Tree-based language complexity of Thompson's group F

Jennifer Taback
Bowdoin College

Sharif Younes
NuFit Media

Follow this and additional works at: https://digitalcommons.bowdoin.edu/mathematics-faculty-publications

Research Article

Jennifer Taback* and Sharif Younes

Tree-based language complexity of Thompson's group *F*

Abstract: The definition of graph automatic groups by Kharlampovich, Khoussainov and Miasnikov and its extension to -graph automatic by Elder and the first author raise the question of whether Thompson's group *F* is graph automatic. We define a language of normal forms based on the combinatorial "caret types", which arise when elements of *F* are considered as pairs of finite rooted binary trees. The language is accepted by a finite state machine with two counters, and forms the basis of a 3-counter graph automatic structure for the group.

Keywords: Thompson's group *F*, automatic group, graph automatic group, -graph automatic group

1 Introduction

It is not known whether Thompson's group *F* is automatic; *F* has some characteristics of an automatic group such as quadratic Dehn function, type FP_∞ and word problem solvable in O $n \log n)$ time see [10], [2] and [13], respectively). Yet an automatic structure for the group remains elusive. Guba and Sapir present a regular language of normal forms for elements of *F* in [10], and Cleary and the first author show that *F* has no regular language of geodesics in [5].

With the extension of the notion of an automatic group to a graph automatic group by Kharlampovich, Khoussainov and Miasnikov in [12], and further to a -graph automatic group by Elder and the first author in [7], a natural question is whether *F* is captured by one of these larger classes of groups. To summarize these definitions for a group *G* with finite generating set *S*:

1) *G* has an automatic structure if there is a normal form for group elements that is recognized by a finite state machine, as well as finite state machines $_s$ that accept the language of pairs $g, gs)$, for each $s \in S$. The machines $_s$ are called the multiplier automata.

2) *G* has a graph automatic structure if there is a finite symbol alphabet used to define a normal form for group elements that is recognized by a finite state machine, as well as finite state machines $_s$ that accept pairs of normal form words corresponding to group elements that differ by *s*, for each $s \in S$.

3) *G* has a -graph automatic structure, where is a class of languages – for example regular, context free, counter or context-sensitive – and all languages in the previous definition are now allowed to lie in the class .

The introduction of a symbol alphabet used to express normal forms for group elements seems quite suited to Thompson's group *F*. If group elements are expressed as reduced pairs of finite rooted binary trees, there is combinatorial information that can be used to express the group element uniquely. Namely, these trees are constructed from nodes, or *carets*, and there are several descriptions of caret types given in the literature that depend on the placement of the carets within the tree. We follow those given by Elder, Fusy and Rechnitzer in [6] and define a quasi-geodesic normal form for elements of *F*.

Corresponding author: Jennifer Taback: Department of Mathematics, Bowdoin College, Brunswick, ME 04011, USA, e-mail: jtaback@bowdoin.edu
Sharif Younes: NuFit Media, 1 Kendall Square, Suite B2101, Cambridge, MA 02139, USA, e-mail: sharif@nuftmedia.com

We show below that *F* is not graph automatic with respect to this natural normal form language over this set of symbols. This does not rule out the result with respect to a different normal form, or an alternate symbol alphabet. However, any symbol alphabet which captures the tree structure in any way seems unlikely to the authors to yield the result that *F* is graph automatic.

In [8], Elder and the first author show that the standard infinite normal form for elements of *F* is the basis of a 1-counter graph automatic structure. The quasigeodesic normal form we construct below based on caret types yields a 3-counter graph automatic structure for the group. This could be improved to a 2-counter graph automatic structure, but the increase in complexity of the argument did not balance out the decrease in counters, given the result in [8].

We remark that a third possibility for constructing a -graph automatic structure for *F* would be to use the regular normal form language of Guba and Sapir [10]. In preparing this article we considered the complexity of the multiplier automata for this language and found that the x_1 multiplier automaton would require at least four counters, and hence the resulting structure is less efficient overall than the structures obtained in [8] and this paper.

The paper is organized as follows. In Section 2 we recall the definition of -graph automatic groups introduced in [7] which generalizes the definition of graph automatic groups introduced by Kharlampovich, Khoussainov and Miasnikov in [12]. In Section 3 we present relevant background on Thompson's group *F*. In Sections 4 and 5 we show that the language based on the symbol alphabet consisting of caret types forms a 3-counter graph automatic structure for *F*.

2 Background on generalizations of automaticity

There are many definitions in the literature of counter automata; we begin with our definition.

Definition 2.1 Counter automaton). A *nonblind deterministic k-counter automaton* is a deterministic finite state automaton augmented with *k* integer counters: these are all initialized to zero, and can be incremented, decremented, compared to zero and set to zero during operation. For each configuration of the machine and subsequent input letter, there is at most one possible move. The automaton accepts a word if upon reading the word it reaches an accepting state with all counters returned to zero.

In drawing a counter automaton, we label transitions by the input letter to be read, with subscript to denote the possible counter instructions:
- "= 0" to indicate the edge may only be traversed if the value of the counter is 0;
- "+1" to increment the counter by 1;
- "−1" to decrement the counter by 1.

In this paper we focus entirely on counter graph automatic structures, which will be defined below. The class of counter languages is closed under homomorphism, inverse homomorphism, intersection with regular languages, and finite intersection see [11], for example). Moreover, the intersection of a *k*-counter language with a regular language is *k*-counter and the intersection of *k*- and *l*-counter languages is a *k* + *l*)-counter language, as proven in [7].

We introduce the notion of a convolution of strings in a language, following [12], for ease of notation in the multiplier languages we later define. Let *G* be a group with symmetric generating set *X*, and Λ a finite set of symbols. In general we do not assume that *X* is finite. The number of symbols letters) in a word $u \in \Lambda^*$ is denoted $|u|_\Lambda$.

