2B,4B,4B,3A)	1	66	(2B,4B,4B,4B)	1	36
	(2B,2C,3A,3A)	1	30	(2B,2C,4B,3A)	1	24
	(2B,2C,4B,4B)	1	24	(2B,2B,3A,7B)	1	21
	(2B,2B,3A,7A)	1	21	(2B,2B,3A,6A)	1	30
	(2B,2B,3A,4C)	1	24	(2B,2B,4B,7B)	1	14
	(2B,2B,4B,7A)	1	14	(2B,2B,4B,6A)	1	24
	(2B,2B,4B,4C)	1	24	(2B,2B,2C,7B)	1	7
	(2B,2B,2C,7A)	1	7	(2B,2B,2B,3A,3A)	1	864
	(2B,2B,2B,4B,3A)	1	648	(2B,2B,2B,4B,4B)	1	456
	(2B,2B,2B,2C,3A)	1	216	(2B,2B,2B,2C,4B)	1	192
	(2B,2B,2B,2B,7B)	1	147	(2B,2B,2B,2B,7A)	1	147
	(2B,2B,2B,2B,6A)	1	216	(2B,2B,2B,2B,4C)	1	192
	(2B,2B,2B,2B,2B,3A)	1	6480	(2B,2B,2B,2B,2B,4B)	1	4800
	(2B,2B,2B,2B,2B,2C)	1	1680	(2B,2B,2B,2B,2B,2B,2B)	1	48960

Table B.2: The Genus Zero System of AGL$(4, 2)$ Part 1

Using BRAID					
ramification type	# of orbits	largest orbit	ramification type	# of orbits	largest orbit
(2B,5A,15B)	1	1	(2B,2B,3B,7B)	1	7
(2B,5A,15A)	1	1	(2B,2B,3B,7A)	1	7
(2B,5A,14B)	1	1	(2B,2B,4B,5A)	1	80
(2B,5A,14A)	1	1	(2B,2B,4B,6C)	1	96
(2B,6C,15B)	1	1	(2B,2B,6A,5A)	1	120
(2B,6C,15A)	1	1	(2B,2B,6A,6C)	1	108
(2B,6C,14B)	1	1	(2B,2D,4D,5A)	1	90
(2B,6C,14A)	1	1	(2B,2D,4D,6C)	1	78
(2D,4F,15B)	1	1	(2B,2D,3A,5A)	1	60
(2D,4F,15A)	1	1	(2B,2D,3A,6C)	1	72
(2D,6A,15B)	3	1	(2B,2B,2B,2D,5A)	1	650
(2D,6A,15A)	3	1	(2B,2B,2B,2D,6C)	1	648
(2D,6A,14B)	2	1	(4B,4B,4D)	12	1
(2D,6A,14A)	2	1	(6A,4B,4D)	18	1
(2B,2D,2D,15B)	1	15	(6A,4B,4B)	32	1
(2B,2D,2D,15A)	1	15	(6A,6A,4F)	12	1
(2B,2D,2D,14B)	1	14	(6A,6A,4B)	52	1
(2B,2D,2D,14A)	1	14	(6A,6A,6A)	72	1
(4D,4F,5A)	6	1	(2B,2D,4B,4F)	1	96
(4D,4F,6C)	4	1	(2B,2D,4B,4B)	1	216
(4D,3B,7B)	1	1	(2B,2D,6A,4F)	1	84
(4D,3B,7A)	1	1	(2B,2D,6A,4B)	1	312
(4D,4B,5A)	6	1	(2B,2D,6A,6A)	1	414
(4D,4B,6C)	12	1	(2B,4F,4D,3B)	1	24
(4D,6A,5A)	18	1	(2B,3A,4D,3B)	1	30
(4D,6A,6C)	12	1	(2B,3A,3A,3B)	1	24
(3A,4F,5A)	2	1	(2D,2D,4D,4F)	1	88
(3A,4F,6C)	4	1	(2D,2D,4D,4B)	1	192
(3A,6A,5A)	10	1	(2D,2D,4D,6A)	1	336
(3A,6A,6C)	12	1	(2D,2D,3A,4F)	1	56
(2B,2B,4F,5A)	1	30	(2D,2D,3A,6A)	1	216
(2B,2B,4F,6C)	1	30	(2B,2B,2B,4D,3B)	1	216
(2B,2B,2B,3A,3B)	1	216	(2B,2B,2D,2D,4F)	1	576

Table B.3: The Genus Zero System of AGL$(4, 2)$ Part 2

Using Matching Algorithm							
ramification type	# of nodes	# of orbits	orbit length	ramification type	# of nodes	# of orbits	orbit length
(2B,2B,2D,2D,6A)	170	1	2448	(2B,2B,2B,2B,2B,3B)	107	1	1782
(2B,2D,2D,2D,4D)	63	1	1920	(2B,2B,2D,2D,4B)	151	1	1920
(2B,2B,2B,2D,2D,2D)	903	1	15168	(2B,2D,2D,2D,3A)	56	1	1512

