THE COMPUTER PROGRAM NETMAP

1. Introduction

The program NETmap makes computations for nearly Euclidean Thurston maps, NET maps for short. These special Thurston maps were introduced in [1] and further investigated in [2], [3], [4], [5] and [7]. They are those Thurston maps $f\colon S^2 \to S^2$ with exactly four postcritical points such that the local degree of f at each of its critical points is 2.

NETmap is a command line program written in C. It begins with a plain text input file prepared by the user. This input file contains information which describes a virtual NET map presentation, defined in Section 6 of [3]. The word "virtual" indicates that such a presentation does not quite describe a NET map. It describes a NET map up to a translation term, which is an ordered pair of integers. Since the equivalence class of the map depends only on the values of these integers modulo 2, there are essentially four choices for this translation term. The program does not require the translation term because most of its computations are independent of the translation term. Computations which require the translation term are usually made four times, once for each translation term. So with each run, the program makes computations for up to four closely related Thurston equivalence classes of NET maps.

In addition to the input file, NETmap also asks for some input from the user at the keyboard. It then writes output to a number of output files. All of this is carefully described in the following two sections, one for input and one for output.

2. Input

During the keyboard input phase of the program, the program asks for information. Every response concludes by pressing the enter key.

The input file. Most of the input is in a plain text file, which should be prepared in the format of the sample file sample.input. This file contains the data corresponding to the virtual presentation diagram in Figure 2. The name of the file must have the form filename.input. The program begins by asking for filename. This may be an absolute path name or a path name relative to NETmap's location. The program reads the data in filename.input. It writes output to files in the same directory with names based on filename. We continue with an explanation of the input.

The vectors λ_1 and λ_2. NETmap makes computations for a NET map expressed as $f = h \circ g$, where $g\colon S^2 \to S^2$ is a Euclidean NET map and $h\colon S^2 \to S^2$ is a push homeomorphism taking the postcritical set P_g of g into $g^{-1}(P_g)$. The map g is induced by an affine map $\Phi\colon \mathbb{R}^2 \to \mathbb{R}^2$. We assume that $\Phi(x) = Ax + b$, where x is a column vector in $\mathbb{R}^2$ and A is a 2×2 matrix of integers with $\det(A) = \deg(g) = \deg(f) \geq 2$. Let λ_1 and λ_2 be the columns of A. Then b is a integral linear combination of λ_1 and λ_2. The vectors λ_1 and λ_2 form a basis of a proper sublattice Λ_1 of the standard lattice $\mathbb{Z}^2$. Let Γ_1 be the group of Euclidean isometries of the form $x \mapsto 2\lambda \pm x$ for some $\lambda \in \Lambda_1$. Then the map $\Phi\colon \mathbb{R}^2 \to \mathbb{R}^2$ induces the map

Date: March 22, 2019.

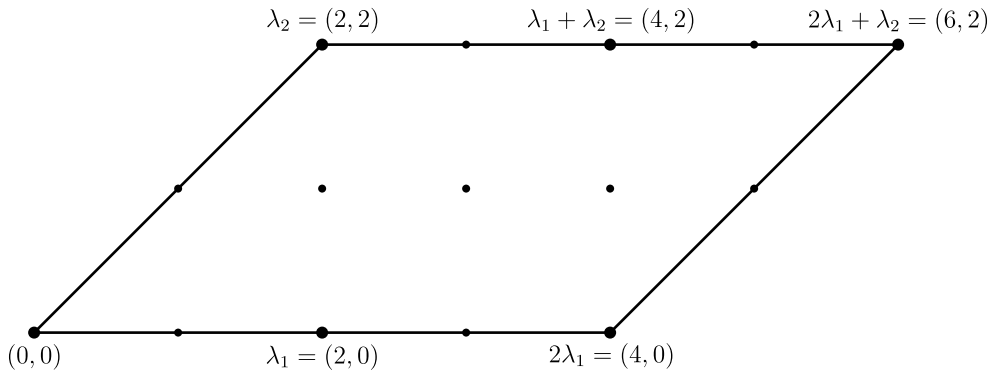


FIGURE 1. The fundamental domain F_1 with corners 0, $2\lambda_1$, λ_2 and $2\lambda_1 + \lambda_2$

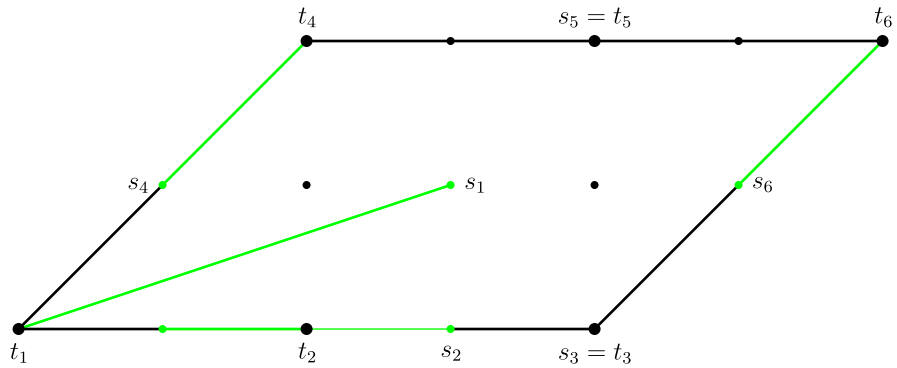


FIGURE 2. The green line segments and the points $s_1, t_1, \ldots, s_6, t_6$

$g\colon \mathbb{R}^2/\Gamma_1 \to \mathbb{R}^2/\Gamma_1$ in the straightforward way. This determines g in terms of λ_1, λ_2 and b (up to conjugation by a homeomorphism depending on the identification of $\mathbb{R}^2/\Gamma_1$ with S^2). This map g is a Euclidean NET map. The program requires the coordinates of λ_1 and λ_2.

For example, the first line of the input file sample.input is "lambda1". The next line has the form "integer space integer". Since the first integer is 2 and the second integer is 0, this means that $\lambda_1 = (2,0)$, using row notation now. Similarly, the next two lines of sample.input mean that $\lambda_2 = (2,2)$.

We discuss the translation term b in this paragraph. The program requires the vectors λ_1 and λ_2. This determines the matrix A. Thus g is determined except for the translation term b. The program does not require b. For one thing, most of the computations are independent of b. For those computations which depend on b, we note that Γ_1 contains all translations of the form $x \mapsto x + \lambda$, where $\lambda \in 2\Lambda_1$. This implies that changing b by adding an element of $2\Lambda_1$ does not change f, and so we may assume that $b \in \{0, \lambda_1, \lambda_2, \lambda_1 + \lambda_2\}$. Computations which depend on b are usually made four times, once for each of the values 0, λ_1, λ_2 and $\lambda_1 + \lambda_2$. The reason for the word "usually" is that there are atypical cases in degrees 2 and 4 for which the map f resulting from some choice(s) of b has fewer than four postcritical points, and so is not a NET map.

