2009-11-19

Growth and Geodesics of Thompson's Group F

Jennifer L. Schofield
Brigham Young University - Provo

Growth and Geodesics of Thompson's Group F

Jennifer Schofield

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

James W. Cannon, Chair
Stephen P. Humphries
Eric L. Swenson

Department of Mathematics

Brigham Young University

December 2009

ABSTRACT

Growth and Geodesics of Thompson's Group F

Jennifer Schofield

Department of Mathematics

Master of Science

In this paper our goal is to describe how to find the growth of Thompson's group F with generators a and b. Also, by studying elements through pipe systems, we describe how adding a third generator c affects geodesic length. We model the growth of Thompson's group F by producing a grammar for reduced pairs of trees based on Blake Fordham's tree structure. Then we change this grammar into a system of equations that describes the growth of Thompson's group F and simplify. To complete our second goal, we present and discuss a computer program that has led to some discoveries about how generators affect the pipe systems. We were able to find the growth function as a system of 11 equations for generators a and b.

Keywords: Thompson's group F, growth function, reduced pairs of trees, pipe systems

ACKNOWLEDGMENTS

Thanks to my advisor James Cannon for introducing me to the subject matter contained in this thesis. Thanks to him also for seeing my progress even when I did not. Thanks to my friend Jeanne Fischer for English support. Thanks to Eric Swenson and Stephen Humphries for reviewing the text. I would like to thank Greg Conner for his help and support as the graduate coordinator. I would like to thank my family for their patience and support throughout the process of writing this thesis.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 PREFACE

Richard Thompson [7] [8] first described his group F in 1965. He used it as an ingredient in constructing groups with unsolvable word problems and in constructing finitely presented infinite simple groups. This group F can be described in many different ways; for example, it can be described as a subgroup of the piecewise linear homeomorphism group of the unit interval with emphasis either on function composition or on the individual elements as graphs, as a group whose elements can be described either as pairs of binary rooted trees or as forest trees [4], as planar pipe systems, or as a group with explicit finite and infinite group presentations.

Here are the standard finite and infinite group presentations. We start with two generators x_0 and x_1 and define, in terms of them, infinitely many other generators $x_{j+1} = x_0^{-1} x_j x_0$ for each $j \geq 1$. The standard finite presentation is then

$$F =< x_0, x_1 \,|\, x_1^{-1} x_2 x_1 = x_3 \text{ and } x_1^{-1} x_3 x_1 = x_4 >,$$

and the standard infinite presentation is

$$F =< x_0, x_1, x_2, \ldots \mid \text{ for each } i < j,\ x_i^{-1} x_j x_i = x_{j+1} > .$$

For convenience throughout the rest of the paper, we let $a = x_0$ and $b = x_1$ with inverses $A = x_0^{-1}$ and $B = x_1^{-1}$. We will occasionally use the symbol F_{ab} to designate group F as generated by a and b.

Blake Fordham [5] has discovered an efficient algorithm for determining the minimal length of a group element in terms of generators x_0 and x_1. He begins with a pair of rooted binary trees representing the element with the pair *reduced* in the sense that no smaller trees suffice to represent the element. His method mechanically decorates the tree pair with certain numbers whose sum gives the desired length.

Our principal goal is to define a grammar that constructs each reduced pair of rooted binary trees, together with the Fordham decoration, in a unique way and to use that grammar to calculate the growth function of the group F with respect to the generating set $\{x_0, x_1\}$. We are only partially successful in attaining this goal. We reduce the calculation to eleven complicated expressions.

In this paper we look at two different topics: The first of these comprises Chapters 2, 3, 4, and Appendix A where we consider our principal goal the growth of Thompsons group F. Chapter 2 will be a review of everything that is known about Thompson's group F. In Chapter 3, we describe a Fordham tree that will have the carets labelled according to his labeling. In Chapter 4, we produce a grammar that will describe a reduced pair of Fordham trees with Fordham's caret labeling. Also in Chapter 4, we will change the grammar into a system of equations that can produce both the number N caret trees and the number of geodesic elements of length N in Thompson's group F. The second topic comprises Chapter 5 and Appendix B, where we study group F with respect to a generating set that is geometrically symmetric, formed by adding to generators a and b another generator c that is the geometric reflection of b. Thompson's group F could be generated by just a and c or by a and b because b and c are symmetric. We have created a computer program that pictures group elements as planar pipe systems and allows us to see quickly how the pipe system changes when we multiply by a generator. One can use this program to explore geodesics in group F. In Chapter 5, we will demonstrate an example of the output of the program. Also we will try to describe how all three generators, a, b, c, can work together by first investigating in detail how each one changes a pipe system.

2 REVIEW OF THOMPSON'S GROUP F

Thompson's group F can be described in many ways. This chapter gives three ways of describing the elements of Thompson's group F. The first is Thompson's own description

as a group of piecewise linear homeomorphisms from the unit interval to itself. The second describes elements as pairs of planar, binary, rooted trees [2],[5], [6]. The third describes elements as planar pipe system. Each of these representations gives unique insight into Thompson's group F.

2.1 Representing group elements as homeomorphisms of the unit interval

In 1965, Richard Thompson invented an infinite, non-abelian group that contains no rank 2 free subgroup.

Definition 2.1.1 (Thompson's group F)**.** Define F to be the set of piecewise linear homeomorphisms that map the closed unit interval $[0, 1]$ onto itself, fixing 0 and 1, such that each homeomorphism is differentiable except at finitely many dyadic rational numbers such that, on any differentiable interval, the slope is an integer power of 2. It can be shown that the elements of F map the set of dyadic rationals to itself bijectively. It follows that F is closed under function composition and that it is a subgroup of all homeomorphisms from $[0, 1]$ to $[0, 1]$.

Definition 2.1.2 (Generators for the group F)**.** The following two functions generate the group F:

$$G(x) = \begin{cases} \frac{x}{2} & 0 \leq x \leq \frac{1}{2} \\ x - \frac{1}{4} & \frac{1}{2} \leq x \leq \frac{3}{4} \\ 2x - 1 & \frac{3}{4} \leq x \leq 1 \end{cases} \qquad H(x) = \begin{cases} x & 0 \leq x \leq \frac{1}{2} \\ \frac{x}{2} + \frac{1}{4} & \frac{1}{2} \leq x \leq \frac{3}{4} \\ x - \frac{1}{8} & \frac{3}{4} \leq x \leq \frac{7}{8} \\ 2x - 1 & \frac{7}{8} \leq x \leq 1 \end{cases}$$

The function G acts like the generator a of the group F_{ab}, and the function H acts like b. The group F is isomorphic to the group F_{ab}. Here are the graphs of the functions G and H.

Note that the domain and range of G are broken into three intervals. The domain intervals are [0, 1/2], [1/2, 3/4], and [3/4, 1]. The range intervals are [0, 1/4], [1/4, 1/2], and [1/2, 1].

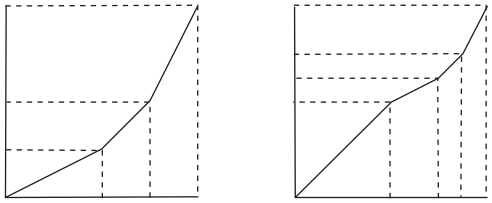


Figure 2.1: Graphs for the generating functions
G(x) (on the left) and H(x) (on the right).

The domain and range of H are broken into four intervals. The domain intervals are [0, 1/2], [1/2, 3/4], [3/4, 7/8], and [7/8, 1]. The range intervals are [0, 1/2], [1/2, 5/8], [5/8, 3/4], and [3/4, 1]. Such intervals can be used to describe Thompson's group F in another way which we shall discuss in the next section. We summarize:

Theorem 2.1.3. The homeomorphism group F generated by functions G and H is isomorphic to the group $F_{a,b}$ generated by elements a and b by an isomorphism that sends G to a and H to b.

2.2 Representing group elements by reduced pairs of trees

Let $f \in F$. Then f can be written as a composition of functions G and H and their inverses. Composition is hard to do quickly. If we keep track of the intervals into which the domain and range are divided, it is easier to compose two elements. Binary trees can record these intervals. We will now show how this can be done.

Definition 2.2.1 (Rooted binary tree)**.** A rooted binary tree S is a planar tree with the following three properties:

(1) If S is not empty, then S has a distinguished root vertex v_0.

(2) If S contains vertices other than v_0, then v_0 has valence 2.

(3) If v is a vertex of S other than v_0, then either

(a) v has valence 1, or

(b) v has valence 3 and there are two edges $e_{v,vL}$ and $e_{v,vR}$ which contain v but

are not in the path geodesic from v to v_0.

A concrete example of a rooted binary tree is the *family tree* or *pedigree chart* of an individual. The individual is represented by the root vertex v_0, the father by v_0R, and the mother by v_0L. In general, a vertex v represents some ancestor of the individual, the father of this ancestor by vR, and the mother by vL.

In mathematics, the terminology is usually inverted from that of the family tree. The vertex vL is called the *left child* of v and the vertex vR is called the *right child* of v. The vertex v is called the *parent* of vL and of vR.

The configuration of v, vL, vR with the two edges $e_{v,vL}$ and $e_{v,vR}$ joining them is viewed as a *caret* (caret = ˆ), with v at the top, vL at the lower left, and vR at the lower right. Thus the rooted binary tree with more than one vertex is a union of carets, and every rooted binary tree is also called a *caret tree*. A vertex with no children is called a *leaf*. A caret in which neither child has a child is said to be an *exposed caret* or a *trivial caret*.

We commonly represent a caret tree by a planar graph, where the root vertex is at the top, (mathematical) parents are above (mathematical) children, and left child is to the left of right child. It is sometimes convenient to arrange to have all leaves on the same bottom level. When the leaves are all at the same bottom level, it becomes obvious that the carets have a natural left to right ordering as demonstrated in the following figure. Likewise, the leaves have a natural left to right ordering.

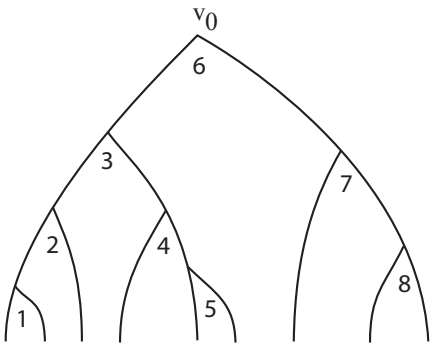


Figure 2.2: Caret order of a tree
The trival carets are 1, 5 and 8.

We sometimes invert the entire graph so that the root vertex is on the bottom, parents are below children, and all leaves are on the top level.

The caret tree is associated with a partition of $[0, 1]$ into subintervals as follows:

The vertex v_0 represents the entire interval $[0, 1]$. If a parent represents the interval $[x, y]$, then the left child represents the interval $[x, (x + y)/2]$ and the right child represents the interval $[(x + y)/2, y]$. Thus each vertex represents a basic *dyadic* interval.

An element of F can then be represented by a pair of trees, each having the same number of leaves (and, necessarily, the same number of carets). The first tree is viewed as the domain tree. The second tree is viewed as the range tree. Each leaf represents a basic dyadic interval. In each tree, the intervals are naturally ordered from left to right and represent a partition of $[0, 1]$ into intervals. The intervals of the domain tree are to be mapped linearly to the corresponding intervals of the range tree. The resulting mapping is an element of group F.

This will be demonstrated in the drawing of the domain tree of our function G. Recall that the domain intervals are [0, 1/2], [1/2, 3/4], and [3/4, 1]. Each element of F can be

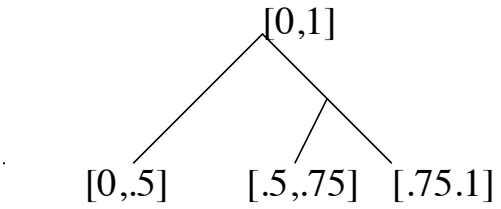


Figure 2.3: Tree intervals
Function G Domain tree

represented by infinitely many different pairs of trees. In order to obtain a unique representative pair, one must delete matching pairs of exposed carets. That is, if leaves vL and vR of the domain tree are matched with leaves wL and wR of the range tree, then these leaves may be deleted and v matched with w without changing the homeomorphism. This figure gives an example:

When no more trivial pairs can be deleted, the tree pair is called *reduced*. The reduced representative is unique.

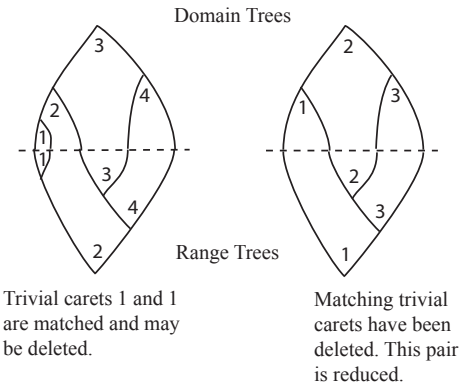


Figure 2.4: Reducing carets

Definition 2.2.2 (Sides of a tree)**.** The right side of a tree is the maximal path that begins at the root and connects root to right child and that right child to the next right child, and so forth. The left side of a tree is the maximal path that begins at the root and connects root to left child and that left child to the next left child, and so forth.

In the family tree or pedigree chart, the right side of the tree is the path that connects an individual to his father and to his father's father, to father's father's father, and so forth. The left side is the path that connects an individual to his mother and to his mother's mother, to mother's mother's mother, and so forth.

Theorem 2.2.3. Every nonempty tree has an odd number of vertices.

Proof. Every vertex except the root vertex has a unique left or right sibling to which it can be matched. □

Suppose that we have two pairs (D_1, R_1) and (D_2, R_2) of rooted binary trees representing elements ϕ_1 and ϕ_2 of group F. We may perform the composition (multiplication) $\phi_2 \circ \phi_1$ of the two elements as follows. By adding trivial caret pairs to the first pair and to the second pair, we find that we may assume that the range tree R_1 is precisely equal to the domain tree D_2. After that condition has been obtained, the composition is represented by (D_1, R_2).

2.3 Representing group elements by pipe systems

A pipe system is a simplified picture of a pair of trees representing an element of Thompson's group F. The domain tree and the range tree have the same number of carets. We have noted the natural left to right ordering of those carets in each tree. The carets of the domain tree may be matched with the carets of the range tree in an order-preserving manner. We replace each matched pair of carets by a vertical segment in the plane whose endpoints have the same height as the vertices representing the two caret parents. This vertical segment is called a *pipe*. Pipes are to be ordered left-to-right with the same ordering given the carets. Here is an example:

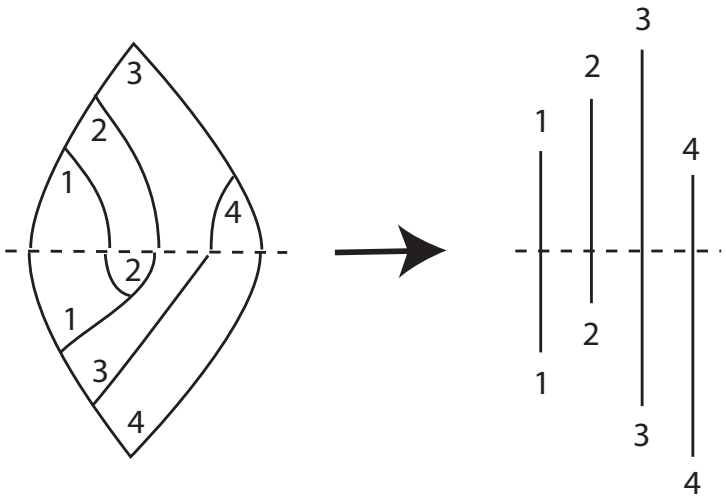


Figure 2.5: From trees to pipes

The numberings are only given to indicate the left-to-right ordering of the carets and pipes; these numbers are not a part of the pipe system.

Definition 2.3.1. A pipe system consists of a horizontal (dotted) line segment with a finite number of vertical line segments crossing it, each perpendicular to the horizontal segment. The vertical line segments are called pipes. We require that no two pipe endpoints have the same height unless there is an intervening pipe that hides the one endpoint from the other. Note that this is always true of the pipe system arising from a pair of binary trees.

Also we may call the endpoints of each vertical line segment a vertex. Since they are vertical each pipe has a upper vertex and a lower vertex.

The domain tree can be recovered from the upper half of the pipe system in the following way: Each upper vertex v represents the parent in a caret. This parent v is connected by descending paths to the left and right to the highest upper vertices not obscured by an intervening pipe. If there is no vertex visible to the left or right, then the path descends to a point of the dotted line where a leaf is created. Thus the left child vL of v is highest among those vertices that are to the left of v yet are visible from v (that is, are not obscured by an intervening pipe that is higher than the pipe containing v). If no such upper vertex exists, then the left child vL is a leaf and is to appear on the dotted line. The right child vR of v is defined similarly to the right of v.

In a similar manner, the range tree can be recovered from the lower half of the pipe system.

Since the actual physical realization of a tree pair is not unique, the same is true of a pipe system. Only the combinatorial properties of the pipe system are important. Thus pipes can be slid to the right or left in the plane provided that the order of pipes from left to right is maintained. Any individual pipe can be lengthened or shortened provided only that the relative height or depth of a pipe with respect to those pipes visible to it does not change. Thus in the pipe system showing the transition from trees to pipes, the upper vertex of pipe 4 can be raised or lowered provided that it does not exceed or equal the height of pipe 3. The relative heights of pipe 4 and pipe 2 are not important since intervening pipe 3 obscures pipe 2 from pipe 4. The upper vertex of pipe 2 can be raised almost up to the height of pipe 3 or lowered almost to the height of pipe 1.

A pair of matched trivial carets in a tree corresponds to a pipe that is shorter than its two neighbors on top and shorter than its two neighbors on the bottom. Such pipes may be inserted or deleted at will without changing the element of F represented by the pipe system. We call such pipes *trivial pipes*. If the pipe system has no trivial pipes, then the

system is called *reduced*.

Thus we have the following theorem.

Theorem 2.3.2. There is a one-to-one correspondence between tree pairs and pipe systems. There is also a one-to-one correspondence between reduced pairs of trees and reduced pipe systems.

Although pipe systems and tree-pairs are logically equivalent, pipe systems have three advantages: (1) It is easier to draw a complicated pipe system. (2) It is a trivial matter to multiply a pipe system by the standard generators a, $A = a^{-1}$, b, $B = b^{-1}$, or by the new generators c and $C = c^{-1}$ geometrically symmetric to b and B. (3) It is easy to preserve a history of multiplications. We will discuss these three advantages in Chapter 5.

2.4 Three ways to draw the element a

In this chapter, we have described three different ways of describing an element of the Thompson's group F. We now present those three ways again using the element a as an example. In Section 1, we discuss how the element was first realized by a graph where group multiplication was given by composition of functions. Our function G was given by

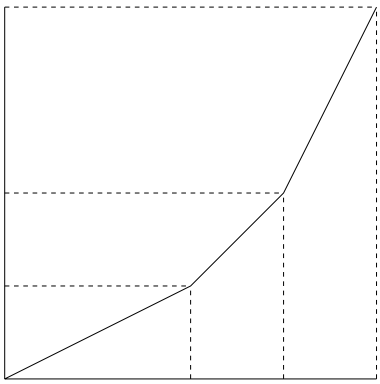


Figure 2.6: Graph for element a

. It also has a domain tree and a range tree. A detailed description can be found in figure 2.3, but now we draw the trees on top of each other and get

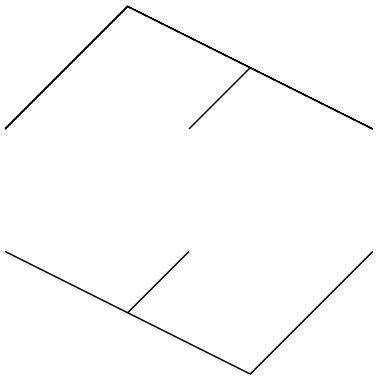


Figure 2.7: Reduced pair of trees for element a

From this figure, we can easily see how to draw the pipe system.

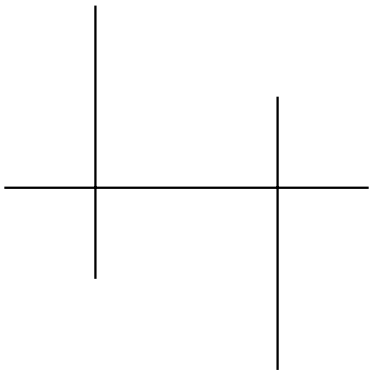


Figure 2.8: Pipe system for element a

Thus we can describe element a in F_{ab} in the three ways pictured.

3 BINARY TREES

Our attempts to calculate the growth of Thompsons group F, which are described in Chapter 3, Chapter 4, and Appendix A, are based on two facts. The first fact is a theorem of Blake Fordham, which shows that word length in F_{ab} may be calculated from its reduced-tree-pair representation by classifying each caret as one of seven types. The second fact is that, for any integer n, there is a formula calculating the number of reduced pairs of n caret trees.

In this chapter, we will describe as a grammar the set of binary trees with carets classified and labeled as by Fordham. Without the Fordham labelling, the grammar would be simple,

but the labelling is used by Fordham in calculating minimal word length and thus plays a role in calculating growth. In the first section, we will review Fordham's tree description. In Section 2, we will change the look of the tree. This change will make it easier to describe the grammar for binary trees. In Section 3, we will describe a grammar based on Fordham's tree description and will show that this correctly explains tree growth. Finally, we will look at patterns and repetitions in the grammar and simplify the grammar in preparation for the grammar of Chapter 4, which describes reduced pairs of trees, with Fordham labelling.

3.1 The Fordham tree

Recall that a rooted, binary tree is a planar tree formed from a finite collection of carets and that these carets have a natural order from left to right. The vertices of a caret consist of one parent and two children. A caret attached to one of the children is called a child of that caret.

A caret is called a *left caret* if it has an edge on the left edge of the tree. It is called a *right caret* if it is not a left caret and if it has an edge on the right edge of the tree. It is called an *interior caret* if neither of its two edges lies on the left or right edges of the tree. A caret is *exposed* or *trivial* if it has no children.

A *Fordham tree* is a rooted, binary tree, each of whose carets has been assigned a label between 1 and 7 in the following way:

1: The caret is the left-most caret in the left-right ordering.

2: The caret is a left caret but is not the left-most caret.

3: The caret is a right caret that is followed in the left-right order only by right carets.

4: The caret is a right caret one of whose successors in the left-right order is an interior caret but whose immediate successor is a right caret.

5: The caret is a right caret whose immediate successor in the left-right order is an interior caret.

6: The caret is an interior caret with no right child.

7: The caret is an interior caret with a right child.

If a caret is trivial, we append the letter t to its numerical label. The following figure gives a tree with its carets labelled.

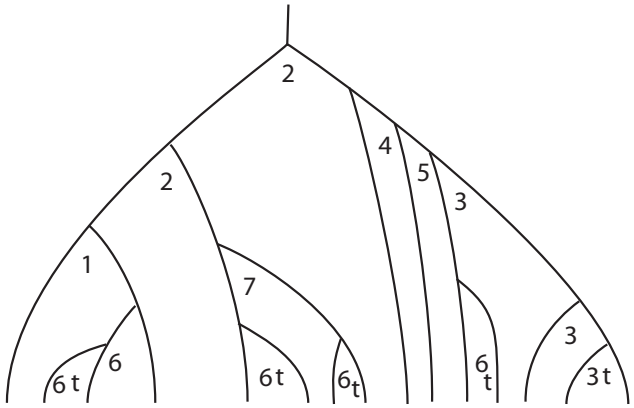


Figure 3.1: Fordham tree
The carets, in left-to-right order therefore have labels $1, 6t, 6, 2, 6t, 7, 6t, 2, 4, 5, 6t, 3, 3, 3t$.

Given a reduced tree pair with carets labeled in both the domain tree and the range tree, the carets, and hence also labels, come in matched pairs. Each pair of labels contributes to the geodesic length of the element as given by Fordhams table in the following theorem.

Theorem 3.1.1. The geodesic length of an element g of group F may be calculated from the reduced tree-pair representative of g as follows. For each matched and labeled pair of carets, look up the label pair in the following table. The sum of the entries obtained is the geodesic length of the element g.