Table B.4: Genus Zero Systems for Other Affine Primitive Groups of Degree 16

Group	ramification	# of	largest	Group	ramification	# of	largest
	type	orbits	orbit		type	orbits	orbit
$2^4 : D(2*5)$	(2A,5B,4C)	1	1	$2^4.A_6$	(2B,5B,5B)	4	1
	(2A,5B,4B)	1	1		(2B,5A,5A)	4	1
	(2A,5B,4A)	1	1		(3A,3B,5B)	2	1
	(2A,5A,4C)	1	1		(3A,3B,5A)	2	1
	(2A,5A,4B)	1	1		(2B,2B,2B,5B)	2	30
	(2A,5A,4A)	1	1		(2B,2B,2B,5A)	2	30
	(2A,2A,2A,4C)	1	12		(2B,2B,3A,3B)	1	36
	(2A,2A,2A,4B)	1	12		(2B,2B,2B,2B,2B)	2	864
	(2A,2A,2A,4A)	1	12				
$(A_4 \times A_4) : 2$	(2A,6B,6C)	1	1	$2^4 : S_5$	(2C,5A,12A)	1	1
	(2A,6A,6D)	1	1		(2C,5A,8A)	1	1
	(2A,2A,3E,3A)	1	1		(2E,6C,12A)	1	1
	(2A,2A,3D,3B)	1	1		(2E,6C,8A)	1	1
$(2^4 : 5).4$	(2A,4B,8B)	1	1		(2E,4E,12A)	1	1
	(2A,4A,8A)	1	1		(2C,2E,2E,12A)	1	6
$2^4 : S_3 \times S_3$	(2E,6B,6C)	3	1		(2C,2E,2E,8A)	1	8
	(2D,2E,2E,6C)	1	12		(2D,6C,5A)	3	1
	(2C,2E,2E,6B)	1	12		(2D,4E,5A)	3	1
	(2C,2D,2E,6A)	1	3		(2C,2E,2D,5A)	1	15
	(2C,2D,2E,2E,2E)	1	48		(2E,2E,2D,6C)	1	18
$2^4.3^2 : 4$	(2C,4D,8B)	2	1		(2E,2E,2D,4E)	1	24
	(2C,4C,8A)	2	1		(2C,2E,2E,2E,2D)	1	120
	(3A,4C,4D)	3	1				
$(S_4 \times S_4) : 2$	(2E,6B,8A)	1	1	$2^4 : A_5$	(2C,5A,5B)	3	1
	(2C,4F,12A)	1	1		(2C,6C,5B)	1	1
	(2C,6C,8A)	1	1		(2C,6C,5A)	1	1
	(2E,2C,2D,8A)	1	4		(2C,6B,5B)	1	1
	(2E,2C,2C,12A)	1	2		(2C,6B,5A)	1	1
	(2F,4F,6B)	3	1		(2C,6A,5B)	1	1
	(2E,2C,2F,6B)	1	6		(2C,6A,5A)	1	1
	(2E,2C,3A,4F)	1	6		(2C,2C,2C,5B)	1	30
	(2C,2D,2F,4F)	1	12		(2C,2C,2C,5A)	1	30
	(2C,2C,2F,6C)	1	6		(2C,2C,2C,6C)	1	18
	(2E,2E,2C,2C,3A)	1	12		(2C,2C,2C,6B)	1	18
	(2E,2C,2C,2D,2F)	1	24		(2C,2C,2C,6A)	1	18
					(2C,2C,2C,2C,2C)	1	576
$A\Gamma L(2,4)$	(2C,4C,5A)	1	1	$2^4.A_7$	(2B,4A,14B)	2	1
	(2C,4C,15A)	1	1		(2B,4A,14A)	2	1
	(3B,4C,6C)	4	1		(2B,7B,6A)	2	1
	(2B,2C,3B,4C)	1	20		(2B,7A,6A)	2	1
ASL$(2,4) : 2$	(2C,5A,6A)	2	1		(2B,5A,7B)	2	1
	(2B,6A,6A)	2	1		(2B,5A,7A)	2	1
	(2B,2C,2C,5A)	1	10		(3B,3A,7B)	1	1
	(2B,2B,2C,6A)	1	12		(3B,3A,7A)	1	1
	(4A,4A,3A)	2	1		(3B,4A,6A)	6	1
	(2B,2B,2B,2C,2C)	1	80		(3B,4A,5A)	10	1
$2^4.S_6$	(2B,2B,3B,5A)	1	10		(2B,2B,2B,7B)	2	21
	(6B,4D,3B)	2	1		(2B,2B,2B,7A)	2	21
	(6B,6B,3B)	6	1		(4A,4A,4A)	24	1
	(2B,2D,4D,3B)	1	12		(2B,2B,3B,4A)	1	192
	(2B,2D,6B,3B)	1	24				
	(2B,2B,2D,2D,3B)	1	108				

Table B.5: Genus Zero Systems for Affine Primitive Groups of Degree 32

group	ramification type	# of orbits	largest orbit	ramification type	# of orbits	largest orbit
$ASL(5,2)$	(2D,3B,31A)	1	1	(2D,8A,6F)	16	1
	(2D,3B,31B)	1	1	(2D,12A,6F)	16	1
	(2D,3B,31C)	1	1	(2D,6E,6F)	22	1
	(2D,3B,31E)	1	1	(2D,5A,6F)	18	1
	(2D,3B,31D)	1	1	(4A,4A,6F)	12	1
	(2D,3B,31F)	1	1	(4A,3B,8A)	12	1
	(2D,3B,30A)	1	1	(4A,3B,12A)	12	1
	(2D,3B,30B)	1	1	(4A,3B,6E)	24	1
	(2D,4J,21B)	2	1	(4A,3B,5A)	18	1
	(2D,4J,21A)	2	1	(4I,3B,4J)	18	1
	(6C,3B,4J)	12	1	(2D,2D,2D,6F)	1	720
	(2B,2D,3B,4J)	1	84	(2D,2D,4A,3B)	1	624

Table B.6: Genus Zero Systems for Affine Primitive Groups of Degree 64

group	ramification type	# of orbits	largest orbit	ramification type	# of orbits	largest orbit
$2^6 : 3^2 : S_3$	(2E,3F,12A)	1	1	(2E,6C,12B)	1	1
$2^6 : 7 : 6$	(2E,3B,12B)	1	1	(2E,3A,12A)	1	1
$2^6 : (3^2 : 3) : D_8$	(2G,4D,6D)	3	1	(2F,4D,6E)	3	1
$2^6 : (3^2 : 3) : SD_{16}$	(2E,4G,8D)	1	1	(2E,4G,8B)	1	1
$2^6 : (6 \times GL(3,2))$	(2F,3C,14A)	1	1	(2F,3C,14B)	1	1
$2^6 : S_7$	(2I,4N,6K)	4	1	(2I,4D,7A)	3	1
$2^6 : (\mathrm{GL}(2,2) \wr S_3)$	(2L,4N,6I)	4	1			
$2^6 : (\mathrm{GL}(3,2) \wr 2)$	(2J, 4Q, 14H)	1	1	(2J, 4Q, 14G)	1	1
	(2I, 2J, 2J, 7B)	1	1	(2I, 2J, 2J, 7A)	1	1
$2^6 : 7^2 : S_3$	(2C,3A,14C)	1	1	(2C,3A,14D)	1	1
	(2C,3A,14E)	1	1	(2C,3A,14F)	1	1
	(2C,3A,14G)	1	1	(2C,3A,14H)	1	1
$2^6 : A_7$	(2D,4F,7A)	2	1	(2D,4F,7B)	2	1
$2^6 : \mathrm{GL}(3,2)$	(2G,4F,8D)	1	1	(2G,4F,8B)	1	1
	(2G,4D,6C)	2	1			
$2^6 : S_8$	(2C,6L,6K)	4	1	(2C,4O,7A)	6	1
$2^6 : O-(6,2)$	(4H,6C,12I)	2	1	(2C,8E,12I)	6	1
$AGL(6,2)$	(2B,3B,15D)	4	1	(2B,3B,15E)	4	1