Definition 2.2 Convolution [12, Definition 2.3]). Let Λ be a finite set of symbols, $\diamond$ a symbol not in Λ, and let $L_1, \ldots, L_k$ be a finite set of languages over Λ. Set $\Lambda_\diamond = \Lambda \cup \{\diamond\}$. Define the *convolution of a tuple* $w_1, \ldots, w_k) \in L_1 \times \cdots \times L_k$ to be the string $\otimes w_1, \ldots, w_k)$ of length max $|w_i|_\Lambda)$ over the alphabet $\Lambda_\diamond)^k$

as follows. The ith symbol of the string is

$$\begin{pmatrix} \lambda_1 \\ \vdots \\ \lambda_k \end{pmatrix},$$

where λ_j is the ith letter of w_j if $i \le |w_j|_\Lambda$ and $\diamond$ otherwise. Then

$$\otimes L_1, \ldots, L_k) = \{\otimes\ w_1, \ldots, w_k) \mid w_i \in L_i\}.$$

As an example, if $w_1 = aa$, $w_2 = bbb$ and $w_3 = a$ then

$$\otimes\ w_1, w_2, w_3) = \begin{pmatrix} a \\ b \\ a \end{pmatrix} \begin{pmatrix} a \\ b \\ \diamond \end{pmatrix} \begin{pmatrix} \diamond \\ b \\ \diamond \end{pmatrix}.$$

When $L_i = \Lambda^*$ for all i, the definition in [12] is recovered.

We require that the normal form language on which a -graph automatic structure is based be quasigeodesic.

Definition 2.3 Quasigeodesic normal form). A *normal form for* $G, X, \Lambda)$ is a set of words $L \subseteq \Lambda^*$ in bijection with G. A normal form L is *quasigeodesic* if there is a constant D so that

$$|u|_\Lambda \le D\ \|u\|_X + 1)$$

for each $u \in L$, where $\|u\|_X$ is the length of a geodesic in X^* for the group element represented by u.

The $\|u\|_X + 1$ in the definition allows for normal forms where the identity of the group is represented by a nonempty string of length at most D. We denote the image of $u \in L$ under the bijection with G by $\overline{u}$.

We now state the definition of a -graph automatic group, following [7], which generalizes the notion of a graph automatic group introduced in [12].

Definition 2.4 -graph automatic group). Let be a formal language class, $G, X)$ a group and symmetric generating set, and Λ a finite set of symbols. We say that $G, X, \Lambda)$ is *-graph automatic* if there is a normal form $L \subset \Lambda^*$ in the language class , such that for each $x \in X$ the language

$$L_x = \{\otimes\ u, v) \mid u, v \in L,\ \overline{v} =_G \overline{u}x\}$$

is in the class .

The following observation is proven in [7].

Lemma 2.5 [7, Lemma 2.5]). *Let L_1 and L_2 be k- and l-counter languages, respectively. Then $\otimes\ L_1, L_2)$ is a $k + l)$-counter language.*

We end this section with a lemma describing when certain languages of convolutions are regular; this lemma will be used to streamline many of the proofs in later sections.

Lemma 2.6. *Fix a symbol alphabet Λ and let a_1, a_2, b_1, b_2 be fixed possibly empty) words in Λ^*. Then the following languages are all regular.*
i) $L_1 = \{\otimes\ a_1 w, a_1 b_1 w) \mid w \in \Lambda^*\}$;
ii) $L_2 = \{\otimes\ a_1 w, b_1 w) \mid w \in \Lambda^*\}$ *and* $L_3 = \{\otimes\ wa_2, wb_2) \mid w \in \Lambda^*\}$;
iii) $L_4 = \{\otimes\ w_1 a_1 w_2 a_2 w_3, w_1 b_1 w_2 b_2 w_3) \mid w_1, w_2, w_3 \in \Lambda^*\}$.

Proof.   i) Since $\{\otimes\ a_1, a_1)\}$ is regular and the set of regular languages is closed under concatenation, this problem reduces to recognizing the language $L = \{\otimes\ w, bw)\}$ with a finite state automaton.

Let $|b| = p$. Enumerate all strings of length p in Λ^*: $s_1, s_2, s_3, \ldots, s_n$. For each s_i, create a state S_i and add a path of p edges from the start state q_0 to S_i labeled by the successive letters in $\otimes\ s_i, b)$. Next, for each pair $\lambda_1, \lambda_2) \in \Lambda \times \Lambda$ add an edge from S_i to S_j labeled $\lambda_1, \lambda_2)$ if and only if $s_i = \lambda_2 x$ and $s_j = x\lambda_1$ for $x \in \Lambda^*$. Finally, for each s_i, construct a path from S_i to an accept state F labeled by $\otimes\ \epsilon, s_i) = \diamond^p, s_i)$.

The edge labels ensure that if a prefix of $\otimes\, u, v)$ describes a path from the start state to state S_i, then the last p letters read from u are exactly the string s_i. Reading the next letter of $\otimes\, u, v)$, say $\binom{x}{y}$, to transition to state S_j means that the final p letters now read from u are s_j and that the initial letter of s_i was y. In this way we check that the string bw contains the string w shifted p places by the insertion of b.

Cases ii) and iii) are proven by similar arguments and we omit them here.   □

3 Background on Thompson's group F

Thompson's group F can be equivalently viewed from three perspectives:
1) As the group defined by the infinite presentation

$$\mathcal{P}_{\text{inf}} = \langle x_0, x_1, x_2, \ldots \mid x_j x_i = x_i x_{j+1} \text{ whenever } i < j \rangle$$

or the finite presentation

$$\mathcal{P}_{\text{fin}} = \langle x_0, x_1 \mid [x_0^{-1} x_1, x_0^{-1} x_1 x_0], [x_0^{-1} x_1, x_0^{-2} x_1 x_0^2] \rangle.$$

2) As the set of piecewise linear homeomorphisms of the interval $[0, 1]$ satisfying the following:
   - each homeomorphism has finitely many linear pieces;
   - all breakpoints have coordinates which are dyadic rationals;
   - all slopes are powers of two.
3) As the set of pairs of finite rooted binary trees with the same number of nodes, or carets.

