

Deciding Conjugacy in Thompson’s Group F in Linear Time

Nabil Hossain, Robert W. McGrail, and James Belk
 The Laboratory for Algebraic and Symbolic Computation
 Reem Kayden Center for Science and Computation
 Bard College
 Annandle-on-Hudson, New York 12528

Francesco Matucci
 Département de Mathématiques
 Faculté des Sciences d’Orsay
 Université Paris-Sud 11
 Bâtiment 425, Bureau 21 F-91405 Orsay Cedex, France

Abstract—We present an efficient implementation of the solution to the conjugacy problem in Thompson’s group F , a certain infinite group whose elements are piecewise-linear homeomorphisms of the unit interval $[0, 1]$. This algorithm checks for conjugacy by constructing and comparing directed graphs called strand diagrams. We provide a comprehensive description of our solution algorithm, including the data structure that stores strand diagrams and methods to simplify them. We prove that our algorithm theoretically achieves an $O(n)$ bound in the size of the input, and we present a $O(n^2)$ working solution.

I. INTRODUCTION

Given a finitely-presented group G , the **conjugacy problem** is the decision problem of determining whether two elements $g, h \in G$ are conjugate, i.e. whether there exists an element $k \in G$ so that $g = khk^{-1}$. This problem cannot be solved in general [1], but solutions are known for many classes of infinite groups, including free groups, braid groups, and so forth [2], [3]. Moreover, the conjugacy problem in free groups has been proven to be solvable in linear time [2].

Thompson’s group F is a certain infinite group of piecewise-linear homeomorphisms of the unit interval. It can be described by a presentation with two generators and two relations:

$$F = \langle x_0, x_1 \mid x_2x_1 = x_1x_3, x_3x_1 = x_1x_4 \rangle$$

where x_n is shorthand for $x_0^{1-n}x_1x_0^{n-1}$ for $n \geq 2$.

This group is well-known in geometric group theory, and has been studied extensively. See [4] for a comprehensive introduction to F .

Guba and Sapir [5], [6] provided a solution to the conjugacy problem in Thompson’s group F using graphs called *diagrams*. Building upon this solution, Belk and Matucci [7] introduced certain directed graphs called **strand diagrams**, and described a reduction of the conjugacy problem in group F to isotopy of strand diagrams. In this paper we describe a further reduction of this problem to isomorphism of planar graphs and include an implementation of this reduction. This reduction algorithm takes as input two words in $\{x_0, x_1\}$ of length at most n , and produces two planar graphs in

Francesco Matucci gratefully acknowledges the Fondation Mathématique Jacques Hadamard (FMJH - ANR - Investissement d’Avenir for the support received during the development of this work.

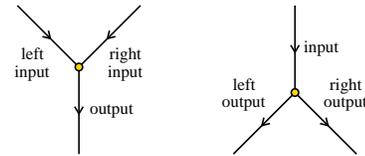


Fig. 1. A merge and a split (image taken from [7]).

$O(n)$ time. Given a linear time algorithm for determining isomorphism of planar graphs, as is theoretically possible according to Hopcroft and Wong [8], this gives rise to a linear time algorithm for the conjugacy problem in Thompson’s Group F . The Hopcroft-Wong algorithm is quite complex, and no implementation of it currently exists. Therefore, we also describe a $O(n^2)$ Java program for checking conjugacy in F .

For an alternate approach to solving the conjugacy problem in F , the reader is referred to [9].

II. ANNULAR STRAND DIAGRAMS

All of the material in this section are taken from [7].

Definition 1. A **simple annular strand diagram** is a finite directed multigraph drawn on an annulus without edge crossings, having the following properties:

- 1) Each vertex has degree three, and is either a merge or a split (see Fig. 1).
- 2) Every directed cycle winds counterclockwise around the central hole.

An **annular strand diagram** is a simple annular strand diagram along with a finite number of **free loops**, which are directed cycles without any vertices.

In [7], Belk and Matucci describe a procedure for obtaining an annular strand diagram corresponding to any element of F . Specifically, let $B(x_0)$, $B(x_1)$, $B(x_0^{-1})$ and $B(x_1^{-1})$ denote the four building blocks shown in Fig. 2. Given any word $s_1s_2 \cdots s_n$ in F , i.e. any finite sequence of elements from $\{x_0, x_0^{-1}, x_1, x_1^{-1}\}$, the **corresponding annular strand diagram** is obtained by gluing together the building blocks $B(s_1), B(s_2), \dots, B(s_n)$ in counterclockwise order around

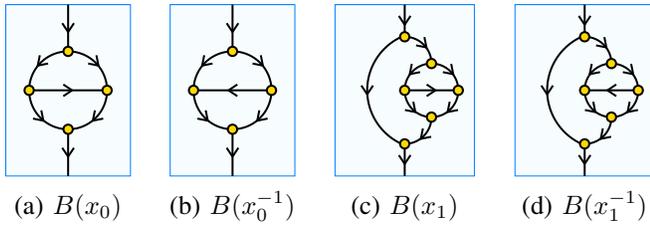


Fig. 2. The four building blocks $B(x_0), B(x_0^{-1}), B(x_1), B(x_1^{-1})$, corresponding to the generators for F and their inverses.