The six lattice points paired with $0, \lambda_1, 2\lambda_1, \lambda_2, \lambda_1 + \lambda_2, 2\lambda_1 + \lambda_2$. The program requires input data for h. For this we need a fundamental domain F_1 for Γ_1. The program always takes F_1 to be the closed parallelogram with corners 0, $2\lambda_1$, λ_2 and $2\lambda_1 + \lambda_2$. The rotation of order 2 about λ_1 identifies the closed line segments $[0, \lambda_1]$ and $[\lambda_1, 2\lambda_1]$. The rotation of

order 2 about $\lambda_1 + \lambda_2$ identifies the closed line segments $[\lambda_2, \lambda_1 + \lambda_2]$ and $[\lambda_1 + \lambda_2, 2\lambda_1 + \lambda_2]$. The translation $x \mapsto x + 2\lambda_1$ identifies the closed line segments $[0, \lambda_2]$ and $[2\lambda_1, 2\lambda_1 + \lambda_2]$. Identifying the points on the boundary of F_1 using these boundary identifications obtains our 2-sphere.

The map h is a push map. It is defined as follows. There are four disjoint arcs β_1, β_2, β_3, β_4 in S^2, each with initial endpoint in the postcritical set P_g of g and terminal endpoint in the postcritical set P_f of f. We choose four disjoint closed topological disks D_1, D_2, D_3, D_4 in $\mathbb{R}^2/\Gamma_1$ with $\beta_i \subseteq \text{int}(D_i)$ for $i \in \{1, 2, 3, 4\}$. The map $h\colon \mathbb{R}^2/\Gamma_1 \to \mathbb{R}^2/\Gamma_1$ is a homeomorphism such that $h(x) = x$ for every $x \notin D_1 \cup D_2 \cup D_3 \cup D_4$ and h maps the initial endpoint of β_i to its terminal endpoint for every $i \in \{1, \ldots, 4\}$. Theorem 6.1 of [3] shows that every NET map has such a presentation for which the inverse image of $\beta_1 \cup \beta_2 \cup \beta_3 \cup \beta_4$ in the fundamental domain F_1 is a union of line segments. `NETmap` assumes that the inverse image of $\beta_1 \cup \beta_2 \cup \beta_3 \cup \beta_4$ in the fundamental domain F_1 is a union of line segments. (Some line segments might be trivial, consisting of just a point each.) We call these line segments the green line segments. We color the nontrivial ones green.

So here is the information that `NETmap` needs for h. See Figures 1 and 2. Let $t_1, \ldots, t_6$ be the elements of $F_1 \cap \Lambda_1$ in the following order.

$$t_1 = 0 \quad t_2 = \lambda_1 \quad t_3 = 2\lambda_1 \quad t_4 = \lambda_2 \quad t_5 = \lambda_1 + \lambda_2 \quad t_6 = 2\lambda_1 + \lambda_2$$

Each of $t_1, \ldots, t_6$ is contained in a green line segment, a connected component of the inverse image in F_1 of $\beta_1 \cup \beta_2 \cup \beta_3 \cup \beta_4$. Let s_i be an endpoint of the green line segment which contains t_i for $i \in \{1, \ldots, 6\}$. We choose s_i so that $s_i \neq t_i$ unless the green line segment which contains t_i is trivial, consisting of just a point. `NETmap` requires the points $s_1, \ldots, s_6$.

We return to the file sample.input. One line contains "the_lattice_point_paired_with_0". The line following this has the form "integer space integer". The 3 and the 1 here mean that $s_1 = (3, 1)$, which is correct because $(3, 1)$ is the other endpoint of the green line segment with endpoint t_1. Similarly, $s_2 = (3, 0)$ because $(3, 0)$ is one endpoint of the green line segment which contains t_2. Equivalently, we may take $s_2 = (1, 0)$. We must take $s_3 = t_3 = (4, 0)$ because the green line segment containing t_3 is trivial. We must take $s_4 = (1, 1)$ because $(1, 1)$ is the other endpoint of the green line segment with one endpoint t_4. We must take $s_5 = t_5 = (4, 2)$. We must take $s_6 = (5, 1)$.

This is all of the information in the input file. So the information in the input file consists of λ_1, λ_2 and the six lattice points $s_1, \ldots, s_6$.

More keyboard input. A bit more keyboard input is usually needed. The program will probably ask for a positive integer which bounds the absolute values of the numerators and denominators of slopes to be considered. For simple computations, 25 is a good choice, resulting in a quick and effective computation. The program also asks for the smallest and largest x-coordinates to be used for the half-space postscript output file. These are floating point numbers. It is difficult to guess appropriate bounds on x. In general, a reasonable strategy is to run the program using the values -10 and 10, examine the file to refine these bounds and run the program again. The bounds on numerators and denominators and the bounds on x are made at the keyboard because making changes is quicker at the keyboard than by file.

This completes the input phase of the program. The program has the information which it needs to make computations.

3. Output

The program first performs several checks on the input to verify that it is valid. For example, it checks that the images in the quotient space $\mathbb{R}^2/\Gamma_1$ of the green line segments are disjoint. If it finds an error in the input, then it aborts with an error message which describes the difficulty. Most of these checks occur immediately after the user inputs the file name. This is the main reason for the statement above that the program will "probably" ask for a positive integer which bounds the absolute values of the numerators and denominators of slopes to be considered.

Output is contained in the terminal, several postscript files and three text files. These text files are filename_Main.output, filename_MOD.output and filename_Table.output. Except for error messages, which often appear only in the terminal, the content of the terminal output is the same as filename_Main.output.

All conclusions reached by the program are based on integer computations. That is, no conclusion involves a floating point computation. So roundoff error cannot affect any conclusion. However some output does appear in decimal form. This is simply because in these cases the decimal form is generally more informative than giving the number in fraction form. These decimal expressions are computed by converting a rational number to decimal form solely for output.

To explain the output, we now fix some definitions and notation regarding slopes. We begin with a NET map $f\colon S^2 \to S^2$ with postcritical set P_f. The input file provides a presentation for f, except for omitting the translation term.

This presentation determines a way [1, Section 4] to assign slopes, elements of $\overline{\mathbb{Q}} = \mathbb{Q} \cup \{\infty\}$, in a bijective manner to homotopy classes of simple closed curves in $S^2 \setminus P_f$ which are neither inessential nor peripheral. The map f determines a pullback map on these homotopy classes and therefore a slope function $\mu_f\colon \overline{\mathbb{Q}} \to \overline{\mathbb{Q}} \cup \{\odot\}$. The symbol $\odot$, called the nonslope, corresponds to homotopy classes of simple closed curves which are either inessential or peripheral. Let $s \in \overline{\mathbb{Q}}$. Let γ be a simple closed curve in $S^2 \setminus P_f$ with slope s. If every connected component of $f^{-1}(\gamma)$ is either inessential or peripheral, then $\mu_f(s) = \odot$. Otherwise, the connected components of $f^{-1}(\gamma)$ which are neither inessential nor peripheral have the same slope, $\mu_f(s)$. Let $c(s)$ be the number of these connected components. Theorem 4.1 of [1] implies that f maps each of these connected components to γ with the same degree $d(s)$. The fraction $c(s)/d(s)$ is the Thurston multiplier of s.

The intersection number of two simple closed curves in $S^2 \setminus P_f$ is gotten by putting them in general position and counting intersection points. If the curves have slopes $\frac{p}{q}$ and $\frac{r}{s}$, then this intersection number is 2 times the absolute value of the determinant of the matrix whose columns are $\left[\begin{smallmatrix} p \\ q \end{smallmatrix}\right]$ and $\left[\begin{smallmatrix} r \\ s \end{smallmatrix}\right]$. We call this determinant absolute value the intersection pairing of the two slopes.

We continue with a subsection for each output file.