Table B.7: Genus Zero Systems for Affine Primitive Groups of Degree 9

group	ramification type	# of orbits	largest orbit	ramification type	# of orbits	largest orbit
$3^2 : 4$	(2A,4A,4A)	2	1	(2A,4B,4B)	2	1
$3^2 : D(2 \times 4)$	(2C,4A,6A)	1	1	(2A,4A,6B)	1	1
	(2A,2C,2C,6A)	1	2	(2A,2A,2C,6B)	1	2
	(2A,2C,2B,4A)	1	4	(2A,2A,2C,2C,2B)	1	8
$3^2 : (2'A_4)$	(3B,4A,3E)	1	1	(3B,6B,4A)	1	1
	(3B,6A,3D)	1	1	(3A,4A,3D)	1	1
	(3A,6B,3E)	1	1	(3A,6A,4A)	1	1
	(3B,3B,3B,2A)	1	1	(3A,3A,3A,2A)	1	1
$A\Gamma L(1,9)$	(2A,4A,8A)	1	1	(2A,4A,8B)	1	1
$\mathrm{AGL}(2,3)$	(2A,3C,8B)	1	1	(2A,3C,8A)	1	1
	(2A,6A,8B)	1	1	(2A,6A,8A)	1	1
	(2A,2A,2A,8B)	1	16	(2A,2A,2A,8A)	1	16
	(2A,2A,3A,3C)	1	12	(2A,2A,3A,4A)	1	12
	(2A,2A,3A,6A)	1	12	(2A,2A,2A,2A,3A)	1	216

Table B.8: Genus Zero Systems for Affine Primitive Groups of Degree 27

group	ramification type	# of orbits	largest orbit	ramification type	# of orbits	largest orbit
$3^3.A_4$	(2A,3B,9D)	1	1	(2A,3B,9B)	1	1
	(2A,3A,9C)	1	1	(2A,3A,9A)	1	1
$3^3(A_4 \times 2)$	(2B,3D,12B)	1	1	(2B,3D,12A)	1	1
	(2A,2B,2B,3D)	1	24			
$3^3.S_4$	(2B,4A,9B)	1	1	(2B,4A,9A)	1	1
$3^3(S_4 \times 2)$	(2E,4A,6G)	4	1	(2B,2E,2E,4A)	1	16
$\mathrm{ASL}(3,3)$	(2A,3F,13D)	2	1	(2A,3F,13C)	2	1
	(2A,3F,13B)	2	1	(2A, 3F,13A)	2	1
$\mathrm{AGL}(3,3)$	(2C,4A,13D)	1	1	(2C,4A,13C)	1	1
	(2C,4A,13B)	1	1	(2C,4A,13A)	1	1
	(3E,6E,4A)	8	1			

Table B.9: Genus Zero Systems for Affine Primitive Groups of Degree 49

group	ramification type	# of orbits	largest orbit	ramification type	# of orbits	largest orbit
$7^2 : 4$	(2A,4B,4B)	12	1	(2A,4A,4A)	12	1
$7^2 : 3 : D(2 * 4)$	(2A,4A,6C)	3	1	(2A,4A,6B)	3	1

Table B.10: Genus Zero Systems for Affine Primitive Groups of Degree 81

group	ramification type	# of orbits	largest orbit	ramification type	# of orbits	largest orbit
$3^4 : (\mathrm{GL}(1,3) \wr S_4)$	(6S,4C,6K)	2	1			
$3^4 : (2 \times S_5)$	(6K,4A,6M)	1	1			
$3^4 : S_5$	(6E,12A,3G)	1	1			
$\mathrm{AGL}(4,3)$	(2C,5A,8E)	1	1	(2C,5A,8F)	1	1

Table B.11: Genus Zero Systems for Affine Primitive Groups of Degree 121

group	ramification type	# of orbits	largest orbit	ramification type	# of orbits	largest orbit
$11^2 : 3$	(3A,3A,3A)	40	1	(3B,3B,3B)	40	1
$11^2 : 4$	(2A,4A,4A)	30	1	(2A,4B,4B)	30	1
$11^2 : 6$	(2A,3B,6B)	20	1	(2A,3A,6A)	30	1
$11^2 : (Q_8 : D_6)$	(2B,3A,8A)	5	1	(2B,3A,8B)	5	1

Table B.12: Genus Zero Systems for Affine Primitive Groups of Degree 25 and 125

group	ramification type	# of orbits	largest orbit	ramification type	# of orbits	largest orbit
$5^2 : 3$	(3B,3B,3B)	8	1	(3A,3A,3A)	8	1
$5^2 : 6$	(2A,3B,6B)	4	1	(2A,3A,6B)	8	1
$5^2 : S_3$	(2A,3A,10D)	1	1	(2A,3A,10C)	1	1
	(2A,3A,10B)	1	1	(2A,3A,10A)	1	1
$5^2 : D(2*6)$	(2A,2B,2C,3A)	1	12			
$5^2 : D(2*4) : 2$	(2A,2C,2D,4C)	1	1	(2A,2C,2D,4A)	1	1
$5^2 : O+(2,5)$	(2C,4F,8A)	1	1	(2C,4E,8B)	1	1
$5^2 : ((Q_8 : 3)'2)$	(2B,3B,12B)	1	1	(2B,3B,12A)	1	1
	(2B,3A,12D)	1	1	(2B,3A,12C)	1	1
$5^2 : ((Q_8 : 3)'4)$	(4F,3A,4E)	1	1	(4D,3A,4G)	1	1
$\mathrm{ASL}(2,5) : 2$	(2B,3A,20D)	1	1	(2B,3A,20C)	1	1
	(2B,3A,20B)	1	1	(2B,3A,20A)	1	1
$5^3 : 4^2 : S_3$	(2B,3A,8B)	4	1	(2B,3A,8A)	4	1