	2	3	4	5	6	7
2	2	1	1	1	2	2
3	1	0	2	2	1	3
4	1	2	2	2	1	3
5	1	2	2	2	3	3
6	2	1	1	3	2	4
7	2	3	3	3	4	4

Table 3.1: Lengths from Ordered Pairs

Finding the geodesic length is simple. We start with a reduced pair of trees and label all the carets with the Fordham labels. We put carets into domain-range pairs by means of the left to right ordering. Each pair of carets is then represented by an ordered pair of labels. Each pair of carets can be represented by an ordered pair. Table 3.1 gives us the first number in the ordered pair on the left side of the table and the second number across the top. Each ordered pair outputs a number. When added, these numbers tell us the geodesic length of the element.

3.2 New tree drawing

It would be nice to reduce the number of types of subtrees. Carets of types 2 and 7 behave much alike when they have an interior right child. This similarity will allow a simplification of the grammar. It is helpful to indicate such a caret by bending its right edge in the following manner (Figure 3.2):

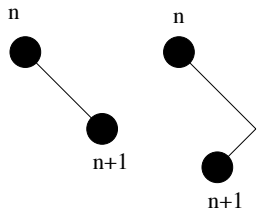


Figure 3.2: Tree grammar

Note that there is an obvious bijection between the set of trees whose edges are not bent and the set of those whose edges are bent. Here is an example of a tree and its bent partner.

This bending makes every internal subtree look like things are only attached on the left, making it easier to build the tree left to right

3.3 Simple grammar of the Fordham tree description

Now that we have our Fordham tree description and our new tree with bent edges, we are ready to build our grammar. Building a grammar for a normal rooted binary tree is actually

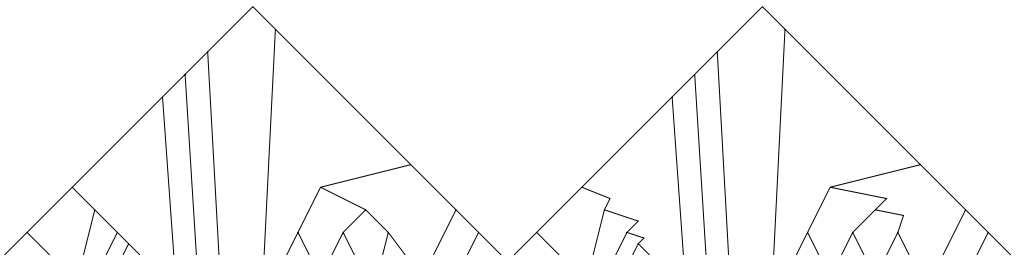


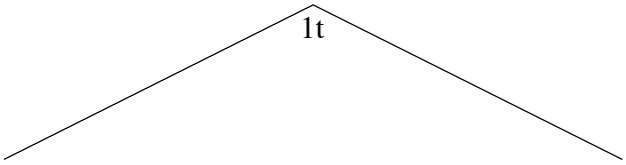
Figure 3.3: Two-tree example
Note these are the same tree. On the left is a normal binary tree. On the right is the special new tree. Note how every interior tree looks like a left tree because it has the funny forward back line.

very simple (build left trees from bottom up and build right trees from top down), but we want to add to the tree description a Fordham-type labelling to help us classify the geodesic length of the elements of group F. We build this grammar from left to right, starting from the bottom-left, going up, and inserting all subtrees as needed. Then we build the right tree from the top down. This means that any left child will receive a label before the right parent does. The label of each caret or node or edge in the tree indicates the actions that are to follow. We explain how the production rules of the grammar are connected with the geometry of the tree in the following pages. A tree construction ends when all labels are caret labels and are of the form n or nt, where n is one of the integers 1, 2, 3, 4, 5, 6, or 7.

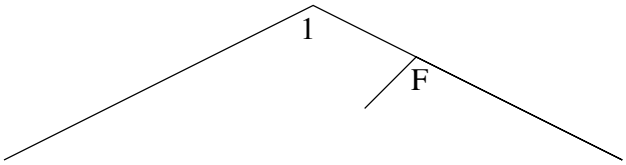
In the following simple grammar, we use the numerical assignments described earlier in the chapter along with letters that tell us how to build our tree. Once the simple grammar is explained, we will describe a more complex grammar based on the patterns that appear in building trees.

Five production rules or actions are possible after start. Here they are:

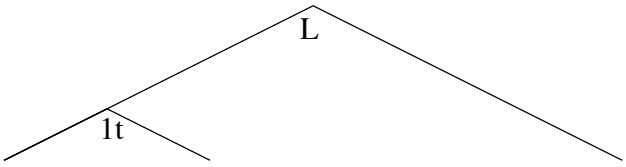
- Symbol ϵ indicates that the tree is complete and is empty.
- Symbol 1t indicates that the tree is a one caret tree, is complete, and that the caret is trivial.



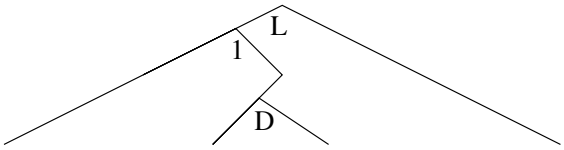
- Symbol 1 F tells us to draw one left caret that will be the left-most in the left-right ordering and to draw a right caret based on the directions assigned to F, explained below. Symbol F indicates that the left edge of the tree is (F)inished, with one caret, and that we are ready to build the right side of the tree.



- Symbol 1t L tells us to draw one left trivial caret and another left caret based on the directions assigned to L, explained below. The symbol t indicates that no subtree is to be attached to the caret 1t so that this caret is trivial, and the L indicates that the the left edge of the tree above caret 1t is not yet complete.

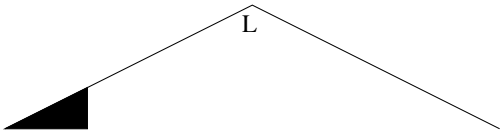


- Symbol 1 D L tells us to draw one left caret that is the leftmost caret and to label it 1. We also draw two more carets, one subtree based on the directions assigned to D

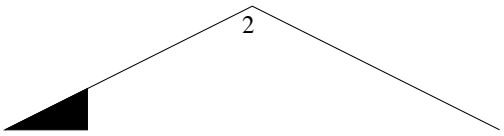


explained below and one left caret based on the directions assigned to L. The symbol D indicates that a tree is growing (D)ownward at that point.

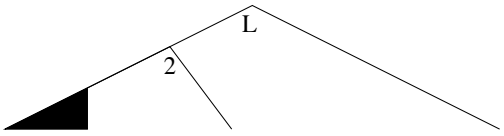
As noted above, an L indicates that the left edge of the tree is not yet complete and requires the insertion of another left caret above those already created. This can be done in four different ways (four production rules):



- Symbol 2 tells us to add one left caret above those already existing and to label it with a 2. This symbol indicates that the left edge of the tree is complete and that no carets will be added to the right. This caret will be the last caret in the left-to-right ordering.

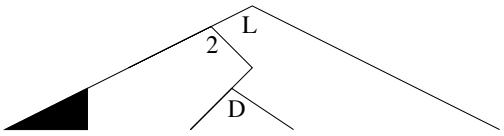


- Symbol 2 L tells us tells us to draw one left caret that is not the leftmost caret with the label 2 and one left caret based on the directions assigned to L. The L indicates that the left edge of the tree is still not complete.

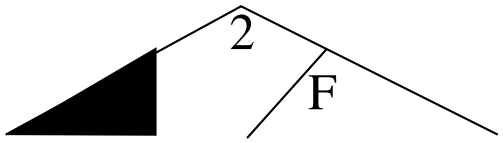


- Symbol 2 D L tells us to draw one left caret that is not the leftmost caret and to label it 2. We also draw two more carets, one subtree based on the directions assigned to D

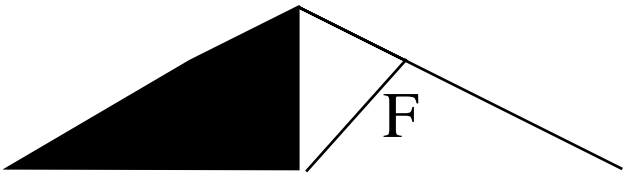
explained below and one left caret based on the directions assigned to L. The symbol D indicates that a tree is growing (D)ownward at that point.



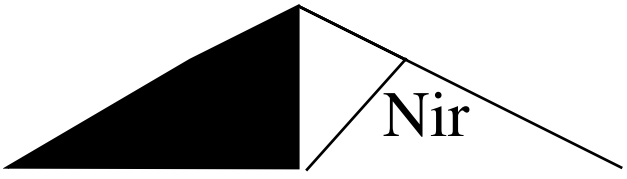
- Symbol 2 F tells us tells us to draw one left caret that is not the leftmost caret with the label 2 and one right caret based on the directions assigned to F, explained below. The symbol F indicates that the left edge of the graph is (F)inished.



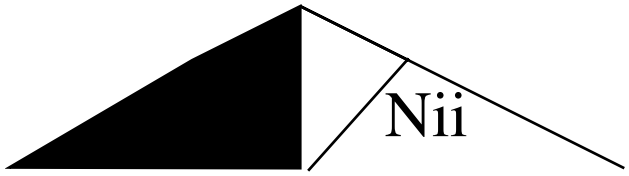
The symbol F indicates that the left edge of the tree is (F)inished and that we are ready to build the right edge. There are three possible productions following from the symbol F. All of them change the labelling on the caret labeled F.



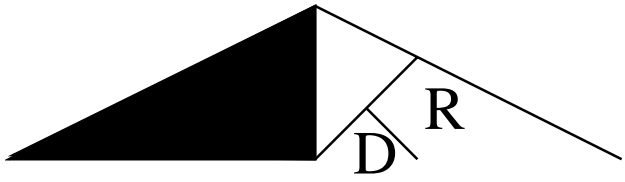
- Symbol Nir replaces label F by symbol Nir. One is then to apply the production rules assigned to symbol Nir. This symbol Nir indicates that all descendants of this caret will be on the right edge of the tree.



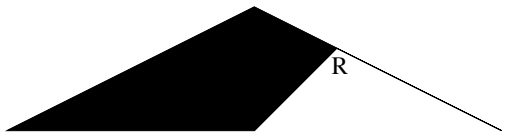
- Symbol Nii replaces label F by symbol Nii. One then applies the production rules assigned to Nii. The symbol Nii indicates that this caret will be followed at some point in the left-to-right order by an interior caret.



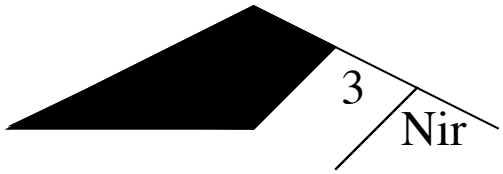
- Symbol D R replaces the label F by the label R and adds a caret labeled D as left child of R. The symbol D indicates that a tree is growing (D)ownward at that point. The symbol R postpones a final labelling of the original caret, but it indicates that a (possibly empty) tree is to grow from its right child.



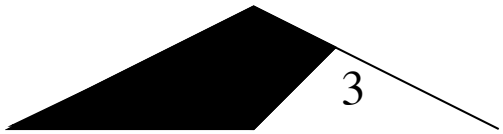
The symbol R on a caret appears just to the right after an interior subtree. This means that this right caret has a left child. There are four actions that we can apply following R.



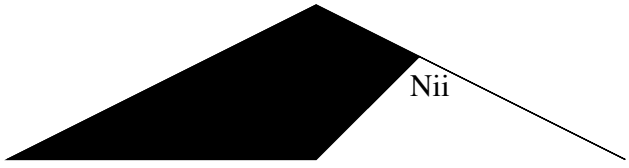
- Symbol 3 tells us to change the symbol R to the symbol 3 . This caret becomes the last caret in the left-to-right ordering. This caret labeled 3 is not a trivial caret since R had a left child.



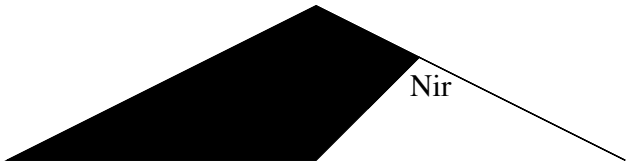
- The label 3 tells us to draw one right caret.



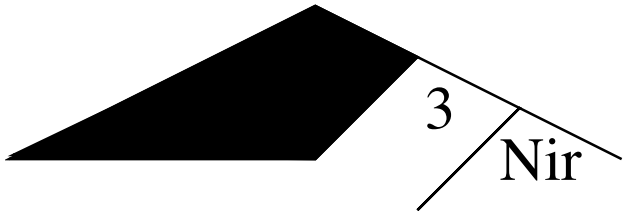
- Symbol Nii tells us to replace the symbol R with the symbol Nii. The symbol Nii indicates that this caret will be followed at some point in the left-to-right order by an interior caret.



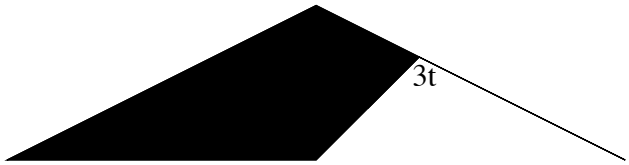
The symbol Nir on a caret indicates that no interior caret will follow in the left-to-right order. There are two possible actions that we can apply following Nir.



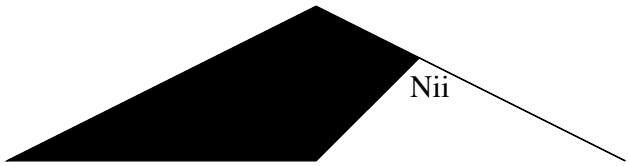
- Symbol 3 Nir changes the label of the caret to 3 and adds a new caret as right child with label Nir.



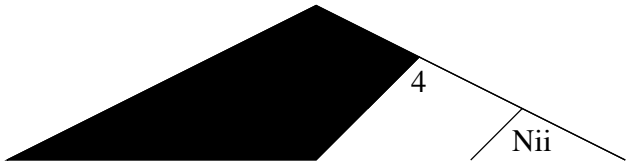
- Symbol 3t changes the label from Nir to 3t. This caret is trivial and is the last caret in the left-to-right ordering.



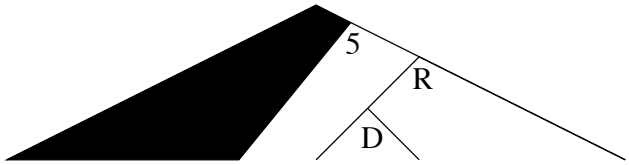
The label Nii on a caret indicates that this caret will be followed at some point in the left-to-right order by an interior caret. There are two actions that we can apply following Nii.



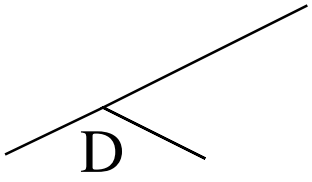
- Symbol 4 Nii changes the Nii label to a 4 and adds a right child with label Nii.



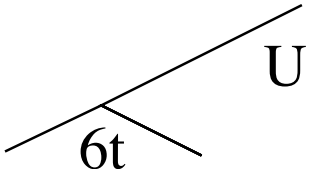
- Symbol 5 D R changes the Nii label to a 5, adds a right child labeled R and a left child to that right child with label D.



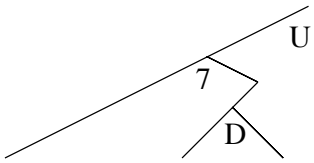
Label D on a caret indicates that an interior subtree is being built (D)ownward, beginning at that caret. The left edge of such a subtree is built in the same manner as the left edge of the global tree, namely from the bottom up. There are two possible actions.



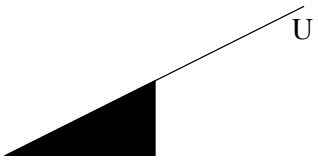
- Symbol 6t U places the symbol U on the edge just above the caret and replaces the label D on the caret with the symbol 6t. The symbol U indicates the possibility of inserting vertices on that edge to allow further branching to the right. The symbol U acts for that edge much as the symbol L acted for the left edge of the total tree.



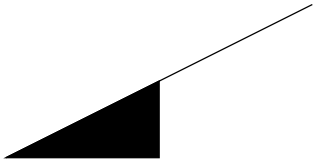
- Symbol 7 D U places the symbol U on the edge just above the caret, replaces the label D on the caret with the symbol 7, and adds a new caret as right child with label D. The right edge of caret 7 is bent as in the picture.



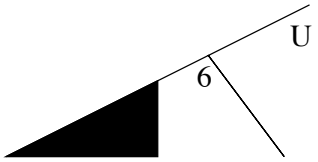
Label U on an edge indicates the possibility of introducing vertices along that edge as parents of carets to allow further branching to the right. The action is much like that of an L on the left edge of the total tree. There are three possible actions.



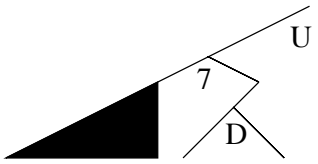
- Symbol ϵ simply removes the letter U from the edge without introducing any new carets.



- Symbol 6 U introduces one new caret branching from the edge, labels that caret 6, and leaves symbol U on the edge just above the new caret.



- Symbol 7 D U introduces one new caret labeled 7 branching from the edge, leaves the symbol U on the edge above this new caret, and adds a caret labeled D as right child of the new caret 7.



3.4 Example of the simple grammar in action

Now that we have described the grammar, we wish to demonstrate how to build a tree. The first the tree for which the grammar is being built is shown in the following figure.

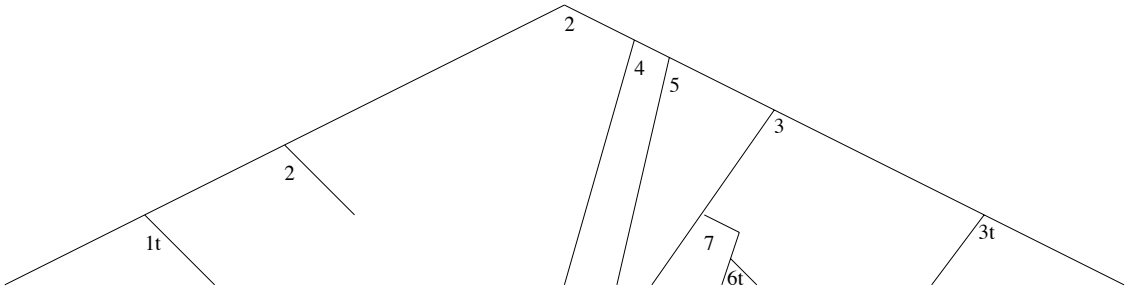


Figure 3.4: End tree built by grammar

All trees begin with Start. Since the leftmost caret is of type 1t we choose the option 1t L and begin drawing our tree.

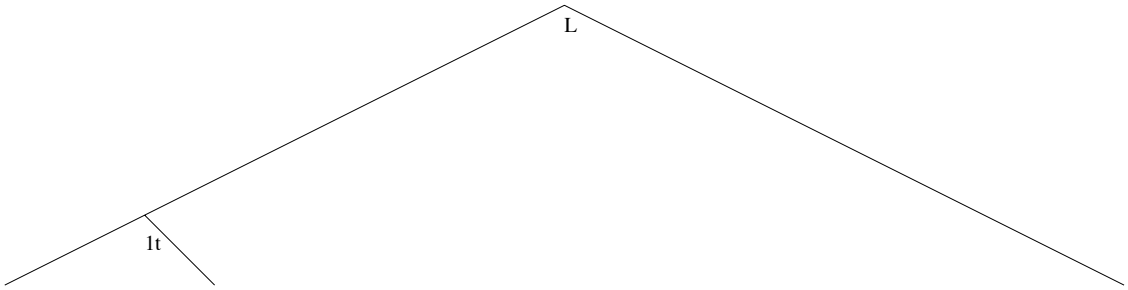


Figure 3.5: Second stage of grammar

Now we look at our 2nd leftmost caret and find that it is of type 2 and that it has no internal subtree and has a parent. We replace the L with a 2L as shown below.

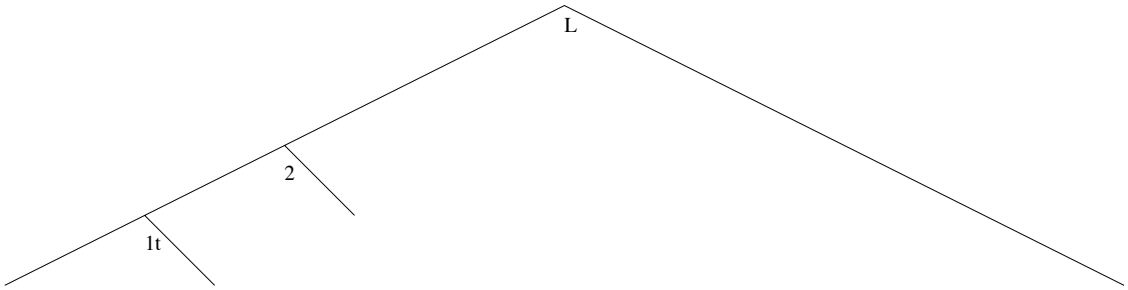


Figure 3.6: Third stage of grammar

Now we look at our 3rd leftmost caret and find that it is of type 2 and that it is the root vertex with carets to the right so we replace the L with a 2F as shown.

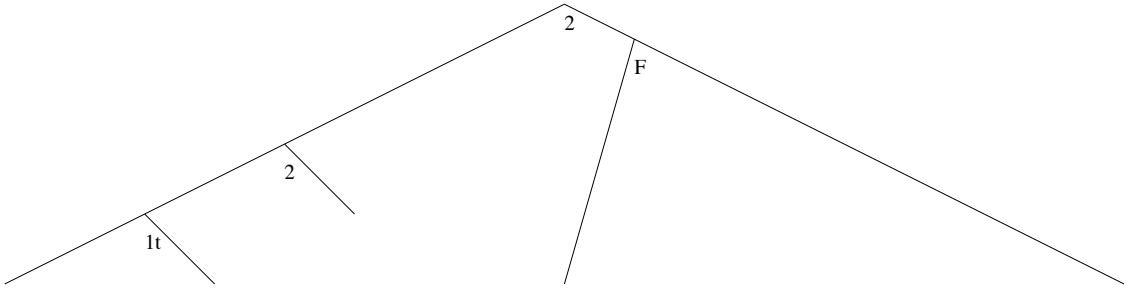


Figure 3.7: Fourth stage of grammar

Now we look at our 4th leftmost caret and find that it is of type 4, so we replace the F with a 4 Nii as shown below.

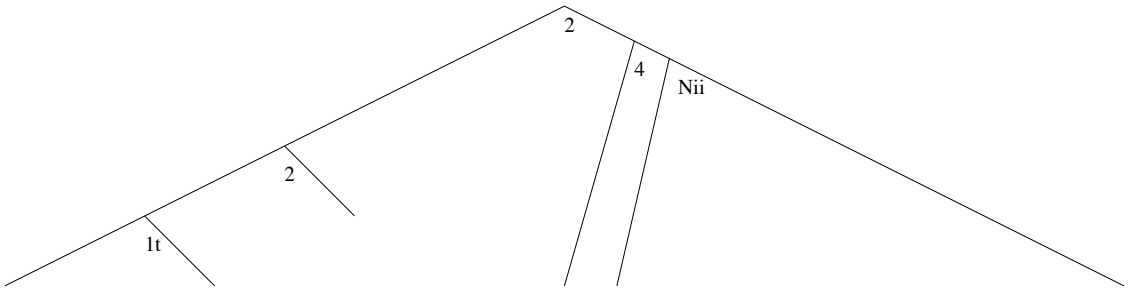


Figure 3.8: Fifth stage of grammar

Now we look at our 5th leftmost caret and find that it is of type 5, so we replace the Nii with a 5 D R as shown below.

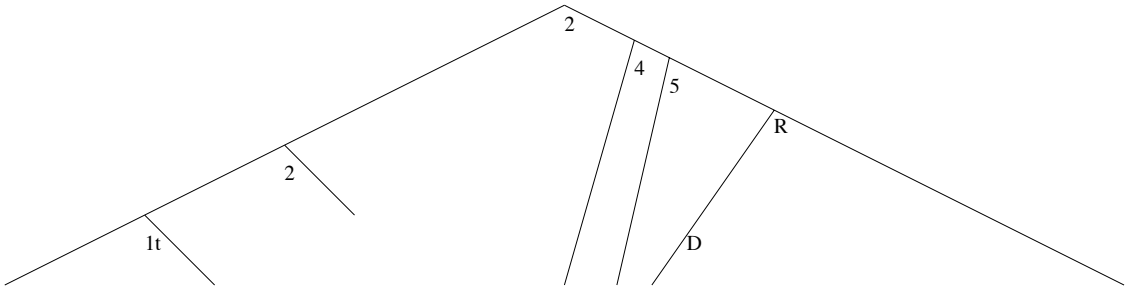


Figure 3.9: Sixth stage of grammar

Now we look at our 6th leftmost caret and find that it is of type 7, so we replace the D with a 7 D U as shown below.