3.1 The file filename_Main.output

Order of the postcritical set. NETmap always computes the order of the postcritical set for every translation term $b \in \{0, \lambda_1, \lambda_2, \lambda_1 + \lambda_2\}$. There are some cases in degrees 2 and 4 for which the resulting Thurston map has fewer than four postcritical points and so is not a NET map. If there are fewer than four postcritical points for every choice of b, then the program aborts with an explanatory message, since the remaining computations are not meaningful.

Primitivity. We say that f is imprimitive if there exist NET maps f_1 and f_2 both with postcritical sets P_f such that f_1 is Euclidean and $f = f_1 \circ f_2$. If there do not exist such maps f_1 and f_2, then we say that f is primitive. NETmap next states whether or not f is primitive.

Degree of f. NETmap gives the degree of f.

Pure modular group Hurwitz classes for translations. NET maps f_1 and f_2 with postcritical sets P_{f_1} and P_{f_2} are Hurwitz equivalent for the modular group if there exist orientation-preserving homeomorphisms $\varphi, \psi \colon (S^2, P_{f_1}) \to (S^2, P_{f_2})$ such that $\varphi \circ f_1 = f_2 \circ \psi$. If in addition $\varphi|_{P_{f_1}} = \psi|_{P_{f_2}}$, then f_1 and f_2 are Hurwitz equivalent for the pure modular group. The resulting equivalence classes are called modular group Hurwitz classes and pure modular group Hurwitz classes, respectively. The NET maps which arise from the four possible translation terms lie in the same modular group Hurwitz class, but not necessarily the same pure modular group Hurwitz class. NETmap next gives a partition of $\{0, \lambda_1, \lambda_2, \lambda_1 + \lambda_2\}$ according to pure modular group Hurwitz equivalence.

Finiteness of the number of Thurston equivalence classes. NETmap next states whether or not the pure modular group Hurwitz class of f contains infinitely many Thurston equivalence classes. If there are only finitely many, then NETmap states whether or not the modular group Hurwitz class of f contains infinitely many Thurston equivalence classes.

Number of pure modular group Hurwitz classes in the modular group Hurwitz class. NETmap next gives the number of pure modular group Hurwitz classes in the modular group Hurwitz class of f.

Multipliers. NETmap next lists all slope multipliers. These multipliers are expressed as $c(s)/d(s)$ without reducing the fraction, and so the numerator and denominator might not be relatively prime.

Half-space computations. NETmap essentially always reports on the rationality of f. If it verifies that f is Thurston equivalent to a rational map, then it says so with details. If it verifies that f is not Thurston equivalent to a rational map, then it says so with details. Otherwise it reports that its efforts are inconclusive with details.

The main tool used to verify that f is Thurston equivalent to a rational map is the half-space theorem, Theorem 6.7 in [1]. The half-space computation produces the postscript file filenameHalfSpace.ps and the text file filename_Table.output. The file filename_Main.output contains a summary of these results. We describe the postscript file and the summary here. The half-space table is described in Section 3.3.

The half-space theorem determines open half-spaces in the upper half-plane. It provides one such half-space for every slope $\frac{p}{q}$ such that $\mu_f(\frac{p}{q}) \neq \frac{p}{q}$ and $\mu_f(\frac{p}{q}) \neq \odot$. This open half-space contains no fixed point of the pullback map σ_f. Moreover, the open interval of $\mathbb{R} \cup \{\infty\}$ in the boundary of this half-space does not contain the negative reciprocal of the slope of an obstruction for f. We call such intervals and connected unions of them excluded intervals. The shaded region in filenameHalfSpace.ps is the union of those half-spaces which arise from slopes $\frac{p}{q}$ such that $|p|$ and $|q|$ are bounded by the bound input at the keyboard. Increasing the numerator-denominator bound creates more half-spaces. This enlarges the shaded region and the set of excluded points (possibly not strictly).

NETmap outputs in the summary the set of excluded points in $\mathbb{R}$ as a disjoint union of open intervals. Although not graphically appealing like the postscript file, this list of intervals is

numerically much more precise. It also handles the entire real line, whereas the postscript file necessarily handles only a finite interval.

The endpoints of the excluded intervals arising from the half-space theorem are in general numbers in real quadratic number fields. They are not always rational. In order for NETmap to reach its conclusions using only integer arithmetic, it shrinks these intervals slightly to obtain intervals with rational endpoints. The endpoints of the subintervals are typically within 10^{-15} of the endpoints of the original intervals.

If the orbifold of f is hyperbolic, if every point in $\mathbb{R}$ is excluded and if ∞ is not an obstruction, then f is unobstructed. In this case, `NETmap` announces in the summary that the half-space theorem shows that f is Thurston equivalent to a rational map.

Now suppose that the half-space computation does not exclude every real number, and that the determination of whether or not f is Thurston equivalent to a rational map has not been made by some other means. In this case, `NETmap` performs the supplemental half-space computation. The main tool in the supplemental half-space computation is the extended half-space theorem, discussed in Section 8 of [2]. To state a qualitative version of the extended half-space theorem, we first restate the part of the half-space theorem which deals with obstructions. If $s \in \overline{\mathbb{Q}}$, $\mu_f(s) \neq s$ and $\mu_f(s) \neq \odot$, then the half-space theorem determines an open interval containing $-1/s$ such that no number in this open interval is the negative reciprocal of a Thurston obstruction. On the other hand, if $s \in \overline{\mathbb{Q}}$ and either $\mu_f(s) = s$ or $\mu_f(s) = \odot$, then the extended half-space theorem determines an open interval containing $-1/s$ such that no extended rational number in this open interval *other than* $-1/s$ is the negative reciprocal of a Thurston obstruction.

In this paragraph we very briefly describe the supplemental half-space computation. The half-space computation yields an open set of excluded real numbers. `NETmap` searches for a rational number not in this set, trying to minimize absolute values of numerator and denominator. (This is done without regard for the numerator-denominator bound input at the keyboard. These numerators and denominators can be much larger than the numerator-denominator bound input at the keyboard.) After it finds a suitable rational number t, it checks whether or not t is the negative reciprocal of an obstruction. If an obstruction has not been found, then `NETmap` applies either the half-space theorem or the extended half-space theorem to $-1/t$, whichever is appropriate. This enlarges the excluded set of real numbers. This procedure is then iterated.

In the summary `NETmap` lists every interval which it finds in this way together with t and whether this interval was obtained by applying the half-space theorem (HST) or the extended half-space theorem (EXTENDED HST) to $-1/t$. Instead of either HST or EXTENDED HST, the output might be EXTENDED HST $->$ HST. This means that during the extended half-space computation for t, a half-space theorem excluded interval was found which contains t. The supplemental half-space computation often concludes with a set of excluded real numbers whose complement is a finite set of rational numbers. (A forthcoming paper will show that if the orbifold of f is hyperbolic and there is no obstruction with multiplier 1, then this always happens theoretically.) In this situation it is a straightforward matter to determine whether or not f is obstructed. If `NETmap` fails to determine whether or not f is obstructed, then it presents all of the excluded real numbers which it has found as a list of disjoint open intervals.

Slope function dynamics. The slope function dynamics for Euclidean NET maps are simple, and `NETmap` gives a quite complete description in this case. `NETmap`'s output of

slope function dynamics is never complete in the case of a hyperbolic orbifold. For hyperbolic orbifolds, NETmap proceeds very differently depending on whether f is obstructed or unobstructed.