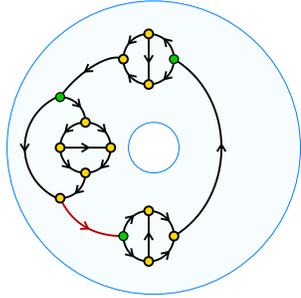


Fig. 3. Annular strand diagram corresponding to the word $x_0 x_0^{-1} x_1$. Each green split is marker for the start of its building block. The red edge glues the last building block to the first, creating a directed cycle.

the annulus, as shown in Fig. 3. Note that different words for the same element of F will correspond to different annular strand diagrams.

Before stating the conjugacy theorem from [7], we need a few more definitions:

- A **reduction** of an annular strand diagram is a simplification of the directed graph using one of the three moves shown in Fig. 4. This set of reductions is confluent and terminating [7], so every annular strand diagram reduces to a unique normal form. We say that an annular strand diagram has been **reduced** if it is a normal form for this rewriting system. Fig. 5 shows the reductions performed on an annular strand diagram until it is reduced.
- Two annular strand diagrams are said to be **isotopic** if the second can be obtained from the first by some continuous motion in the annulus. The edges of the diagram are not

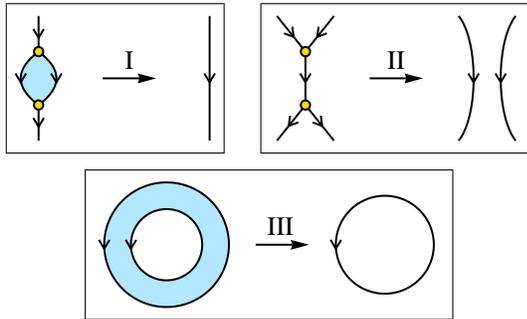


Fig. 4. The three reduction rules for annular strand diagrams. The blue indicates “empty regions”, i.e. regions which are devoid of vertices and edges, and do not contain the central hole of the annulus.

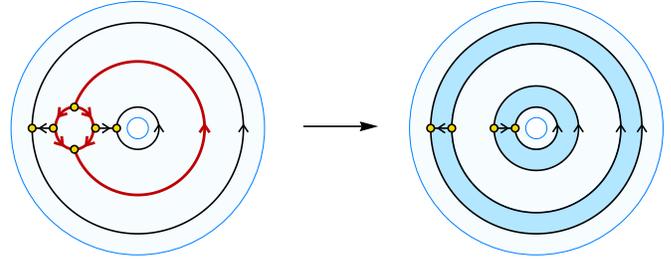


Fig. 6. A type II move at the red region splits the connected annular strand diagram into two connected components.

allowed to cross during this motion, and no edge may move across the central hole of the annulus.

We now present the solution to the conjugacy problem in F described by Belk and Matucci [7].

Theorem 1. *Let $a = a_1 \cdots a_m$ and $b = b_1 \cdots b_n$ be words representing elements of Thompson’s group F . Let A and B be the corresponding annular strand diagrams, and let A' and B' be reduced annular strand diagrams obtained by reducing A and B , respectively. Then a and b represent conjugate elements of F if and only if A' and B' are isotopic.*

Our algorithm simplifies annular strand diagrams through the rewrite rules of Fig. 4 in linear time before performing any check for isotopy of annular strand diagrams.

III. CHECKING FOR ISOTOPY

During the process of reduction, an annular strand diagram might become disconnected into several components. For example, Fig. 6 shows an annular strand diagram that splits into two components after a type II reduction. Keeping track of the relative positions of these components is one of the most difficult aspects of our algorithm.

Because every vertex of an annular strand diagram has at least one outgoing edge, every component must have at least one directed cycle. Since directed cycles are required to surround the central hole, it follows that the connected components of an annular strand diagram are arranged concentrically around the central hole of the annulus. We will use the following concept to keep track of the concentric order of these components.

Definition 2. Let A be an annular strand diagram. A **cutting path** in A is a path from the central hole of the annulus to the outside circle such that the cutting path crosses an edge only from left to right with respect to the orientation of the edge (see Fig. 7).

The following theorem explains the usefulness of cutting paths.

Theorem 2. *Let A_1, \dots, A_m be the connected components of a reduced annular strand diagram A' in concentric order, and let e_1, \dots, e_n be the sequence of edges crossed by a cutting path*