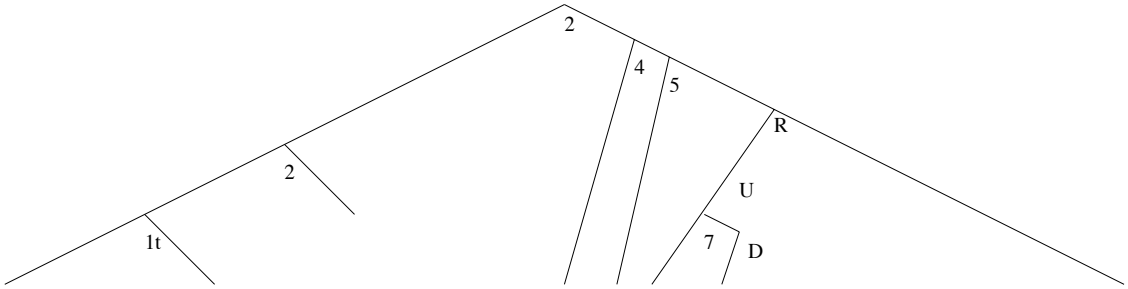


Figure 3.10: Seventh stage of grammar

Now we look at our 7th left most caret and find that it is of type 6t, so we replace the D with a 6t U as shown below.

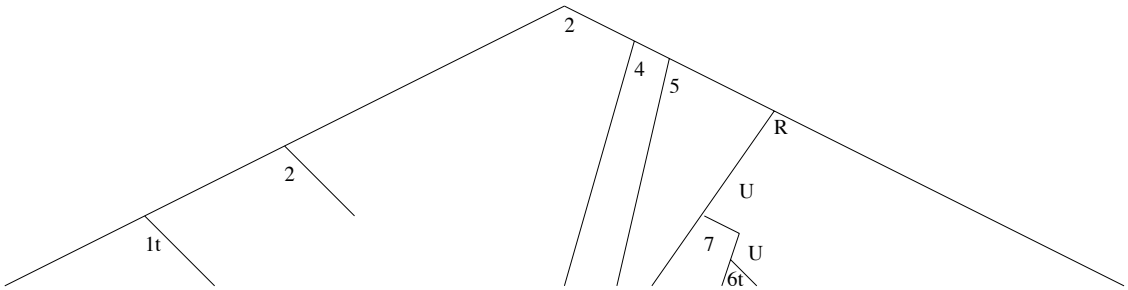


Figure 3.11: Eighth stage of grammar

Note that where the two Us are, there are no carets in the final tree. Thus we replace both Us with ϵs, which means that we erase both Us. We then look at the actions associated with R.

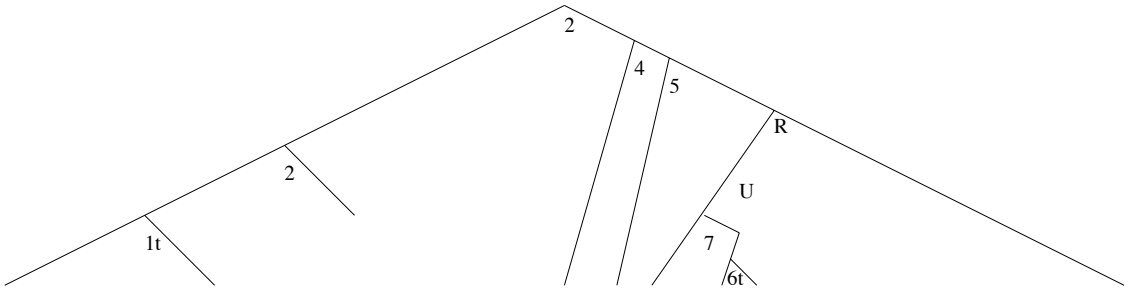


Figure 3.12: Ninth stage of grammar

Now we look at our 8th leftmost caret and find that it is of type 3, so we replace the R with a 3 Nir as shown below.

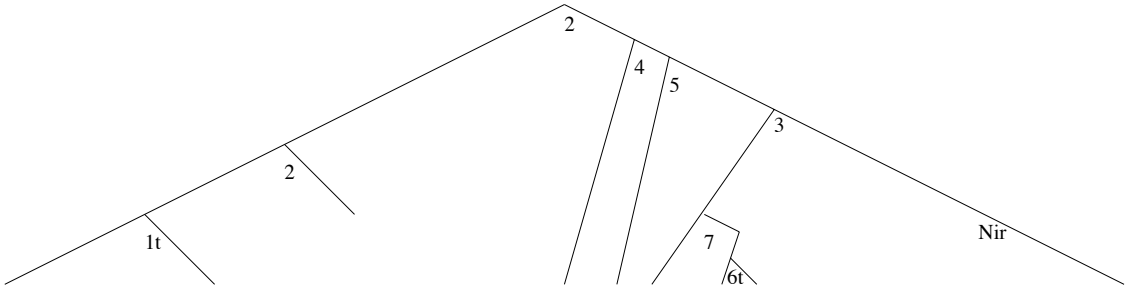


Figure 3.13: Tenth stage of grammar

Now we look at our 9th leftmost and final caret and find that it is of type 3t so we replace the Nir with a 3t as shown below.

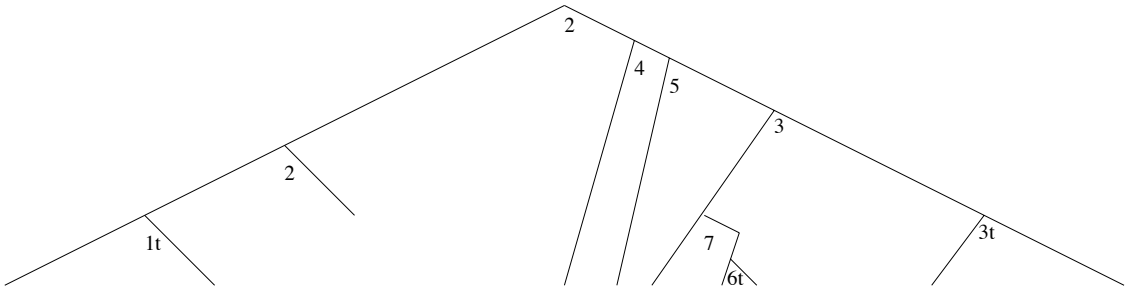


Figure 3.14: End tree built by grammar

Note that we have the tree we started with.

3.5 Changes

We have discussed all of these actions/production rules in a previous section. Here is a shortened version of the grammar. We list all of the actions on the left followed by a right arrow that points to the following productions rules separated by a vertical line shown below.

- $Start \rightarrow \epsilon \mid 1\ F \mid 1t\ L \mid 1\ D\ L|1t$
- $L \rightarrow 2\ D\ L \mid 2\ L \mid 2\ F|2$
- $F \rightarrow\ Nir \mid Nii \mid DR$
- $Nir \rightarrow 3\ Nir \mid 3t$

- $Nii \rightarrow 4\ Nii \mid 5\ DR$
- $R \rightarrow 3\ Nir \mid 3\ \mid Nii$
- $D \rightarrow\ 6t\ U \mid 7\ D\ U$
- $U \rightarrow\ \epsilon \mid 6\ U \mid 7\ D\ U$

In the previous section we demonstrated how to use the grammar to draw a tree. We were able to draw and label the carets in a left to right manner. This method can be done for any Fordham tree. Note that this grammar is great when describing 1 tree, but we want to be able to describe reduced pair of trees. This grammar is a context-free grammar which means that it can be represented by a push down automaton, or 1 stack. We have just seen that this context free grammar constructs all Fordham trees and that the carets of those trees can be constructed, when desired, in left-to-right order. When that order is followed, successive strings in the production have an interesting property: Each such string alternates between strings on the terminal characters (1, 1t, 2, 3, 3t, 4, 5, 6, 6t, 7), which we call number strings and strings on the non-terminal characters (Start, L, F, Nir, Nii, R, D, U), which we call letter strings." The letter strings take on only finitely many (indexed) types $(Start, L, DL, UL, DR, UR, DU^mL, U^mL, DU^mR, U^mR, F, Nir, Nii, R)$, where m may be any positive integer. Hence we may write an equivalent language that uses these strings of intermediate characters as its new letters. Here is the resulting language:

$Start \rightarrow \epsilon \mid 1\ F \mid 1t\ L \mid 1\ D\ L|1t$

$L \rightarrow 2\ D\ L \mid 2\ L \mid 2\ F|2$

$DL \rightarrow\ 6t\ UL \mid 7\ D\ UL$

$UL \rightarrow\ L \mid 6\ UL \mid 7\ D\ UL$

$DR \rightarrow\ 6t\ UR \mid 7\ D\ UR$

$UR \rightarrow\ R \mid 6\ UR \mid 7\ D\ UR$

$DU^mL \rightarrow\ 6t\ U^{m+1}L \mid 7\ D\ U^{m+1}L$

$U^mL \rightarrow\ U^{m-1}L \mid 6\ U^mL \mid 7\ D\ U^mL$

$DU^mR \rightarrow$ 6t $U^{m+1}R$ | 7 D $U^{m+1}R$

$U^mR \rightarrow$ $U^{m-1}R$ | 6 U^mR | 7 D U^mR

$F \rightarrow$ Nir | Nii | DR

$Nir \rightarrow 3$ Nir | 3t

$Nii \rightarrow 4$ Nii | 5 DR

$R \rightarrow 3$ Nir | 3 | Nii

Note that m works as long as U^0 does not exist. This new grammar describes tree building so that each production rule is only replaced with one caret and one production rule. There are still 14 rules that we will reduce. Observe how if m=0 then we have

$DU^0L \rightarrow$ 6t UL | 7 D UL

$DU^0R \rightarrow$ 6t UR | 7 D UR

For later convenience, note that DL and DU^0L have the same produduction rule:

$UL \rightarrow$ U^0L | 6 UL | 7 D UL

$UR \rightarrow$ U^0R | 6 UR | 7 DUR

Note that if we let U^0L mean L and U^0R mean R, we can remove four grammar rules and modify the others appropriately to obtain our final grammar for one tree.

$Start \rightarrow \epsilon$ | 1 F | 1t U^0L | 1 D U^0L|1t

$U^0L \rightarrow 2$ D U^0L | 2 U^0L | 2 F | 2

$DU^mL \rightarrow$ 6t $U^{m+1}L$ | 7 D $U^{m+1}L$

$U^mL \rightarrow$ $U^{m-1}L$ | 6 U^mL | 7 D U^mL

$DU^mR \rightarrow$ 6t $U^{m+1}R$ | 7 D $U^{m+1}R$

$U^mR \rightarrow$ $U^{m-1}R$ | 6 U^mR | 7 D U^mR

$F \rightarrow \epsilon$ | Nir | Nii | DU^0R

$Nir \rightarrow 3$ Nir | 3t

$Nii \rightarrow 4$ Nii | 5 DU^0R

$U^0R \rightarrow 3$ Nir | 3 | Nii

4 PAIRS OF TREES

4.1 Grammar for reduced pairs of trees

We are now ready to construct the grammar for a pair of trees. We can keep track of a pair of trees with an ordered pair where the first value corresponds to the input tree and the second to the output tree. We shall now begin the grammar. Note that we never see Start again after the beginning. This means that to start drawing a pair of trees, we begin with (Start,Start), but Start can not be paired with any other production rule, so we shall call (Start,Start) just S. Recall $Start \rightarrow \epsilon \mid 1\ F \mid 1t\ U^0L \mid 1\ D\ U^0L$. The reduced pair of trees must be the same length, or have the same number of carets, so ϵ can not be paired with the other 3. Now we look at all possible combinations. $(1F, 1F), (1F, 1tU^0L), (1F, 1DU^0L)(1tU^0L, 1F)$, $(1tU^0L, 1tU^0L), (1tU^0L, 1DU^0L)(1DU^0L, 1F), (1DU^0L, 1tU^0L), (1DU^0L, 1DU^0L)$. We may only have trees of the same length. This means that when we create this grammar, numbers will show up in pairs. Instead of writing (1F,1F), we write (1,1)(F,F). (1,1) represents a label of the reduced pair of trees; (F,F) represents an action in the reduce pair of trees; the combination represents a rule in a pair of trees. The way a grammar works is this: We start with the action and replace it with one of the rules. We continue until we have no more actions left since a rule can be a label followed by an action or can be just a label. Also we can not have (1t,1t) since that will make a non-reduced pair. Therefore there are 8 rules produced by the Start action shown below.

$$S \rightarrow (1,1)\Big[(F,F) \mid (DU^0L, DU^0L) \mid (F, DU^0L) \mid (DU^0L, F)\Big] \mid$$
$$(1t,1)\Big[(U^0L, F) \mid (U^0L, DU^0L)\Big] \mid (1,1t)\Big[(F, U^0L) \mid (DU^0L, U^0L)\Big]$$

A similar argument can be made for the next 80 production rules. After Start, there are 9 more actions to take care of in a single tree. We argue that if there are a possible 9 actions on the top and 9 actions on the bottom, there are 81 possible actions. Note that Nir means that the rightmost caret will be trivial; therefore, the action (Nir, Nir) does not exist

in reduced pairs of trees. Every other combination is valid, however. Now that we know all possible actions, we look at all possible rules for each action.

The action U^0L may be paired with any action of the second tree so that gives 9 actions with U^0L in the first as follows.

- $(U^0L, U^0L) \rightarrow (2,2)|(2.,2)\Big[(U^0L, U^0L)|(DU^0L, U^0L)|(F, U^0L)|(U^0L, DU^0L)|$
  $(DU^0L, DU^0L)|(F, DU^0L)|(U^0L, F)|(DU^0L, F)|(F, F)\Big]$
- $(U^0L, DU^nL) \rightarrow (2,7)\Big[(U^0L, DU^{n+1}L) \mid (DU^0L, DU^{n+1}L) \mid (F, DU^{n+1}L)\Big] \mid$
  $(2,6t)\Big[(U^0L, U^{n+1}L) \mid (DU^0L, U^{n+1}L) \mid (F, U^{n+1}L)\Big]$
- $(U^0L, U^nL) \rightarrow 2,6)\Big[(U^0L, U^nL) \mid (DU^0L, U^nL) \mid (F, U^nL)\Big] \mid$
  $(2,7)\Big[(U^0L, DU^nL) \mid (DU^0L, DU^nL) \mid (F, DU^nL)\Big] \mid (U^0L, U^{n-1}L)$
- $(U^0L, DU^nR) \rightarrow (22,7)\Big[(U^0L, DU^{n+1}R) \mid (DU^0L, DU^{n+1}R) \mid (F, DU^{n+1}R)\Big] \mid$
  $(2,6t)\Big[(U^0L, U^{n+1}R) \mid (DU^0L, U^{n+1}R) \mid (F, U^{n+1}R)\Big]$
- $(U^0L, U^nR) \rightarrow (2,6)\Big[(U^0L, U^nR) \mid (DU^0L, U^nR) \mid (F, U^nR)\Big] \mid$
  $(2,7)\Big[(U^0L, DU^nR) \mid (DU^0L, DU^nR) \mid (F, DU^nR)\Big] \mid (U^0L, U^{n-1}R)$
- $(U^0L, F) \rightarrow (U^0L, DU^0R) \mid (U^0L, Nir) \mid (U^0L, Nii)$
  $7\ (U^0L, Nir) \rightarrow (2,3t) \mid (2,3)\Big[(U^0L, Nir) \mid (DU^0L, Nir) \mid ((F, Nir)\Big]$
- $(U^0L, Nii) \rightarrow (2,4)\Big[(U^0L, Nii) \mid (DU^0L, Nii)|(F, Nii)\Big] \mid (2,5)\Big[(U^0L, DU^0R) \mid$
  $(DU^0L, DU^0R) \mid (F, DU^0R)\big]$
- $(U^0L, U^0R) \rightarrow (2,3)|(2,3)[(U^0L, Nir)|(DU^0L, Nir)|(F, Nir)]|(U^0L, Nii)$

The action DU^kL may be paired with any action of the second tree so that gives 9 actions with DU^KL in the first as follows.

- $(DU^kL, U^0L) \rightarrow 6t, 2)\Big[(U^{k+1}L, U^0L) \mid (U^{k+1}L, DU^0L) \mid (U^{k+1}L, F)\Big] \mid$
  $(7, 2)\Big[(DU^{k+1}L, U^0L) \mid (DU^{k+1}L, DU^0L) \mid (DU^{k+1}L, F)\Big]$

- $(DU^kL, DU^nL) \rightarrow (6t, 7)(U^{k+1}L, DU^{n+1}L) \mid (7, 7)(DU^{k+1}L, DU^{n+1}L) \mid$
  $(7, 6t)(DU^{k+1}L, U^{n+1}L)$

- $(DU^kL, U^nL) \rightarrow (6t, 6)(U^{k+1}L, U^nL) \mid (7, 6)(DU^{k+1}L, U^nL) \mid (6t, 7)(U^{k+1}L, DU^nL) \mid$
  $(7, 7)(DU^{k+1}L, DU^nL) \mid (DU^kL, U^{n-1}L)$

- $(DU^kL, DU^nR) \rightarrow (6t, 7)(U^{k+1}L, DU^{n+1}R) \mid (7, 7)(DU^{k+1}L, DU^{n+1}R) \mid$
  $(7, 6t)(DU^{k+1}L, U^{n+1}R)$

- $(DU^kL, U^nR) \rightarrow (6t, 6)(U^{k+1}L, U^nR) \mid (7, 6)(DU^{k+1}L, U^nR) \mid (6t, 7)(U^{k+1}L, DU^nR) \mid$
  $(7, 7)(DU^{k+1}L, DU^nR) \mid (DU^kL, U^{n-1}R)$

- $(DU^kL, F) \rightarrow (DU^kL, DU^0R) \mid (DU^kL, Nir) \mid (DU^kL, Nii)$

- $(DU^kL, Nir) \rightarrow (6t, 3)(U^{k+1}L, Nir) \mid (7, 3)(DU^{k+1}L, Nir)$

- $(DU^kL, Nii) \rightarrow (6t, 4)(U^{k+1}L, Nii) \mid (7, 4)(DU^{k+1}L, Nii) \mid (6t, 5)(U^{k+1}L, DU^0R) \mid$
  $(7, 5)(DU^{k+1}L, DU^0R)$

- $(DU^kL, U^0R) \rightarrow (DU^kL, Nii) \mid (DU^kL, Nir)$

The action U^kL may be paired with any action of the second tree so that gives 9 actions with U^kL in the first as follows.

- $(U^kL, U^0L) \rightarrow (6, 2)\Big[(U^kL, U^0L) \mid (U^kL, DU^0L) \mid (U^kL, F)\Big] \mid$
  $(7, 2)\Big[(DU^kL, U^0L) \mid (DU^kL, DU^0L) \mid (DU^kL, F)\Big] \mid (U^{k-1}L, U^0L)$

- $(U^kL, DU^nL) \rightarrow (6, 7)(U^kL, DU^{n+1}L) \mid (7, 7)(DU^kL, DU^{n+1}L) \mid (6, 6t)(U^kL, U^{n+1}L) \mid$
  $(7, 6t)(DU^kL, U^{n+1}L) \mid (U^{k-1}L, DU^nL)$

- $(U^kL, U^nL) \rightarrow (6,7)(U^iL, DU^jL) \mid (7,7)(DU^iL, DU^iL) \mid (6,6t)(U^iL, U^jL) \mid$
  $(7,6t)(DU^kL, U^jL) \mid (6,2)\Big[(U^iL, U^0L) \mid (U^iL, DU^0L) \mid (U^iL, F)\Big] \mid$
  $(7,2)\Big[(DU^iL, U^0L) \mid (DU^iL, DU^0L) \mid (DU^iL, F)\Big] \mid$
  $(2,6)\Big[(U^0L, U^jL) \mid (DU^0L, U^jL) \mid (F, U^lL)\Big] \mid$
  $(2,7)\Big[(U^0L, DU^lL) \mid (DU^0L, DU^jL) \mid (F, DU^jL)\Big] \mid (U^0L, U^0L)$
  For $i \in \{1,2,...,k\}$ and $j \in \{1,2,...n\}$

- $(U^kL, DU^nR) \rightarrow (6,7)(U^kL, DU^{n+1}R) \mid (7,7)(DU^kL, DU^{n+1}R) \mid (6,6t)(U^kL, U^{n+1}R) \mid$
  $(7,6t)(DU^kL, U^{n+1}R) \mid (U^{k-1}L, DU^nR)$

- $(U^kL, U^nR) \rightarrow (6,7)(U^iL, DU^jR) \mid (7,7)(DU^iL, DU^iR) \mid (6,6t)(U^iL, U^jR) \mid$
  $(7,6t)(DU^iL, U^jR) \mid (6,3)(U^iL, Nir) \mid (7,3)(DU^iL, Nir) \mid (6,4)(U^iL, Nii) \mid$
  $(7,4)(DU^iL, Nii) \mid (6,5)(U^iL, DU^0R) \mid (7,5)(DU^iL, DU^0R) \mid$
  $(2,6)\Big[(U^0L, U^nR) \mid (DU^0L, U^nR) \mid (F, U^nR)\Big] \mid$
  $(2,7)\Big[(U^0L, DU^nR) \mid (DU^0L, DU^nR) \mid (F, DU^nR)\Big] \mid (U^0L, U^0R)$
  For $i \in \{1,2,...,k\}$ and $j \in \{1,2,...n\}$

- $(U^kL, F) \rightarrow (U^kL, DU^0R) \mid (U^kL, Nir) \mid (U^kL, Nii)$

- $(U^kL, Nir) \rightarrow (6,3)(U^kL, Nir) \mid (7,3)\ (DU^kL, Nir) \mid (U^{k-1}L, Nir)$

- $(U^kL, Nii) \rightarrow (6,4)(U^kL, Nii) \mid (7,4)(DU^kL, Nii) \mid (6,5)(U^kL, DU^0R) \mid$
  $(7,5)(DU^kL, DU^0R) \mid (U^{k-1}L, Nii)$

- $(U^kL, U^0R) \rightarrow (U^{k-1}L, U^0R) \mid (6,3)(U^kL, Nir) \mid (7,3)(DU^kL, Nir) \mid (6,4)(U^kL, Nii) \mid$
  $(7,4)(DU^kL, Nii) \mid (6,5)(U^kL, DU^0R) \mid (7,5)(DU^kL, DU^0R)$

The action DU^kR may be paired with any action of the second tree so that gives 9 actions with DU^kR in the first as follows.

- $(DU^kR, U^0L) \rightarrow (6t, 2)\Big[(U^{k+1}R, U^0L) \mid (U^{k+1}R, DU^0L) \mid (U^{k+1}R, F)\Big] \mid$
  $(7, 2)\Big[(DU^{k+1}R, U^0L) \mid (DU^{k+1}R, DU^0L) \mid (DU^{k+1}R, F)\Big]$

- $(DU^kR, DU^nL) \rightarrow (6t, 7)(U^{k+1}R, DU^{n+1}L) \mid (7, 7)(DU^{k+1}R, DU^{n+1}L) \mid$
  $(7, 6t)(DU^{k+1}R, U^{n+1}L)$

- $(DU^kR, U^nL) \rightarrow (6t, 6)(U^{k+1}R, U^nL) \mid (7, 6)(DU^{k+1}R, U^nL) \mid (6t, 7)(U^{k+1}R, DU^nL) \mid$
  $(7, 7)(DU^{k+1}R, DU^nL) \mid (DU^kR, U^{n-1}L)$

- $(DU^kR, DU^nR) \rightarrow (6t, 7)(U^{k+1}R, DU^{n+1}R) \mid (7, 7)(DU^{k+1}R, DU^{n+1}R) \mid$
  $(7, 6t)(DU^{k+1}R, U^{n+1}R)$

- $(DU^kR, U^nR) \rightarrow (6t, 6)(U^{k+1}R, U^nR) \mid (7, 6)(DU^{k+1}R, U^nR) \mid (6t, 7)(U^{k+1}R, DU^nR) \mid$
  $(7, 7)(DU^{k+1}R, DU^nR) \mid (DU^kR, U^{n-1}R)$

- $(DU^kR, F) \rightarrow (DU^kR, DU^0R) \mid (DU^kR, Nir) \mid (DU^kR, Nii)$

- $(DU^kR, Nir) \rightarrow (6t, 3)(U^{k+1}R, Nir) \mid (7, 3)(DU^{k+1}R, Nir)$

- $(DU^kR, Nii) \rightarrow (6t, 4)(U^{k+1}R, Nii) \mid (7, 4)(DU^{k+1}R, Nii) \mid (6t, 5)(U^{k+1}R, DU^0R) \mid$
  $(7, 5)(DU^{k+1}R, DU^0R)$

- $(DU^kR, U^0R) \rightarrow (DU^kR, Nii) \mid (DU^kR, Nir)$

The action U^kR may be paired with any action of the second tree so that gives 9 actions with U^kR in the first as follows.