NETmap always searches for all slope function fixed points. It usually finds all of them, even when there are infinitely many. It does this by a method analogous to how it determines rationality using the half-space theorem and the extended half-space theorem. Analogous to the half-space theorem, Corollary 8.3 of [7] provides intervals containing no slope function fixed points. There is also an analog of the extended half-space theorem.

Suppose that the orbifold of f is hyperbolic and that f is unobstructed. In this case NETmap searches for slope function cycles. The basic idea is to simply compute orbits of slopes under the slope function and to record the cycles at the ends of these orbits. However, it does something more efficient. Recall that the user inputs a positive integer B which bounds the absolute values of numerators and denominators of slopes to consider. Let S be this set of slopes. Let $\frac{p}{q} \in S$. NETmap computes $\mu_f(\frac{p}{q})$. Suppose that this is unequal to $\odot$ and equal to $\frac{p'}{q'}$. If $\frac{p'}{q'} \in S$ and $|p'| + |q'| < |p| + |q|$, then NETmap terminates the computation for $\frac{p}{q}$ and proceeds to the next slope. The cycle for $\frac{p}{q}$ will be found in the computation for $\frac{p'}{q'}$. This reduction from $\frac{p}{q}$ to $\frac{p'}{q'}$ is what usually happens. This phenomenon usually makes this cycle computation rather fast. If the computation for $\frac{p}{q}$ does not reduce to $\frac{p'}{q'}$, then the program proceeds in the straightforward manner to compute successive elements of the orbit of $\frac{p}{q}$. Doing so, it might obtain slopes not in the set S. It will tolerate 29 consecutive slopes not in S. However, upon reaching 30 consecutive slopes not in S, it quits computing the orbit of $\frac{p}{q}$, conceding failure. It then proceeds to the next slope. Another way that NETmap might concede failure is to eventually obtain a slope whose numerator or denominator is at least 100,000 in absolute value.

Now suppose that the orbifold of f is hyperbolic and that f is obstructed. In this case instead of working with the set of slopes whose numerators and denominators are bounded by the bound B input by the user, NETmap attempts to work with the set S of slopes whose intersection pairing with the obstruction slope is at most B. This is an infinite set of slopes. NETmap is often able to reduce this infinite set to a finite set by utilizing the stabilizer of the obstruction in the group of modular group liftables. When it is unable to do so, it reverts to the algorithm used in the unobstructed case.

The slope function dynamics for obstructed NET maps can be much more complicated than for unobstructed NET maps. NETmap's computations and output in the former case are accordingly also more complicated. For example, there could be infinitely many fixed points. There could be an "infinite cycle", an infinite set of slopes on which μ_f acts as translation by 1 acts on $\mathbb{Z}$. There could be an infinite set of slopes on which μ_f acts as multiplication by 2 acts on $\mathbb{Z}$. More complicated behaviors occur. Slopes in such infinite sets are always expressed as fractions $(aN + b)/(cN + d)$. The N represents an arbitrary integer and is always present. The a, b, c and d are specific integers.

NETmap always describes the results of this slope function computation, including failures. For example, if these orbits all end in either a fixed point or a nontrivial cycle or the nonslope, then the program says so. The program lists the fixed points and nontrivial cycles which it finds. Because these orbits need not remain in the set S, the slopes in the cycles listed might not be in S either. Moreover, the search for all slope function fixed points might find a fixed point not in S.

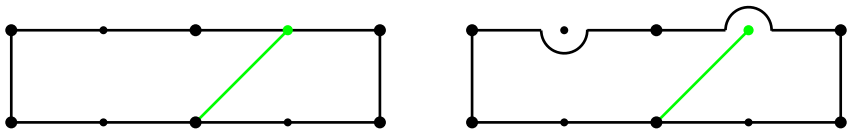


FIGURE 3. Modifying F_1' in an exceptional case

The fixed points of the slope function are listed in a table with seven columns. The first column contains the fixed point s. Column 2 contains $c = c(s)$. Column 3 contains $d = d(s)$. The remaining four columns deal with whether or not s is the slope of a mating equator. Unlike most of these computations, this depends on the translation term b in the presentation for f. So there is one remaining column for every choice of b among $0, \lambda_1, \lambda_2, \lambda_1 + \lambda_2$. Below the value of b is either "Yes" or "No". "Yes" means that s is the slope of a mating equator for the NET map defined with this value of b, and "No" means that s is not the slope of a mating equator.

Because `NETmap` is unable to print the symbol $\odot$, instead of saying that $\mu_f(s) = \odot$, it says that $\mu_f(s)$ is the nonslope.

Miscellaneous results. If f is Euclidean, then `NETmap` reports this. If f is a flexible Lattès map, then `NETmap` reports this. If every NET map in the pure modular group Hurwitz class of f is rational, then `NETmap` reports this. If every NET map in the modular group Hurwitz class of f is rational, then `NETmap` reports this. If $\mu_f(s) = \odot$ for every slope s, equivalently, σ_f is constant, then `NETmap` reports this. In the case of a constant pullback map, `NETmap` usually gives the value of this constant. At the other extreme, if there is no slope s such that $\mu_f(s) = \odot$, then `NETmap` reports this.

For each of these properties, if `NETmap` says nothing about the property, then the property does not hold. For example, if `NETmap` does not say that every NET map in the pure modular group Hurwitz class of f is rational, then some NET map in the pure modular group Hurwitz class of f is not rational.

Wreath recursions. The last output to appear in filename_Main.output is a fundamental group wreath recursion for every value of the translation term b which yields a NET map. This information is given in terms of four generators a, b, c and d of the fundamental group of the complement of the postcritical set of f in S^2. (Please excuse the double meaning of b.) The rest of this discussion of wreath recursions is about the choices made when computing this wreath recursion.

We work with the fundamental domain F_1, the green line segments and the points $s_1, \ldots, s_6$ described in the section on input. Let $\varphi\colon \mathbb{R}^2 \to \mathbb{R}^2/\Gamma_1$ be the canonical quotient map. Let F_1' be the set of points in the interior of F_1 which are not contained in any of the green line segments. We want F_1' to be connected, which is usually true. However, F_1 might be as in the left portion of Figure 3. In this case we remedy this defect by modifying F_1' as indicated in the right portion of Figure 3. This handles the case in which a green line segment joins the top and bottom of F_1. We modify F_1' in a similar manner if some green line segment joins the sides of F_1.

Thus F_1' is connected, an open topological disk. This property is largely responsible for the validity of the following discussion. We choose a point $x_0 \in F_1'$, and we take $\varphi(x_0)$ to be the basepoint for our fundamental group. We next define the group elements a, b, c, d.

Suppose that s_3 is not in the interior of F_1. We choose an arc α in F_1' from x_0 to a point near s_1. Then a is the homotopy class of a loop which traverses $\varphi(\alpha)$, then a very small simple closed curve around $\varphi(s_1)$ in the counterclockwise direction and then the inverse of $\varphi(\alpha)$. For b, we choose an arc β in F_1' from x_0 to a point near s_2. Then b is the homotopy class of a loop which traverses $\varphi(\beta)$, then a very small simple closed curve around $\varphi(s_2)$ in the counterclockwise direction and then the inverse of $\varphi(\beta)$.

If s_3 is in the interior of F_1, then α goes from x_0 to a point near s_2 and β goes from x_0 to a point near s_3. This defines a and b.