For the equivalence of these three interpretations of this group, as well as a more complete introduction to the group, we refer the reader to [3]. If $g = T, S)$ is an element of F given as a pair of finite rooted binary trees, then the tree T determines the linear intervals in the domain of g, and the tree S determines the linear intervals in the range; again see [3] for details.

3.1 Terminology and notation

In this section we define the terminology that we use in creating the 2-counter language of normal forms for elements of F. A *caret* consists of a vertex in a tree T together with two edges that we draw with a downward orientation. A vertex in the tree T
1) is called a *leaf* iff it has valence one;
2) is the *root* of the tree iff it has valence two;
3) has valence three otherwise.

A caret in T has a left edge and a right edge; we refer to a caret D as the *left resp. right) child* of caret C if the vertex of D is the endpoint of the left resp. right) edge of C. A caret is called *exterior* if at least one of its edges lies on the left or right side of the tree, and *interior* otherwise. A caret is *exposed* if it has no children.

Let $T, S)$ denote a pair of finite rooted binary trees with the same number of carets, which necessarily corresponds to a unique element of Thompson's group F. We refer to this as a tree pair diagram representing the element. The carets of T and S are numbered in increasing infix order: beginning with the leftmost descendant of the root caret, the left child of any caret is numbered before the caret, and the right child is numbered after it. We then pair carets with the same infix number from the two trees and refer to a *caret pair* with caret number n.

3.2 Reduction criterion

To ensure a bijective correspondence between elements of F and pairs of finite rooted binary trees with the same number of carets, we impose an additional requirement. Viewing an element of F as a piecewise linear homeomorphism of the interval, subject to the above conditions, we can always trivially add additional

breakpoints within a linear component without altering the function. However, such breakpoints are superfluous. We define a *reduced* tree pair diagram to correspond to a function without unnecessary breakpoints. Exposed carets with identical infix numbers in both trees introduce a superfluous breakpoint to the analytic representation of a group element. By removing these unnecessary caret pairs we obtain a reduced tree pair diagram. An element $g \in F$ is thus represented by an equivalence class of tree pair diagrams with a unique reduced diagram. When we refer to a tree pair diagram representing a group element, we mean the reduced diagram unless otherwise indicated.

3.3 Multiplication of tree pair diagrams

Multiplication of elements of F when viewed as piecewise linear homeomorphisms of the interval $[0, 1]$ is simply composition of functions; "multiplication" of tree pair diagrams is a combinatorial construction which mimics composition of functions. Consider $g = T_1, T_2)$ and $h = S_1, S_2)$ in F; create unreduced representatives of both g and h, which we denote by $T_1', T_2')$ and $S_1', S_2')$, respectively, so that $T_2' = S_1'$. The product hg is then represented by the possibly unreduced tree pair diagram $T_1', S_2')$. For explicit examples of multiplication of elements of F we refer the reader to [4].

Multiplication by the generators x_0 and x_1 and their inverses induces particular combinatorial rearrangements of a given tree pair diagram which are explicitly described in Section 5.

3.4 Caret labels

In Section 4 below we develop a set of normal forms for elements of F using an alphabet consisting of *caret types* or *labels*. We give each caret a label based on its position in the tree, and form these strings into pairs by infix order.

There are several existing sets of caret labels in the literature; Fordham in [9] uses caret labels to assign weights to each caret pair. He then proves that the sum of the weights corresponding to a reduced) element is exactly its word length with respect to the finite generating set $\{x_0, x_1\}$. Unfortunately, Fordham's caret labels are not in one-to-one correspondence with configurations of carets in a tree, as illustrated in Figure 1. Fordham defines a caret to be *interior* if neither edge comprising the caret lies on the left or right side of the tree, and further subdivides these carets into type I_0, which have no right child, and type I_R, which do. However, the configurations of carets in Figure 1 both correspond to the string of labels I_R, I_0, I_0, where the labels are listed in infix order.
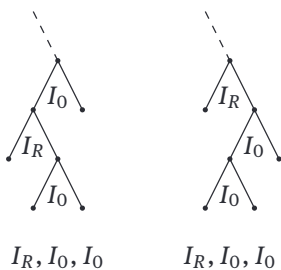

I_R, I_0, I_0 I_R, I_0, I_0

Figure 1. Fordham's caret labels are not in bijective correspondence with binary trees: two different interior subtrees corresponding to the same string of labels. Labels are listed in infix order.

We refine Fordham's definition of interior caret types to avoid the above problem, and condense his left and right caret types comprising carets with an edge on either the left or right side of the tree) into a single exterior type, while distinguishing the root caret. Define the following caret types within a finite rooted binary tree:
1) the root caret is labeled "r";
2) all other exterior carets are labeled "e";
3) interior carets are labeled as in Table 1.

Caret	Is the left child of an interior caret or the child of an exterior caret	Is neither the left child of an interior caret nor the child of an exterior caret
Has a right child		b
Has no right child	a	)

Table 1. Labeling of interior carets based on their position in the tree.
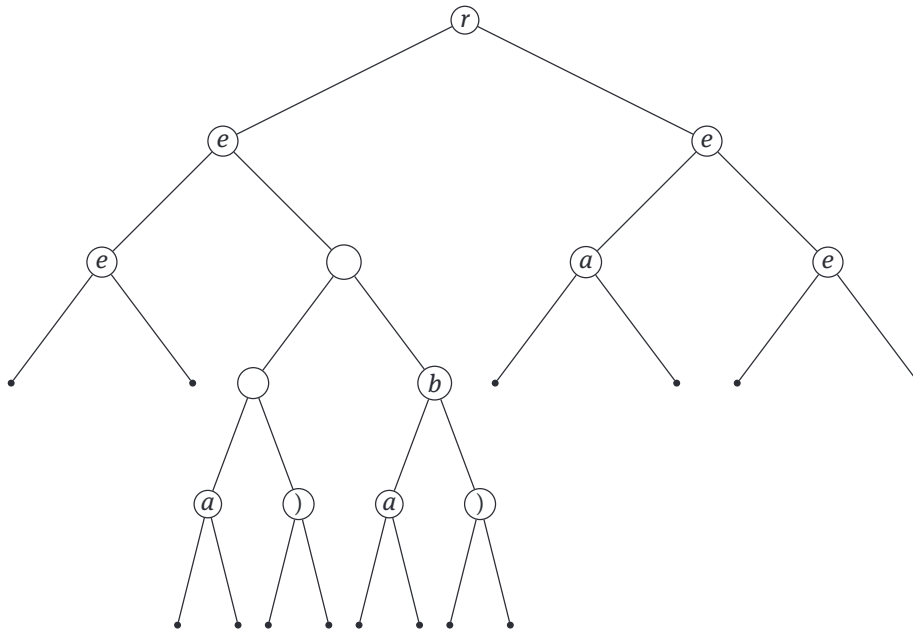