- $(U^kR, U^0L) \rightarrow (6, 2)\Big[(U^kR, U^0L) \mid (U^kR, DU^0L) \mid (U^kR, F)\Big] \mid$
  $(7, 2)\Big[(DU^kR, U^0L) \mid (DU^kR, DU^0L) \mid (DU^kR, F)\Big] \mid (U^{k0-01}R, U^0L)$

- $(U^kR, DU^nL) \rightarrow (6, 7)(U^kR, DU^{n+1}L) \mid (7, 7)(DU^kR, DU^{n+1}L) \mid (6, 6t)(U^kR, U^{n+1}L) \mid$
  $(7, 6t)(DU^kR, U^{n+1}L) \mid (U^{k-1}R, DU^nL)$

- $(U^kR, U^nL) \rightarrow (6,7)(U^iR, DU^jL) \mid (7,7)(DU^iR, DU^jL) \mid (6,6t)(U^iR, U^jL) \mid$
  $(7,6t)(DU^iR, U^jL) \mid (6,2)\Big[(U^iR, U^0L) \mid (U^iR, DU^0L) \mid (U^iR, F)\Big] \mid$
  $(7,2)\Big[(DU^iR, U^0L) \mid (DU^iR, DU^0L) \mid (DU^iR, F)\Big] \mid (U^0R, U^0L) \mid$
  $(3,6)(Nir, U^jL) \mid (3,7)(Nir, DU^jL) \mid (4,6)(Nii, U^jL) \mid$
  $(4,7)(Nii, DU^jL) \mid (5,6)(DU^0R, U^jL) \mid (5,7)(DU^0R, DU^jL)$
  For $i \in \{1,2,...,k\}$ and $j \in \{1,2,...n\}$

- $(U^kR, DU^nR) \rightarrow (6,7)(U^kR, DU^{n+1}R) \mid (7,7)(DU^kR, DU^{n+1}R) \mid$
  $(6,6t)(U^kR, U^{n+1}R) \mid (7,6t)(DU^kR, U^{n+1}R) \mid (U^{k-1}R, DU^nR)$

- $(U^kR, U^nR) \rightarrow (6,7)(U^iR, DU^jR) \mid (7,7)(DU^iR, DU^jR) \mid (6,6t)(U^kR, U^{n+1}R) \mid$
  $(7,6t)(DU^iR, U^jR) \mid (6,3)(U^iR, Nir) \mid (7,3)(DU^iR, Nir) \mid (6,4)(U^iR, Nii) \mid$
  $(7,4)(DU^iR, Nii) \mid (6,5)(U^iR, DU^0R) \mid (7,5)(DU^kR, DU^0R) \mid (U^0R, U^0R) \mid$
  $(3,6)(Nir, U^jR) \mid (3,7)(Nir, DU^jR) \mid (4,6)(Nii, U^jR) \mid$
  $(4,7)(Nii, DU^jR) \mid (5,6)(DU^0R, U^jR) \mid (5,7)(DU^0R, DU^jR)$
  For $i \in \{1,2,...,k\}$ and $j \in \{1,2,...n\}$

- $(U^kR, F) \rightarrow (U^kR, DU^0R) \mid (U^kR, Nir) \mid (U^kR, Nii)$

- $(U^kR, Nir) \rightarrow (6,3)(U^kR, Nir) \mid (7,3)(DU^kR, Nir) \mid (U^{k-1}R, Nir)$

- $(U^kR, Nii) \rightarrow (6,4)(U^kR, Nii) \mid (7,4)(DU^kR, Nii) \mid (6,5)(U^kR, DU^0R) \mid$
  $(7,5)(DU^kR, DU^0R) \mid (U^{k-1}R, Nii)$

- $(U^kR, U^0R) \rightarrow (U^{k-1}R, U^0R) \mid (6,3)(U^kR, Nir) \mid (7,3)(DU^kR, Nir) \mid (6,4)(U^kR, Nii) \mid$
  $(7,4)(DU^kR, Nii) \mid (6,5)(U^kR, DU^0R) \mid (7,5)(DU^kR, DU^0R)$

The action F may be paired with any action of the second tree so that gives 9 actions with F in the first as follows.

- $(F, U^0L) \rightarrow (DU^0R, U^0L) \mid (Nir, U^0L) \mid (Nii, U^0L)$

- $(F, DU^nL) \rightarrow (DU^0R, DU^nL) \mid (Nir, DU^nL) \mid (Nii, DU^nL)$

- $(F, U^nL) \to (DU^0R, U^nL) \mid (Nir, U^nL) \mid (Nii, U^nL)$
- $(F, DU^nR) \to (DU^0R, DU^nR) \mid (Nir, DU^nR) \mid (Nii, DU^nR)$
- $(F, U^nR) \to (DU^0R, U^nR) \mid (Nir, U^nR) \mid (Nii, U^nR)$
- $(F, F) \to (DU^0R, F) \mid (Nir, F) \mid (Nii, F)$
- $(F, Nir) \to (DU^0R, Nir) \mid (Nii, Nir)$
- $(F, Nii) \to (DU^0R, Nii) \mid (Nir, Nii) \mid (Nii, Nii)$
- $(F, U^0R) \to (DU^0R, U^0R) \mid (Nir, U^0R) \mid (Nii, U^0R)$

The action Nir may be paired with any action of the second tree so that gives 8 actions with Nir in the first since Nir will mean that the right most carrot will be trivial therefore the action (Nir, Nir) is impossible for reduced pairs of trees but every other combination is valid.

- $(Nir, U^0L) \to (3t, 2)|(3, 2)\Big[(Nir, U^0L) \mid (Nir, DU^0L) \mid (Nir, F)\Big]$
- $(Nir, DU^nL) \to (3, 7)(Nir, DU^{n+1}L) \mid (3, 6t)(Nir, U^{n+1}L)$
- $(Nir, U^nL) \to (3, 7)(Nir, DU^nL) \mid (3, 6)(Nir, U^nL) \mid (Nir, U^{n-1}L)$
- $(Nir, DU^nR) \to (3, 7)(Nir, DU^{n+1}R) \mid (3, 6t)(Nir, U^{n+1}R)$
- $(Nir, U^nR) \to (3, 7)(Nir, DU^nR) \mid (3, 6)(Nir, U^nR) \mid (Nir, U^{n-1}R)$
- $(Nir, F) \to (Nir, DU^0R) \mid (Nir, Nii)$
- $(Nir, Nii) \to (3, 4)(Nir, Nii) \mid (3, 5)(Nir, DU^0R)$
- $(Nir, U^0R) \to (3t, 3)|(Nir, Nii)$

The action Nii may be paired with any action of the second tree so that gives 9 actions with U^kL in the first as follows.

- $(Nii, U^0L) \to (4,2)\Big[(Nii, U^0L) \mid (Nii, DU^0L) \mid (Nii, F)\Big] \mid$
  $(5,2)\Big[(DU^0R, U^0L) \mid (DU^0R, DU^0L) \mid (DU^0R, F)\Big]$
- $(Nii, DU^nL) \to (4,6t)(Nii, U^{n+1}L) \mid (4,7)(Nii, DU^{n+1}L) \mid (5,6t)(DU^0R, U^{n+1}L) \mid$
  $(5,7)(DU^0R, DU^{n+1}L)$
- $(Nii, U^nL) \to (4,6)(Nii, U^nL) \mid (4,7)(Nii, DU^nL) \mid (5,6)(DU^0R, U^nL) \mid$
  $(5,7)(DU^0R, DU^nL) \mid (Nii, U^{n-1}L)$
- $(Nii, DU^nR) \to (4,6t)(Nii, U^{n+1}R) \mid (4,7)(Nii, DU^{n+1}R) \mid (5,6t)(DU^0R, U^{n+1}R) \mid$
  $(5,7)(DU^0R, DU^{n+1}R)$
- $(Nii, U^nR) \to (4,6)(Nii, U^nR) \mid (4,7)(Nii, DU^nR) \mid (5,6)(DU^0R, U^nR) \mid$
  $(5,7)(DU^0R, DU^nR) \mid (Nii, U^{n-1}R)$
- $(Nii, F) \to (Nii, DU^0R) \mid (Nii, Nir) \mid (Nii, Nii)$
- $(Nii, Nir) \to (4,3)(Nii, Nir) \mid (5,3)(DU^0R, Nir)$
- $(Nii, Nii) \to 4,4)(Nii, Nii) \mid (5,4)(DU^0R, Nii) \mid (4,5)(Nii, DU^0R) \mid$
  $(5,5)(DU^0R, DU^0R)$
- $(Nii, U^0R) \to (Nii, Nii) \mid (Nii, Nir)$

Thus, by extension we have come up with all posible pairs of trees.

4.2 Equations

Our goal is to count both the number of reduced pairs of trees with n carets, and the number of unique geodesic lengths m in F_{ab}. We can do these two different counts at the same time since every geodesic element of length m in F_{ab} corresponds to some reduced pair of trees where each pair of carets needs a fixed number of a, b, a^{-1}, b^{-1}. See Table 3.1. We have just presented a grammar that gives a reduced pair of trees. Chomsky and Schutzenberger

[3] proved that every context-free grammar has a growth function that is algebraic. They gave a method for converting a contex-free grammar into a system of equations by making every action into a function of x and every label into an x. If there are many rules that can replace one action, they are added together. Thus there is a way (see [3]) to transform such a grammar into a power series. Remember every (m,n) where $m, n \in \{1, 1t, 2, 3, 3t, 4, 5, 6, 6t, 7\}$ represents a pair of nodes in our reduced pair of trees. We would like to reduce the systems of equations only once instead of twice. In Chapter 5 we will need every (m,n) to be replaced by x while in Chapter 6 we will need replacing every (m,n) by x^k where k is a number in table 3.1. Note that the numbers in the table range from 0 to 4; therefore, we will adopt a convention. We plan on having 5 variables a,b,c,d, and e. In appendix A.1 we let a=b=c=d=e=x. In appendix A.2 we let $a = 1$, $b = x$, $c = x^2$, $d = x^3$ and $e = x^4$. Now that we have the grammar we can transform it into equations as follows. Every arrow becomes an equal sign, every OR symbol becomes a plus sign, and every production rule becomes a function. Every (m,n) where $m, n \in \{1, 1t, 2, 3, 3t, 4, 5, 6, 6t, 7\}$ is replaced by a,b,c,d, or e depending on the number in table 3.1.

1. $S = a\Big[(F,F) + (DU^0L, DU^0L) + (F, DU^0L) + (DU^0L, F)\Big] +$
   $a\Big[(U^0L, F) + (U^0L, DU^0L)\Big] + a\Big[(F, U^0L) + (DU^0L, U^0L)\Big]$

2. $(U^0L, U^0L) = c + c\Big[(U^0L, U^0L) + (DU^0L, U^0L) + (F, U^0L) + (U^0L, DU^0L) +$
   $(DU^0L, DU^0L) + (F, DU^0L) + (U^0L, F) + (DU^0L, F) + (F, F)\Big]$

3. $(U^0L, DU^nL) = c\Big[(U^0L, DU^{n+1}L) + (DU^0L, DU^{n+1}L) + (F, DU^{n+1}L)\Big] +$
   $c\Big[(U^0L, U^{n+1}L) + (DU^0L, U^{n+1}L) + (F, U^{n+1}L)\Big]$

4. $(U^0L, U^nL) = c\Big[(U^0L, U^nL) + (DU^0L, U^nL) + (F, U^nL)\Big] +$
   $c\Big[(U^0L, DU^nL) + (DU^0L, DU^nL) + (F, DU^nL)\Big] + (U^0L, U^{n-1}L)$

5. $(U^0L, DU^nR) = c\Big[(U^0L, DU^{n+1}R) + (DU^0L, DU^{n+1}R) + (F, DU^{n+1}R)\Big] +$
$c\Big[(U^0L, U^{n+1}R) + (DU^0L, U^{n+1}R) + (F, U^{n+1}R)\Big]$

6. $(U^0L, U^nR) = c\Big[(U^0L, U^nR) + (DU^0L, U^nR) + (F, U^nR)\Big] +$
$c\Big[(U^0L, DU^nR) + (DU^0L, DU^nR) + (F, DU^nR)\Big] + (U^0L, U^{n-1}R)$

7. $(U^0L, F) = (U^0L, DU^0R) + (U^0L, Nir) + (U^0L, Nii)$

8. $(U^0L, Nir) = b + b\Big[(U^0L, Nir) + (DU^0L, Nir) + ((F, Nir)\Big]$

9. $(U^0L, Nii) = b\Big[(U^0L, Nii) + (DU^0L, Nii) + (F, Nii)\Big] +$
$b\Big[(U^0L, DU^0R) + (DU^0L, DU^0R) + (F, DU^0R)\Big]$

10. $(U^0L, U^0R) = b + b[(U^0L, Nir) + (DU^0L, Nir) + (F, Nir)] + (U^0L, Nii)$

11. $(DU^kL, U^0L) = c\Big[(U^{k+1}L, U^0L) + (U^{k+1}L, DU^0L) + (U^{k+1}L, F)\Big] +$
$c\Big[(DU^{k+1}L, U^0L) + (DU^{k+1}L, DU^0L) + (DU^{k+1}L, F)\Big]$

12. $(DU^kL, DU^nL) = e(U^{k+1}L, DU^{n+1}L) + e(DU^{k+1}L, DU^{n+1}L) + e(DU^{k+1}L, U^{n+1}L)$

13. $(DU^kL, U^nL) = c(U^{k+1}L, U^nL) + e(DU^{k+1}L, U^nL) + e(U^{k+1}L, DU^nL) +$
$e(DU^{k+1}L, DU^nL) + (DU^kL, U^{n-1}L)$

14. $(DU^kL, DU^nR) = e(U^{k+1}L, DU^{n+1}R) + e(DU^{k+1}L, DU^{n+1}R) + e(DU^{k+1}L, U^{n+1}R)$

15. $(DU^kL, U^nR) = c(U^{k+1}L, U^nR) + e(DU^{k+1}L, U^nR) + e(U^{k+1}L, DU^nR) +$
$e(DU^{k+1}L, DU^nR) + (DU^kL, U^{n-1}R)$

16. $(DU^kL, F) = (DU^kL, DU^0R) + (DU^kL, Nir) + (DU^kL, Nii)$

17. $(DU^kL, Nir) = b(U^{k+1}L, Nir) + d(DU^{k+1}L, Nir)$

18. $(DU^kL, Nii) = b(U^{k+1}L, Nii) + d(DU^{k+1}L, Nii) + d(U^{k+1}L, DU^0R) +$
$d(DU^{k+1}L, DU^0R)$

19. $(DU^kL, U^0R) = (DU^kL, Nii) + (DU^kL, Nir)$

20. $(U^kL, U^0L) = c\Big[(U^kL, U^0L) + (U^kL, DU^0L) + (U^kL, F)\Big] +$
$c\Big[(DU^kL, U^0L) + (DU^kL, DU^0L) + (DU^kL, F)\Big] + (U^{k-1}L, U^0L)$

21. $(U^kL, DU^nL) = e(U^kL, DU^{n+1}L) + e(DU^kL, DU^{n+1}L) + c(U^kL, U^{n+1}L) +$
$e(DU^kL, U^{n+1}L) + (U^{k-1}L, DU^nL)$

22. $(U^kL, U^nL) = (U^0L, U^0L) + \sum_{i=1}^{k}\Bigg(c\Big[(U^iL, U^0L) + (U^iL, DU^0L) + (U^iL, F)\Big] +$
$c\Big[(DU^iL, U^0L) + (DU^iL, DU^0L) + (DU^iL, F)\Big]\Bigg) +$
$\sum_{j=1}^{n}\Bigg(c\Big[(U^0L, U^jL) + (DU^0L, U^jL) + (F, U^lL)\Big] +$
$c\Big[(U^0L, DU^lL) + (DU^0L, DU^jL) + (F, DU^jL)\Big]\Bigg) +$
$\sum_{i=1}^{k}\sum_{j=1}^{n}\Big[e(U^iL, DU^jL) + e(DU^iL, DU^iL) + c(U^iL, U^jL) + e(DU^kL, U^jL)\Big]$

23. $(U^kL, DU^nR) = e(U^kL, DU^{n+1}R) + e(DU^kL, DU^{n+1}R) + c(U^kL, U^{n+1}R) +$
$e(DU^kL, U^{n+1}R) + (U^{k-1}L, DU^nR)$

24. $(U^kL, U^nR) = (U^0L, U^0R) + \sum_{i=1}^{k}\Big[b(U^iL, Nir) + d(DU^iL, Nir) +$
$b(U^iL, Nii) + d(DU^iL, Nii) + d(U^iL, DU^0R) + d(DU^iL, DU^0R))\Big] +$
$\sum_{j=1}^{n}\Bigg(c\Big[(U^0L, U^jR) + (DU^0L, U^jR) + (F, U^jR)\Big] +$
$c\Big[(U^0L, DU^nR) + (DU^0L, DU^nR) + (F, DU^nR)\Big]\Bigg) +$
$\sum_{i=1}^{k}\sum_{j=1}^{n}\Big[e(U^iL, DU^jR) + e(DU^iL, DU^iR) + c(U^iL, U^jR) + e(DU^iL, U^jR))\Big]$

25. $(U^kL, F) = (U^kL, DU^0R) + (U^kL, Nir) + (U^kL, Nii)$

26. $(U^kL, Nir) = b(U^kL, Nir) + d(DU^kL, Nir) + (U^{k-1}L, Nir)$

27. $(U^kL, Nii) = b(U^kL, Nii) + d(DU^kL, Nii) + d(U^kL, DU^0R) + d(DU^kL, DU^0R) +$
$(U^{k-1}L, Nii)$

28. $(U^kL, U^0R) = (U^{k-1}L, U^0R) + b(U^kL, Nir) + d(DU^kL, Nir) + b(U^kL, Nii) +$
$d(DU^kL, Nii) + d(U^kL, DU^0R) + d(DU^kL, DU^0R)$

29. $(DU^kR, U^0L) = c\Big[(U^{k+1}R, U^0L) + (U^{k+1}R, DU^0L) + (U^{k+1}R, F)\Big] +$
$c\Big[(DU^{k+1}R, U^0L) + (DU^{k+1}R, DU^0L) + (DU^{k+1}R, F)\Big]$

30. $(DU^kR, DU^nL) = e(U^{k+1}R, DU^{n+1}L) + e(DU^{k+1}R, DU^{n+1}L) + e(DU^{k+1}R, U^{n+1}L)$

31. $(DU^kR, U^nL) = c(U^{k+1}R, U^nL) + e(DU^{k+1}R, U^nL) + e(U^{k+1}R, DU^nL) +$
$e(DU^{k+1}R, DU^nL) + (DU^kR, U^{n-1}L)$

32. $(DU^kR, DU^nR) = e(U^{k+1}R, DU^{n+1}R) + e(DU^{k+1}R, DU^{n+1}R) + e(DU^{k+1}R, U^{n+1}R)$

33. $(DU^kR, U^nR) = c(U^{k+1}R, U^nR) + e(DU^{k+1}R, U^nR) + e(U^{k+1}R, DU^nR) +$
$e(DU^{k+1}R, DU^nR) + (DU^kR, U^{n-1}R)$

34. $(DU^kR, F) = (DU^kR, DU^0R) + (DU^kR, Nir) + (DU^kR, Nii)$

35. $(DU^kR, Nir) = b(U^{k+1}R, Nir) + d(DU^{k+1}R, Nir)$

36. $(DU^kR, Nii) = b(U^{k+1}R, Nii) + d(DU^{k+1}R, Nii) + d(U^{k+1}R, DU^0R) +$
$d(DU^{k+1}R, DU^0R)$

37. $(DU^kR, U^0R) = (DU^kR, Nii) + (DU^kR, Nir)$

38. $(U^kR, U^0L) = c\Big[(U^kR, U^0L) + (U^kR, DU^0L) + (U^kR, F)\Big] +$
$c\Big[(DU^kR, U^0L) + (DU^kR, DU^0L) + (DU^kR, F)\Big] + (U^{k0-01}R, U^0L)$

39. $(U^kR, DU^nL) = e(U^kR, DU^{n+1}L) + e(DU^kR, DU^{n+1}L) + c(U^kR, U^{n+1}L) +$
$e(DU^kR, U^{n+1}L) + (U^{k-1}R, DU^nL)$

40. $(U^kR, U^nL) = \sum_{i=1}^{k}\sum_{j=1}^{n}\Big[e(U^iR, DU^jL) + e(DU^iR, DU^jL) + c(U^iR, U^jL) + e(DU^iR, U^jL)\Big] +$

$\sum_{j=1}^{n}\Big[b(Nir, U^jL) + \delta(Nir, DU^jL) + b(Nii, U^jL) + \ d(Nii, DU^jL) +$

$d(DU^0R, U^jL) + d(DU^0R, DU^jL)\Big] + \sum_{i=1}^{k}\Bigg(c\Big[(U^iR, U^0L) + (U^iR, DU^0L) + (U^iR, F)\Big] +$

$c\Big[(DU^iR, U^0L) + (DU^iR, DU^0L) + (DU^iR, F)\Big]\Bigg) + (U^0R, U^0L)$

41. $(U^kR, DU^nR) = e(U^kR, DU^{n+1}R) + e(DU^kR, DU^{n+1}R) + c(U^kR, U^{n+1}R) +$

$e(DU^kR, U^{n+1}R) + (U^{k-1}R, DU^nR)$

42. $(U^kR, U^nR) = \sum_{i=1}^{k}\sum_{j=1}^{n}\Big[e(U^iR, DU^jR) + e(DU^iR, DU^jR) + c(U^kR, U^{n+1}R) +$

$\epsilon(DU^iR, U^jR)\Big] + \sum_{i=1}^{k}\Big[b(U^iR, Nir) + d(DU^iR, Nir) + b(U^iR, Nii) + d(DU^iR, Nii) +$

$d(U^iR, DU^0R) + d(DU^kR, DU^0R)\Big] + (U^0R, U^0R) + \sum_{j=1} n\Big[b(Nir, U^jR) +$

$d(Nir, DU^jR) + b(Nii, U^jR) + d(Nii, DU^jR) + d(DU^0R, U^jR) + d(DU^0R, DU^jR)\Big]$

43. $(U^kR, F) = (U^kR, DU^0R) + (U^kR, Nir) + (U^kR, Nii)$

44. $(U^kR, Nir) = b(U^kR, Nir) + d(DU^kR, Nir) + (U^{k-1}R, Nir)$

45. $(U^kR, Nii) = b(U^kR, Nii) + d(DU^kR, Nii) + d(U^kR, DU^0R) + \delta(DU^kR, DU^0R) +$

$(U^{k-1}R, Nii)$

46. $(U^kR, U^0R) = (U^{k-1}R, U^0R) + b(U^kR, Nir) + d(DU^kR, Nir) + b(U^kR, Nii) +$