We define c and d analogously, using arcs γ and δ instead of α and β. If s_6 is not in the interior of F_1, then γ goes from x_0 to a point near s_5 and δ goes from x_0 to a point near s_4. If s_6 is in the interior of F_1, then γ goes from x_0 to a point near s_6 and δ goes from x_0 to a point near s_5. This defines c and d.

This completes the definition of a, b, c and d. Our fundamental group has presentation $\langle a, b, c, d : abcd = 1\rangle$.

We next discuss the elements of $f^{-1}(\varphi(x_0))$ and how they are indexed. The preimages of F_1 under the affine map Φ are 2×1 rectangles with sides parallel to the coordinate axes. If the translation term of Φ is either 0 or λ_2, then four of these rectangles meet at $(0, 0)$. Otherwise, four of these rectangles meet at $(1, 0)$. The centers of exactly $\deg(f)$ of these rectangles lie in either the interior of F_1 or in the "left half" of the boundary of F_1. ("Left" means in the direction of $-\lambda_1$.) Being correct up to homotopy, we compute as though the images under φ of these centers are the elements of $f^{-1}(\varphi(x_0))$. We order these $\deg(f)$ centers starting with the lowest leftmost one. That is center number 1. The centers above it have indices $2, 3, 4, \ldots$ in order. The next center is the lowest leftmost of those which remain. We iterate this procedure to index all of these centers. For each of these centers we choose an arc from x_0 to it so that the interior of this arc is in F_1'. If this center is in the interior of F_1 and on one of the green line segments, then this arc is chosen so that if it were extended to pass through the green line segment and return to x_0 within F_1', then it would encircle a segment of the green line segment in the counterclockwise direction.

These choices determine this wreath recursion.

3.2 The file filename_Mod.output

Let G be either the pure modular group, the modular group or the extended modular group, which allows for reversal of orientation. We say that an element φ of G is liftable if it is represented by a homeomorphism φ_0 for which there exists a homeomorphism $\widetilde{\varphi}_0$ representing an element $\widetilde{\varphi}$ of G such that $f \circ \varphi = \widetilde{\varphi}_0 \circ f$. The set of all such elements φ is a subgroup of G. If G is the pure modular group, then the assignment $\varphi \mapsto \widetilde{\varphi}$ defines the pure modular group virtual endomorphism. For the other two groups, this assignment is a multi-function and the result is a multi-endomorphism.

The file filename_MOD.output contains information about the subgroups of liftable elements relative to the pure modular group, the modular group and the extended modular group. Except for translations, all of this information is given in terms of the pullback action of these groups on the upper half-plane. The meaning of most of this information is hopefully clear. For the first two groups we have index (of the image) in $\mathrm{PSL}(2, \mathbb{Z})$, minimal number of generators (of the image), number of equivalence classes of cusps, genus and representatives of the cusp equivalence classes. For the second group there is also information about elliptic elements.

Next comes a list of cusps at the fundamental domain. These cusps are the extended rational numbers which are on either side of an edge of the fundamental domain in filenameTreePMOD.ps, filenameTreeMOD.ps or filenameTreeEMOD.ps, except that if an edge is fixed by a reflection, then the list contains only the extended rational number in the unshaded region of filenameSigma.ps.

Here is what NETmap does with the cusps for PMOD and MOD. For each of these cusps, if the pullback map σ_f maps the cusp to a cusp, then this image is given. Otherwise NET map gives the pseudoimages of this cusp. This calls for some explanation. Here, instead of dealing with simple closed curves, we deal with core arcs of simple closed curves. A core arc is simply an arc in S^2 joining distinct postcritical points whose interior avoids the postcritical points of f. Every core arc has a slope. Let α and β be disjoint core arcs. Their four endpoints are the four elements of P_f, and they have the same slope s. The connected components of the inverse image $f^{-1}(\alpha \cup \beta)$ of $\alpha \cup \beta$ consists of two arcs and some simple closed curves. Suppose that $\mu_f(s) = \odot$. Then one of these connected components either contains at least three elements of P_f or it contains two elements of P_f and separates the other two elements of P_f. It is a union of core arcs. We call the slopes of these core arcs pseudoimages of s. The pseudoimages given by NETmap are negative reciprocals of the pseudoimages of the slope s corresponding to the cusp $-1/s$.

Here is what NETmap does with the cusps for EMOD. In this case NETmap tries to determine the image of the cusp under the pullback map σ_f even if this image lies in the upper half-plane. Of course, if the image is a cusp, then this is given. Sometimes NETmap can only determine that the image lies in a geodesic, in which case it gives this geodesic. If the image in the upper half-plane can be determined, then this image is given as the intersection of two or (redundantly) three geodesics. The corresponding values of c and d are also given.

Next comes a list of generators for the pullback action of the group on the upper half-plane. Matrix(a, b, c, d) means $\left[\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right]$. These generating sets arise from the fundamental domains in the files filenameTreePMOD.ps, filenameTreeMOD.ps and filenameTreeEMOD.ps. These fundamental domains correspond to fundamental domains for the pullback actions of these groups on the upper half-plane. The generators given pair edges of the latter fundamental domains, and NETmap describes these edge pairings. So if $(r_1, r_2) -> (r_3, r_4)$ appears in the column with heading "EDGE PAIRING", then this generator takes the geodesic with endpoints r_1 and r_2 to the geodesic with endpoints r_3 and r_4.

Next comes another listing of the same generators together with their images under the modular group virtual endomorphism.

NETmap next gives information on the image of the group of liftables under the modular group virtual endomorphism.

For the modular and pure modular groups, σ_f induces a branched covering map from the upper half-plane modulo the action of the group of liftables to the upper half-plane modulo the image of this group under the virtual endomorphism. This induced map is a finite branched cover from one finitely punctured compact Riemann surface to another. The degree of this induced map is given next. For PMOD NETmap also gives the degrees of Sarah Koch's [6] maps X and Y.

The map Y is a Belyi map. Hence there is a dessin d'enfant associated to it. Unable to draw the dessin, NETmap gives equivalent information, a permutation triple [8] for Y. This is a triple of permutations $(\pi_1; \pi_2; \pi_3)$ with $\pi_1\pi_2\pi_3 = 1$. The permutations π_1, π_2 and π_3 describe the monodromy of Y at 0, 1 and ∞.

Here is an explicit construction of the dessin from this permutation triple. The dessin is a bipartite graph embedded in an oriented closed surface whose genus is the genus of the pure modular group liftables, mentioned above. Every edge of this graph has a white vertex and a black vertex. The permutations act on the edges of this graph. The number of edges is the degree of Y. Every cycle in π_1, including trivial cycles, corresponds to a white vertex and the cycle describes the edges which contain this vertex in positive cyclic order. In the same way, the permutation π_2 describes the black vertices and the edges which contain them. This determines the dessin.

NETmap gives DeckMod(f), the subgroup of the modular group represented by homeomorphisms φ such that $f \circ \varphi = f$. These are always given by translation by either 0, λ_1, λ_2 or $\lambda_1 + \lambda_2$.

NETmap gives the subgroup of the modular group liftables represented by translations. Beware that these elements map injectively to the modular group of f exactly when DeckMod(f) is trivial.