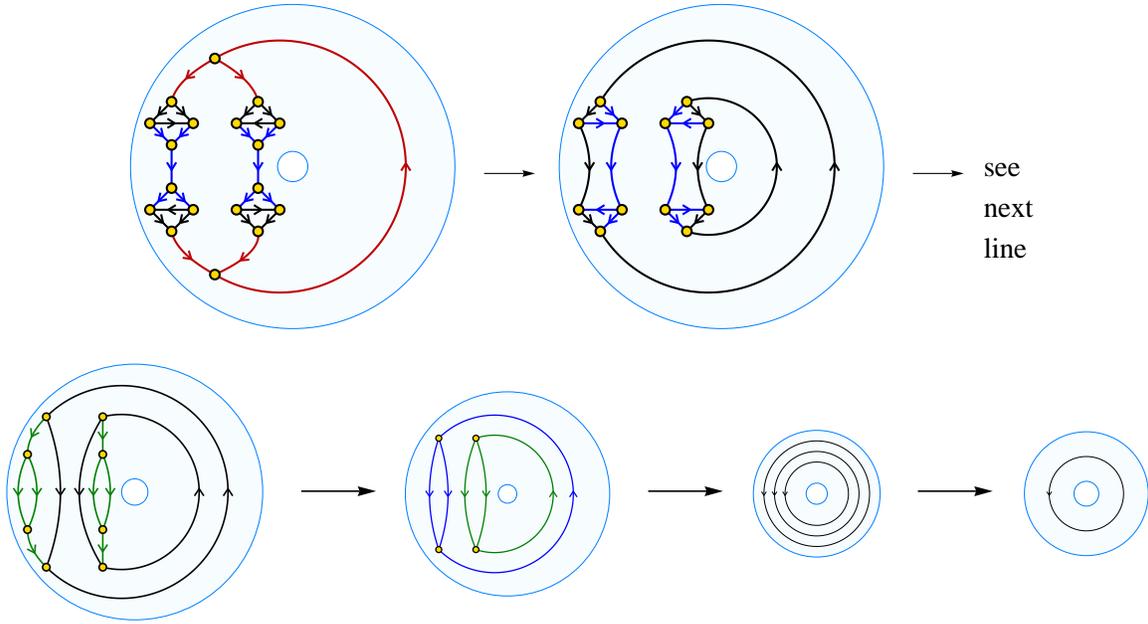


Fig. 5. Reducing an annular strand diagram to a free loop. The green regions are subject to type I moves, and the red and blue regions are each subject to type II moves.

for A' . Then e_1, \dots, e_n consists of one or more edges from A_1 followed by one or more edges from A_2 and so forth, ending with one or more edges from A_m .

To prove this theorem, we require the following lemma on the structure of reduced annular strand diagrams.

Lemma 1. *Let A' be a reduced annular strand diagram. Then each component of A' that is not a free loop lies in a closed annular region bounded by two directed cycles.*

Proof: Let A_i be a component of A' that is not a free loop, and consider any directed cycle of A_i . If this cycle were to have both merges and splits, then at some point there would be a merge followed by a split, which would be subject to a type II reduction. Since A' is reduced, it follows that every directed cycle in A_i consists solely of either merges or splits. Therefore, all of the edges attached to the outermost directed cycle of A_i must lie on the inside of the cycle, and all of the edges attached to the innermost cycle of A_i must lie on the outside of the cycle, so A_i lies in the annular region between these two cycles. ■

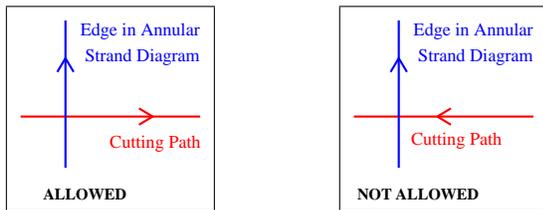


Fig. 7. The cutting path crosses each edge in the annular strand diagram from left to right.

Proof of Theorem 2: Clearly the cutting path must cross each of the components A_i at least once. Moreover, because of the edge crossing rules for cutting paths, a cutting path can cross a directed cycle only one time. Since every component is either a free loop or is bounded by two directed cycles, the result follows. ■

Our algorithm keeps track of a cutting path for the annular strand diagram of each element, modifying the path as the annular strand diagram is reduced. After reduction, we use Theorem 2 to reconstruct the concentric order of the components, a necessary step in checking for isotopy. Indeed, because of the concentric arrangement of the components, we can check isotopy for each pair of components separately:

Proposition 1. *Let A be an annular strand diagram with components A_1, \dots, A_m in concentric order, and let B be an annular strand diagram with components B_1, \dots, B_n in concentric order. Then A and B are isotopic if and only if $m = n$ and A_i is isotopic to B_i for each i . □*

We use the following theorem to check isotopy for connected components:

Theorem 3. *Let A_i and B_i be connected annular strand diagrams. Then A_i and B_i are isotopic if and only if there exists an isomorphism $\varphi: A_i \rightarrow B_i$ of directed multigraphs satisfying the following conditions:*

- 1) *For every split vertex v in A_i , the isomorphism φ maps the left output of v to the left output of $\varphi(v)$, and the right output of v to the right output of $\varphi(v)$.*
- 2) *For every merge vertex v in A_i , the isomorphism φ maps the left input of v to the left input of $\varphi(v)$, and the right input of v to the right input of $\varphi(v)$.*

input of v to the right input of $\varphi(v)$.

Proof: Observe that an isomorphism φ satisfies conditions (1) and (2) if and only if it preserves the counterclockwise order of the edges incident on each vertex. That is, φ satisfies (1) and (2) if and only if φ respects the “rotation systems” associated to A_i and B_i (see [10]). Therefore, there exists an isomorphism φ satisfying (1) and (2) if and only if A_i and B_i are isotopic as directed graphs on a **sphere**.