Chapter C

GAP code

```
# Functions for Generating ramification types that fit the Riemann Hurwitz number
# Assign the group g first, then fix the list of conjugacy class representatives

CC:=[];
ct:=[];

CheckingTheGroup:=function(group)
 local i;

 CC:=List(ConjugacyClasses(group),Representative);
  ct:=CharacterTable(group);
end;

# The index of a permutation

PermIndex:=function(perm,group,degree)
 local n;

 return degree-Length(Orbits(Group(perm), [1..degree]));
end;

# The indices of conjugacy class representatives for a group

FindInd:=function(group,degree)
 local i,IndexSet,t,Ind;
 IndexSet:=[];
 for i in [2..Length(CC)] do
  t:=CC[i];
  Ind:=PermIndex(t,group,degree);
  Append(IndexSet,[rec(pos:=i,index:=Ind)]);
 od;
 return IndexSet;
end;

# Find all possible ramification types for a group with fixed degree and genus

Dim:=[];
RamiTypes:=function(group,degree,genus)
 local dim,n,RH,IndexSet,Indices,PossibleCombinations,i,a,b,
       Temp,j,k,c,RamificationTypes;

 dim:=Length(FactorsInt(degree));
 n:=Filtered(NormalSubgroups(g),x->Size(x)=degree)[1];
 RamificationTypes:=[];
 RH:=2*(degree+genus-1);
 IndexSet:=FindInd(group,degree);
 Indices:=List(IndexSet,x->x.index);
 PossibleCombinations:=RestrictedPartitions(RH,Elements(Indices));
 for i in [1..Length(PossibleCombinations)] do
  Temp:=[];
  a:=PossibleCombinations[i];
  b:=Elements(a);
  for j in [1..Length(b)] do
   k:=Length(Filtered(a,x->x=b[j]));
   c:=List(Filtered(IndexSet,x->x.index=b[j]),x->x.pos);
   Append(Temp,[UnorderedTuples(c,k)]);
  od;
  Temp:=Cartesian(Temp);
  for j in [1..Length(Temp)] do
   Append(RamificationTypes,[Concatenation(Temp[j])]);
  od;
 od;
 for i in [1..Length(RamificationTypes)] do
  a:=RamificationTypes[i];
  b:=List(a,x->Size(Centralizer(n,CC[x])));
  c:=[];
  for j in [1..Length(b)] do
```

```
    if b[j]<> 1 then
     Append(c,[dim-Length(FactorsInt(b[j]))]) ;
    else
    Append(c,[dim]);
    fi;
   od;
  if Sum(c)<2*dim then
    Unbind(RamificationTypes[i]);
   else
    Append(Dim,[c]);
   fi;
  od;
  return Elements(RamificationTypes);
end;

AddOneGenerator:=function(SubgroupList,NewGen,group)
 local NewSubgroupList,a,b,x,Cgx,gg,c,t,y,j,r,h,flag,hh;

 NewSubgroupList:=[];
 for a in [1..Length(CC)] do
  NewSubgroupList[a]:=[];
 od;

 for b in [1..Length(CC)] do
  x:=CC[b];
  Cgx:=Centralizer(group,x);
  for gg in SubgroupList[b] do
   for c in DoubleCosetRepsAndSizes(group,Centralizer(group,NewGen),
                                   Normalizer(Cgx,gg)) do
    t:=NewGen^(c[1]);
    y:=x*t;
    j:=1;
    while j < Length(CC)+1 do
     r:=RepresentativeAction(group,y,CC[j]);
     if not r=fail then
      break;
     else
      j:=j+1;
     fi;
    od;
    h:=Group(Concatenation(GeneratorsOfGroup(gg),[t]))^r;
    flag:=0;
    for hh in NewSubgroupList[j] do
     if IsConjugate(Centralizer(group,CC[j]),hh,h) then
      flag:=1;
      break;
     fi;
    od;
    if flag=0 then
     Add(NewSubgroupList[j],h);
    fi;
   od;
  od;
 od;
  return NewSubgroupList;
end;

AddOneGenerator1:=function(SubgroupList,NewGen,group,p)
 local NewSubgroupList,a,b,x,Cgx,gg,c,t,y,r,h,stop;

 stop:=0;
 for b in [1..Length(CC)] do
  x:=CC[b];
  Cgx:=Centralizer(group,x);
  for gg in SubgroupList[b] do
   for c in DoubleCosetRepsAndSizes(group,Centralizer(group,NewGen),
                                    Normalizer(Cgx,gg)) do
    t:=NewGen^(c[1]);
    y:=x*t;
    r:=RepresentativeAction(group,y,CC[p]);
    if r<>fail then
     h:=Group(Concatenation(GeneratorsOfGroup(gg),[t]))^r;
     if h=group then
      stop:=1;
      break;
     fi;
    fi;
   od;
  od;
 od;
  return stop;
end;
```

```
GeneratingType:=function(group,degree,genus)

 local RamificationTypes,GeneratingTypes,i,SubgroupList,
       ClassRepTuple,NewGen,m,j,k,n,a,p;

 GeneratingTypes:=[];
 RamificationTypes:=Reversed(RamiTypes(group,degree,genus));

 for i in [1..Length(RamificationTypes)] do
  Print("\r","Checking the ramification type ",i," with ",
         Length(RamificationTypes)-i," remaining ","\c");
   SubgroupList:=[];
   p:=0;
   ClassRepTuple:=List(RamificationTypes[i],x->CC[x]);
   for k in [1..Length(CC)] do
    SubgroupList[k]:=[];
    if IsConjugate(group,CC[k],Inverse(ClassRepTuple[Length(ClassRepTuple)])) then
     p:=k;
    fi;
   od;
   SubgroupList[RamificationTypes[i][1]]:=[Group(ClassRepTuple[1])];
   for j in [2..Length(ClassRepTuple)-1] do
    NewGen:=ClassRepTuple[j];
    if j=Length(ClassRepTuple)-1 then
     m:=AddOneGenerator1(SubgroupList,NewGen,group,p);
     if m=1 then
      Append(GeneratingTypes,[RamificationTypes[i]]);
     fi;
    else
     m:=AddOneGenerator(SubgroupList,NewGen,group);
    fi;
    SubgroupList:=m;
   od;
 od;
 n:=Concatenation("CasesFor",String(group),"genus",String(genus));
 if Length(GeneratingTypes)<>0 then
  AppendTo(n,"GeneratingTypes:=",GeneratingTypes,";\n");
  AppendTo(n,"group:=",group,";\n");
  AppendTo(n,"CC:=",CC,";\n");
 fi;
 Print("\n");
 return GeneratingTypes;
```

```
end;

# Find primitive affine groups containing p^t

FindAffine:=function(degree,p)
 local AffineGroupList,GroupList,i,g,syl,InvClass;