$d(DU^kR, Nii) + d(U^kR, DU^0R) + d(DU^kR, DU^0R)$

47. $(F, U^0L) = (DU^0R, U^0L) + (Nir, U^0L) + (Nii, U^0L)$

48. $(F, DU^nL) = (DU^0R, DU^nL) + (Nir, DU^nL) + (Nii, DU^nL)$

49. $(F, U^nL) = (DU^0R, U^nL) + (Nir, U^nL) + (Nii, U^nL)$

50. $(F, DU^nR) = (DU^0R, DU^nR) + (Nir, DU^nR) + (Nii, DU^nR)$

51. $(F, U^nR) = (DU^0R, U^nR) + (Nir, U^nR) + (Nii, U^nR)$

52. $(F, F) = (DU^0R, F) + (Nir, F) + (Nii, F)$

53. $(F, Nir) = (DU^0R, Nir) + (Nii, Nir)$

54. $(F, Nii) = (DU^0R, Nii) + (Nir, Nii) + (Nii, Nii)$

55. $(F, U^0R) = (DU^0R, U^0R) + (Nir, U^0R) + (Nii, U^0R)$

56. $(Nir, U^0L) = b + b\Big[(Nir, U^0L) + (Nir, DU^0L) + (Nir, F)\Big]$

57. $(Nir, DU^nL) = d(Nir, DU^{n+1}L) + b(Nir, U^{n+1}L)$

58. $(Nir, U^nL) = d(Nir, DU^nL) + b(Nir, U^nL) + (Nir, U^{n-1}L)$

59. $(Nir, DU^nR) = d(Nir, DU^{n+1}R) + b(Nir, U^{n+1}R)$

60. $(Nir, U^nR) = d(Nir, DU^nR) + b(Nir, U^nR) + (Nir, U^{n-1}R)$

61. $(Nir, F) = (Nir, DU^0R) + (Nir, Nii)$

62. $(Nir, Nii) = c(Nir, Nii) + c(Nir, DU^0R)$

63. $(Nir, U^0R) = a + (Nir, Nii)$

64. $(Nii, U^0L) = b\Big[(Nii, U^0L) + (Nii, DU^0L) + (Nii, F)\Big] +$
$b\Big[(DU^0R, U^0L) + (DU^0R, DU^0L) + (DU^0R, F)\Big]$

65. $(Nii, DU^nL) = b(Nii, U^{n+1}L) + d(Nii, DU^{n+1}L) + d(DU^0R, U^{n+1}L) +$
$d(DU^0R, DU^{n+1}L)$

66. $(Nii, U^nL) = b(Nii, U^nL) + d(Nii, DU^nL) + d(DU^0R, U^nL) + d(DU^0R, DU^nL) +$
$(Nii, U^{n-1}L)$

67. $(Nii, DU^nR) = b(Nii, U^{n+1}R) + d(Nii, DU^{n+1}R) + d(DU^0R, U^{n+1}R) +$
$d(DU^0R, DU^{n+1}R)$

68. $(Nii, U^nR) = b(Nii, U^nR) + d(Nii, DU^nR) + d(DU^0R, U^nR) + d(DU^0R, DU^nR) +$
$(Nii, U^{n-1}R)$

69. $(Nii, F) = (Nii, DU^0R) + (Nii, Nir) + (Nii, Nii)$

70. $(Nii, Nir) = c(Nii, Nir) + c(DU^0R, Nir)$

71. $(Nii, Nii) = c(Nii, Nii) + c(DU^0R, Nii) + c(Nii, DU^0R) + c(DU^0R, DU^0R)$

72. $(Nii, U^0R) = (Nii, Nii) + (Nii, Nir)$

73. $(U^0R, U^0L) = b + b\Big[(Nir, U^0L) + (Nir, DU^0L) + (Nir, F)\Big] + (Nii, U^0L)$

74. $(U^0R, DU^nL) = (Nii, DU^nL) + (Nir, DU^nL)$

75. $(U^0R, U^nL) = (U^0R, U^{n-1}L) + b(Nir, U^nL) + d(Nir, DU^nL) + b(Nii, U^nL) +$
$d(Nii, DU^nL) + d(DU^0R, U^nL) + d(DU^0R, DU^nL)$

76. $(U^0R, DU^nR) = (Nii, DU^nR) + (Nir, DU^nR)$

77. $(U^0R, U^nR) = (U^0R, U^{n-1}R) + b(Nir, U^nR) + d(Nir, DU^nR) + b(Nii, U^nR) +$
$d(Nii, DU^nR) + d(DU^0R, U^nR) + d(DU^0R, DU^nR)$

78. $(U^0R, F) = (U^0R, DU^0R) + (U^0R, Nir) + (U^0R, Nii)$

79. $(U^0R, Nir) = a + (Nii, Nir)$

80. $(U^0R, Nii) = (Nir, Nii) + (Nii, Nii)$

81. $(U^0R, U^0R) = a + (Nir, Nii) + (Nii, Nir) + (Nii, Nii)$

4.3 Simplifications

A system of 81 equations is a lot to work with, so in this section we use several facts to reduce the number of equations from 81 to 11. The actual computation is left in appendix A.1.

Fact 4.3.1. if x=y+z and a=w+y+z then a=w+x.

Fact 4.3.2. If we have an equation like $c_n = b_n + c_{n-1}$ for $n \geq 1$ then $c_n = \sum_{j=1}^{n} b_j + c_0$.

Fact 4.3.3. If a=ax+c then $a = \frac{c}{1-x}$

The first reduction we will perform is that, for this power series, (a,b)=(b,a) for all production rules a and b. This will reduce our 81 equations to 45 equations. We will also change notation on the 11 equation since ordered pairs take up so much room. We will also replace long symbols pairs by single symbols as follows:

A. $S \to S$

B. $(U^0L, DU^nL) \to L_L^n$

C. $(U^0L, DU^nR) \to B_L^n$

D. $(DU^kL, DU^nL) \to L_n^k$

E. $(DU^kL, DU^nR) \to B_n^k$

F. $(DU^kR, DU^nR) \to R_n^k$

G. $(DU^kR, U^0R) \to R^k$

H. $(DU^kL, U^0R) \to B^k$

I. $(U^kL, U^nL) \to l_n^k$

J. $(U^kL, U^nR) \to b_n^k$

K. $(U^kR, U^nR) \to r_n^k$

Because these are just simple college algebra tricks, we do not present the simplication process here but instead leave it as an appendix. We put the 11 simplified equations on the next page.

$$S = a\left[\frac{1+b}{(1-c)(1-b)}\left[2R^0 + R_0^0\right] + 2\frac{1}{1-b}\left(b + B^0 + B_L^0 + B_0^0\right) + 2L_L^0 + L_0^0\right]$$

$$L_L^n = c\Bigg[2L_L^0 + L_0^0 + \sum_{j=1}^{n+1}\left(L_L^j + cl_j^1 + L_{j-1}^0 + cb_1^j + B_0^j + B^j\right) + \frac{ac+cS}{a-ac} + \frac{1}{1-b}\Bigg(b + B^0 + \frac{b}{1-c}\left[2R^0 + R_0^0\right] + B_L^0 + B_0^0\Bigg)\Bigg]$$

$$B_L^n = c\left[\sum_{j=1}^{n+1}\left(B_L^j + cb_j^1 + B_j^0 + cr_j^1 + R_j^0 + R^j\right) + \frac{1}{1-b}\left(b + B^0 + \frac{1}{1-c}\left[2R^0 + R_0^0\right] + B_L^0 + B_0^0\right) + a\right]$$

$$L_n^k = e\left(\sum_{j=1}^{k+1}\left[cl_{n+2}^j + L_{n+1}^{j-1}\right] + L_L^{n+1} + L_{n+1}^{k+1} + \sum_{j=1}^{n+1}\left[cl_j^{k+2} + L_{j-1}^{k+1}\right] + L_L^{k+1}\right)$$

$$B_n^k = e\left(\sum_{j=1}^{k+1}\left[cb_{n+2}^j + B_{n+1}^{j-1}\right] + B_L^{n+1} + B_{n+1}^{k+1} + \sum_{j=1}^{n+1}\left[cb_j^{k+2} + B_{j-1}^{k+1}\right] + R^{k+1}\right)$$

$$R_n^k = e\left(\sum_{j=1}^{k+1}\left[cr_{n+2}^j + R_{n+1}^{j-1}\right] + R^{n+1} + R_{n+1}^{k+1} + \sum_{j=1}^{n+1}\left[cr_j^{k+2} + R_{j-1}^{k+1}\right] + R^{k+1}\right)$$

$$R^k = ab + \sum_{j=1}^{k+1}\left[bR^{j-1} + dcr_1^j + dR_1^{j-1}\right] + \frac{cb}{1-c}\left[2R^0 + R_0^0\right] + d(R^{k+1} + R^0 + R_0^{k+1})$$

$$B^k = \sum_{j=1}^{k+1}\left[bB^{j-1} + dcb_1^j + dB_0^{j-1}\right] + dB^{k+1} + \frac{b^2}{1-b}\left[1 + B^0 + \frac{1}{1-c}\left[2R^0 + R_0^0\right] + B_L^0 + B_0^0\right] + dB_L^0 + dB_0^{k+1}$$

$$l_n^k = \sum_{i=1}^{k}\sum_{j=1}^{n}\left[L_{j-1}^{i-1} + cL_j^i\right] + \sum_{i=1}^{k} L_L^{i-1} + \sum_{j=1}^{n} L_L^{j-1} + \frac{ac+cS}{a-ac}$$

$$b_n^k = \sum_{i=1}^{k}\sum_{j=1}^{n}\left[B_{j-1}^{i-1} + cb_j^i\right] + \sum_{i=1}^{k} B^{j-1} + \sum_{j=1}^{n} B_L^{j-1} + \frac{b}{1-b}\left[1 + B^0 + \frac{1}{1-c}\left[2R^0 + R_0^0\right] + B_L^0 + B_0^0\right]$$

$$r_n^k = \sum_{i=1}^{k}\sum_{j=1}^{n}\left[R_{j-1}^{i-1} + cr_j^i\right] + \sum_{i=0}^{k-1} R^i + \sum_{j=0}^{n-1} R^j + a + \frac{c}{1-c}[2R^0 + R_0^0\Big]$$

5 PIPES

Recall from Chapter 2 that each element of Thompson's group F can be uniquely represented by a pipe system with no trival pipes. In this chapter we look at how it is a trivial matter to multiply a pipe system by a, b, A and by B and introduce a third generator C. We use the rules of generator multiplication to create a computer program that allows us to manipulate the pipes.

5.1 Generator multiplication on pipes

In this section, we demonstrate how to build a pipe system, and in the next section, we discuss a computer program that takes advantage of two benefits of representing elements of Thompson's group F using pipes: (1) It is a trivial matter to multiply a pipe system by $a^{\pm 1}$ and by $b^{\pm 1}$ and (2) it is easy to preserve a history of multiplications.

First, to multiply by a generator we introduce some terminology. Recall everything above the horizontal line corresponds to the top tree. Simerly everything below the horizontal line corresponds to the bottom tree.

Definition 5.1.1. An upper half pipe is the part of the pipe that is above the horizontal line.

Definition 5.1.2. A lower half pipe is the part of the pipe that is below the horizontal line.

Definition 5.1.3. Both the top of a pipe and the bottom of a pipe are assigned level numbers. The level describes how high the top of a pipe is or how low the bottom of a pipe is. The higher the top of the pipe, the smaller the top level number is. The lower the bottom of the pipe, the smaller the bottom level number is. The level number ranks vertices by distance to the root, with the root vertex labeled 1, its children labeled 2, their children labeled 3, etc. In the top half of a pipe system, only one pipe has level 1, up to two can have label 2, up to four can have label 3, In the top half of a pipe system, there can be only one level 1 pipe and there can be up to two level 2 pipes, one on each side of the level 1 pipe. In left to right order, the level 2 pipe on the left side of level one is called the "left" level 2 pipe and similarly, the right level 2 pipe is on the right side of level 1 pipe and is called the "right" level 2 pipe. Similarly, there are up to four level 3 pipes. There is the right right level 3 pipe which we get from going to the right of the right level 2 pipe. The other 3 are labeled right left, left right, and left left.

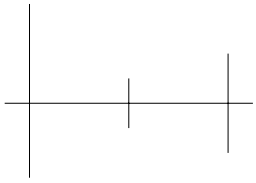
Note that both the upper half pipe and the lower half pipe have a level. We now describe how to multiply by the generators. When multiplying on the right by a we raise the left

level 2 upper half pipe to a level 1. When multiplying on the right by A we raise the right level 2 upper half pipe to a level 1. When multiplying on the right by b we raise the right left level 3 upper half pipe to a level 2. When multiplying on the right by B we raise the right right level 3 upper half pipe to a level 2.

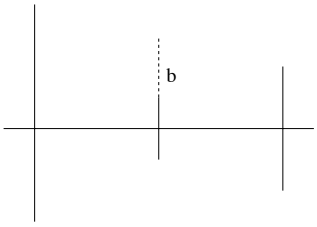
Since there are two more possible level 3 pipes, we introduce a generator C symmetric to B: When multiplying on the right by c we raise the left right level 3 upper half pipe to a level 2. When multiplying on the right by C we raise the left left level 3 upper half pipe to a level 2.

Since an appropriate level 1, 2, or 3 pipe may not exist, we may have to insert one or two or even three trivial pipes before we apply our operation of raising a pipe.

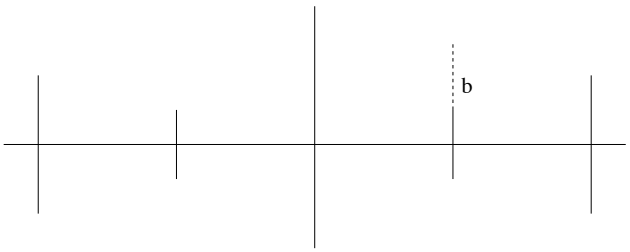
We will now show how to build the pipe system that represents the element bcA . The first letter b requires that we raise the right left level 3 pipe to a level 2, but there is no right left level 3 pipe. We create the required pipe by inserting the following trivial pipe system which represents the identity element of the group.



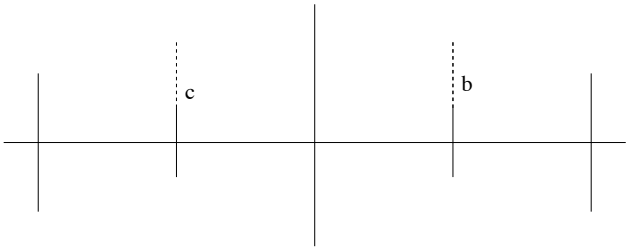
Now we can rais the right left level 3 pipe to a level 2.



Next we have c. We do the same proces and insert two more trivial pipes as shown below.



Then we raise the left right level 3 pipe to a level 2 shown below.



Now we look at A. Since there is a right level 2 pipe, we raise it to a level 1 pipe as shown below.

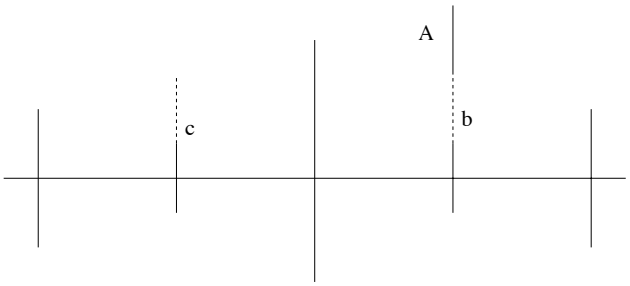


Figure 5.1: Pipe system for element bcA

Theorem 5.1.4. The groups F_{ab} and F_{ac} are isomorphic in the sense that generator is sent to a generator.

Proof. Let $f : F_{ac} \to F_{ab}$ be defined by $f(a) = A$ and $f(b) = c$. Draw any pipe system with generators a and b. Flip the pipe system from left to right. Every left left raise becomes a right right raise. Every left right raise becomes a right left raise. Every left raise became a right raise. Every right raise became a left raise. Thus they have the same growth.

□

5.2 Pipe computer program

Drawing a pipe system for an element is done with ease. It is just an algorithm with a series of rules that tell us what to do next. Therefore, a computer program can be made to keep track of the actions and to produce lots of examples. A single pipe can be realized by the upper level and the lower level which are just integers. At first, we built the program just to manipulate pipes and keep track of the history of multiplications. Through research during the past two years, we have changed the program to answer different questions that we came across. For a while, we thought that nothing good would come of the computer program, and we started thinking about the generating functions of two generators as talked about in Chapters 3 and 4, and we began adding the Fordham tree labeling to the pipe program. While doing so, we found a way to keep some type of history that greatly resembled Fordham tree labeling. Instead of putting the label on the pipe that was raised, we now describe a new type of history and label a pipe system from a word of generators.

(a) Generator A gives rise to 2 labels

- the label X is put on the pipe that was raised from a 2 to a 1
- The label Y is put on the pipe that was lowered from a 1 to a 2

(b) Generator a gives rise to 2 labels

- the label x is put on the pipe that was raised from a 2 to a 1
- The label y is put on the pipe that was lowered from a 1 to a 2

(c) Generator B is put on the pipe that was lowered from a 2 to a 3

(d) Generator b is put on the pipe that was raised from a 3 to a 2

(e) Generator C is put on the pipe that was lowered from a 2 to a 3

(f) Generator c is put on the pipe that was raised from a 3 to a 2

Fordhams table allows us to calculate the geodesic length of any given element of Thompsons group. The pipe program allows us to experiment with different algorithms for finding geodesic reductions to the identity element of the group. We have observed that, if we label the pipes as indicated during alternative geodesic reductions, the final labellings seem to be independent of the particular geodesic reduction used. We have not yet been able to prove or disprove this result.

5.3 Experiments with the three-generator presentation

The standard, asymmetrical, two-generator presentation of group F, with generators a and b, is not as geometrically natural as the three-generator presentation with generators a, b, and c; but the two-generator presentation is easier to deal with algorithmically because one has fewer choices to make in seeking geodesic element-representatives. In his BYU Ph.D. dissertation [5], Benjamin M. Woodruff was able to discover Blake Fordhams types and table for the two-generator presentation by computer computation; but similar computations with the three-generator presentation failed to yield corresponding types and tables. In his BYU Masters thesis [1], Aaron Peterson describes the subtle details necessary to consider in finding the shortest path to a first pipe cancellation in the three-generator presentation. Our experiments with the pipe program indicate that 28 types or more may be necessary to understand type mutation under multiplication in the three-generator presentation. Thus a Fordham table might have more than 784 = 28 x 28 entries. With 28 types, pipe type does seem to change nicely under right multiplication by the generators a, A, b, B, c, C. We reserve the details for later work.

6 CONCLUSION

In this paper, we have discussed properties of Thompson's group F with two different sets of generators. First, we have discussed the growth function of Thompson's group F with

just Generators a and b. We were not able to reduce the growth function to one equation, but we were able to reduce the growth function to a well-defined system of 11 equations which, if someone desired, could easily be used to calculate how many elements there are of any desired length N. Secondly, we talked about the Thompson's group F with three Generators A, B, and C. Our goal was again to figure an algorithm that would tell us how to write elements geodesically in these three generators. We wrote a computer program that allowed us to manipulate pipes using these generators. This program helped us to discover patterns that allowed us to reduce pipe systems. Although we were unable to find an efficient algorithm that yields geodesic three-generator representations of group elements, we were able to discover how each generator affects a pipe system. We hope that, with further research, we will be able to discover an efficient algorithm

A HEAVY DUTY COMPUTATIONS

In this appendix we present heavy-duty computations that were omitted in the body of the thesis.

A.1 Reducing 81 equations to 11

We now simplify and plug in new symbols for the 45 equations.

1. $(F, Nir) = (DU^0R, Nir) + (Nii, Nir)$

2. $(F, Nii) = (DU^0R, Nii) + (Nir, Nii) + (Nii, Nii)$

3. $(F, DU^nR) = (DU^0R, DU^nR) + (Nir, DU^nR) + (Nii, DU^nR)$

4. $(Nii, Nii) = c(Nii, Nii) + c(DU^0R, Nii) + c(Nii, DU^0R) + c(DU^0R, DU^0R)$ Then by fact 4.3.3 and the new notation we have
   $(Nii, Nii) = \frac{c}{1-c}[2(DU^0R, Nii) + R_0^0]$

5. $(Nir, Nii) = c(Nir, Nii) + c(Nir, DU^0R)$ Then by fact 4.3.3 we have
   $(Nir, Nii) = \frac{c}{1-c}(Nir, DU^0R)$

6. $(F, F) = (F, DU^0R) + (F, Nir) + (F, Nii)$ Then by subsitution we have
   $(F, F) = (DU^0R, DU^0R) + (Nir, DU^0R) + (Nii, DU^0R) + (DU^0R, Nir) + (Nii, Nir) +$
   $(DU^0R, Nii) + (Nir, Nii) + (Nii, Nii)$
   $(F, F) = R_0^0 + 2(DU^0R, Nir) + 2(DU^0R, Nii) + 2(Nir, Nii) + (Nii, Nii)$ $(F, F) = R_0^0 +$
   $2(DU^0R, Nir) + 2(DU^0R, Nii) + 2\frac{c}{1-c}(DU^0R, Nir) + \frac{c}{1-c}[2(DU^0R, Nii) + R_0^0]$. Simplifying
   we have $(F, F) = [R_0^0 + 2(DU^0R, Nir) + 2(DU^0R, Nii)]\frac{1}{1-c}$. Plugging in we have $(F, F) =$
   $[R_0^0 + 2(DU^0R, U^0R)]\frac{1}{1-c}$. Changing notation again, we have $(F, F) = [R_0^0 + 2R^0]\frac{1}{1-c}$

7. $(U^0L, Nir) = b + b\Big[(U^0L, Nir) + (DU^0L, Nir) + (F, Nir)\Big]$ The by fact 4.3.3 we have
   $(U^0L, Nir) = \frac{b}{1-b}\Big[1 + (DU^0L, Nir) + (F, Nir)\Big]$

8. $(U^0L, Nii) = b\Big[(U^0L, Nii) + (DU^0L, Nii) + (F, Nii)\Big] + b\Big[(U^0L, DU^0R) + (DU^0L, DU^0R) +$
   $(F, DU^0R)\Big]$ The by fact 4.3.3 we have $(U^0L, Nii) = \frac{b}{1-b}\Big[(DU^0L, Nii) + (F, Nii) +$
   $(U^0L, DU^0R) + (DU^0L, DU^0R) + (F, DU^0R)\Big]$ With the new notation, we have $(U^0L, Nii) =$
   $\frac{b}{1-b}\Big[(DU^0L, Nii) + (F, Nii) + B_L^0 + B_0^0 + (F, DU^0R)\Big]$

9. $(U^0L, U^0R) = b + b[(U^0L, Nir) + (DU^0L, Nir) + (F, Nir)] + (U^0L, Nii)$ Then by fact 4.3.1
   we have $(U^0L, U^0R) = (U^0L, Nir) + (U^0L, Nii)$

10. $(F, U^0L) = (DU^0R, U^0L) + (Nir, U^0L) + (Nii, U^0L)$. Plugging in, $(F, U^0L) = (DU^0R, U^0L) +$
    $\frac{b}{1-b}\Big[1 + (DU^0L, Nir) + (F, Nir)\Big] + \frac{b}{1-b}\Big[(DU^0L, Nii) + (F, Nii) + B_L^0 + B_0^0 + (F, DU^0R)\Big]$
    Then $(F, U^0L) = (DU^0R, U^0L) + \frac{b}{1-b}\Big[1 + (DU^0L, Nir) + (F, F) + (DU^0L, Nii) + B_L^0 + B_0^0\Big]$
    Then $(F, U^0L) = B_L^0 + \frac{b}{1-b}\Big[1 + (DU^0L, Nir) + [R_0^0 + 2R^0]\frac{1}{1-c} + (DU^0L, Nii) + B_L^0 + B_0^0\Big]$
    Then $(F, U^0L) = B_L^0 + \frac{b}{1-b}\Big[1 + (DU^0L, U^0R) + [R_0^0 + 2R^0]\frac{1}{1-c} + B_L^0 + B_0^0\Big]$ Then
    $(F, U^0L) = B_L^0 + \frac{b}{1-b}\Big[1 + B^0 + [R_0^0 + 2R^0]\frac{1}{1-c} + B_L^0 + B_0^0\Big]$

11. $(F, DU^nL) = (DU^0R, DU^nL) + (Nir, DU^nL) + (Nii, DU^nL)$. Then
$(F, DU^nL) = (DU^0R, DU^nL) + (U^0R, DU^nL)$. With new notation we have
$(F, DU^nL) = B_0^n + B^n$

12. $S = a\Big[(F,F) + (DU^0L, DU^0L) + (F, DU^0L) + (DU^0L, F)\Big] +$
$a\Big[(U^0L, F) + (U^0L, DU^0L)\Big] + a\Big[(F, U^0L) + (DU^0L, U^0L)\Big]$. By symmetry we have
$S = a\Big[(F,F) + (DU^0L, DU^0L) + 2(F, DU^0L)\Big] + 2a\Big[(F, U^0L) + (U^0L, DU^0L)\Big]$. With new notation we have $S = a\Big[(F,F) + L_0^0 + 2(F, DU^0L) + 2(F, U^0L) + 2L_L^0\Big]$ Plugging in, we have $S = a\Big[[R_0^0 + 2R^0]\frac{1}{1-c} + L_0^0 + 2(B_0^0 + B^0) + 2(B_L^0 + \frac{b}{1-b}\Big[1 + B^0 + [R_0^0 + 2R^0]\frac{1}{1-c} + B_L^0 + B_0^0\Big]) + 2L_L^0\Big]$. Finally combining like terms
$S = a\Big[\frac{1+b}{(1-c)(1-b)}\Big[2R^0 + R_0^0\Big] + 2\frac{1}{1-b}\Big(b + B^0 + B_L^0 + B_0^0\Big) + 2L_L^0 + L_0^0\Big]$.