To relate isotopy on the sphere with isotopy on the annulus, observe that the region of A_i containing the central hole is the only region whose boundary is a counterclockwise directed cycle. Similarly, the region of A_i corresponding to the outside of the annulus is the only region whose boundary is a clockwise directed cycle. The same holds true for B_i . Therefore, given any isotopy from A_i to B_i on the sphere, the regions containing the central holes must correspond, as must the outer regions. It follows that A_i and B_i are isotopic on the sphere if and only if they are isotopic on the annulus. ■

IV. THE ALGORITHM

In this section, we describe and analyze the algorithm for the conjugacy problem in F . This algorithm refines the solution stated in Theorem 1 to achieve the best possible running time.

The steps in the algorithm are summarized in Fig. 8. We believe that the following key points will make it easier for the reader to understand the analysis of the algorithm presented later in this section:

- 1) The two inputs w_1 and w_2 are strings in the alphabet $\{x_0, x_1, x_0^{-1}, x_1^{-1}\}$.
- 2) The algorithm keeps track of a cutting path. After the annular strand diagrams are reduced, their connected components are labeled in concentric order using the sequence of edges in this cutting path (see Theorem 2).
- 3) We reduce the problem of determining whether two reduced annular strand diagrams are isotopic to the problem of determining whether two planar graphs are

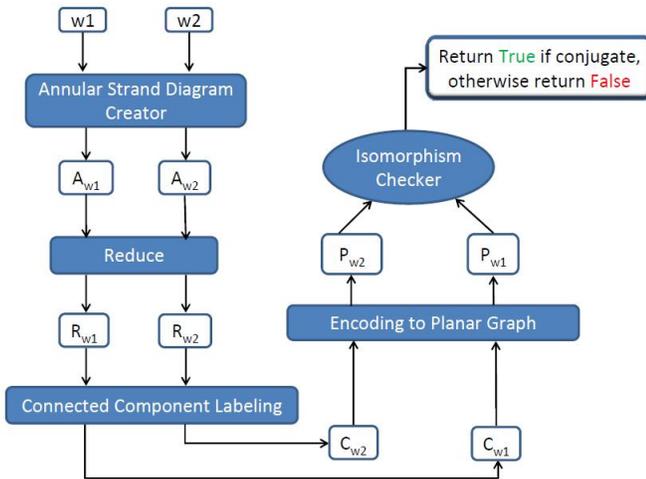


Fig. 8. Overview of the algorithm for the conjugacy problem in Group F

isomorphic. This is done by applying a one-to-one encoding on each connected component to convert it to a connected planar graph. The purpose of this step is to apply the $O(|V|)$ algorithm proposed by Hopcroft and Wong [8] for the isomorphism problem in planar graphs to make our algorithm achieve a linear running time. Whether w_1 and w_2 represent conjugate elements is then decided by using Proposition 1.

Theorem 4. *Given two input words w_1 and w_2 in $\langle x_0, x_1 \rangle$ representing two elements of F , the proposed algorithm for the conjugacy problem decides whether the two elements are conjugate in $O(n)$, where $n = |w_1| + |w_2|$.*

The rest of this section proves this theorem.

A. The Data Structure

Table I shows the data structure for representation and manipulation of annular strand diagrams.

Below we discuss the data structure, with particular emphasis on key fields and methods.

Class: Edge

This class holds edges in annular strand diagrams.

- 1) The field `type` is an array of two integers that records the **type** to which the edge belongs (see Section IV-D for a discussion of “type”). In this array, the first integer denotes the input type and the second integer denotes the output type for the edge. These integers can be the following:
 - 0 → free loop
 - 1 → left input or left output
 - 2 → right input or right output
- 2) The `node` field denotes the container node in the linked list representing the cutting path to which the edge belongs.
- 3) Given an edge e_1 with source vertex s , the `combineEdge()` method takes an edge e_2 with target vertex t as input, and then merges the two edges. As a result, both e_1 and e_2 are the same edge with source vertex s , target vertex t , and their `type` and `node` fields are modified if required.

Class: Vertex

The `Vertex` class represents merges and splits.

The field `type` denotes the vertex type, which can be either “merge” or “split”. Using the `type` field, we can safely decide which of the four `Edge` fields are valid for a vertex, as shown in Table II.

TABLE II
THE EDGE OBJECTS ASSOCIATED WITH CERTAIN VERTEX TYPES.

	merge	split
<code>leftParentEdge</code>	✓	✓
<code>rightParentEdge</code>	✓	✗
<code>leftChildEdge</code>	✓	✓
<code>rightChildEdge</code>	✗	✓

TABLE I
THE JAVA MODEL OF THE DATA STRUCTURE USED IN THE ALGORITHM. NOTE THAT ALL THE LINKED LISTS ARE DOUBLY LINKED.