Figure 2. An example of a tree corresponding to the string of labels eea) $ab)raee$.
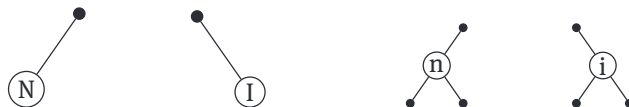

Figure 3. Leaf and caret labels used in [6]. Leaves are labeled "N" or "I" and carets are labeled "n" or "i". By convention, the root caret is labeled "n" and the left-most leaf is not labeled.

Listing the caret types of the carets in a binary tree in infix order yields a word over the alphabet $\Sigma_T = \{r, e, \ ,), a, b\}$. For example, the tree in Figure 2 is encoded by the word eea) $ab)raee$.

These caret labels are easily translated into those used by Elder, Fusy and Rechnitzer in [6] which are based on diagrams in [1]); their labels mark only interior subtrees. Leaves are labeled "N" or "I" and carets are labeled "n" or "i" as in Figure 3.

There is a straightforward translation between the notation in [6] and our own:

1) if a caret is labeled "n", the caret is a left child or else the root caret;
2) if the caret is labeled "i", then the caret is a right child;
3) if the leaf directly to the right of a caret is labeled "N", then the caret has a right child; if the leaf directly right of a caret is labeled "I", then the caret does not.

Following these rules, replace any appearance of "nN" with " ", "iI" with ")", "nI" with "a", and "iN" with "b".

4 A 2-counter language of normal forms for elements of *F*

We now define a language $\mathcal{F}$ of normal forms for elements of F based on the caret types defined in Section 3.4 which is accepted by a nonblind deterministic 2-counter automaton. This forms the basis of a 3-counter graph automatic structure for F; the additional counter is required to construct the multiplier automaton for x_1.

Theorem 4.1. *Thompson's group F is nonblind deterministic 3-counter graph automatic with quasigeodesic normal form with respect to the generating set $S = \{x_0, x_1\}$ and the symbol alphabet $\Sigma_T = \{e, r, ,), a, b\}$ of caret types.*

This theorem is proved in the subsequent sections. Moreover, we show that the language of normal forms we define over this symbol alphabet can never be accepted by a 1-counter automaton.

4.1 Word acceptor

The language $\mathcal{F}$ is constructed in stages, always taking the symbol alphabet to consist of the caret labels $\{e, r, ,), a, b\}$.
 1) We first define a 1-counter language L_{int} containing all strings that uniquely describe possible interior subtrees of a finite rooted binary tree. Exterior caret labels are not used.
 2) Next we extend L_{int} to a 1-counter language L_{tree} whose strings uniquely describe finite rooted binary trees.
 3) We define a 2-counter language L_T which is the intersection of $\otimes L_{tree}, L_{tree})$ with a regular language ensuring that no $\diamond$ symbols appear in any convolutions. This language describes all pairs of finite rooted binary trees with the same number of carets.
 4) Finally, we check that strings in L_T correspond to reduced pairs of binary trees by intersecting with a regular language R which checks a simple reduction criterion. Then $\mathcal{F} = L_T \cap R$.

4.1.1 A language describing interior subtrees

In [6] a simple criterion is given for when a string of caret labels consisting of "*N*", "*I*", "*n*" and "*i*" as defined in Figure 3) represents an interior subtree of a finite rooted binary tree. We restate this lemma using the alphabet $\Sigma_{int} = \{ ,), a, b\}$ of interior caret labels. A translation between the two sets of labels is given in Section 3.4.

Lemma 4.2 [6, Lemma 6]). *Let $w = s_1 \cdots s_n$ be a word in $\Sigma_{int})^*$. Let w_j denote the prefix $s_1 \cdots s_j$ and let $|u|_y$ denote the number of occurrences of "y" in u. Then w encodes a possibly empty) interior subtree if and only if the following conditions hold:*
 i) $s_{j+1} \in \{ , a\}$ *for any $0 \le j < n$ such that $|w_j| = |w_j|_)$;*
 ii) $|w| = |w|_)$.

Let L_{int} denote the set of all strings satisfying the conditions in Lemma 4.2 including the empty string). Then L_{int} is a nonblind deterministic 1-counter language accepted by a machine $_{int}$, where the input is read from left to right and the counter instructions are as follows:
 1) the counter increments by 1 after reading " ";
 2) read ")" only when the value of the counter is non-zero, and subsequently decrement the counter by 1;
 3) proceed to a fail state if the counter is zero and "*b*" is read.

These counter instructions correspond exactly to the following arrangement of carets in a binary tree. A caret with label " " corresponds to the root caret of an interior subtree with a nonempty right subtree; there will be a corresponding caret labeled ")" which is the final caret in infix order) of this subtree.