13. $(U^0L, U^0L) = c + c\Big[(U^0L, U^0L) + (DU^0L, U^0L) + (F, U^0L) + (U^0L, DU^0L) +$
$(DU^0L, DU^0L) + (F, DU^0L) + (U^0L, F) + (DU^0L, F) + (F,F)\Big]$ Then by fact 4.3.1 we have $(U^0L, U^0L) = c + c\Big[(U^0L, U^0L) + \frac{S}{a}\Big]$ Then by fact 4.3.3 we have
$(U^0L, U^0L) = \dfrac{c + \frac{cS}{a}}{1-c} = \dfrac{ac + cS}{a - ac}$

14. $(U^0L, DU^nL) = c\Big[(U^0L, DU^{n+1}L) + (DU^0L, DU^{n+1}L) + (F, DU^{n+1}L)\Big] + c\Big[(U^0L, U^{n+1}L) +$
$(DU^0L, U^{n+1}L) + (F, U^{n+1}L)\Big]$ With the new notation we have
$L_L^n = c\Big[L_L^{n+1} + L_{n+1}^0 + (F, DU^{n+1}L) + (U^0L, U^{n+1}L) + (DU^0L, U^{n+1}L) + (F, U^{n+1}L)\Big]$.
Plugging in, $L_L^n = c\Big[L_L^{n+1} + L_{n+1}^0 + B_0^{n+1} + B^{n+1} + \sum_{j=1}^{n+1} L_L^{j-1} + \dfrac{ac + cS}{a - ac} + \sum_{j=1}^{n+1} cl_j^1 + L_{j-1}^0 +$
$L_L^0 + (F, U^{n+1}L)\Big]$

15. $(U^0L, U^nL) = c\Big[(U^0L, U^nL) + (DU^0L, U^nL) + (F, U^nL)\Big] + c\Big[(U^0L, DU^nL) +$
$(DU^0L, DU^nL) + (F, DU^nL)\Big] + (U^0L, U^{n-1}L)$ Then by fact 4.3.1 and equation 3 we have

$(U^0L, U^nL) = (U^0L, DU^{n-1}L) + (U^0L, U^{n-1}L)$. Then by fact 4.3.2 we have $(U^0L, U^nL) = \sum_{j=1}^{n}(U^0L, DU^{j-1}L) + (U^0L, U^0L)$. With the new notation we have $(U^0L, U^nL) = \sum_{j=1}^{n} L_L^{j-1} + \frac{ac+cS}{a-ac}$.

16. $(DU^kL, DU^nL) = e(U^{k+1}L, DU^{n+1}L) + e(DU^{k+1}L, DU^{n+1}L) + e(DU^{k+1}L, U^{n+1}L)$ With the new notation we have $L_n^k = e\Big[(DU^{n+1}L, U^{k+1}L) + L_{n+1}^{k+1} + (DU^{k+1}L, U^{n+1}L)\Big]$

17. $(DU^kL, U^nL) = c(U^{k+1}L, U^nL) + e(DU^{k+1}L, U^nL) + e(U^{k+1}L, DU^nL) +$ $e(DU^{k+1}L, DU^nL) + (DU^kL, U^{n-1}L)$ Then by fact 4.3.1 and equation 10 we have $(DU^kL, U^nL) = c(U^{k+1}L, U^nL) + (DU^kL, DU^{n-1}L) + (DU^kL, U^{n-1}L)$. Then by fact 4.3.2 we have $(DU^kL, U^nL) = \sum_{j=1}^{n} c(U^{k+1}L, U^jL) + (DU^kL, DU^{j-1}L) + (DU^kL, U^0L)$. With the new notation we have $(DU^kL, U^nL) = \sum_{j=1}^{n} cl_j^{k+1} + L_{j-1}^k + L_L^k$.

18. $(U^0L, DU^nR) = c\Big[(U^0L, DU^{n+1}R) + (DU^0L, DU^{n+1}R) + (F, DU^{n+1}R)\Big] +$ $c\Big[(U^0L, U^{n+1}R) + (DU^0L, U^{n+1}R) + (F, U^{n+1}R)\Big]$ With the new notation we have $B_L^n = c\Big[B_L^{n+1} + B_{n+1}^0 + (F, DU^{n+1}R) + (U^0L, U^{n+1}R) + (DU^0L, U^{n+1}R) + (F, U^{n+1}R)\Big]$

19. $(U^0L, U^nR) = c\Big[(U^0L, U^nR) + (DU^0L, U^nR) + (F, U^nR)\Big] + c\Big[(U^0L, DU^nR) +$ $(DU^0L, DU^nR) + (F, DU^nR)\Big] + (U^0L, U^{n-1}R)$ Then by fact 4.3.1 and equation 5 we have $(U^0L, U^nR) = (U^0L, DU^{n-1}R) + (U^0L, U^{n-1}R)$. Then by fact 4.3.2 we have $(U^0L, U^nR) = \sum_{j=1}^{n}(U^0L, DU^{j-1}R) + (U^0L, U^0R)$ With the new notation we have $(U^0L, U^nR) = \sum_{j=1}^{n} B_L^{j-1} + (U^0L, U^0R)$

20. $(DU^kL, DU^nR) = e(U^{k+1}L, DU^{n+1}R) + e(DU^{k+1}L, DU^{n+1}R) + e(DU^{k+1}L, U^{n+1}R)$

21. $(DU^kL, U^nR) = c(U^{k+1}L, U^nR) + e(DU^{k+1}L, U^nR) + e(U^{k+1}L, DU^nR) +$ $e(DU^{k+1}L, DU^nR) + (DU^kL, U^{n-1}R)$ Then by fact 4.3.1 and equation 12 we have $(DU^kL, U^nR) = c(U^{k+1}L, U^nR) + (DU^kL, DU^{n-1}R) + (DU^kL, U^{n-1}R)$. Then by fact

4.3.2 we have $(DU^kL, U^nR) = \sum_{j=1}^{n} c(U^{k+1}L, U^jR) + (DU^kL, DU^{j-1}R) + (DU^kL, U^0R)$.

With new notation we have $(DU^kL, U^nR) = \sum_{j=1}^{n} cb_j^{k+1} + B_{j-1}^k + B^k$

22. $(DU^kL, Nir) = b(U^{k+1}L, Nir) + d(DU^{k+1}L, Nir)$

23. $(DU^kL, Nii) = b(U^{k+1}L, Nii) + d(DU^{k+1}L, Nii) + d(U^{k+1}L, DU^0R) + d(DU^{k+1}L, DU^0R)$

24. $(DU^kL, U^0R) = (DU^kL, Nii) + (DU^kL, Nir)$

25. $(U^kL, U^nL) = \sum_{i=1}^{k}\sum_{j=1}^{n}\Big[e(U^iL, DU^jL) + e(DU^iL, DU^iL) + c(U^iL, U^jL) + e(DU^kL, U^jL)\Big] +$
$\sum_{i=1}^{k}\Big(c\Big[(U^iL, U^0L) + (U^iL, DU^0L) + (U^iL, F)\Big] + c\Big[(DU^iL, U^0L) + (DU^iL, DU^0L) +$
$(DU^iL, F)\Big]\Big) + \sum_{j=1}^{n}\Big(c\Big[(U^0L, U^jL) + (DU^0L, U^jL) + (F, U^lL)\Big] +$
$c\Big[(U^0L, DU^lL) + (DU^0L, DU^jL) + (F, DU^jL)\Big]\Big) + (U^0L, U^0L)$

26. $(U^kL, DU^nR) = e(U^kL, DU^{n+1}R) + e(DU^kL, DU^{n+1}R) + c(U^kL, U^{n+1}R) +$
$e(DU^kL, U^{n+1}R) + (U^{k-1}L, DU^nR)$

27. $(U^kL, U^nR) = \sum_{i=1}^{k}\sum_{j=1}^{n}\Big[e(U^iL, DU^jR) + e(DU^iL, DU^iR) + c(U^iL, U^jR) + e(DU^iL, U^jR))\Big]$
$+ \sum_{i=1}^{k}\Big[b(U^iL, Nir) + d(DU^iL, Nir) + b(U^iL, Nii) + d(DU^iL, Nii) + d(U^iL, DU^0R) +$
$d(DU^iL, DU^0R))\Big] + \sum_{j=1}^{n}\Big(c\Big[(U^0L, U^jR) + (DU^0L, U^jR) + (F, U^jR)\Big] +$
$c\Big[(U^0L, DU^nR) + (DU^0L, DU^nR) + (F, DU^nR)\Big]\Big) + (U^0L, U^0R)$

28. $(U^kL, Nir) = b(U^kL, Nir) + d(DU^kL, Nir) + (U^{k-1}L, Nir)$ Then by fact 4.3.1 and equation 14 we have $(U^kL, Nir) = (DU^{k-1}L, Nir) + (U^{k-1}L, Nir)$ Then by fact 4.3.2 we have $(U^kL, Nir) = \sum_{j=1}^{k}(DU^{j-1}L, Nir) + (U^0L, Nir)$

29. $(U^kL, Nii) = b(U^kL, Nii) + d(DU^kL, Nii) + d(U^kL, DU^0R) + d(DU^kL, DU^0R) +$
$(U^{k-1}L, Nii)$ Then by fact 4.3.1 and equation 15 we have

$(U^kL, Nir) = (DU^{k-1}L, Nii) + (U^{k-1}L, Nii)$ Then by fact 4.3.2 we have
$(U^kL, Nii) = \sum_{j=1}^{k}(DU^{j-1}L, Nii) + (U^0L, Nii)$

30. $(U^kL, U^0R) = (U^{k-1}L, U^0R) + b(U^kL, Nir) + d(DU^kL, Nir) + b(U^kL, Nii) +$
$d(DU^kL, Nii) + d(U^kL, DU^0R) + d(DU^kL, DU^0R)$

31. $(DU^kR, DU^nR) = e(U^{k+1}R, DU^{n+1}R) + e(DU^{k+1}R, DU^{n+1}R) + e(DU^{k+1}R, U^{n+1}R)$

32. $(DU^kR, U^nR) = c(U^{k+1}R, U^nR) + e(DU^{k+1}R, U^nR) + e(U^{k+1}R, DU^nR) +$
$e(DU^{k+1}R, DU^nR) + (DU^kR, U^{n-1}R)$

33. $(DU^kR, Nir) = b(U^{k+1}R, Nir) + d(DU^{k+1}R, Nir)$

34. $(DU^kR, Nii) = b(U^{k+1}R, Nii)+d(DU^{k+1}R, Nii)+d(U^{k+1}R, DU^0R)+d(DU^{k+1}R, DU^0R)$

35. $(DU^kR, U^0R) = (DU^kR, Nii) + (DU^kR, Nir)$

36. $(U^kR, U^nR) = \sum_{i=1}^{k}\sum_{j=1}^{n}\Big[e(U^iR, DU^jR) + e(DU^iR, DU^jR) + c(U^kR, U^{n+1}R) +$
$e(DU^iR, U^jR)\Big] + \sum_{i=1}^{k}\Big[b(U^iR, Nir) + d(DU^iR, Nir) + b(U^iR, Nii) + \delta(DU^iR, Nii) +$
$d(U^iR, DU^0R)+d(DU^kR, DU^0R)\Big] + (U^0R, U^0R)+\sum_{j=1} n\Big[b(Nir, U^jR)+d(Nir, DU^jR)+$
$b(Nii, U^jR) + d(Nii, DU^jR) + d(DU^0R, U^jR) + d(DU^0R, DU^jR)\Big]$

37. $(U^kR, Nir) = b(U^kR, Nir) + d(DU^kR, Nir) + (U^{k-1}R, Nir)$

38. $(U^kR, Nii) = b(U^kR, Nii) + d(DU^kR, Nii) + d(U^kR, DU^0R) + \delta(DU^kR, DU^0R) +$
$(U^{k-1}R, Nii)$

39. $(U^kR, U^0R) = (U^{k-1}R, U^0R) + b(U^kR, Nir) + d(DU^kR, Nir) + b(U^kR, Nii) +$
$d(DU^kR, Nii) + d(U^kR, DU^0R) + d(DU^kR, DU^0R)$

40. $(F, U^nL) = (DU^0R, U^nL) + (Nir, U^nL) + (Nii, U^nL)$

41. $(F, U^nR) = (DU^0R, U^nR) + (Nir, U^nR) + (Nii, U^nR)$

42. $(F, U^0R) = (DU^0R, U^0R) + (Nir, U^0R) + (Nii, U^0R)$

43. $(Nir, U^0R) = a + (Nir, Nii)$

44. $(Nii, U^0R) = (Nii, Nii) + (Nii, Nir)$

45. $(U^0R, U^0R) = a + (Nir, Nii) + (Nii, Nir) + (Nii, Nii)$

A.2 Growth function for reduced pairs of trees

Recall from 4.2 that, in order to find the growth function for reduced pairs of trees, we set each of a, b, c, d, and e equal to x. When we make that substitution in our 11 equations, we obtain the following:

$$S = x\left[\tfrac{1+x}{(1-x)(1-x)}\left[2R^0 + R_0^0\right] + 2\tfrac{1}{1-x}\left(x + B^0 + B_L^0 + B_0^0\right) + 2L_L^0 + L_0^0\right]$$

$$L_L^n = x\left[2L_L^0 + L_0^0 + \sum_{j=1}^{n+1}\left(L_L^j + xl_j^1 + L_{j-1}^0 + xb_1^j + B_0^j + B^j\right) + \frac{xx + xS}{x - xx} + \frac{1}{1-x}\left(x + B^0 + \frac{x}{1-x}\left[2R^0 + R_0^0\right] + B_L^0 + B_0^0\right)\right]$$

$$B_L^n = x\left[\sum_{j=1}^{n+1}\left(B_L^j + xb_j^1 + B_j^0 + xr_j^1 + R_j^0 + R^j\right) + \frac{1}{1-x}\left(x + B^0 + \frac{1}{1-x}\left[2R^0 + R_0^0\right] + B_L^0 + B_0^0\right) + x\right]$$

$$L_n^k = x\left(\sum_{j=1}^{k+1}\left[xl_{n+2}^j + L_{n+1}^{j-1}\right] + L_L^{n+1} + L_{n+1}^{k+1} + \sum_{j=1}^{n+1}\left[xl_j^{k+2} + L_{j-1}^{k+1}\right] + L_L^{k+1}\right)$$

$$B_n^k = x\left(\sum_{j=1}^{k+1}\left[xb_{n+2}^j + B_{n+1}^{j-1}\right] + B_L^{n+1} + B_{n+1}^{k+1} + \sum_{j=1}^{n+1}\left[xb_j^{k+2} + B_{j-1}^{k+1}\right] + R^{k+1}\right)$$

$$R_n^k = x\left(\sum_{j=1}^{k+1}\left[xr_{n+2}^j + R_{n+1}^{j-1}\right] + R^{n+1} + R_{n+1}^{k+1} + \sum_{j=1}^{n+1}\left[xr_j^{k+2} + R_{j-1}^{k+1}\right] + R^{k+1}\right)$$

$$R^k = x^2 + \sum_{j=1}^{k+1}\left[xR^{j-1} + x^2r_1^j + xR_1^{j-1}\right] + \frac{x^2}{1-x}\left[2R^0 + R_0^0\right] + x(R^{k+1} + R^0 + R_0^{k+1})$$

$$B^k = \sum_{j=1}^{k+1}\left[xB^{j-1} + xxb_1^j + xB_0^{j-1}\right] + xB^{k+1} + \frac{x^2}{1-x}\left[1 + B^0 + \frac{1}{1-x}\left[2R^0 + R_0^0\right] + B_L^0 + B_0^0\right] + xB_L^0 + xB_0^{k+1}$$

$$l_n^k = \sum_{i=1}^{k}\sum_{j=1}^{n}\left[L_{j-1}^{i-1} + xL_j^i\right] + \sum_{i=1}^{k} L_L^{i-1} + \sum_{j=1}^{n} L_L^{j-1} + \frac{xx + xS}{x - xx}$$

$$b_n^k = \sum_{i=1}^{k}\sum_{j=1}^{n}\left[B_{j-1}^{i-1} + xb_j^i\right] + \sum_{i=1}^{k} B^{j-1} + \sum_{j=1}^{n} B_L^{j-1} + \frac{x}{1-x}\left[1 + B^0 + \frac{1}{1-x}\left[2R^0 + R_0^0\right] + B_L^0 + B_0^0\right]$$

$$r_n^k = \sum_{i=1}^{k}\sum_{j=1}^{n}\left[R_{j-1}^{i-1} + xr_j^i\right] + \sum_{i=0}^{k-1} R^i + \sum_{j=0}^{n-1} R^j + x + \frac{x}{1-x}\left[2R^0 + R_0^0\right]$$

A.3 Growth function for elements of Thompson's group F

Recall from 4.2 that, in order to find the growth function for geodesics elements of Thompson's group F, we set $a = 1$, $b = x$, $c = x^2$, $d = x^3$ and $e = x^4$. When we make that substitution in our 11 equations, we obtain the following: $S = 1\left[\frac{1+x}{(1-x^2)(1-x)}\left[2R^0 + R_0^0\right] + 2\frac{1}{1-x}\left(x + B^0 + B_L^0 + B_0^0\right) + 2L_L^0 + L_0^0\right]$

$$L_L^n = x^2\left[2L_L^0 + L_0^0 + \sum_{j=1}^{n+1}\left(L_L^j + x^2 l_j^1 + L_{j-1}^0 + x^2 b_1^j + B_0^j + B^j\right) + \frac{1x^2 + x^2 S}{1 - 1x^2} + \frac{1}{1-x}\left(x + B^0 + \frac{x}{1-x^2}\left[2R^0 + R_0^0\right] + B_L^0 + B_0^0\right)\right]$$

$$B_L^n = x^2\left[\sum_{j=1}^{n+1}\left(B_L^j + x^2 b_j^1 + B_j^0 + b^2 r_j^1 + R_j^0 + R^j\right) + \frac{1}{1-x}\left(b + B^0 + \frac{1}{1-b^2}\left[2R^0 + R_0^0\right] + B_L^0 + B_0^0\right) + 1\right]$$

$$L_n^k = x^4\left(\sum_{j=1}^{k+1}\left[x^2 l_{n+2}^j + L_{n+1}^{j-1}\right] + L_L^{n+1} + L_{n+1}^{k+1} + \sum_{j=1}^{n+1}\left[x^2 l_j^{k+2} + L_{j-1}^{k+1}\right] + L_L^{k+1}\right)$$

$$B_n^k = x^4\left(\sum_{j=1}^{k+1}\left[x^2 b_{n+2}^j + B_{n+1}^{j-1}\right] + B_L^{n+1} + B_{n+1}^{k+1} + \sum_{j=1}^{n+1}\left[x^2 b_j^{k+2} + B_{j-1}^{k+1}\right] + R^{k+1}\right)$$

$$R_n^k = x^4\left(\sum_{j=1}^{k+1}\left[x^2 r_{n+2}^j + R_{n+1}^{j-1}\right] + R^{n+1} + R_{n+1}^{k+1} + \sum_{j=1}^{n+1}\left[x^2 r_j^{k+2} + R_{j-1}^{k+1}\right] + R^{k+1}\right)$$

$$R^k = 1x + \sum_{j=1}^{k+1}\left[xR^{j-1} + x^3 x^2 r_1^j + x^3 R_1^{j-1}\right] + \frac{x^2 x}{1-x^2}\left[2R^0 + R_0^0\right] + x^3(R^{k+1} + R^0 + R_0^{k+1})$$

$$B^k = \sum_{j=1}^{k+1}\left[xB^{j-1} + x^3 x^2 b_1^j + x^3 B_0^{j-1}\right] + x^3 B^{k+1} + \frac{x^2}{1-x}\left[1 + B^0 + \frac{1}{1-x^2}\left[2R^0 + R_0^0\right] + B_L^0 + B_0^0\right] + x^3 B_L^0 + x^3 B_0^{k+1}$$

$$l_n^k = \sum_{i=1}^{k}\sum_{j=1}^{n}\left[L_{j-1}^{i-1} + x^2 L_j^i\right] + \sum_{i=1}^{k} L_L^{i-1} + \sum_{j=1}^{n} L_L^{j-1} + \frac{1x^2 + x^2 S}{1 - 1x^2}$$

$$b_n^k = \sum_{i=1}^{k}\sum_{j=1}^{n}\left[B_{j-1}^{i-1} + x^2 b_j^i\right] + \sum_{i=1}^{k} B^{j-1} + \sum_{j=1}^{n} B_L^{j-1} + \frac{x}{1-x}\left[1 + B^0 + \frac{1}{1-x^2}\left[2R^0 + R_0^0\right] + B_L^0 + B_0^0\right]$$

$$r_n^k = \sum_{i=1}^{k}\sum_{j=1}^{n}\left[R_{j-1}^{i-1} + x^2 r_j^i\right] + \sum_{i=0}^{k-1} R^i + \sum_{j=0}^{n-1} R^j + 1 + \frac{x^2}{1-x^2}[2R^0 + R_0^0]\Big]$$

B COMPUTER PIPE PROGRAM

Computers make recursive mathematical processes happen more quickly which means knowledge can be gained more quickly, as well. This is why we created a java program that allows one to analyze how multiplication by a generators works. This program is a work in progress that someone can change to give them new insight into Thompson's group F. In this chapter we will explain the organization of the program and how one can change it to solve different problems.

B.1 The files and purposes of each

This java program is made up of 9 files, each of which has a purpose.