Class	Vertex	Edge	Annular	Graph
Fields	type: String leftParentEdge: Edge rightParentEdge: Edge leftChildEdge: Edge rightChildEdge: Edge ID: Integer node: Node<Vertex> isPaired: Boolean correspondent: Vertex	source: Vertex target: Vertex ID: Integer type: Integer isFreeLoop: Boolean node: Node<Edge> flagged: Boolean	vertices: LinkedList <Vertex> stackReduceSplits: Stack<Vertex> cuttingPath: LinkedList<Edge>	vertices: List<Vertex> edges: List <Edge>
Methods	getLeftParent(): Vertex getRightParent(): Vertex getLeftChild(): Vertex getRightChild(): Vertex	combineEdge(Edge) makeFreeLoop()	reduce() getComponents() encodeToPlanarGraph()	

Note that the `Vertex` data structure keeps track of the counterclockwise order of the edges (i.e. the rotation system) since it keeps track of the left and right parents of a merge, and similarly the left and the right children of a split. Therefore, by Theorem 3, this data structure is sufficient to keep track of the isotopy class of the annular strand diagram.

Class: Graph

The `Graph` data structure is used to hold the planar graphs that are generated from the components of reduced annular strand diagram (discussed in Section IV-D). A list of the vertices and a list of the undirected edges are sufficient to represent these planar graphs.

Class: Annular

This data structure represents elements of F in annular strand diagram forms.

- 1) An `Annular` object is constructed by going through the input word from left to right, and creating and gluing the corresponding building blocks together.
- 2) The stack `stackReduceSplits` stores vertices that may be involved in reduction (discussed in Sections IV-B and IV-C).
- 3) The field `cuttingPath` is a linked list that stores a sequence of edges in a particular cutting path in the annular strand diagram.
- 4) The `reduce()` method performs all the possible reduction moves on an annular strand diagram, thereby reducing it.
- 5) The `getComponents()` method returns a concentric ordered list of the connected components in the annular strand diagram. These connected components are also `Annular` objects.
- 6) The method `encodeToPlanarGraph()` encodes connected components to planar graphs, which are `Graph` objects.

Now we begin a thorough discussion and analysis of the algorithm for the conjugacy problem.

B. Annular Strand Diagram Generation

Each building block for creating an annular strand diagram has a constant number of vertices and edges (see Fig. 2).

Therefore, construction of the annular strand diagram corresponding to the input word requires $O(n)$ vertices and edges, and $O(n)$ gluing of the building blocks, proving that creation of an annular strand diagram is $O(n)$. Note the following key points:

- 1) During the construction of an annular strand diagram, we put all the splits at the gluing points into a stack called `stackReduceSplits` as we know that these splits mark the regions of all possible reductions that can be performed on the annular strand diagram at that instant. To be precise, these regions are exposed to type II reduction moves. For instance, in Fig. 3, these are the green splits.
- 2) The edge that glues the last building block to the first is added to a doubly linked list called `cuttingPath` that represents the cutting path which the algorithm will dynamically keep track of.

C. Reduce

We now analyze the `reduce()` method and prove that it takes $O(n)$ to reduce an annular strand diagram. For our purposes, it suffices to show that the cutting path update, the number of reductions, and the number of checks for reductions collectively take $O(n)$.

Cutting Path Update: Fig. 9 shows the strategy we employ to update the cutting path. Reductions are performed by first removing edges and vertices, and then combining edges using `combineEdge()`. Each new edge created after a reduction represents both the edges that were combined to create the new edge. This means that in the case of a reduction I, we do not need to worry about updating the cutting path if it crosses edges e_1 or e_4 prior to the reduction because the reduction will update the cutting path accordingly. Similarly, in the case of a reduction II, we do not need to update the cutting path if it does not cross e_3 prior to the reduction. Hence, the only cases where the algorithm has to update the cutting path are the cases shown in Fig. 9. Also recall that the `node` field for edges ensures that each edge knows whether it is in the cutting path. Therefore, it is easy to see that deciding whether the cutting path needs to be updated during a reduction, and also updating the cutting path during a reduction both take

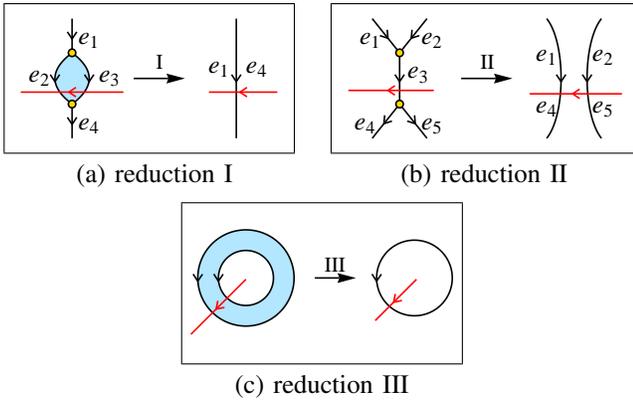


Fig. 9. Update of the cutting path (colored red) for each reduction move.

$O(1)$. Fig. 10 shows the edges in an annular strand diagram that the cutting path intersects before and after the annular strand diagram is reduced.

Number of Reductions: Observe that each type I or type II move deletes two vertices from an annular strand diagram. Since the number of vertices in the annular strand diagram after its creation is $O(n)$, it follows that the number of type I or type II moves is also $O(n)$. The number of type III moves is also bounded by $O(n)$ because each of these moves merge two concentric edges, and there can be at most $O(n)$ concentric edges in the annular strand diagram.