As every element in L_{int} corresponds to a possible interior subtree of a finite rooted binary tree, we can formalize this identification by defining a mapping $\tau_{int} : L_{int} \to \{$interior subtrees$\}$. To show that τ_{int} is well-

defined we must verify that any word $w \in L_{int}$ encodes a unique interior subtree. This is easily checked by considering all combinations of pairs of caret labels in $\Sigma_{int} = \{ \, ,), a, b\}$ and verifying that each combination precisely defines where the corresponding carets must be positioned in the interior subtree; see Figure 4. If $\alpha \in L_{int}$ is a string of caret types from Σ_{int}, we build a unique interior subtree as follows. Each adjacent pair of caret types beginning on the left side of the string uniquely determines the relative placement of the carets in the subtree being constructed as shown in Figure 4. Hence τ_{int} is a bijection between the set of possible interior subtrees of a finite rooted binary tree and the set of words in the language L_{int}.

4.1.2 A language describing binary trees

Next we construct a language L_{tree} over the alphabet

$$\Sigma_T = \{e, r, \, ,), a, b\}$$

which accepts exactly those words corresponding to finite rooted binary trees. Let $_{tree}$ denote the machine which accepts L_{tree}. A binary tree containing no interior carets will correspond to a word of the form $e^{n_1} r e^{n_2}$, where $n_1, n_2 \in \mathbb{N} \cup \{0\}$. In general, a binary tree has interior subtrees attached to any collection of exterior carets. Thus, a string w in Σ_T^* describing such a tree must satisfy the following conditions:

1) w begins with either "e" or "r";
2) w ends with either "e" or "r";
3) w contains exactly one "r";
4) for any substring $s_{i-1} s_i \cdots s_j s_{j+1}$, if $s_{i-1}, s_{j+1} \in \{e, r\}$ and $s_i \cdots s_j \in \Sigma_{int}^*$, then $s_i \cdots s_j$ is a word in L_{int}.

Let L_{tree} be the set of words satisfying conditions 1) through 4). Since L_{int} is a 1-counter language, condition 4) can be checked with a single counter, as follows. Note that substrings $s_{i-1} s_i \cdots s_j s_{j+1}$ and $s_{k-1} s_k \cdots s_l s_{l+1}$ may overlap only at the first and final letters, which lie in $\{e, r\}$. Upon reading "e" or "r" with a subsequent letter from Σ_{int}, a machine scanning the word simulates the machine for L_{int} on the substring $s_i \cdots s_j s_j$, and repeats this process when it encounters the next letter from $\{e, r\}$.

The three remaining conditions are easily recognized with finite state automata. Hence L_{tree} is a nonblind deterministic 1-counter language.

We can now extend the function $\tau_{int} : L_{int} \to \{\text{interior subtrees}\}$ to a bijection

$$\tau_{tree} : L_{tree} \to \{\text{finite rooted binary trees}\}.$$

4.1.3 A language describing pairs of binary trees

To describe pairs of binary trees with the same number of carets, we use the convolution of two strings from L_{tree} which are the same length. Namely, L_T is the intersection of $\otimes L_{tree}, L_{tree})$ with a regular language of convolutions of elements of Σ_T^* containing no $\diamond$ symbols. It follows from Lemma 2.5 that L_T is a 2-counter language. Analogously, there is a bijection

$$\tau : L_T \to \{\text{pairs of finite rooted binary trees with the same number of carets}\}.$$

If $\otimes u, v) \in L_T$ then the top line in the convolution determines the first tree in the pair, and the bottom line the second tree in the pair. Let $_T$ denote the machine which accepts L_T.

4.1.4 Reduction criterion

Next we test whether a string in L_T corresponds to a reduced pair of trees. We first observe sequences of labels which must appear in a string corresponding to a tree pair diagram that is unreduced. For each sequence, we intersect L_T with the complement of the corresponding regular language, using the fact that the set of regular languages is closed under complementation.

Combination: aa

Combination: ab

Combination: a

Combination: $a)$

Combination: ba

Combination: bb

Combination: b

Combination: $b)$

Combination: a

Combination: b

Combination:

Combination: $)$

Combination: $)a$

Combination: $)b$

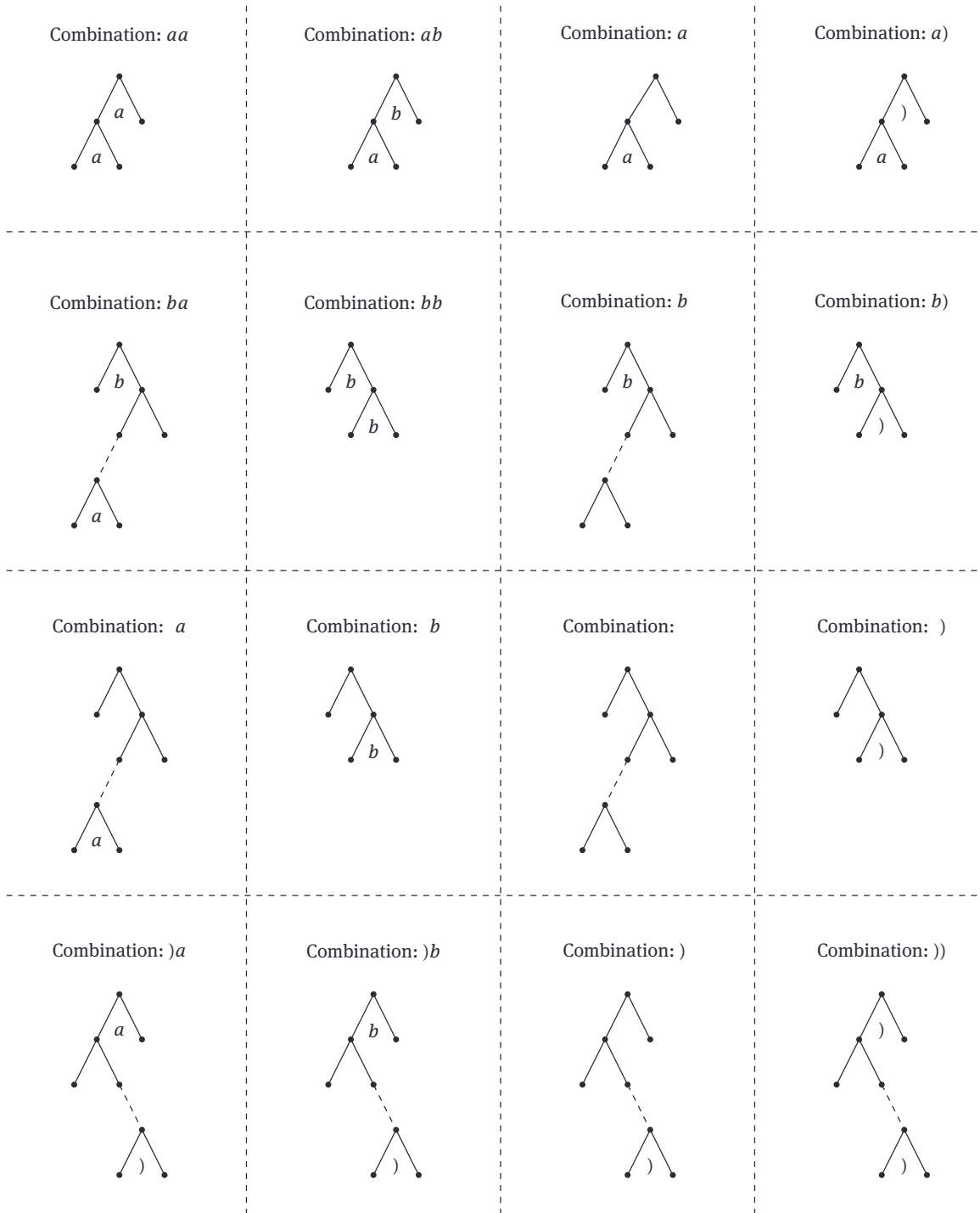Combination: $)$

Combination: $))$

Figure 4. Every possible pair of caret labels in L_{int} and the relative placement of the carets they encode.

Given a finite rooted binary tree T, it is easily checked that:

1) the first exterior caret of T is exposed iff the word encoding it begins with "ee" or "er";
2) the last exterior caret of T is exposed iff the word encoding it ends with "re" or "ee";
3) a symbol "y" encodes an exposed interior caret of T iff $y \in \{), a\}$ and is preceded by some $x \in \{e, r, \,, b\}$. Recall that ")" and "a" are the two interior caret types which label carets that have no right child.

There are three ways that a pair of binary trees representing a non-identity element of F may be unreduced:

1) the first exterior caret of each tree is exposed; this occurs iff the word in L_T describing the pair of trees begins with $\binom{e}{e}\binom{y_1}{y_2}$ for $y_i \in \{e, r\}$;
2) the last exterior caret of each tree is exposed; this occurs iff the word in L_T describing the pair of trees ends with: $\binom{x_1}{x_2}\binom{e}{e}$ for $x_i \in \{e, r\}$;
3) each tree has an exposed interior caret with infix number n; this occurs iff the word in L_T describing the pair of trees contains any of the following strings: $\binom{x_1}{x_2}\binom{y_1}{y_2}$ for $x_i \in \{e, r, \,, b\}$ and $y_i \in \{), a\}$, where the y_i represent the exposed interior carets.

Each of the above cases can be detected with a finite state machine. Let R denote the set of all strings that *do not* contain any of these sequences. Then R is regular and, moreover, R consists of exactly those strings that do not fail any of the three reduction criteria. Let $\mathcal{F} = L_T \cap R$. Since each word in L_T encodes a pair of binary trees with the same number of carets, each word in $\mathcal{F}$ encodes a reduced pair of binary trees, that is, a unique element of F. Furthermore, because L_T is a 2-counter language, it follows that $\mathcal{F}$ is as well.

We can now restrict the bijection τ to obtain another bijection

$$\tau_{\text{red}} : \mathcal{F} \to \{\text{reduced pairs of finite rooted binary trees with the same number of carets}\}.$$

Thus we have shown that $\mathcal{F}$ is a 2-counter language that is a normal form for Thompson's group F.

Let $\nu = \tau^{-1}$. If (T, S) is a pair of binary trees, then $\nu(T, S)$ is the string in L_T representing the caret types from the pair. We abuse this notation in several ways. If $g = (T, S)$, we write $\nu(g)$ interchangeably with $\nu(T, S)$. If $g = (T, S)$ is reduced, then we understand that $\nu(g) = \tau^{-1}(g) = \tau_{\text{red}}^{-1}(g)$. Thus in this case $\nu(g) \in \mathcal{F}$.

We now show that the language $\mathcal{F}$ is a quasigeodesic normal form for F.

Proposition 4.3. *The normal form language $\mathcal{F}$ for Thompson's group F is quasigeodesic.*

Proof. It follows from work of Fordham [9] that the number of carets in either tree in a reduced tree pair diagram for $g \in F$ is coarsely equivalent to the word length of g with respect to the finite generating set $\{x_0, x_1\}$. The length of an element of $\mathcal{F}$ is exactly the number of carets in either tree in the reduced tree pair diagram representing $g \in F$, hence $\mathcal{F}$ is a quasigeodesic normal form. $\square$

We next show that the language $\mathcal{F}$ of normal forms is not a context-free language. Since 1-counter languages form a subset of the set of context-free languages, it follows that we can not improve on Theorem 4.1. We prove Lemma 4.5 using Ogden's lemma, which is a corollary to the standard pumping lemma for context-free languages, and is stated below.

Lemma 4.4 Ogden's lemma [11, Lemma 6.2]). *Let L be a context free language. Then there is a constant p so that if z is any word in L, and we mark any p or more positions of z as* distinguished, *then we can write z = uvwxy so that*

i) *v and x together have at least one distinguished position;*
ii) *vwx has at most p distinguished positions;*
iii) *for all $i \geq 0$ we have $uv^i wx^i y \in L$.*

Ogden's lemma allows us to omit the first letter of any string in the following proof from the substrings v and x in the application of the lemma, an assumption one cannot make when applying the standard pumping lemma for context-free languages.