1. Filenode.java allows multiple internal files.

2. Gfile.java is one internal frame. It controls the saving and opening of files.

3. GPicture.java remembers a pipe system and the entire drawing in one internal frame. This is the file we edit and change to test new ideas

4. GroupF.java is the file that starts the program.

5. Line.java keeps track of a single line in a previous pipes

6. LineNode.java keeps track on one internal frame of many lines that you want to draw of old pipe systems

7. Word.java saves a string and the location where it should be placed.

8. Wordnode.java saves multiple words to be drawn at various locations.

```
package Groupf;

// Filenode.java
public class Filenode {

    Gfile element;
    Filenode next;

    public Filenode(Gfile e, Filenode n) {
        element = e;
        next = n;
    }

    public Filenode(Gfile e) {
        this(e, null);
    }
}
```

```
 package Groupf;
import java.io.*;
import java.util.logging.Level;
import java.util.logging.Logger;
/**
 * Gfile.java
 * @author jenniferschofield */
public class Gfile extends javax.swing.JInternalFrame {
    // This is the pannel where all the drawing gose
    private GPicture gPicture1;
    // This is the name of the file of the internal frame
    private File name;
    public Gfile() {
        initComponents();
        name = null;
    }
    public Gfile(File n) throws IOException {
        int t;
        int b;
    }

    private void initComponents() {
        gPicture1 = new GPicture();
        setClosable(true);
        setIconifiable(true);
        setMaximizable(true);
        setResizable(true);
```

```
        this.gPicture1.setOpaque(true);
        this.setContentPane(this.gPicture1);
        pack();
    }
    public void save() throws IOException {

    }
    public void saveas(File n) throws IOException {
        name = n;
        n.createNewFile();
        this.setTitle(name.getName());
        this.save();
    }
    public File getname() {
        return name;
    }
    public void setname(File name) {
        this.name = name;
    }
    public GPicture getGPicture1() {
        return gPicture1;
    }

    public void setGPicture1(GPicture gPicture1) {
        this.gPicture1 = gPicture1;
    }
}
```

```
package Groupf;
import javax.swing.*;
import java.awt.*;
public class GPicture extends javax.swing.JPanel {
private Linkedpipes pipes;
private DrawingPane drawingPane;
private Linenode lines;
private Wordnode words;
private String element;
private int top;
private Dimension area;
private int pages;
private boolean at;
private boolean se;
private boolean rm;
private int w = 35;
private int h = 4;
private int right;
private int tb;
private int tt;
private int tts;
private int tbs;
private boolean left;
private boolean leftb;
private boolean leftS;
private boolean leftbS;
private int slength;
```

```
private Linenode rect;
public GPicture() {
    super(new BorderLayout());
    pages = 1;
    lines = null;
    element = "";
    words = null;
    rect = null;
    slength = 5;
    area = new Dimension(200, 200);
    pipes = new Linkedpipes();
    pipes.insertAfter(1, 1, pipes.head());
    drawingPane = new DrawingPane();
    drawingPane.setBackground(Color.white);
    //Put the drawing area in a scroll pane.
    JScrollPane scroller = new JScrollPane(drawingPane);
    drawingPane.setPreferredSize(area);
    add(scroller, BorderLayout.CENTER);
    top = 15;

}
public void delete() {

    element = "";
    pipes = new Linkedpipes();
    pipes.insertAfter(1, 1, pipes.head());
    drawingPane.repaint();
```

```
}
public void newelement() {
    savetree();
    delete();
}
public class DrawingPane extends javax.swing.JPanel {
    @Override
    protected void paintComponent(Graphics g) {
        int wordab = 0;
        int wordac = 0;
        int ac;
        int ab;
        int wordabcmin = 0;
        double wordabc = 0;
        left = true;
        leftb = true;
        leftS = true;
        leftbS = true;
        int x;
        int y;
        super.paintComponent(g);
        g.setColor(Color.BLACK);
        Wordnode tempw = words;
        while (tempw != null) {
            g.drawString(tempw.element.w,
                    tempw.element.x, tempw.element.y);
            tempw = tempw.next;
```

```
}
Linenode templ = lines;
while (templ != null) {
    g.drawLine(templ.element.x1, templ.element.y1,
            templ.element.x2, templ.element.y2);
    templ = templ.next;
}
templ = rect;
while (templ != null) {
    g.fillRect(templ.element.x1, templ.element.y1,
            templ.element.x2, templ.element.y2);
    templ = templ.next;
}
int bottom = 550 + top;
PipeIterator tempp = pipes.head();
tempp.next();
tt = tempp.pos.top;
tb = tempp.pos.bottom;
tbs = tempp.pos.bottom;
tts = tempp.pos.top;
g.drawLine(0, 175 + top, 31 * pipes.count, 175 + top);
for (int i = 1; i <= pipes.count; i++) {
    g.drawLine(i * 30, bottom - h *
            tempp.pos.bottom, i * 30,
            top + h * tempp.pos.top);
    g.drawString(Integer.toString(
            tempp.pos.bottom), i * 30 + 2,
```

```
            bottom - h * tempp.pos.bottom);
    Wordnode tempe = tempp.pos.element;
    while (tempe != null) {
        g.drawString(tempe.element.w, i * 30 + 2,
                100 + 15 * tempe.element.y);
        tempe = tempe.next;
    }
    g.drawString(Integer.toString(
            tempp.pos.top), i * 30 + 2,
            top + h * tempp.pos.top);
    x = findtypet(tempp.pos.top, tempp.pos.next.top,
            tempp.pos.previous.top);
    y = findtypeb(tempp.pos.bottom,
            tempp.pos.next.bottom,
            tempp.pos.previous.bottom);
    ab = findcountab(x, y);
    ac = findcountac(x, y);
    g.drawString(Integer.toString(Math.min(ab, ac)),
            i * 30 + 2, bottom - 320);
    g.drawString(Integer.toString(tempp.pos.ablength),
            i * 30 + 2, bottom - 295);
    g.drawString(Integer.toString(ab),
            i * 30 + 2, bottom - 270);
    g.drawString(Integer.toString(tempp.pos.aclength),
            i * 30 + 2, bottom - 225);
    g.drawString(Integer.toString(ac), i
            * 30 + 2, bottom - 200);
```

```
        wordab += ab;
        wordac += ac;
        wordabcmin += Math.min(ab, ac);
        ;
        g.drawString(findtypets(tempp.pos.top,
                tempp.pos.next.top,
                tempp.pos.previous.top),
                i * 30 + 2,
                top + h * tempp.pos.top + 15);

        g.drawString(findtypebs(tempp.pos.bottom,
                tempp.pos.next.bottom,
                tempp.pos.previous.bottom),
                i * 30 + 2,
                bottom - h * tempp.pos.bottom - 22);
        tempp.next();
    }
    g.drawString(element, w * pipes.count
            + 2 * w, top + 20);
    g.drawString(Integer.toString(element.length()),
            w * pipes.count + 2 * w, top + 40);
    g.drawString(Integer.toString(pipes.count),
            w * pipes.count + 4 * w, top + 40);
    g.drawString(Integer.toString(wordab),
            w * pipes.count + 2 * w, top + 60);
    g.drawString(Integer.toString(wordac),
            w * pipes.count + 4 * w, top + 60);
```

```
            g.drawString(Integer.toString(wordabcmin),
                    w * pipes.count + 6 * w, top + 60);
            g.drawString(Integer.toString(slength),
                    w * pipes.count + 8 * w, top + 40);
            g.drawString("min of ab and ac",
                    w * pipes.count + 2 * w, bottom - 320);
            g.drawString("ab", w * pipes.count + 2 * w,
                    bottom - 295);
            g.drawString("# of letters to reduce this pipe" +
                    " with genersters a and b",
                    w * pipes.count + 2 * w, bottom - 270);
            g.drawString("ac", w * pipes.count + 2 * w,
                    bottom - 225);
            g.drawString("# of letters to reduce this pipe" +
                    " with genersters a and c",
                    w * pipes.count + 2 * w, bottom - 200);
            System.out.println(wordab);
        }
    }
    public void savelength() {
        slength = 1 * element.length();
        sp();
    }
    public int findtypet(int a, int b, int c) {
        if (left) {
            if (a == 1) {
                left = false;
```

```
            if (a == tt) {
                tt = 2;
                return 12;
            } else {
                if (b == 0) {
                    return 11;
                } else {
                    return 10;
                }
            }
        } else if (a == tt) {
            return 1;
        } else if (a == tt - 1) {
            tt = a;
            if (c == tt + 1) {
                return 2;
            } else {
                return 0;
            }
        } else {
            return findtype(a, b, c);
        }
    } else {
        if (b == 0) {
            return 3;
        } else if (a == tt) {
            tt = tt + 1;
```

```
            if (b == tt) {
                return 4;
            } else {
                return 5;
            }
        } else {
            return findtype(a, b, c);
        }
    }
}
public int findtypeb(int a, int b, int c) {
    if (leftb) {
        if (a == 1) {
            leftb = false;
            if (a == tb) {
                tb = 2;
                return 12;
            } else {
                if (b == 0) {
                    return 11;
                } else {
                    return 10;
                }
            }
        } else if (a == tb) {
            return 1;
        } else if (a == tb - 1) {
```

```
            tb = a;

            if (c == tb + 1) {
                return 2;
            } else {
                return 0;
            }
        } else {
            return findtype(a, b, c);
        }
    } else {
        if (b == 0) {
            return 3;
        } else if (a == tb) {
            if (b == tb + 1) {
                tb = b;
                return 4;
            } else {
                tb = tb + 1;
                return 5;
            }
        } else {
            return findtype(a, b, c);

        }
    }
```

```
}
public String findtypets(int a, int b, int c) {
    if (leftS) {
        if (a == 1) {
            leftS = false;
            if (a == tts) {
                tts = 2;
                if (b == tts) {
                    return "FT";
                } else {
                    return "FRT";
                }
            } else {
                if (b == 0) {
                    if (c == 2) {
                        return "ET";
                    } else {
                        return "ELT";
                    }
                } else {
                    if (c == 2) {
                        if (b == 2) {
                            return "T";
                        } else {
                            return "TL";
```

```
                }
            } else {
                    if (b == 2) {
                        return "TR";
                    } else {
                        return "TB";
                    }
            }
        }
    }
} else if (a == tts) {

    if (b == tts -1) {
        return "F";
    } else {
        return "FR";
    }
} else if (a == tts - 1) {
    tts = a;
    if (b == tts - 1) {

        if (c == tts +1) {
            return "L";
        } else {
            return "LL";
        }
    } else {
```

```
            if (c == tts +1) {
                return "LR";
            } else {
                return "LB";
            }
        }
    } else {
        return findtypei(a, b, c);
    }
} else {
    if (b == 0) {

        if (c == tts -1) {
            return "E";
        } else {
            return "EL";
        }
    } else if (a == tts) {
        tts = tts + 1;
        if (b == tts) {

            if (c == tts-2) {
                return "R";
            } else {
                return "RR";
            }
```

```
            } else {

                if (c == tts-2) {
                    return "RL";
                } else {
                    return "RB";
                }
            }
        } else {
            return findtypei(a, b, c);
        }
    }
}
public String findtypebs(int a, int b, int c) {
    if (leftbS) {
        if (a == 1) {
            leftbS = false;
            if (a == tbs) {
                tbs = 2;
                if (b == tbs) {
                    return "FT";
                } else {
                    return "FRT";
                }
            } else {
                if (b == 0) {
                    if (c == 2) {
```

```
                return "ET";
            } else {
                return "ELT";
            }
        } else {
            if (c == 2) {
                if (b == 2) {
                    return "T";
                } else {
                    return "TL";
                }
            } else {
                    if (b == 2) {
                        return "TR";
                    } else {
                        return "TB";
                    }
            }
        }
    } else if (a == tbs) {

        if (b == tbs -1) {
            return "F";
        } else {
            return "FR";
        }
```

```
    } else if (a == tbs - 1) {
        tbs = a;
        if (b == tbs - 1) {

            if (c == tbs +1) {
                return "L";
            } else {
                return "LL";
            }
        } else {

            if (c == tbs +1) {
                return "LR";
            } else {
                return "LB";
            }
        }
    } else {
        return findtypei(a, b, c);
    }
} else {
    if (b == 0) {

        if (c == tbs -1) {
            return "E";
        } else {
            return "EL";
```

```
            }
        } else if (a == tbs) {
            tbs = tbs + 1;
            if (b == tbs) {

                if (c == tbs-2) {
                    return "R";
                } else {
                    return "RR";
                }
            } else {

                if (c == tbs-2) {
                    return "RL";
                } else {
                    return "RB";
                }
            }
        } else {
            return findtypei(a, b, c);
        }
    }
}
public int findtype(int a, int b, int c) {
    if (b > a) {
        if (c > a) {
            return 9;
```

```
        } else {
            if (a == b + 1) {
                return 7;
            } else {
                return 17;
            }
        }
    } else {
        if (c > a) {

            if (a == b + 1) {
                return 6;
            } else {
                return 16;
            }
        } else {
            if (a == b + 1) {
                return 8;
            } else {
                return 18;
            }
        }
    }
}
public String findtypei(int a, int b, int c) {
    if (b > a) {
        if (c > a) {
```

```
            return "ib";
        } else {
            if (a == b + 1) {
                return "ir";
            } else {
                return "17";
        }
        }} else {
        if (c > a) {

            if (a == c + 1) {
                return "iL";
            } else {
                return "16";
            }
        } else {
            if (a == c - 1) {
                return "itr";
            } else {
                return "itL";
            }
        }
    }
}
public int findcountab(int a, int b) {
    if (a == 11) {
        a = 10;
```

```
}
if (b == 11) {
    b = 10;
}
if (a > 12) {
    a = a - 10;
}
if (b > 12) {
    b = b - 10;
}
if (a == 12) {
    a = 1;
}
if (b == 12) {
    b = 1;
}
if (a == b) {

    switch (a) {
        case 1:
        case 3:
            return 0;
        case 0:
        case 2:
        case 4:
        case 5:
        case 6:
```

```
            case 10:
                return 2;
            case 7:
            case 9:
                return 4;
        }
    } else if (a > b) {
        return findcountab(b, a);
    } else {
        switch (a) {
            case 0:
                switch (b) {
                    case 3:
                    case 4:
                    case 5:
                        return 1;
                    case 2:
                    case 10:
                    case 6:
                    case 7:
                    case 8:
                    case 9:
                        return 2;
                }
            case 2:
                switch (b) {
                    case 3:
```

```
            case 4:
            case 5:
                return 1;
            case 10:
            case 6:
            case 7:
            case 8:
            case 9:
                return 2;
        }
    case 3:
        return 1;
    case 4:
        switch (b) {
            case 5:
                return 2;
            case 6:
            case 8:
            case 10:
                return 1;
            case 7:
            case 9:
                return 3;
        }
    case 5:
        if (b == 10) {
            return 1;
```

```
        } else {
            return 3;
        }
    case 6:
        switch (b) {
            case 8:
            case 10:
                return 2;
            case 7:
            case 9:
                return 4;
        }
    case 7:
        switch (b) {
            case 10:
                return 2;
            case 8:
            case 9:
                return 4;
        }
    case 8:
        switch (b) {
            case 10:
                return 2;
            case 9:
                return 4;
        }
```

```
            case 9:
                return 2;
        }
    }
    return a;
}
public int findcountac(int a, int b) {
    return findcountab(swich(a), swich(b));
}
public int swich(int a) {
    switch (a) {
        case 0:
            return 5;
        case 1:
            return 3;
        case 2:
            return 4;
        case 3:
            return 1;
        case 4:
            return 2;
        case 5:
            return 0;
        case 6:
            return 7;
        case 7:
            return 6;
```

```
            case 11:
                return 12;
            case 12:
                return 11;
            case 16:
                return 17;
            case 17:
                return 16;

        }
        return a;
    }
    public void rmove() {

        if (pipes.count > 1) {
            PipeIterator temp = pipes.head();
            temp.next();
            while (temp.pos.top > 0) {
                if (temp.pos.top > java.lang.Math.max(
                        temp.pos.previous.top,
                        temp.pos.next.top)) {
                    if (temp.pos.bottom > java.lang.Math.max(
                            temp.pos.previous.bottom,
                            temp.pos.next.bottom)) {
                        pipes.remove(temp);
                    }
                }
```

```
            temp.next();
        }
    }


}
private void savetree() {
    int bottom = 2 * h * (2 + pipes.count) + top;
    int x;
    int y;
    int word2 = 0;
    left = true;
    leftb = true;
    Numnode tempn;
    if (bottom > 700 * pages) {

        top = 700 * pages + 1;
        bottom = 2 * h * (2 + pipes.count) + top;
        pages += 1;
    }
    words = new Wordnode(new Word(element,
            w * pipes.count + 2 * w, top + 20), words);
    words = new Wordnode(new Word(
            Integer.toString(element.length()),
            w * pipes.count + 2 * w, top + 40), words);
    words = new Wordnode(new Word(
            Integer.toString(pipes.count),
```

```
        w * pipes.count + 2 * w, top + 60), words);
PipeIterator temp = pipes.head();
temp.next();
tt = temp.pos.top;
tb = temp.pos.bottom;
lines = new Linenode(new Line(0, h * (2 + pipes.count)
        + top,
        w * (1 + pipes.count),
        h * (2 + pipes.count) + top), lines);
for (int i = 1; i <= pipes.count; i++) {
    int t = h * temp.pos.top + top;
    int b = bottom - h * temp.pos.bottom;
    words = new Wordnode(new Word(Integer.toString(
            temp.pos.bottom), w * i + 4, b), words);
    words = new Wordnode(new Word(Integer.toString(
            temp.pos.top), w * i + 4, t + h), words);
    lines = new Linenode(new Line(w * i, b, w * i, t),
            lines);
    x = findtypet(temp.pos.top, temp.pos.next.top,
            temp.pos.previous.top);
    y = findtypeb(temp.pos.bottom, temp.pos.next.bottom,
            temp.pos.previous.bottom);

    word2 = word2 + findcountab(x, y);
    words = new Wordnode(new Word(Integer.toString(x),
            i * 30 + 2, top + h * temp.pos.top + 15),
            words);
```

```
            temp.next();
        }
        words = new Wordnode(new Word(Integer.toString(word2),
                w * pipes.count + 2 * w, top + 80), words);
        top = bottom + 25;
        System.out.println(word2);
    }
    public void sp() {
        if (rm) {
            rmove();
            rmove();
        }
        // if (se)
        // savetree();

        final int W = 21 * pipes.count;
        final int H = top;
        boolean changed = false;
        int x = 0;
        int y = 0;
        Rectangle rect = new Rectangle(x, top - 40, W, H);
        drawingPane.scrollRectToVisible(rect);

        int this_width = (x + W + 20 + 1100);
        if (this_width > area.width) {
            area.width = this_width;
            changed = true;
```

```
        }

        int this_height = (y + H + 1024);
        if (this_height > area.height) {
            area.height = this_height;
            changed = true;
        }

        if (changed) {
            drawingPane.setPreferredSize(area);
            drawingPane.revalidate();
        }
        drawingPane.repaint();
    }
    // 2 to 1 left
    public void drawa(boolean save) {
        element = element + "a";
        PipeIterator temp = pipes.head();
        if (temp.pos.next.top == 1) {
            pipes.insertAfter(2, temp.pos.next.bottom + 1, temp);
        }
        temp.next();
        while (temp.pos.top > 2) {
            temp.pos.top--;
            temp.next();
        }
        temp.pos.top--;
```

```
        //if(se)
         temp.pos.changeabc(save ,"y",3,true);
        temp.pos.changeac(save, "", 15);
        //temp.pos.changeabc(save );
        temp.next();
        while (temp.pos.top > 1) {
            temp.next();
        }
        //if(!se)
        temp.pos.changeabc(save, "x", 5, true);
        temp.pos.changeab(save, "", 15);
        while (temp.pos.top > 0) {
            temp.pos.top++;
            temp.next();
        }
        if (save) {
            sp();
        }
    }
    // 3 to 2 right left
    public void drawb(boolean save) {
        element = element + "b";
        PipeIterator temp = pipes.head();
        PipeIterator topp = pipes.findtop();
        if (temp.pos.previous.top == 1) {
            int b = temp.pos.previous.bottom;
            pipes.insertBefore(3, b + 2, temp);
```

```
            pipes.insertBefore(2, b + 1, temp);
        } else if (topp.pos.next.top == 2) {
            pipes.insertAfter(3, 1 + Math.max(topp.pos.bottom,
                    topp.pos.next.bottom), topp);
        }
        temp.previous();
        while (temp.pos.top > 2) {
            temp.pos.top++;
            temp.previous();
        }
        temp.pos.top++;
        temp.previous();
        while (temp.pos.top > 3) {
            temp.previous();
        }
        temp.pos.changeab(save, "", 15);
        temp.pos.changeabc(save, "b", 2);
        while (temp.pos.top > 1) {
            temp.pos.top--;
            temp.previous();
        }
        if (save) {
            sp();
        }
    }
    // 3 to 2 left right
    public void drawc(boolean save) {
```

```
element = element + "c";
PipeIterator temp = pipes.head();
PipeIterator topp = pipes.findtop();
if (temp.pos.next.top == 1) {
    int b = temp.pos.next.bottom;
    pipes.insertAfter(3, b + 2, temp);
    pipes.insertAfter(2, b + 1, temp);
} else if (topp.pos.previous.top == 2) {
    pipes.insertBefore(3, 1 + Math.max(topp.pos.bottom,
            topp.pos.previous.bottom), topp);
}
temp.next();
while (temp.pos.top > 2) {
    temp.pos.top++;
    temp.next();
}
temp.pos.top++;
//temp.pos.changeab(save);
temp.next();
while (temp.pos.top > 3) {
    temp.next();
}
temp.pos.changeac(save, "", 15);
temp.pos.changeabc(save, "c", 2);
while (temp.pos.top > 1) {
    temp.pos.top--;
    temp.next();
```

```
    }
    if (save) {
        sp();
    }
}
// 2 to 1 right
public void drawA(boolean save) {
    element = element + "A";
    PipeIterator temp = pipes.head();
    if (temp.pos.previous.top == 1) {
        pipes.insertBefore(2, temp.pos.previous.bottom + 1,
                temp);
    }
    temp.previous();
    while (temp.pos.top > 2) {
        temp.pos.top--;
        temp.previous();
    }
    temp.pos.changeabc(save, "X", 4, true);

    temp.pos.changeab(save, "", 15);
    temp.pos.top--;
    temp.previous();
    while (temp.pos.top > 1) {
        temp.previous();
    }
    temp.pos.changeabc(save, "Y",6,true);
```

```
        temp.pos.changeac(save, "", 15);
        while (temp.pos.top > 0) {
            temp.pos.top++;
            temp.previous();
        }
        if (save) {
            sp();
        }
    }
    // 3 to 2 right right
    public void drawB(boolean save) {
        element = element + "B";
        PipeIterator temp = pipes.head();
        if (temp.pos.previous.top == 1) {
            int b = temp.pos.previous.bottom;
            pipes.insertBefore(2, b + 1, temp);
            pipes.insertBefore(3, b + 2, temp);
        } else if (temp.pos.previous.top == 2) {
            pipes.insertBefore(3, temp.pos.previous.bottom + 1,
                    temp);
        }
        temp.previous();
        while (temp.pos.top > 3) {
            temp.pos.top--;
            temp.previous();
        }
        // temp.pos.changeac(save);
```

```
        temp.pos.top--;
        temp.previous();
        while (temp.pos.top > 2) {
            temp.previous();
        }
        temp.pos.changeab(save, "", 15);
        temp.pos.changeabc(save, "B", 1);
        while (temp.pos.top > 1) {
            temp.pos.top++;
            temp.previous();
        }
        if (save) {
            sp();
        }
    }
    // 3 to 2 left left
    public void drawC(boolean save) {
        element = element + "C";
        PipeIterator temp = pipes.head();
        if (temp.pos.next.top == 1) {
            int b = temp.pos.next.bottom;
            pipes.insertAfter(2, b + 1, temp);
            pipes.insertAfter(3, b + 2, temp);
        } else if (temp.pos.next.top == 2) {
            pipes.insertAfter(3, temp.pos.next.bottom + 1, temp);
        }
        temp.next();
```

```
        while (temp.pos.top > 3) {
            temp.pos.top--;
            temp.next();
        }
        temp.pos.top--;
        temp.next();
        while (temp.pos.top > 2) {
            temp.next();
        }
        temp.pos.changeac(save, "", 15);
        temp.pos.changeabc(save, "C", 1);
        while (temp.pos.top > 1) {
            temp.pos.top++;
            temp.next();
        }
        if (save) {
            sp();
        }
    }
    public void undo() {
        boolean draw = true;
        if (pipes.count == 0) {

            pipes = new Linkedpipes();
            pipes.insertAfter(1, 1, pipes.head());
        }
```

```
        char c = element.charAt(element.length() - 1);
        switch (c) {
            case 'a':
                drawA(draw);
                break;
            case 'b':
                drawB(draw);
                break;
            case 'c':
                drawC(draw);
                break;
            case 'A':
                drawa(draw);
                break;
            case 'B':
                drawb(draw);
                break;
            case 'C':
                drawc(draw);
                break;
        }
        element = element.substring(0, element.length() - 2);
        sp();
    }
    public Linkedpipes getPipes() {
        return pipes;
    }
```

```
public void setPipes(Linkedpipes pipes) {
    this.pipes = pipes;
}
public String getElement() {
    return element;
}
public void setElement(String element) {
    this.element = element;
}
public Linenode getLines() {
    return lines;
}
public void setLines(Linenode lines) {
    this.lines = lines;
}
public Wordnode getWords() {
    return words;
}
public void setWords(Wordnode words) {
    this.words = words;
}
public int getTop() {
    return top;
}
public void setTop(int top) {
    this.top = top;
}
```

```
public int getPages() {
    return pages;
}
public void setPages(int pages) {
    this.pages = pages;
}
public boolean isAt() {
    return at;
}
public void setAt(boolean at) {
    this.at = at;
}
public boolean isRm() {
    return rm;
}
public void setRm(boolean rm) {
    this.rm = rm;
}
public boolean isSe() {
    return se;
}
public void setSe(boolean se) {
    this.se = se;
}
}
```

```
package Groupf;
/* GroupF.java
 * @author  Jennifer schofield */
import java.beans.PropertyVetoException;
import java.io.*;
import java.util.logging.*;
import javax.swing.JMenuItem;
public class GroupF extends javax.swing.JFrame {
    private JMenuItem savelength;
    /** Creates new form GroupF */
    public GroupF() {
        initComponents();
        numnewfiles = 0;
        this.newMenuItemActionPerformed();
        last = 6;
    }
    // This function sets up the application for Groupf
    private void initComponents() {
        last = -1;
        jToolBar1 = new javax.swing.JToolBar();
        jButtonA = new KeyButton(this, 3, "a12",
                javax.swing.KeyStroke.getKeyStroke('a'));
        jButtonB = new KeyButton(this, 4, "b123",
                javax.swing.KeyStroke.getKeyStroke('b'));
        jButtonC = new KeyButton(this, 5, "c321",
                javax.swing.KeyStroke.getKeyStroke('c'));
        JButtoma = new KeyButton(this, 0, "A21",
```

```
            javax.swing.KeyStroke.getKeyStroke('r'));
jButtonb = new KeyButton(this, 1, "B132",
            javax.swing.KeyStroke.getKeyStroke('e'));
jButtonc = new KeyButton(this, 2, "C231",
            javax.swing.KeyStroke.getKeyStroke('w'));
app = new javax.swing.JDesktopPane();
menuBar = new javax.swing.JMenuBar();
fileMenu = new javax.swing.JMenu();
editmenu = new javax.swing.JMenu();
doMenu = new javax.swing.JMenu();
newMenuItem = new javax.swing.JMenuItem();
openMenuItem = new javax.swing.JMenuItem();
jSeparator1 = new javax.swing.JSeparator();
saveMenuItem = new javax.swing.JMenuItem();
saveAsMenuItem = new javax.swing.JMenuItem();
savelength = new javax.swing.JMenuItem();
jSeparator2 = new javax.swing.JSeparator();
printMenuItem = new javax.swing.JMenuItem();
jSeparator3 = new javax.swing.JSeparator();
exitMenuItem = new javax.swing.JMenuItem();
helpMenu = new javax.swing.JMenu();
contentMenuItem = new javax.swing.JMenuItem();
aboutMenuItem = new javax.swing.JMenuItem();
randomMenuItem = new javax.swing.JMenuItem();
randomelementMenuItem = new javax.swing.JMenuItem();
randompipesMenuItem = new javax.swing.JMenuItem();
op = new javax.swing.JMenu();
```

```
se = new javax.swing.JCheckBoxMenuItem();
at = new javax.swing.JCheckBoxMenuItem();
rm = new javax.swing.JCheckBoxMenuItem();
undoMenuItem = new javax.swing.JMenuItem();
dEMenuItem = new javax.swing.JMenuItem();
;
nEMenuItem = new javax.swing.JMenuItem();
coppyEMenuItem = new javax.swing.JMenuItem();
pasteMenuItem = new javax.swing.JMenuItem();
element = null;
setDefaultCloseOperation(
        javax.swing.WindowConstants.EXIT_ON_CLOSE);
jToolBar1.setBorder(null);
jToolBar1.setRollover(true);
jToolBar1.add(jButtonA);
jToolBar1.add(jButtonB);
jToolBar1.add(jButtonC);
jToolBar1.add(JButtoma);
jToolBar1.add(jButtonb);
jToolBar1.add(jButtonc);
fileMenu.setText("File");
newMenuItem.setText("New");
newMenuItem.setAccelerator(
        javax.swing.KeyStroke.getKeyStroke('n'));
newMenuItem.addActionListener(
        new java.awt.event.ActionListener() {
    public void actionPerformed(
```

```
            java.awt.event.ActionEvent evt) {
        newMenuItemActionPerformed();
    }
});
fileMenu.add(newMenuItem);
openMenuItem.setText("Open");
openMenuItem.addActionListener(
        new java.awt.event.ActionListener() {
    public void actionPerformed(
            java.awt.event.ActionEvent evt) {
        openMenuItemActionPerformed();
    }
});
fileMenu.add(openMenuItem);
fileMenu.add(jSeparator1);
saveMenuItem.setText("Save");
saveMenuItem.addActionListener(
        new java.awt.event.ActionListener() {
    public void actionPerformed(
            java.awt.event.ActionEvent evt) {
        saveMenuItemActionPerformed();
    }
});
fileMenu.add(saveMenuItem);
saveAsMenuItem.setText("Save As ...");
saveAsMenuItem.addActionListener(
        new java.awt.event.ActionListener() {
```

```
    public void actionPerformed(
            java.awt.event.ActionEvent evt) {
        saveAsMenuItemActionPerformed();
    }
});
fileMenu.add(saveAsMenuItem);
fileMenu.add(jSeparator2);
printMenuItem.setText("Print");
printMenuItem.addActionListener(
        new java.awt.event.ActionListener() {
    public void actionPerformed(
            java.awt.event.ActionEvent evt) {
        printMenuItemActionPerformed();
    }
});
fileMenu.add(printMenuItem);
fileMenu.add(jSeparator3);
exitMenuItem.setText("Exit");
exitMenuItem.setAccelerator(
        javax.swing.KeyStroke.getKeyStroke('q'));
exitMenuItem.addActionListener(
        new java.awt.event.ActionListener() {
    public void actionPerformed(
            java.awt.event.ActionEvent evt) {
        exitMenuItemActionPerformed();
    }
});
```

```
fileMenu.add(exitMenuItem);
menuBar.add(fileMenu);
editmenu.setText("Edit");
undoMenuItem.setText("Undo");
undoMenuItem.setAccelerator(
        javax.swing.KeyStroke.getKeyStroke('u'));
undoMenuItem.addActionListener(
        new java.awt.event.ActionListener() {
    public void actionPerformed(
            java.awt.event.ActionEvent evt) {
        undo();
    }
});
editmenu.add(undoMenuItem);
dEMenuItem.setText("Delete Element");
dEMenuItem.setAccelerator(
        javax.swing.KeyStroke.getKeyStroke('d'));
dEMenuItem.addActionListener(
        new java.awt.event.ActionListener() {
    public void actionPerformed(
            java.awt.event.ActionEvent evt) {
        getcurrent().getGPicture1().delete();
    }
});
savelength.setText("savelength");
dEMenuItem.setAccelerator(
        javax.swing.KeyStroke.getKeyStroke('l'));
```

```
dEMenuItem.addActionListener
        (new java.awt.event.ActionListener() {

    public void actionPerformed(
            java.awt.event.ActionEvent evt) {
        getcurrent().getGPicture1().savelength();
    }
});
editmenu.add(dEMenuItem);
nEMenuItem.setText("New Element");
nEMenuItem.setAccelerator(
        javax.swing.KeyStroke.getKeyStroke('e'));
nEMenuItem.addActionListener(
        new java.awt.event.ActionListener() {
    public void actionPerformed(
            java.awt.event.ActionEvent evt) {
        getcurrent().getGPicture1().newelement();
    }
});
editmenu.add(nEMenuItem);
coppyEMenuItem.setAccelerator(
        javax.swing.KeyStroke.getKeyStroke(
        java.awt.event.KeyEvent.VK_C,
        java.awt.event.InputEvent.SHIFT_MASK));
coppyEMenuItem.setText(("Copy Element")); // NOI18N
coppyEMenuItem.setName("jMenuItem6"); // NOI18N
coppyEMenuItem.addActionListener(
```

```
        new java.awt.event.ActionListener() {

    public void actionPerformed(
            java.awt.event.ActionEvent evt) {
        element = getcurrent().
                getGPicture1().getElement();
    }
});
editmenu.add(coppyEMenuItem);
pasteMenuItem.setAccelerator(
        javax.swing.KeyStroke.getKeyStroke(
        java.awt.event.KeyEvent.VK_P,
        java.awt.event.InputEvent.SHIFT_MASK));
pasteMenuItem.setText(("Paste Element"));
pasteMenuItem.setName("jMenuItem7");
editmenu.add(pasteMenuItem);
menuBar.add(editmenu);
doMenu.setText("Do");
randomMenuItem.setText("Random");
randomMenuItem.setAccelerator(
        javax.swing.KeyStroke.getKeyStroke('R'));
randomMenuItem.addActionListener(
        new java.awt.event.ActionListener() {
    public void actionPerformed(
            java.awt.event.ActionEvent evt) {
        random();
    }
```

```
});
randomelementMenuItem.setText("Random element");
randomelementMenuItem.setAccelerator(
        javax.swing.KeyStroke.getKeyStroke('e'));
randomelementMenuItem.addActionListener(
        new java.awt.event.ActionListener() {
    public void actionPerformed(
            java.awt.event.ActionEvent evt) {
        int last = 10;
        String intStr;
        intStr = javax.swing.JOptionPane.
                showInputDialog(
                null, "Enter word length.");
        if (intStr != null) {
            int length = Integer.parseInt(intStr);
            newMenuItemActionPerformed();
            while (length > getcurrent().getGPicture1().
                    getElement().length()) {
                random();
            }
            getcurrent().getGPicture1().setElement("");
            getcurrent().getGPicture1().sp();
        }
    }
});
randompipesMenuItem.setText("Random pipes");
randompipesMenuItem.setAccelerator(
```

```
        javax.swing.KeyStroke.getKeyStroke('p'));
randompipesMenuItem.addActionListener(
        new java.awt.event.ActionListener() {
    public void actionPerformed(
            java.awt.event.ActionEvent evt) {
        String intStr;
        intStr = javax.swing.JOptionPane.
                showInputDialog(null,
                "Enter an number of pipes");
        if (intStr != null) {
            int n = Integer.parseInt(intStr);
            int length = n;
            newMenuItemActionPerformed();
            while (length > getcurrent().getGPicture1().
                    getPipes().count) {
                random();
                getcurrent().getGPicture1().rmove();
            }
            javax.swing.JOptionPane.
                    showConfirmDialog(
                    null, "the word length is "
                    + getcurrent().
                    getGPicture1().getElement()
                    .length());
            getcurrent().getGPicture1()
                    .setElement("");
            getcurrent().getGPicture1().sp();
```

```
        } else {

            javax.swing.JOptionPane.
                    showConfirmDialog(null,
                    "the word length is ");
        }
    }
});
doMenu.add(randompipesMenuItem);
doMenu.add(randomelementMenuItem);
doMenu.add(randomMenuItem);
menuBar.add(doMenu);
op.setText(("Options"));
op.setName("op");
se.setSelected(false);
se.setText(("Seperate"));
se.setName("se"); //
op.add(se);
at.setSelected(true);
at.setText(("All together"));
at.setName("at");
op.add(at);
rm.setSelected(true);
rm.setText(("Remove trival pipes"));
rm.setName("rm");
op.add(rm);
menuBar.add(op);
```

```
helpMenu.setText("Help");
contentMenuItem.setText("Contents");
helpMenu.add(contentMenuItem);
aboutMenuItem.setText("About");
helpMenu.add(aboutMenuItem);
menuBar.add(helpMenu);
setJMenuBar(menuBar);
org.jdesktop.layout.GroupLayout layout =
        new org.jdesktop.layout.
        GroupLayout(getContentPane());
getContentPane().setLayout(layout);
layout.setHorizontalGroup(layout.
        createParallelGroup(
        org.jdesktop.layout.
        GroupLayout.LEADING).add(jToolBar1,
        org.jdesktop.layout.
        GroupLayout.DEFAULT_SIZE, 496,
        Short.MAX_VALUE).add(app));
layout.setVerticalGroup(layout.
        createParallelGroup(
        org.jdesktop.layout.
        GroupLayout.LEADING)
        .add(layout.createSequentialGroup().
        add(jToolBar1,
        org.jdesktop.layout.
        GroupLayout.PREFERRED_SIZE,
        org.jdesktop.layout.
```

```
                GroupLayout.DEFAULT_SIZE,
                org.jdesktop.layout.GroupLayout.
                PREFERRED_SIZE).addPreferredGap(
                org.jdesktop.layout.LayoutStyle.
                RELATED).add(app)));
        pack();
    }
    // Produces a randon number between 1 and 6 and send
     //it to the act function
     //to help create a computer generated woord.
    public int random() {
        double r = java.lang.Math.random();
        int ra = (int) java.lang.Math.round(r * 100) % 6;
        act(ra, false);
        return ra;
    }
    /* This function will undo the last
     * thing in the current Gfile*/
    public void undo() {
        this.getcurrent().getGPicture1().undo();
    }
    /* @param ra tells what button was press a
     * nd calls the right action
     * on the current Gfile
     * @param draw decides if the picture will change*/
    public void act(int ra, boolean draw) {
        getcurrent().getGPicture1().setAt(at.getState());
```

```
        getcurrent().getGPicture1().setSe(se.getState());
        getcurrent().getGPicture1().setRm(rm.getState());
        switch (ra) {
            case 0:
                getcurrent().getGPicture1().drawA(draw);
                break;
            case 1:
                getcurrent().getGPicture1().drawB(draw);
                break;
            case 2:

                getcurrent().getGPicture1().drawC(draw);
                break;
            case 3:
                getcurrent().getGPicture1().drawa(draw);
                break;
            case 4:
                getcurrent().getGPicture1().drawb(draw);

                break;
            case 5:
                getcurrent().getGPicture1().drawc(draw);
                break;
        }
    }
    // @return the active Gfile
    public Gfile getcurrent() {
```

```
        Filenode temp = files;
        while (!app.getSelectedFrame().
                equals(temp.element)) {
            temp = temp.next;
        }
        return temp.element;
    }
    //Makes a new Gfile
    private void newMenuItemActionPerformed() {
        try {
            numnewfiles++;
            Gfile temp = new Gfile();
            temp.setVisible(true);
            temp.setBounds(20, 50, 276, 248);
            app.add(temp, javax.swing.
                    JLayeredPane.DEFAULT_LAYER);
            temp.setTitle("Pipes " +
                    java.lang.Integer.
                    toString(numnewfiles));
            files = new Filenode(temp, files);
            app.setSelectedFrame(temp);
            temp.setSelected(true);
        } catch (PropertyVetoException ex) {
            Logger.getLogger("global").
                    log(Level.SEVERE, null, ex);
        }
    }
```

```
//Opens a previous saved file made from this program.
 private void openMenuItemActionPerformed() {
     try {
         javax.swing.JFileChooser open =
                 new javax.swing.JFileChooser();
         if (open.showOpenDialog(this) ==
                 javax.swing.JFileChooser.APPROVE_OPTION) {
             java.io.File file = open.getSelectedFile();
             Gfile temp = new Gfile(file);
             temp.setVisible(true);
             temp.setBounds(20, 50, 276, 248);
             app.add(temp, javax.swing.
                     JLayeredPane.DEFAULT_LAYER);
             temp.setTitle(file.getName());
             files = new Filenode(temp, files);
         }
     } catch (IOException ex) {
         Logger.getLogger("global").
                 log(Level.SEVERE, null, ex);
     }
 }
 /** Saves the current Gfile     */
 private void saveMenuItemActionPerformed() {
     if (this.getcurrent().getName() == null) {
         this.saveAsMenuItemActionPerformed();
     } else {
         try {
```

```
            this.getcurrent().save();
        } catch (IOException ex) {
            Logger.getLogger("global").log(Level.SEVERE,
                    null, ex);
        }
    }
}
/** Allows the user to specifie the name of an Gfile
 * and then saves in     */
private void saveAsMenuItemActionPerformed() {
    try {
        javax.swing.JFileChooser saveas =
                new javax.swing.JFileChooser();
        if (saveas.showSaveDialog(this) ==
                javax.swing.JFileChooser.APPROVE_OPTION) {
            java.io.File file = saveas.getSelectedFile();
            this.getcurrent().saveas(file);
        }
    } catch (IOException ex) {
        Logger.getLogger("global").log(Level.SEVERE,
                null, ex);
    }
}
/** prints the current Gfile      */
private void printMenuItemActionPerformed() {
    PrintUtilities.printComponent(
            this.getcurrent().getGPicture1());
```

```
}
/**
 * This shut the program down after it give the option to
 * save each Gfile
 */
private void exitMenuItemActionPerformed() {
    int o = javax.swing.JOptionPane.showConfirmDialog(
            this, "Do you want to save?");
    switch (o) {
        case javax.swing.JOptionPane.YES_OPTION:
            this.saveMenuItemActionPerformed();
            System.exit(0);
            break;
        case javax.swing.JOptionPane.NO_OPTION:
            System.exit(0);
            break;
    }
}
/**This starts the program.
 * @param args the command line argumen
 */
public static void main(String[] args) {
    java.awt.EventQueue.invokeLater(new Runnable() {

        public void run() {
            new GroupF().setVisible(true);
        }
```

```
    });
}
private javax.swing.JDesktopPane app;
private KeyButton jButtonA;
private javax.swing.JButton JButtoma;
private javax.swing.JButton jButtonB;
private javax.swing.JButton jButtonC;
private javax.swing.JButton jButtonb;
private javax.swing.JButton jButtonc;
private javax.swing.JSeparator jSeparator1;
private javax.swing.JSeparator jSeparator2;
private javax.swing.JSeparator jSeparator3;
private javax.swing.JToolBar jToolBar1;
private javax.swing.JMenuBar menuBar;
private javax.swing.JMenu fileMenu;
private javax.swing.JMenu editmenu;
private javax.swing.JMenu doMenu;
private javax.swing.JMenu helpMenu;
private javax.swing.JMenuItem contentMenuItem;
private javax.swing.JMenuItem exitMenuItem;
private javax.swing.JMenuItem aboutMenuItem;
private javax.swing.JMenuItem newMenuItem;
private javax.swing.JMenuItem openMenuItem;
private javax.swing.JMenuItem printMenuItem;
private javax.swing.JMenuItem saveAsMenuItem;
private javax.swing.JMenuItem saveMenuItem;
private javax.swing.JMenuItem randomMenuItem;
```

```
    private javax.swing.JMenuItem randomelementMenuItem;
    private javax.swing.JMenuItem randompipesMenuItem;
    private javax.swing.JMenuItem undoMenuItem;
    private javax.swing.JMenuItem dEMenuItem;
    private javax.swing.JMenuItem nEMenuItem;
    private javax.swing.JMenuItem coppyEMenuItem;
    private javax.swing.JMenuItem pasteMenuItem;
    private javax.swing.JMenu op;
    private javax.swing.JCheckBoxMenuItem at;
    private javax.swing.JCheckBoxMenuItem se;
    private javax.swing.JCheckBoxMenuItem rm;
    private javax.swing.JMenuItem MenuItem;
    private Filenode files;
    private int numnewfiles;
    private int last;
    private String element;
}
```

```
package Groupf;

import java.awt.event.ActionEvent;
import javax.swing.*;
import javax.swing.plaf.ActionMapUIResource;

public class KeyButton extends javax.swing.JButton {

    public KeyButton(GroupF o, int i, String t, KeyStroke ke) {
        owner = o;
        this.setText(t);

        addActionListener(new java.awt.event.ActionListener() {

            public void actionPerformed(
                    java.awt.event.ActionEvent evt) {

                owner.act(num, true);
            } });
        this.setAccelerator(ke);
        num = i;
        setFocusable(false);
        setHorizontalTextPosition(
                javax.swing.SwingConstants.CENTER);
        setVerticalTextPosition(
                javax.swing.SwingConstants.BOTTOM);
    }
```

```
    public void setAccelerator(KeyStroke ke) {
        InputMap keyMap = new ComponentInputMap(this);
        keyMap.put(ke, "action");

        ActionMapUIResource actionMap =
                new ActionMapUIResource();
        actionMap.put("action", this.action);

        SwingUtilities.replaceUIActionMap(this, actionMap);
        SwingUtilities.replaceUIInputMap(this,
                JComponent.WHEN_IN_FOCUSED_WINDOW, keyMap);
    }
    private Action action = new AbstractAction("Action Name") {

        public void actionPerformed(ActionEvent evt) {
            owner.act(num, true);
        }};
    GroupF owner;
    int num;
}
```

```
package Groupf;

/*
 * Line.java
 */

/**
 *
 * @author jenniferschofield
 */
public class Line {

    int x1;

    int x2;
    int y1;
    int y2;

    public Line(int a, int b, int c, int d) {
        x1=a;
        y1=b;
        x2=c;
        y2=d;
    }

}
```

```
package Groupf;



/**
 * Linenode.java
 * @author jenniferschofield
 */
public class Linenode {
    Line element;
    Linenode next ;

    public Linenode(Line e, Linenode n) {
        element =e;
        next =n;
    }
    public Linenode(Line e){
        this(e,null);
    }

}
```

```
package Groupf;
/* LinkedList.java
 * @author Jennifer Schofield
 */

public class Linkedpipes {

    public PipeNode head;
    int count;
    public Linkedpipes() {
        head = new PipeNode(null, 0, 0, null);
        head.next = head.previous = head;
        count = 0;
    }
    void insertAfter(int t, int b, PipeIterator cursor) {
        PipeNode newItem = new PipeNode(cursor.pos, t, b,
                cursor.pos.next);
        newItem.next.previous = newItem;
        cursor.pos.next = newItem;
        count++;
    }
    void insertBefore(int t, int b, PipeIterator cursor) {
        PipeNode newItem = new PipeNode(cursor.pos.previous,
                t, b, cursor.pos);
        newItem.previous.next = newItem;
        cursor.pos.previous = newItem;
        count++;
```

```
    }
    void remove(PipeIterator cursor) {
        cursor.pos.previous.next = cursor.pos.next;
        cursor.pos.next.previous = cursor.pos.previous;
        count--;
    }
    final PipeIterator head() {
        return new PipeIterator(this, head);
    }
    PipeIterator findtop() {
        PipeIterator temp = head();
        temp.next();
        while (temp.pos.top > 1) {
            temp.next();
        }
        return temp;
    }
}
```

```
package Groupf;
// PipeNode.java  @author jenniferschofield
public class PipeNode {
    int top;
    int bottom;
    PipeNode next;
    PipeNode previous;
    int ablength;
    int aclength;
    double abclength;
    Wordnode element;
    public PipeNode(PipeNode p, int t, int b,  PipeNode n) {
        previous=p;
        top=t;
        bottom=b;
        next=n;
        ablength=0;
        aclength=0;
        abclength=0;
        element=null;
    }
    public void changeab(boolean record, String k, int h) {
        if (record) {
            ablength++;
            element = new Wordnode(new Word(k, 0, h),
                    element);
        }    }
```

```
    public void changeabc(boolean record, String k, int h) {
        if (record) {
            abclength++;
            element = new Wordnode(new Word(k, 0, h),
                    element);
        }    }
    public void changeabc(boolean record, String k, int h,
            boolean t) {
        if (record) {
            abclength += .5;
            element = new Wordnode(new Word(k, 0, h),element);
        }    }
    public void changeac(boolean record, String k, int h) {
        if (record) {
            aclength++;
            element = new Wordnode(new Word(k, 0, h),
                    element);
        }   }
}
```

```
package Groupf;

/* PipeItorator.java
 * @author jenniferschofield
 */
final class PipeIterator {
    Linkedpipes owner;
    PipeNode pos;
    PipeIterator(Linkedpipes owner, PipeNode pos) {
        this.owner = owner;
        this.pos = pos;
    }
    public boolean belongsTo(Object owner) {
        return this.owner == owner;
    }
    public void head() {
        pos = owner.head;
    }
    public void next() {
        pos = pos.next;
    }
    public void previous() {
        pos = pos.previous;
    }
}
```

```
package Groupf;

/*
 * Word.java
 * @author jenniferschofield
 */
public class Word {

    String w;
    int x;
    int y;
    public Word(String a, int b, int c) {
        w=a;
        x=b;
        y=c;
    }

}
```

```
package Groupf;



/**
 *
 * @author jenniferschofield
 */
public class Wordnode {
    Word element;

    Wordnode next ;

    public Wordnode(Word e, Wordnode n) {
        element =e;
        next =n;
     }
    public Wordnode(Word e){
        this(e,null);
    }
}
```