Number of Checks for Reductions: Observe that a reduction can create a new region of reduction nearby. We perform reductions locally and keep track of all possible regions in which new reductions may appear as a result of a local reduction. Notice that both type I or type II moves happen around a split. We use `stackReduceSplits` to store all possible splits that may be involved in these reduction moves. After any of these reduction moves are performed, we push onto `stackReduceSplits` the splits connected to the newly created edges (for instance, the splits connected to e_1 in Fig. 9.(a) after the reduction) because such a split might be now exposed to a reduction move. In this way we can check for all possible type I and type II moves. Observe that these reductions involve a constant number of pushes onto the stack. Because there are $O(n)$ reductions, there are at most $O(n)$ pushes onto the stack, and hence $O(n)$ checks for reduction I and II. We perform the type III moves after all possible type I and type II moves are performed. The type III moves are detected by finding all the adjacent free loops in the cutting path, which involve $O(n)$ checks since the cutting path can have at most $O(n)$ edges in it. This proves that the `reduce()` method takes $O(n)$.

After the annular strand diagrams are reduced, the connected components are detected in concentric order.

Connected Component Labeling: Recall that the data structure for edges holds the source and target vertices, and the data structure for vertices holds the connected edges. Therefore, given the cutting path for a reduced annular strand diagram, for each edge that meets the cutting path in order,

we can perform a breadth first search along both directions of the edge to discover the connected component to which the edge belongs. Because all the components collectively have a sum of vertices and edges bounded by $O(n)$, it follows that all connected components are identified in concentric order in $O(n)$.

D. Encoding to Planar Graphs

In this section, we explain and analyze the part of the algorithm which reduces the problem of determining isotopic annular strand diagrams to the problem of determining isomorphic planar graphs.

Theorem 5. *Any two connected, reduced annular strand diagrams A'_1 and A'_2 can be encoded into two planar graphs p_1 and p_2 respectively such that A'_1 and A'_2 are isotopic if and only if p_1 and p_2 are isomorphic.*

Sketch of Proof: To prove the theorem above, it suffices to demonstrate an encoding that is one-to-one. First we will describe the encoding, and then show that it is one-to-one.

We will now describe the function ϕ that encodes connected, reduced annular strand diagrams into planar graphs. First, notice that there are three possible input types for an edge, namely the left input or the right input to a merge, or the lone input to a split (see Fig. 1). Similarly an edge can have three possible output types. Hence there are nine different input-output combinations for an edge. In addition, an edge can be a free loop. Therefore, we have ten different **types of edges**. We provide unique encodings of each type of edge, as shown in Table III that ϕ will make use of. Note that the type to which an edge e belongs uniquely identifies the corresponding planar graph for the edge using the number of edges created between the vertices u and v_2 in the corresponding planar graph (see Table III).

Let $\phi : X \rightarrow G$ be a function that maps the set of connected, reduced annular strand diagrams X to the set of planar graphs G . Assume that $E = \{e_1, e_2, \dots, e_n\}$ is the edge set of $A' \in X$. To obtain $p_{A'} = \phi(A')$, follow the steps below:

- 1) Create a null graph $p_{A'}$.
- 2) Copy all the vertices from A' to $p_{A'}$.
- 3) For each edge $e \in E$, identify its edge type in Table III, and perform the appropriate encoding of e .

The function ϕ uniquely encodes its input annular strand diagram to a planar graph. Moreover, this process is one-to-one. Indeed if two (reduced) annular strand diagrams encode the same planar graph via ϕ , then it is easy to recover the original vertices and determine the left and right inputs of corresponding merges, and the left and right outputs of corresponding splits. This means that both annular strand diagrams have the same rotation systems. Theorem 3 asserts that they must be isotopic. ■

Analysis of the Encoding Algorithm: Recall that all the connected components together have a total of $O(n)$ vertices and edges. Because the encoding creates a constant number of vertices and edges for each edge in E , it follows that ϕ creates a planar graph with $O(n)$ vertices and edges.

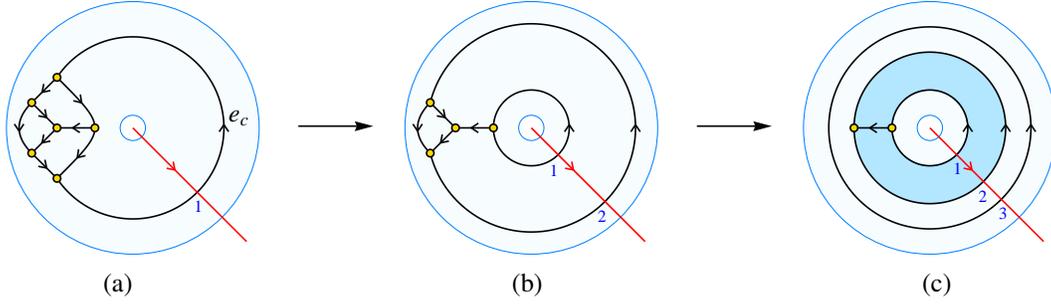


Fig. 10. Status of a cutting path (a) after closing a strand diagram (crosses edge e_c), (b) after performing a type II reduction, and (c) after the annular strand diagram is reduced. The numbers denote the order in which the cutting path crosses the edges of the annular strand diagram.