 AffineGroupList:=[];
 GroupList:=AllPrimitiveGroups(DegreeOperation,degree);
 for i in [1..Length(GroupList)] do
  g:=GroupList[i];
  if g<>AlternatingGroup(degree) and g<>SymmetricGroup(degree) then
   syl:=SylowSubgroup(g,p);
   if Size(Core(g,syl))=degree then
    Append(AffineGroupList,[g]);
   fi;
  fi;
 od;
 return AffineGroupList;
end;

Find3Tuple:=function(RamificationType,group)
 local GeneratingTuples,ClassRepTuple,g1,g2,g3,gg,Cgx,c,t,y,r,h,i,TT,flag;

 ClassRepTuple:=List(RamificationType,x->CC[x]);
 GeneratingTuples:=[];

 g1:=ClassRepTuple[1];
 g2:=ClassRepTuple[2];
 g3:=ClassRepTuple[3];

 gg:=Group(g1);
 Cgx:=Centralizer(group,g1);
 for c in DoubleCosetRepsAndSizes(group,Centralizer(group,g2),
                                    Normalizer(Cgx,gg)) do
  t:=g2^(c[1]);
  y:=g1*t;
  r:=RepresentativeAction(group,y,Inverse(g3));
  if r<>fail then
   h:=Group(Concatenation(GeneratorsOfGroup(gg),[t]));
   if h=group then
    TT:=[g1,t,Inverse(y)];
    flag:=0;
    for i in [1..Length(GeneratingTuples)] do
     if RepresentativeAction(group,TT,GeneratingTuples[i],OnTuples)<>fail then
      flag:=1;
      break;
     fi;
    od;
    if flag=0 then
     Append(GeneratingTuples,[TT]);
    fi;
   fi;
  fi;
 od;
 return GeneratingTuples;
end;

# Load BRAID package first

 ProjectName:=[];
 Read("assemble.g");

# Find the number of orbits in one directory
# m represents the case number
# c is the conjugacy class number

NoOfOrbits:=function(m,c)
 local k,n;

 k:=1;
 while k<>0 do
 n:=Concatenation(String(m),"OrbitsForClass",String(c),"/",String(k));
  if IsExistingFile(n)=true then
   k:=k+1;
  else
   return k-1;
  fi;
 od;
end;

# Restore orbits from the directories.
# m represents the case number
# c is the conjugacy class number
# z is the index of a half orbit

RestoreOrbits:=function(m,c,z)
 local n;

 n:=Concatenation(String(m),"OrbitsForClass",String(c),"/",String(z));
 if IsExistingFile(n)=true then
  Read(n);
 fi;
end;

# Assign the group g first, then fix the list of conjugacy class reps
# Fix the length of the tuple and the length of the left half tuple
# Initialize the list of vertices and matching pairs
# Prepare the list of pure braid elements for connecting process

 ct:=[];
 CC:=[];
 AllPairs:=[];
 AllVertices:=[];
 g:=Group(());
 BraidElements:=[];
 AbsGens1:=[];
 MatchingPairs:=[];

Initialize:=function(group,half,LengthOfTuple)
 local i;

 ct:=CharacterTable(group);
 AbsGens1:=[];
 BraidElements:=[];
 g:=group;
 AllPairs:=[];
 AllVertices:=[];
 CC:=List(ConjugacyClasses(g),Representative);
 NumberOOfGenerators:=LengthOfTuple;
 OurFreeGroup:=FreeGroup(NumberOOfGenerators);
 AbsGens:=GeneratorsOfGroup(OurFreeGroup);
 for i in [1..NumberOOfGenerators] do
  AppendTo("_tmptmp_","f",String(i),":=AbsGens[",String(i),"];\n");
 od;
 Read("_tmptmp_");
 Exec("rm _tmptmp_");

 BraidGroupGenerators1();
 for i in [1..half] do
  Append(BraidElements,[PQ[i][half+1]]);
 od;
 AbsGens1:=ShallowCopy(AbsGens);
end;

# Function to generate orbits of type [x,l]
# m represents the case
# where l is a fixed type and x is every non-identity conjugacy class
# For example: l:=[2,2,2,2], then partition:=[4]

GenerateOrbits:=function(m,l,partition,g)
 local i,tu,x,job,c,T,Par,RamificationType;

 for i in [2..Length(CC)] do
  T:=List(l,x->CC[x]);
  x:=CC[i];
  tu:=Concatenation([x],T);
  Par:=Concatenation([1],partition);
  RamificationType:=Concatenation([i],l);
  c:=ClassStructureCharTable(ct,RamificationType);
  job:=Concatenation(String(m),"OrbitsForClass",String(i));
  AllBraidOrbits(job,g,tu,Par,c);
 od;
end;

# LookUpTuple for finding the normalizer of orbits

NewLookUpTuple1:=function(T)
 local f,a,ConjEle;
 f:=LookUpFingerprint(Fingerprint(T));
 a:=FingerprintTable[f].firstTuple;
 while a<>0 do
  if TupleTable[a].tuple=T then
   return a;
  else
   a:=TupleTable[a].next;
  fi;
 od;
 return a;
end;

# LookUpTuple without adding it to the table
# This is needed for finding position of a tuple

NewLookUpTuple:=function(T,H)
 local f,a;
 f:=LookUpFingerprint(Fingerprint(T));
 a:=FingerprintTable[f].firstTuple;
 while a<>0 do
  if RepresentativeAction(H,TupleTable[a].tuple,T,OnTuples)<>fail then
   return a;
  else
   a:=TupleTable[a].next;
  fi;
 od;
 return a;
end;

# To match the tuples that generate the whole group g,
# where the left half and the right half have the same type
# M is the matching list

Matching:=function(m,M,l)
 local i,j,T1,T2,a,TT,MatchingPairs,n,CosetReps,ClassRep,DC,Cgx;

 Print("\n","Starting to match...","\n");
 ClassRep:=CC[l];
 MatchingPairs:=[];
 Cgx:=Centralizer(g,ClassRep);
 for i in [1..Length(M)] do
  T1:=M[i].tuple;
  for j in [i..Length(M)] do
   T2:=M[j].tuple;
   DC:=DoubleCosets(Cgx,M[i].Norm,M[j].Norm);
   CosetReps:=List(DC,Representative);
   for a in [1..Length(CosetReps)] do
    TT:=Concatenation(Reversed(List(T1,x->x^CosetReps[a])),T2);
    if Group(TT)=g then
     Append(MatchingPairs,[[l,i,DC[a],j]]);
    fi;
   od;
   Print("\r"," Matching the orbit ",i,"\c"," to the orbit ",j,"\c");
  od;
 od;
 n:=Concatenation(String(m),"MatchingPairs",String(l));