BIBLIOGRAPHY

[1] Aaron Peterson, Pipe Diagrams for Thompsons Group F, Masters Thesis, Brigham Young University, 2007, 66 pages.

[2] J. W. Cannon, W. J. Floyd, W. R. Parry, Introductory Notes on Richard Thompsons groups, L'Enseignement Mathématique 42 (1996), 215-256.

[3] Chomsky and Schutzenberger, *The algebraic theory of context-free languages*, Computer Programming and Formal Systems (1963), 118–161,

[4] J. M. Belk and K. S. Brown, *Forest Diagrams for Thompson's group F*, Internat. J. Algebra Comput. **15** (2005), no. 5–6, 815–850.

[5] S. B. Fordham, *Minimal length elements of Thompson's group F*, Geom. Dedicata, **99** (2003), 179–220.

[6] B. Woodruff, *Statistical Properties of Thompson's Group F*, http://contentdm.lib.byu.edu/cdm4/item_viewer.php?CISOROOT=/ETD &CISOPTR=388&CISOBOX=1&REC=1, Accessed July 12, 2007.

[7] R. J. Thompson, *Thompson's notes on the groups F and T*, unpublished manuscript.

[8] R. J. Thompson, *Embeddings into finitely generated simple groups which preserve the word problem*, Word Problems II: The Oxford Book (S. I. Adian, W. W. Boone and G. Higman, es.), Studies in Logic and the Foundations of Mathematics, vol. 95, North-Holland, Amsterdam, 1980, 401–441.