E. Determining Isotopy

To check whether two reduced annular strand diagrams are isotopic, we use Proposition 1 and Theorem 5. In other words, all corresponding planar graphs from both annular strand diagrams are compared to determine whether they are isomorphic using the $O(V)$ algorithm for the isomorphism problem in planar graphs proposed by Hopcroft and Wong [8], where V is the number of vertices in the input. Note that their algorithm accepts planar graphs with loops and multiple edges between vertices. Because the total number of vertices from our planar graph encodings are collectively bounded by $O(n)$, it follows that all the checks for isomorphism between planar graphs take $O(n)$.

This concludes the proof of Theorem 4, confirming that the algorithm for the conjugacy problem is linear in the input size. A Java-style pseudocode version of the algorithm is shown in Algorithm 1.

F. Implementation

To the best of our knowledge, the linear time algorithm for the isomorphism problem in planar graphs proposed in [8] has not been implemented yet. This is due to the complicated design of the algorithm. Moreover, the authors of [8] stated that this algorithm is not practical. As a result, we have not attempted an implementation of this algorithm, and instead programmed a direct isotopy search as a substitute for the following steps:

- 1) planar graph encoding of each component in the two annular strand diagrams, and
- 2) checking for isomorphism between all corresponding pairs of planar graph encodings.

Our implementation, called **ConjugacyF**, is a brute force implementation of Theorem 3. In particular, it substitutes the two steps above with a method that checks the rotation systems of the two reduced annular strand diagrams to determine whether the graphs are isotopic. This involves fixing a reference vertex v_r in one of the reduced annular strand diagrams A'_1 , and then for each vertex v_c in the other reduced annular strand diagram A'_2 as a possible correspondent of v_r , a breadth first expansion is performed in both A'_1 and A'_2 along the edges connected to each of v_r and v_c in counterclockwise order. If any of these expansions produces the same graphs, then A'_1

Input: String w_1 , String w_2

Output: Whether w_1 and w_2 represent conjugate elements

```

1 for  $w$  in  $\{w_1, w_2\}$  do
2   Annular asd = new Annular( $w$ )
3   asd.reduce()
4   List<Annular> components =
   asd.getComponents(asd.cuttingPath)
5    $P_w$  = new List<Graph>()
6   for  $c$  in components do
7      $P_w$ .add( $c$ .encodeToPlanarGraph())
8   end
9 end
10 if  $P_{w_1}.size() \neq P_{w_2}.size()$  then
11   return false
12 for  $i = 0 \rightarrow P_{w_1}.size() - 1$  do
13   Graph  $p_1 = P_{w_1}.get(i)$ 
14   Graph  $p_2 = P_{w_2}.get(i)$ 
15   if  $!(isIsomorphic(p_1, p_2))$  then
16     // the linear method from [8]
17     return false
18 end
19 return true

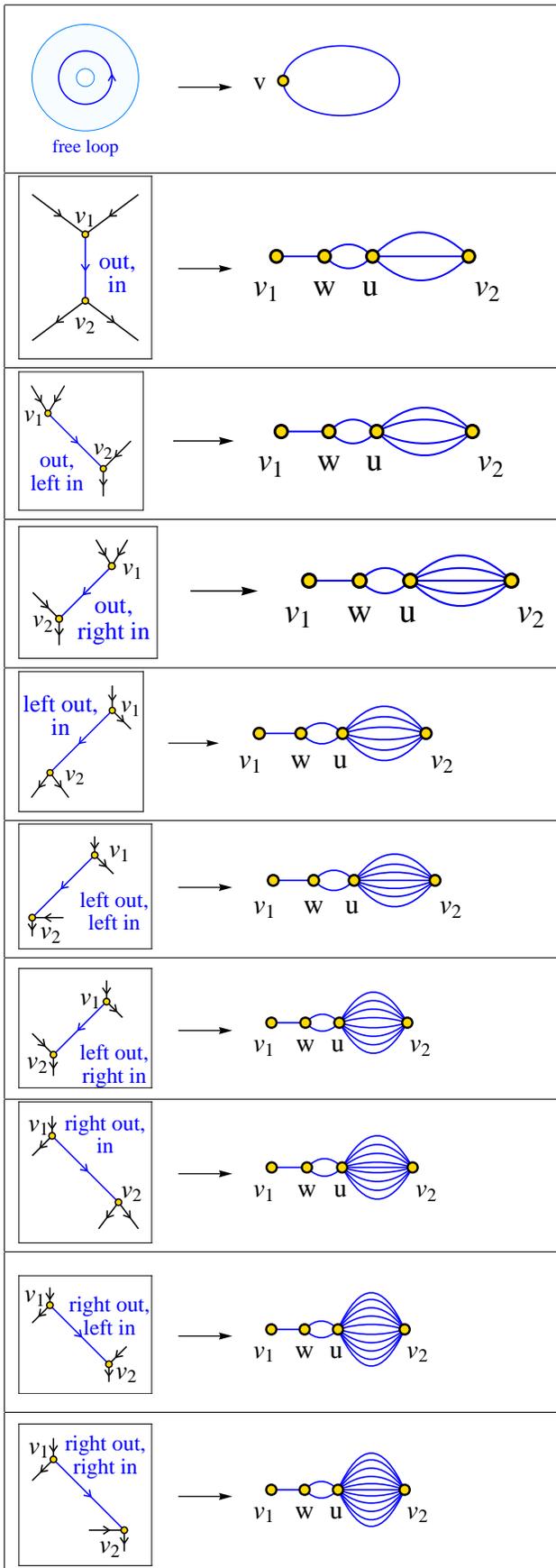
```