 AppendTo(n,"ClassRep:=",ClassRep,";\n");
 AppendTo(n,"MatchingPairs:=",MatchingPairs,";\n");
 Print("\n");
end;

# To match tuples that generate the whole group g.
# where the matching list of the left half is represented by M1, and
# the right half is represented by M2

Matching2:=function(m,M1,M2,l)
 local i,j,p,T1,T2,a,TT,MatchingPairs,n,CosetReps,ClassRep,DC,Cgx;

 Print("\n","Starting to match...","\n");
 ClassRep:=CC[l];
 MatchingPairs:=[];
 Cgx:=Centralizer(g,ClassRep);

 for i in [1..Length(M1)] do
  T1:=M1[i].tuple;
  for j in [1..Length(M2)] do
   T2:=M2[j].tuple;
   DC:=DoubleCosets(Cgx,M1[i].Norm,M2[j].Norm);
   CosetReps:=List(DC,Representative);
   for a in [1..Length(CosetReps)] do
    TT:=Concatenation(List(T1,x->x^CosetReps[a]),T2);
    if Group(TT)=g then
     Append(MatchingPairs,[[l,i,DC[a],j]]);
    fi;
   od;
   Print("\r"," Matching the orbit ",i,"\c"," to the orbit ",j,"\c");
  od;
 od;
 n:=Concatenation(String(m),"MatchingPairs",String(l));
 AppendTo(n,"ClassRep:=",ClassRep,";\n");
 AppendTo(n,"MatchingPairs:=",MatchingPairs,";\n");
 Print("\n");
end;
```

```
# The function to form the matching list
# It's a list of orbit indices with the normalizer

MatchingList:=function(m,l)
 local r1,i,j,t,Ht,T,Cgx,CHt,cc,cc2,Nt,NumberOfOrbits,MList,ClassRep;

 NumberOfOrbits:=NoOfOrbits(m,l);
 ClassRep:=CC[l];
 Cgx:=Centralizer(g,ClassRep);
 MList:=[];

 for i in [1..NumberOfOrbits] do
  RestoreOrbits(m,l,i);
  t:=TupleTable[1].tuple;
  Ht:=Group(t);
  CHt:=Centralizer(Cgx,Ht);
  cc:=List(RightCosets(Cgx,Group(Concatenation(GeneratorsOfGroup(CHt),
           [ClassRep]))),x->Representative(x));
  cc2:=[];
  if Length(cc)<>1 then
   for j in [1..Length(cc)] do
    T:=List(t,x->x^cc[j]);
    if NewLookUpTuple1(T)<>0 then
     Append(cc2,[cc[j]]);
    fi;
   od;
   Nt:=Subgroup(Cgx,Concatenation(GeneratorsOfGroup(CHt),cc2,[ClassRep]));
  else
   Nt:=Cgx;
  fi;
  Append(MList,[rec(tuple:=t,Norm:=Nt)]);
 od;
 return MList;
end;

# Load all the matching pairs into the RAM

LoadingPairs:=function(m)
 local i,j;
 AllPairs:=[];
 AllVertices:=[];
 for i in [2..Length(CC)] do
  Read(Concatenation(String(m),"MatchingPairs",String(i)));
  Append(AllPairs,[MatchingPairs]);
  for j in [1..Length(MatchingPairs)] do
   Append(AllVertices,[[i,j]]);
  od;
 od;
end;

# The function to find all matching pairs
# assign 'case' to be any string representing the case
# m represents the left orbit, n represents the right orbit
# if m=n, then use "Matching", otherwise use "Matching2"

AllMatchingPairs:=function(case,m,n)
 local i,M,M1,M2;

 if m=n then
  for i in [2..Length(CC)] do
   Print("\n","Matching the generating pairs in the class ", i,"\n");
   M:=MatchingList(m,i);
   Matching(case,M,i);
  od;
 else
  for i in [2..Length(CC)] do
   Print("\n","Matching the generating pairs in the class ", i,"\n");
   M1:=MatchingList(m,i);
   M2:=MatchingList(n,i);
   Matching2(case,M1,M2,i);
  od;
 fi;
 LoadingPairs(case);
 Display("The total number of vertices is ");
 return Length(AllVertices);
end;

# Find the position of a tuple where the left and right have the same
# type so when RestoreOrbits, it can look up both half tuples
# simultaneously
```

```
FindPos:=function(m,TT,half)
 local x,i,T,NumberOfOrbits,j,r,t,t1,t2,p,a,Left,Right,r1,r2,b,c,SubList;

 x:=Product(TT{[1..half]});
 for i in [1..Length(CC)] do
  if RepresentativeAction(g,CC[i],x)<>fail then
   p:=i;
   break;
  fi;
 od;
 if p=1 then
  return [1,1];
 else
  r:=RepresentativeAction(g,x,CC[p]);
  T:=List(TT,x->x^r);
  t1:=List(Reversed(T{[1..half]}),x->Inverse(x));
  t2:=T{[half+1..Length(T)]};
  NumberOfOrbits:=NoOfOrbits(m,p);

  a:=0;
  for j in [1..NumberOfOrbits] do
   RestoreOrbits(m,p,j);
   b:=NewLookUpTuple(Concatenation([CC[p]],t1),g);
   if b<>0 then
    t:=TupleTable[b].tuple;
    Left:=j;
    a:=a+1;
    r1:=RepresentativeAction(Centralizer(g,CC[p]),t{[2..Length(t)]},t1,OnTuples);
   fi;
   c:=NewLookUpTuple(Concatenation([CC[p]],t2),g);
   if c<>0 then
    t:=TupleTable[c].tuple;
    Right:=j;
    a:=a+1;
    r2:=RepresentativeAction(Centralizer(g,CC[p]),t{[2..Length(t)]},t2,OnTuples);
   fi;
   if a=2 then
    break;
   fi;
  od;
  if Left>Right then
   SubList:=Filtered(AllPairs[p-1],x->x[2]=Right and x[4]=Left);
   for j in [1..Length(SubList)] do
    if r2*r1^-1 in SubList[j][3] then
     return [p,Position(AllPairs[p-1],SubList[j])];
     break;
    fi;
   od;
  else
   SubList:=Filtered(AllPairs[p-1],x->x[2]=Left and x[4]=Right);
   for j in [1..Length(SubList)] do
    if r1*r2^-1 in SubList[j][3] then
     return [p,Position(AllPairs[p-1],SubList[j])];
     break;
    fi;
   od;
  fi;
 fi;
end;

# Find the position of a tuple if left<>half
# so first it will find left half tuple in "m", then right half in "n"
# "half" represents the length of the left half tuple
# TT is the given tuple

FindPos2:=function(m,n,TT,half)
 local i,T,NumberOfOrbits,j,r,t,t1,t2,p,a,Left,Right,r1,r2,b,c,SubList,x;

 x:=Product(TT{[1..half]});
 for i in [1..Length(CC)] do
  if RepresentativeAction(g,CC[i],x)<>fail then
   p:=i;
   break;
  fi;
 od;