If NETmap draws a guess for σ_f in the file filenameSigma.ps, then the last information to appear in filenameMOD.output deals with symmetries of σ_f. This calls for an explanation. Every liftable element φ induces a pullback map σ_φ on Teichmüller space such that $\sigma_f \circ \sigma_\varphi = \sigma_{\widetilde{\varphi}} \circ \sigma_f$, where $\widetilde{\varphi}$ is an image of φ under the extended modular group virtual multi-endomorphism. (The map $\sigma_{\widetilde{\varphi}}$ is uniquely determined even if $\widetilde{\varphi}$ is not.) Liftables do not always account for all pairs of isometries ψ and $\widetilde{\psi}$ such that $\sigma_f \circ \psi = \widetilde{\psi} \circ \sigma_f$. Such isometries ψ are *symmetries* of σ_f. Symmetries which do not arise from liftables are *extra symmetries*.

NETmap is unable to verify that a general isometry is a symmetry of σ_f. However, it is able to reduce the search for symmetries to a finite set of cosets of the group of liftables. It then checks representatives of these cosets for many conditions which symmetries satisfy. In this way it might determine that σ_f has no extra symmetries. If a representative of a nontrivial coset satisfies all conditions checked, then NETmap reports that it seems to have found an extra symmetry. It gives the potential extra symmetry ψ together with its image $\widetilde{\psi}$ under the modular group endomorphism.

3.3 The file filename_Table.output

For every slope $\frac{p}{q}$ such that both $|p|$ and $|q|$ are less than or equal to the numerator-denominator bound input at the keyboard, NETmap evaluates $\mu_f(\frac{p}{q})$ and applies the half-space theorem to $\frac{p}{q}$ if appropriate. The file filename_Table.output contains the numerical results of these computations.

The output in filename_Table.output appears in eight columns. The first column contains the slopes $\frac{p}{q}$ determined by the numerator-denominator bound with $\gcd(p, q) = 1$ and $q \geq 0$. Column 2 contains $\frac{p'}{q'} = \mu_f(\frac{p}{q})$. This entry is blank if and only if $\mu_f(\frac{p}{q}) = \odot$. Column 3 contains $c = c(\frac{p}{q})$. Column 4 contains $d = d(\frac{p}{q})$. If either $\mu_f(\frac{p}{q}) = \odot$ or $\mu_f(\frac{p}{q}) = \frac{p}{q}$, then the rest of this row is blank. The slope function computation is an implementation of Theorem 5.3 of [1]. The computation of c and d is an implementation of Theorem 4.1 of [1].

Suppose that $\mu_f(\frac{p}{q}) \neq \odot$ and that $\mu_f(\frac{p}{q}) \neq \frac{p}{q}$. Then NETmap applies the half-space theorem to $\frac{p}{q}$. As discussed above, NETmap shrinks the intervals given by the half-space theorem to obtain intervals with rational endpoints. It obtains a half-space H in the upper half-plane. The part of the boundary of H which lies in the upper half-plane is either a Euclidean semicircle or a vertical ray. If the boundary is a semicircle, then its center appears in column

5 as a point on the x-axis. If the boundary is a ray, then the x-value of this ray appears in column 5. The heading of column 6 refers to the shading of H in filenameHalfSpace.ps. Suppose that the boundary of H is a semicircle. If H is within the semicircle, then the shading is in, and otherwise it is out. Suppose that the boundary of H is a ray. If H is left of the ray, then the shading is left, and otherwise it is right. In the case of a semicircle, let C be its center and let R be its radius. The last two columns give the endpoints of the semicircle, $C - R$ and $C + R$. These are the endpoints of an open interval which contains no negative reciprocals of Thurston obstructions.

3.4 The file filenameMod2Corre.ps

If we reduce the numerator and denominator of a slope (with numerator and denominator relatively prime) modulo 2, we obtain either $\frac{0}{1}$, $\frac{1}{1}$ or $\frac{1}{0}$. Allowing the nonslope, we obtain four possible values: 0, 1, ∞ and $\odot$. The file filenameMod2Corre.ps contains a graph with four vertices labeled 0, 1, ∞ and $\odot$. It has a directed edge from vertex x to vertex y if and only if there exists a slope s "congruent to the label of x modulo 2" and the image of s under the slope function is "congruent to the label of y modulo 2". This edge is labeled with the multiplier of slope s. Instead of drawing a distinct edge for every multiplier, `NETmap` draws just one edge with multiple multipliers. It doesn't label edges to the nonslope because these multipliers all equal 0.

3.5 The file filenameDynPortrait.ps

The file filenameDynPortrait.ps contains abbreviated dynamic portraits, one for each translation term for which the resulting Thurston map is a NET map. More precisely, for each such function there is a weighted directed graph. The vertices corresponding to the four postcritical points are labeled A, B, C and D. A preimage in the fundamental domain F_1 of each is given. Every other vertex is labeled with an integer. A vertex labeled with an integer n represents n critical points which are not postcritical and which map to the same postcritical point. The last convention permits the drawing and quick comprehension of dynamic portraits of NET maps with large degrees. These abbreviated dynamic portraits are drawn so that if two of them are isomorphic, then they are identical.

3.6 The file filenameGraphMu.ps

The file filenameGraphMu.ps has a graph of the slope function. The values of x used are those slopes determined by the maximum of 50 and the numerator-denominator bound input at the keyboard.

3.7 The file filenameGraphMuLines.ps

The file filenameGraphMuLines.ps has another graphical representation of the slope function. To explain it, let m be the least common multiple of the lengths of the slope function cycles which the program finds. Let $\frac{p}{q}$ be a slope in the set determined by the maximum of 50 and the numerator-denominator bound input at the keyboard. Suppose that the mth iterate of the slope function at $\frac{p}{q}$ is equal to an extended rational number $\frac{p'}{q'}$, expressed in reduced form with $q' \geq 0$. Then filenameGraphMuLines.ps contains a line segment joining (p, q) and (p', q').

3.8 The file filenameGraphMuTorus.ps

This is another graphical representation of the slope function. Here we view slopes as lying in the 1-point compactification of $\mathbb{R}$. So, disregarding $\odot$, the slope function μ_f maps points on a circle to points on a circle. The resulting graph then lies on a torus. This torus is shown in filenameGraphMuTorus.ps as a square whose opposite sides are to be identified.

The sides of the square in filenameGraphMuTorus.ps are parametrized by a function which is closely related to Minkowski's question mark function. The question mark function ? has the following properties. If $\frac{p}{q}$ and $\frac{r}{s}$ are rational numbers in reduced form in the closed unit interval such that $|ps - qr| = 1$, then

$$? \left(\frac{p+r}{q+s} \right) = \frac{1}{2} \left[? \left(\frac{p}{q} \right) + ? \left(\frac{r}{s} \right) \right].$$

Furthermore, $?(\frac{0}{1}) = \frac{0}{1}$ and $?(\frac{1}{1}) = \frac{1}{1}$. The function which parametrizes the sides of the square in filenameGraphMuTorus.ps satisfies the same functional equation as ? for all extended rational numbers (using both $\frac{-1}{0}$ and $\frac{1}{0}$ for ∞), but for its initial conditions, it maps $\frac{-1}{0}$ to $\frac{0}{1}$ and $\frac{0}{1}$ to $\frac{1}{2}$ and $\frac{1}{0}$ to $\frac{1}{1}$. It maps $\mathbb{Q}$ to the set of dyadic rational numbers in the open unit interval $(0, 1)$.

3.9 The file filenameHalfSpace.ps

The shaded region in filenameHalfSpace.ps is the union of the half-spaces gotten by applying the half-space theorem to slopes $\frac{p}{q}$ such that $|p|$ and $|q|$ are bounded by the numerator-denominator bound input at the keyboard. See the discussion of the half-space computations in Section 3.1 for details.