Lemma 4.5. *Let $\mathcal{F}$ be as above. Then $\mathcal{F}$ is not context-free.*

Proof. Assume for contradiction that $\mathcal{F}$ is context-free. Let p be the constant guaranteed by Ogden's lemma, and consider the convolution

$$w = \binom{e}{e}\binom{a}{a}^{p}\binom{(}{\,}\binom{a}{\,}^{p}\binom{)}{a}^{p}\binom{(}{)}\binom{a}{\,}^{p}\binom{r}{r},$$

which lies in $\mathcal{F}$. Mark all but the first and last symbol as distinguished, and apply Ogden's lemma. As v and x together must have at least one distinguished position, one of these strings must contain either " " or ")" in either the top or bottom line. First suppose vx contains " " in the top line. Then vx contains a symbol from the substring

$$\binom{\,}{a}^{p}\binom{(}{\,}.$$

However, since $|vwx| \leq p$ and $|\binom{a}{\,}^{p}| = p$, it follows that vx cannot contain the symbol ")" in the top line. An almost identical argument shows that if vx contains " " in the bottom line, it cannot also contain ")" in the bottom line. In either case, $uv^{i}wx^{i}y \notin \mathcal{F}$ for $i = 1$, and hence $\mathcal{F}$ is not context free. $\qquad\square$

5 Multiplier languages for the 3-counter graph automatic structure

We now construct the automata $\quad_{x_0}$ and $\quad_{x_1}$, which accept the multiplier languages

$$L_{x_0} = \{\otimes\, u, v) \mid u, v \in \mathcal{F}, \overline{u}x_0 = \overline{v}\}$$

and

$$L_{x_1} = \{\otimes\, u, v) \mid u, v \in \mathcal{F}, \overline{u}x_1 = \overline{v}\}.$$

As noted in [7], when constructing -graph automatic groups, a multiplier automaton which reads the convolution $\otimes\, u, v)$ must check both that u and v lie in the normal form language *and* that $\overline{u}x_1 = \overline{v}$. This is not an issue for either automatic or graph automatic groups; in both of these scenarios, the normal form language is regular, and since intersections of regular languages are regular, it is not necessary for the multiplier automata to check membership in the normal form language. As our normal form languages are not regular, the multiplier automata must also verify this membership condition. The language L_{x_0} is accepted by a nonblind deterministic 2-counter automaton, while L_{x_1} is accepted by a nonblind deterministic 3-counter automaton.

It is proven in [7] that for $\mathcal{R} = \{\text{regular languages}\}$ or $_1 = \{1\text{-counter languages}\}$ we have $L_x \in \mathcal{R}$ resp. $_1$) if and only if $L_{x\,1} \in \mathcal{R}$ resp. $_1$). Hence it suffices to consider only the multiplier languages L_{x_0} and L_{x_1}.

5.1 The multiplier language L_{x_0}

Multiplication of $g = T, S)$ by x_0 induces a particular rearrangement of the subtrees of the original diagram, as shown in Figure 5 when the root caret of T has at least one left child. To compute this product, the tree pair diagram for x_0, which corresponds to $\otimes\, re, er) \in \mathcal{F}$, is placed to the left of the tree pair diagram for g. Then possibly) unreduced representatives of the two elements are created in which the middle trees are identical; this may entail adding an additional caret as a left child to caret 0 in both T and S. The outermost trees together form the product gx_0. The final result of any multiplication of tree pair diagrams is a unique reduced tree pair diagram, but this may include an intermediate stage where the resulting pair of trees is not reduced.

Proposition 5.1. *The multiplier language for the generator x_0 defined by*

$$L_{x_0} = \{\otimes\, u, v) \mid u, v \in \mathcal{F} \text{ and } \overline{u}x_0 =_F \overline{v}\}$$

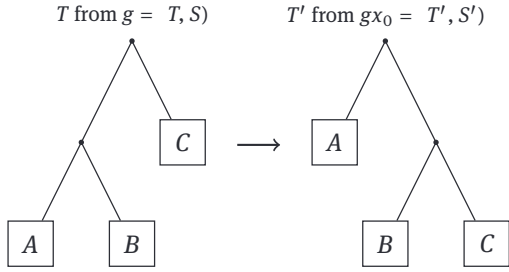*is recognized by a nonblind deterministic 2-counter automaton.*

Figure 5. Part of the tree pair diagrams for $g = (T, S)$ and $gx_0 = (T', S')$. The letters A, B, and C represent possibly empty subtrees, and $S = S'$.

Proof. The language L_{x_0} will be the union of three sub-languages, based on the analysis of multiplication by x_0 given below.

Let $g = (T, S)$ as a reduced pair of trees. Recall that $v(g) \in \mathcal{F}$ is the convolution of strings of caret types describing the trees S and T. To describe $u = v(g)$ and $v(gx_0)$ we divide into three cases depending on whether the root caret of T has no left child, no right child, or has children on both sides. In terms of $u = v(g)$, these cases correspond, respectively, to

1) an initial letter of "r" in the top line of u when written as a convolution;
2) initial letter of "e" and final letter "r" in the top line of u when written as a convolution;
3) neither of the above conditions.

In each case we obtain a unique string in $\mathcal{F}$ which corresponds to gx_0.

Case 1: the root caret of T has no left child. To perform the multiplication by x_0 a caret must be added to the left leaf of the root caret in T and to the left leaf of caret 0 in S, creating an unreduced representative $g^* = (T^*, S^*)$. Let $gx_0 = (T', S')$ denote the product as a reduced tree pair diagram.

In this case we must have $v(g) = \binom{r}{z_0}\binom{\gamma_1}{\gamma_2}$ for some $z_0 \in \{e, r\}$ and $\gamma_1, \gamma_2 \in \Sigma_T^*$, and using the combinatorial rearrangement of carets induced by multiplication by x_0, we can write

- $v(g) = \begin{pmatrix} r \\ z_0 \end{pmatrix}\begin{pmatrix} \gamma_1 \\ \gamma_2 \end{pmatrix}$,

- $v(g^*) = \begin{pmatrix} e \\ e \end{pmatrix}\begin{pmatrix} r \\ z_0 \end{pmatrix}\begin{pmatrix} \gamma_1 \\ \gamma_2 \end{pmatrix}$ and

- $v(gx_0) = \begin{pmatrix} r \\ e \end{pmatrix}\begin{pmatrix} e \\ z_0 \end{pmatrix}\begin{pmatrix} \gamma_1 \\ \gamma_2 \end{pmatrix}$.