Algorithm 1: Algorithm for the conjugacy problem in F .

and A'_2 represent conjugate elements. In the worst case (when the graphs are not isotopic), this algorithm performs two linear time expansions for each vertex v_c in A'_2 . Hence ConjugacyF is $O(n^2)$.

We made ConjugacyF publicly available on [11] as a web application in the form of a Java applet and an executable JAR file (compiled on Windows 7). These applications allow users to check whether two elements of F are conjugate, to sort a list of elements into conjugacy classes, and to compute the reduced annular strand diagram for any element. We also shared the Java source code for ConjugacyF on [11].

TABLE III
DIFFERENT TYPES OF EDGES IN THE ANNULAR STRAND DIAGRAM, AND
THEIR ENCODINGS TO PLANAR GRAPHS



V. CONCLUSION AND FUTURE WORK

We presented a linear time reduction of the conjugacy problem in Thompson's Group F to the isomorphism problem of planar graphs using directed graphs called annular strand diagrams, along with data structures for efficient storage and manipulation of the associated mathematical objects. Given a linear time algorithm for the isomorphism problem of planar graphs [8], this leads to a linear time algorithm for the conjugacy problem in F . Moreover, the conjugacy problem in F requires linear time, so such an algorithm represents the best runtime generally speaking.

Due to the impractical nature of the linear algorithm for the isomorphism problem in planar graphs [8], we implemented a quadratic solution that directly determines whether two reduced annular strand diagrams are isotopic, and we made the software publicly available. This is the first public software implementation of an algorithm for the conjugacy problem in F . It is our hope that our software will be useful to the research community in Thompson's Groups.

For future work, we believe that it would not be too hard to modify our software to create algorithms for the conjugacy problems in the other two Thompson's Groups, namely V and T . The cutting path used in the algorithm for F is representative of the *cutting class* [7] used in solving the conjugacy problem in V . However, the algorithm for V is not expected to be linear because checking whether two cutting paths represent the same cutting class might require Gaussian elimination [12], which is worse than linear.

REFERENCES

- [1] P. S. Novikov, "Unsolvability of the conjugacy problem in the theory of groups. (Russian)," *Izv. Akad. Nauk SSSR. Ser. Mat.*, vol. 18, pp. 485–524, 1954.
- [2] K. Madlener and J. Avenhaus, "String matching and algorithmic problems in free groups." *Revista colombiana de matematicas*, vol. 14, pp. 1–16, 1980.
- [3] F. A. Garside, "The braid group and other groups," *The Quarterly Journal of Mathematics*, vol. 20, no. 1, pp. 235–254, 1969.
- [4] J. W. Cannon, W. J. Floyd, and W. R. Parry, "Introductory notes on Richard Thompson's groups," *Enseignement Mathématique*, vol. 42, pp. 215–256, 1996.
- [5] V. Guba and M. V. Sapir, *Diagram groups*. AMS Bookstore, 1997, vol. 620.
- [6] V. S. Guba and M. V. Sapir, "On subgroups of R. Thompson's group F and other diagram groups," *Sbornik: Mathematics*, vol. 190, no. 8, p. 1077, 1999.
- [7] J. Belk and F. Matucci, "Conjugacy and dynamics in Thompson's groups," *Geometriae Dedicata*, pp. 1–23, 2013. [Online]. Available: <http://dx.doi.org/10.1007/s10711-013-9853-2>
- [8] J. E. Hopcroft and J.-K. Wong, "Linear time algorithm for isomorphism of planar graphs (preliminary report)," in *Proceedings of the sixth annual ACM symposium on Theory of computing*. ACM, 1974, pp. 172–184.
- [9] I. Short and N. Gill, "Conjugacy in Thompson's group F ," *Proceedings of the American Mathematical Society*, vol. 141, pp. 1529–1538, 2013.
- [10] J. L. Gross and S. R. Alpert, "The topological theory of current graphs," *Journal of Combinatorial Theory, Series B*, vol. 17, no. 3, pp. 218–233, 1974.
- [11] N. T. Hossain. (2013, May) Algorithm for the conjugacy problem in Thompson's group F . Accessed: 30 June 2013. [Online]. Available: <http://www.asclab.org/asc/nhossain/conjugacyF>
- [12] J. Edmonds, "Systems of distinct representatives and linear algebra," *J. Res. Nat. Bur. Standards, Sect. B*, vol. 71, no. 4, pp. 241–245, 1967.