 if p=1 then
  return [1,1];
 else
  r:=RepresentativeAction(g,x,CC[p]);
  T:=List(TT,x->x^r);
  t1:=List(Reversed(T{[1..half]}),x->Inverse(x));
  t2:=T{[half+1..Length(T)]};
  NumberOfOrbits:=NoOfOrbits(m,p);
  for j in [1..NumberOfOrbits] do
```

```
   RestoreOrbits(m,p,j);
   b:=NewLookUpTuple(Concatenation([CC[p]],t1),g);
   if b<>0 then
    t:=TupleTable[b].tuple;
    Left:=j;
    r1:=RepresentativeAction(Centralizer(g,CC[p]),t{[2..Length(t)]},t1,OnTuples);
    break;
   fi;
  od;
  NumberOfOrbits:=NoOfOrbits(n,p);
  for j in [1..NumberOfOrbits] do
   RestoreOrbits(n,p,j);
   c:=NewLookUpTuple(Concatenation([CC[p]],t2),g);
   if c<>0 then
    t:=TupleTable[c].tuple;
    Right:=j;
    r2:=RepresentativeAction(Centralizer(g,CC[p]),t{[2..Length(t)]},t2,OnTuples);
    break;
   fi;
  od;
  SubList:=Filtered(AllPairs[p-1],x->x[2]=Left and x[4]=Right);
  for j in [1..Length(SubList)] do
   if r1*r2^-1 in SubList[j][3] then
    return [p,Position(AllPairs[p-1],SubList[j])];
    break;
   fi;
  od;
 fi;
end;


# Pick a random tuple from a matching pair.

PickTuple:=function(m,n,pair)
 local t1,t2,T;

 RestoreOrbits(m,pair[1],pair[2]);
 t1:=Random(TupleTable).tuple;
 t1:=Reversed(List(t1{[2..Length(t1)]},x->x^Representative(pair[3])));
 RestoreOrbits(n,pair[1],pair[4]);
 t2:=Random(TupleTable).tuple;
 T:=Concatenation(List(t1,x->Inverse(x)),t2{[2..Length(t2)]});
 return T;
end;

# Find the neighbors of a selected vertex
# "m" represents the left orbits, "n" represents the right orbits
# "half" represents the length of the left half tuple
# "vertex" is a coordinate for a node, like "[x,y]", x represents
# the conjugacy class and y represents which node it is in that class.

FindNeighbors:=function(m,n,half,vertex)
 local Nei,pair,i,j,T0,T,w,group;

 Nei:=[vertex];
 pair:=AllPairs[vertex[1]-1][vertex[2]];
 T0:=PickTuple(m,n,pair);
 for i in [1..Length(BraidElements)] do
  T:=[];
  for j in [1..Length(T0)] do
   T[j]:=MappedWord(BraidElements[i][j],AbsGens1,T0);
  od;
  if Product(T{[1..half]})<>() then
   if m=n then
    w:=FindPos(m,T,half);
   else
    w:=FindPos2(m,n,T,half);
   fi;
   if Position(Nei,w)=fail then
    Append(Nei,[w]);
   fi;
  fi;
 od;
 if Length(Nei)=1 then
  T0:=Reversed(List(T0,x->Inverse(x)));
  for i in [1..Length(BraidElements)] do
   T:=[];
   for j in [1..Length(T0)] do
    T[j]:=MappedWord(BraidElements[i][j],AbsGens1,T0);
   od;
   T:=Reversed(List(T,x->Inverse(x)));
   if Product(T{[1..half]})<>() then
    if m=n then
     w:=FindPos(m,T,half);
    else
     w:=FindPos2(m,n,T,half);
    fi;
    if Position(Nei,w)=fail then
     Append(Nei,[w]);
    fi;
   fi;
  od;
 fi;
 return Nei;
end;

# Find all neighbors
# "m" represents the left orbits, "n" represents the right orbits
# we can separate all the vertices into different intervals [a,b]
# for parallel computation, or just assign a=1, b=total number

AllNeighbors:=function(case,m,n,a,b)
 local Neighbors,i,j,v,pair,w,T,N,half,first;

 LoadingPairs(case);
 Neighbors:=[];
 for i in [1..b-a+1] do
  Neighbors[i]:=[];
 od;
 first:=1;
 while first<Length(AllPairs)+1 do
  if Length(AllPairs[first])<>0 then
   break;
  else first:=first+1;
  fi;
 od;
 pair:=AllPairs[first][1];
 RestoreOrbits(m,pair[1],pair[4]);
 half:=Length(TupleTable[1].tuple)-1;
 for i in [a..b] do
  Print("\r","The number of vertices checked is ",i-a,
         " ,(  ",b-i+1," remaining )\c");
  v:=AllVertices[i];
  N:=FindNeighbors(m,n,half,v);
  Append(Neighbors[i-a+1],N);
 od;
 Print("\n","Vertices checked from ", a," to ",b,"\n");
 AppendTo(Concatenation(String(case),"Neighbors",String(a),"to",String(b)),
          "Neighbors:=",Neighbors,";\n");
end;



# Connect vertices with neighbors
# "Neighbors" is the result from generating neighbors for each vertex
# each vertex has a coordinate, connect two vertices with their neighbors
# if they share one neighbor

ConnectVertices:=function(case,Neighbors)
 local i,v,j,p,ConnectedGraph,k,AllComponents,a,L;

 AllComponents:=[];
 ConnectedGraph:=ShallowCopy(Neighbors);
 L:=Sum(List(Neighbors,x->Length(x)));
 for a in [1..Length(ConnectedGraph)] do
  if IsBound(ConnectedGraph[a]) then
   i:=1;
   while i< L do
    if IsBound(ConnectedGraph[a][i]) then
     v:=ConnectedGraph[a][i];
     for j in [a+1..Length(ConnectedGraph)] do
      if IsBound(ConnectedGraph[j]) then
       if Position(ConnectedGraph[j],v)<>fail then
        for k in [1..Length(ConnectedGraph[j])] do
         if Position(ConnectedGraph[a],ConnectedGraph[j][k])=fail then
          Append(ConnectedGraph[a],[ConnectedGraph[j][k]]);
         fi;
        od;
        Unbind(ConnectedGraph[j]);
       fi;
      fi;
     od;
     i:=i+1;
    else
     break;
    fi;
   od;
   Append(AllComponents,[ConnectedGraph[a]]);
  fi;

 od;
 Print("\n","The number of components is ",Length(AllComponents),"\n");
 k:=0;
 for i in [1..Length(AllComponents)] do
  if Length(AllComponents[i])=1 then
   k:=k+1;
  fi;
 od;
 Print("\n","The number of loops is ",k,"\n");
 AppendTo(Concatenation(String(case),"AllComponents"),"AllComponents:=",
          AllComponents,";\n");
end;
```