3.10 The file filenamePrenDgm.ps

This file contains a virtual presentation diagram for f. In other words, it is a graphical representation of the input data. It determines the map f except for omitting the translation term. In this virtual presentation diagram, the origin is drawn as a circle rather than a dot.

3.11 The file filenameSigma.ps

The file filenameSigma.ps contains a *guess* at the form of Thurston's pullback map σ_f. It contains two views of the upper half-plane. The top view corresponds to the domain of σ_f, and the bottom view corresponds to the range. In the top view, solid black hyperbolic geodesics are the reflection axes of reflections in the list of EMOD generators given in filenameMOD.output. Dotted black hyperbolic geodesics are not reflection axes of generators. The same is true for the bottom view with "generators" replaced by "generator images". The top view is the upper half-plane analog of filenameEMODTree.ps. It is a fundamental domain for the action of the subgroup of liftables in the extended modular group. The bottom view shows a guess at the image of this fundamental domain under σ_f. It is an interesting problem to combine the visual information in filenameSigma.ps with the numerical information in filenameMOD.output to determine the form of σ_f.

This guess at the form of σ_f is not always correct. If the image of the extended modular group in $\mathrm{GL}(2, \mathbb{Z})$ is a reflection group, then all boundary geodesics are reflection axes, and this guess seems to always be correct. In general this group is not a reflection group, and images of boundary geodesics which are not reflection axes are always very much in doubt. This guess at the image is always a union of fundamental domains for the action of $\mathrm{PGL}(2, \mathbb{Z})$ on the upper half-plane.

NETmap labels axes of liftable reflections, and σ_f respects these labels. For example, a geodesic in the domain labeled "A" maps to the geodesic in the image labeled "A".

NETmap marks folds. A fold is a union of consecutive edges in the domain such that its endpoints have equal images under σ_f and σ_f seems to map the fold into an arc. (The word "seems" is needed because the images of dashed edges are always in doubt.) The fold is folded (noninjectively) into the arc. Such edges in the domain are marked by what might be called an underline. Figure 4 shows two folds, one extending from 1 to $\frac{5}{3}$ and the other from $\frac{5}{3}$ to $\frac{7}{3}$. There is no guarantee that NETmap marks all folds, although this seems to be true; there are some types of complicated folds, (which might not even exist) which it does not recognize.

Reflection axes of extra symmetries (for which, see the end of Section 3.2) which are reflections are drawn in color, as are their images under σ_f, and σ_f respects colors. The reflection axes of four reflections which are extra symmetries are drawn in the top half of Figure 4.

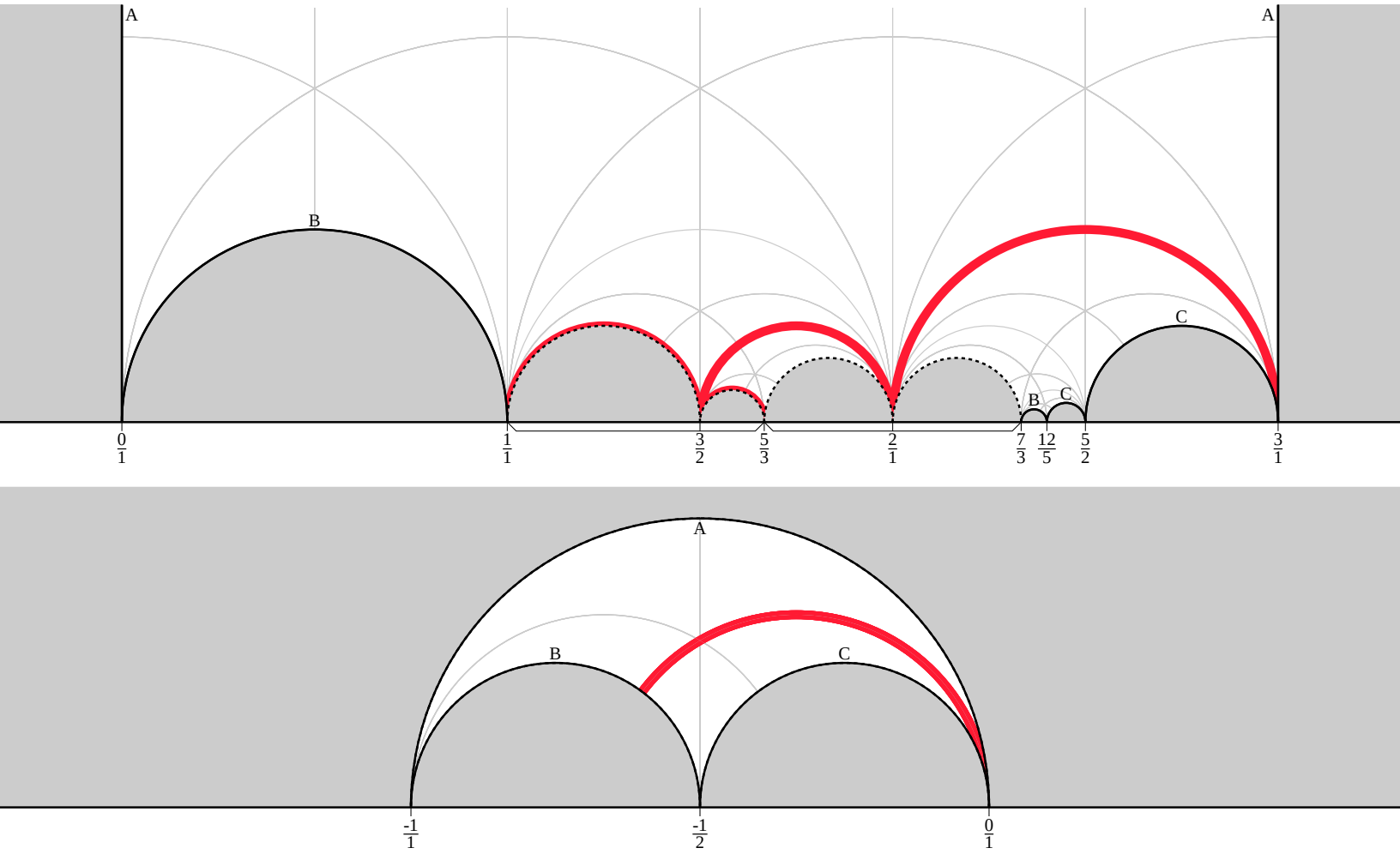
Special points on the x-axis of both views of the upper half-plane are marked by ticks. The program labels these ticks from left to right. If the label of a tick does not overlap the printed label on the left, then this label is printed. If this label overlaps the printed label on the left, then this label is not printed.