Bibliography

[1] M. Aschbacher, *On conjectures of Guralnick and Thompson*, J. Algebra, **135** (1990), no. 2, 277 – 343.

[2] M. Aschbacher, R. Guralnick and K. Magaard, *Rank 3 permutation characters and primitive groups of low genus*, In preparation

[3] M. Badger, *Braid Orbits On Primitive Permutation Groups Of Degree 8 And 16*, Mphil Thesis, University of Birmingham, 2008

[4] M. Fried, Alternating groups and the moduli space lifting invariants, Israel J. Math. **179** (2010), 57 – 125.

[5] M. Fried and H. Völklein, The inverse Galois problem and rational points on moduli spaces, Math. Ann. **290** (1991), 771 – 800.

[6] D. Frohardt, R. Guralnick and K. Magaard, *Genus 0 actions of group of Lie rank 1, Arithmetic fundamental groups and noncommutative algebra* (Berkeley, CA, 1999), 449 – 483, Proc. Sympos. Pure Math., **70**, Amer. Math. Soc., Providence, RI, 2002.

[7] D. Frohardt, R. Guralnick and K. Magaard *Genus 2 point actions of classical groups*, In preparation

[8] D. Frohardt and K. Magaard, *Composition Factors of Monodromy Groups*, Annals of Mathematics, **154** (2001), 327–345.

[9] The GAP Group, *GAP reference manual* October, 2007.

[10] R. Guralnick and K. Magaard *On the Minimal Degree of a Primitive Permutation Group*, J. Algebra, **207**, (1998) 127 – 145.

[11] R. Guralnick and J. Thompson *Finite groups of genus zero*, J. Algebra **131** (1990), no. 1, 303 – 341.

[12] K. Magaard, S. Shpectorov and H. Völklein. *BRAID-a GAP package*, 2001.

[13] K. Magaard, S. Shpectorov and H. Völklein. *A GAP Package for Braid Orbit Computation and Applications*. Experiment. Math. **12** (2003), no. 4, 385 –393.

[14] W. S. Massey. *Algebraic Topology: An Introduction*, Springer-Verlag, 1967.

[15] M. Neubauer, On monodromy groups of fixed genus, J. Algebra 153 (1992), no. 1

[16] T. Shih, A note on groups of genus zero, Comm. Algebra **19** (1991), no. 10, 2813 – 2826.

[17] R. Staszewski, H. Völklein and G. Wiesend, *Counting generating systems of a finite group from given conjugacy classes.* Computational aspects of algebraic curves, 256–263, Lecture Notes Ser. Comput., 13, World Sci. Publ., Hackensack, NJ, 2005.

[18] H. Völklein. *Groups as Galois Groups-An Introduction*, Cambr. Studies in Adv. Math.,53.Cambridge, UK: Cambridge Univ. Press, 1996.