Here is one way in which this information can be used to determine σ_f. Consider sampleSigma.ps. For the domain we see a hyperbolic quadrilateral, and for the range we see a hyperbolic triangle. The image in $\mathrm{GL}(2,\mathbb{Z})$ of our group of EMOD liftables is a reflection group, generated by four reflections. It is not hard to prove that modular group virtual endomorphisms always map reflections to reflections, and Thurston pullback maps always map reflection axes into reflection axes. According to sampleMOD.output and sampleSigma.ps, the geodesics with endpoints 0, ∞ and 0, $\frac{1}{2}$ both map into the geodesic with endpoints -2 and 0. The geodesic with endpoints $\frac{1}{2}$ and 1 maps into the geodesic with endpoints 0 and ∞. The geodesic with endpoints 1 and ∞ maps into the geodesic with endpoints -1 and ∞. We view the domain quadrilateral as a conformal triangle with vertices at $\frac{1}{2}$, 1 and ∞. By the Riemann mapping theorem there exists a unique conformal equivalence from this conformal triangle to the triangle in the bottom half of sampleSigma.ps taking $\frac{1}{2}$ to 0, 1 to ∞ and ∞ to $-1+i$. We extend this map to the entire upper half-plane using the reflection principle. We obtain an analytic map F from the upper half-plane to itself. This map F satisfies all of the functional equations satisfied by σ_f, that is, $F \circ \sigma_\varphi = \sigma_{\widetilde{\varphi}} \circ F$, where φ is a liftable, and $\widetilde{\varphi}$ is the image of this liftable under the virtual multi-endomorphism. The map F also extends continuously in the augmented Teichmüller space topology to the set of extended rational numbers, and it agrees with σ_f at every extended rational number.

Now we pass to the quotients of the extended upper half-plane under the action of the modular group liftables and the image of this group under the virtual multi-endomorphism. The file sampleMOD.output shows that both of these compact Riemann surfaces have genus 0. We see that both σ_f and F induce degree 1 maps from the first Riemann surface to the second and they agree at the three cusps. They must be equal. It follows that $\sigma_f = F$.

After verifying that the potential extra symmetries in Figure 4 are indeed extra symmetries, a similar argument obtains an analogous description of the pullback map of 61HClass8 using Figure 4.

3.12 The file filenameTree PMOD.ps



61HClass8

FIGURE 4. 61HClass8Sigma.ps

This file deals with the group of liftables relative to the pure modular group PMOD. This group acts on a tree which we call the Stern-Brocot tree. The Stern-Brocot tree is a combinatorial model of the tree which consists of all edges of finite hyperbolic length in the standard tesselation of the upper half-plane for $\mathrm{GL}(2,\mathbb{Z})$. The rest of this paragraph is devoted to a very brief discussion of the Stern-Brocot tree. We view this tree as embedded in the plane. The complementary regions are labeled by extended rational numbers. Suppose that two adjacent regions have labels $\frac{r}{s}$ and $\frac{t}{u}$. Then $s \geq 0$, $u \geq 0$, $\gcd(r,s) = 1$ and $\gcd(t,u) = 1$. If in addition $r \geq 0$ and $t \geq 0$, then the region below and adjacent to these regions has label $\frac{r+t}{s+u}$. Reflection about the horizontal line through the valence 2 vertex

contained in the regions with labels $\frac{0}{1}$ and $\frac{1}{0}$ takes region with label $\frac{r}{s}$ to region with label $-\frac{r}{s}$. Every extended rational number is the label of exactly one region. Let e be an edge. One vertex of e has valence 2, and one vertex of e has valence 3. We orient e toward its vertex of valence 3. With this orientation, let $\frac{a}{c}$ be the label of the region on the left, and let $\frac{b}{d}$ be the label of the region on the right. Then $\left|\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right| = \pm 1$. If this determinant is -1, then we multiply one column of $\left[\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right]$ by -1. We obtain an element of $\mathrm{SL}(2,\mathbb{Z})$, which leads to an element of $\mathrm{PSL}(2,\mathbb{Z})$. This map from edge to element of $\mathrm{PSL}(2,\mathbb{Z})$ is a bijection. This bijection leads to a left action of $\mathrm{PSL}(2,\mathbb{Z})$ on the Stern-Brocot tree. A bit more effort leads to an action of $\mathrm{PGL}(2,\mathbb{Z})$.

`NETmap` draws a portion of the Stern-Brocot tree. It always draws the same portion, as shown in sampleTreePMOD.ps. It also draws a fundamental domain for the action of the group of PMOD liftables on the Stern-Brocot tree, or as much as fits in the drawn portion. The green edges form the fundamental domain. The edges containing the leaves of this tree are labeled whether they are in the fundamental domain or not. Edges with equal labels are equivalent under the action of the group.

Because the Stern-Brocot tree is isomorphic to a subgraph of the 1-skeleton of the standard tesselation of the upper-half plane for $\mathrm{GL}(2,\mathbb{Z})$, this Stern-Brocot tree fundamental domain can be viewed as a spine of a fundamental domain for the action of the group on the upper half-plane.

3.13 The file filenameTree MOD.ps

This is the modular group analog of filenameTreePMOD.ps, described in Section 3.12.

3.14 The file filenameTree EMOD.ps

This is the extended modular group analog of filenameTreePMOD.ps, described in Section 3.12. There is a difference here because EMOD allows for reversal of orientation. The difference is that in the case of EMOD, it is possible for an edge of the fundamental domain to be fixed by a reflection. Such edges are red instead of green. Similarly, an edge not in the fundamental domain has a red label if and only if some orientation-reversing group element takes it into the fundamental domain.

4. Integer overflow

Considerable pains have been taken to minimize the possibility of integer overflow. Integers are stored using 32 bits. Their absolute values are at most $2^{31} - 1 = 2,147,483,647$. Whenever there is potential for integer overflow during addition, subtraction or multiplication, the computation is performed using 64 bits. If the result can be stored using 32 bits, then computation proceeds unabated. If the result cannot be stored using 32 bits, then the segment of the program containing this computation is usually aborted and an error message is issued with details.

There are some situations in which integer overflow is in some sense harmless. This can happen during the supplemental half-space computation. Consider how the supplemental half-space computation proceeds. We search for a rational number which has not been excluded. Suppose that we find such a rational number t. We then apply either the half-space theorem or the extended half-space theorem to $-1/t$. Suppose that integer overflow occurs during this application of either the half-space theorem or the extended half-space

theorem. Then the program simply discards t and searches for another rational number. Thus it is possible for the result of the supplemental half-space computation to be valid even when integer overflow occurs.

Nearly all instances of integer overflow generate error messages. These error messages appear at the bottom of the terminal and the main output file. Integer overflow does not invalidate any output.

In theory `NETmap` detects all instances of integer overflow, except one kind: it assumes that all input values can be stored using 32 bits.

References

[1] J. W. Cannon, W. J. Floyd, W. R. Parry and K. M. Pilgrim, *Nearly Euclidean Thurston maps*, Conformal Geometry and Dynamics **16** (2012), 209-255.

[2] W. Floyd, G. Kelsey, S. Koch, R. Lodge, W. Parry, K. M. Pilgrim, E. Saenz, *Origami, affine maps, and complex dynamics*, Arnold Math. J. **3** (2017), 365–395.

[3] W. J. Floyd, W. R. Parry and K. M. Pilgrim, *Presentations of NET maps*, Fundamenta Math. **244** (2019), 49–72.

[4] W. J. Floyd, W. R. Parry and K. M. Pilgrim, *Modular groups, Hurwitz classes and dynamic portraits of NET maps*, Groups, Geometry, and Dynamics, **13** (2019), 47–88.

[5] W. J. Floyd, W. R. Parry and K. M. Pilgrim, *Rationality is decidable for nearly Euclidean Thurston maps*, in preparation.

[6] S. Koch, *Teichmüller theory and critically finite endomorphisms*, Advances in Mathematics Vol. 248, 2013.

[7] W. Parry, *NET map slope functions*, submitted.

[8] J. Sijsling and J. Voight, *On computing Belyi maps*, Publ. Math. Besançon: Algèbre Théorie Nr. 2014/1, Presses Univ. Franche-Comté, Besançon, 73–131.