Notice that this includes g being the identity, in which case $v(g) = \binom{r}{r}$ and $v(gx_0) = \binom{r}{e}\binom{e}{r}$.

Case 2: the root caret of T does have a left child and the following three conditions hold:

i) the root caret of T has no right child;
ii) the left child of the root caret of T has no right child;
iii) the last exterior caret of S is exposed.

Then multiplication by x_0 yields an unreduced representative $(gx_0)^* = (T^*, S^*)$. We must eliminate a redundant caret from both T^* and S^* to obtain $gx_0 = (T', S')$.

Given conditions i), ii) and iii), the string $v(g)$ must end with the symbols $\binom{e}{z_0}\binom{r}{e}$ for some $z_0 \in \{e, r\}$. Hence the string representing the unreduced $(gx_0)^* = (T^*, S^*)$ must end in $\binom{r}{z_0}\binom{e}{e}$. To obtain the reduced tree pair diagram for the product, the final redundant caret must be eliminated from the tree pair diagram (T^*, S^*); specifically, the right child of the root in T^* has two exposed leaves, as does the final (rightmost) caret in S^*. These are removed to obtain the reduced tree pair diagram (T', S') for gx_0.

Thus, for some $\gamma_1, \gamma_2 \in \Sigma_T^*$ and $z_0 \in \{e, r\}$ we have

- $v(g) = \begin{pmatrix} \gamma_1 \\ \gamma_2 \end{pmatrix}\begin{pmatrix} e \\ z_0 \end{pmatrix}\begin{pmatrix} r \\ e \end{pmatrix}$,

- $v(gx_0^*) = \begin{pmatrix} \gamma_1 \\ \gamma_2 \end{pmatrix}\begin{pmatrix} r \\ z_0 \end{pmatrix}\begin{pmatrix} e \\ e \end{pmatrix}$ and

- $v(gx_0) = \begin{pmatrix} \gamma_1 \\ \gamma_2 \end{pmatrix}\begin{pmatrix} r \\ z_0 \end{pmatrix}$.

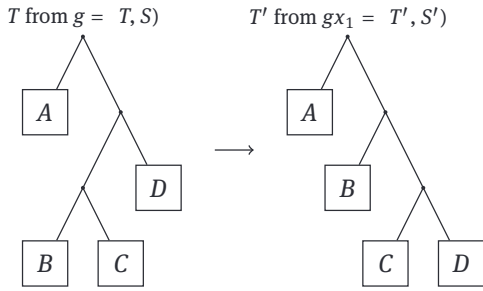T from $g = (T, S)$ T' from $gx_1 = (T', S')$

Figure 6. If $g = (T, S)$ and $gx_1 = (T', S')$, where the tree T contains at least a root caret whose right child has a left child, the combinatorial rearrangement of the subtrees of T yielding T'. Here, A, B, C and D represent possibly empty subtrees and $S = S'$.

Case 3: the element $g = (T, S)$ does not belong to Case 1 or Case 2. In this case, the rearrangement of the subtrees of the domain tree is exactly as depicted in Figure 5. Multiplication has no effect on the range tree: that is, $S = S'$. Let γ_A, γ_B, γ_C denote the substrings that encode the subtrees A, B, and C, respectively, and $\chi_A, \chi_B, \chi_C \in \Sigma_T^*$ the strings of labels for the carets in S paired, respectively, with the carets in subtrees A, B and C of T. Then

- $v(g) = \begin{pmatrix} \gamma_A \\ \chi_A \end{pmatrix} \begin{pmatrix} e \\ z_0 \end{pmatrix} \begin{pmatrix} \gamma_b \\ \chi_B \end{pmatrix} \begin{pmatrix} r \\ z_1 \end{pmatrix} \begin{pmatrix} \gamma_C \\ \chi_C \end{pmatrix}$ and

- $v(gx_0) = \begin{pmatrix} \gamma_A \\ \chi_A \end{pmatrix} \begin{pmatrix} r \\ z_0 \end{pmatrix} \begin{pmatrix} \gamma_b \\ \chi_B \end{pmatrix} \begin{pmatrix} e \\ z_1 \end{pmatrix} \begin{pmatrix} \gamma_C \\ \chi_C \end{pmatrix}$

for some $z_0, z_1 \in \Sigma_T$.

Define a language $\mathbb{N}_0$ to consist of the union of all convolutions $\otimes(u, v)$ of the following forms:

- $u = \begin{pmatrix} r \\ z_0 \end{pmatrix} \begin{pmatrix} \gamma_1 \\ \gamma_2 \end{pmatrix}$ and $v = \begin{pmatrix} r \\ e \end{pmatrix} \begin{pmatrix} e \\ z_0 \end{pmatrix} \begin{pmatrix} \gamma_1 \\ \gamma_2 \end{pmatrix}$;

- $u = \begin{pmatrix} \gamma_1 \\ \gamma_2 \end{pmatrix} \begin{pmatrix} e \\ z_0 \end{pmatrix} \begin{pmatrix} r \\ e \end{pmatrix}$ and $v = \begin{pmatrix} \gamma_1 \\ \gamma_2 \end{pmatrix} \begin{pmatrix} r \\ z_0 \end{pmatrix}$;

- $u = \begin{pmatrix} \gamma_A \\ \chi_A \end{pmatrix} \begin{pmatrix} e \\ z_0 \end{pmatrix} \begin{pmatrix} \gamma_b \\ \chi_B \end{pmatrix} \begin{pmatrix} r \\ z_1 \end{pmatrix} \begin{pmatrix} \gamma_C \\ \chi_C \end{pmatrix}$ and $v = \begin{pmatrix} \gamma_A \\ \chi_A \end{pmatrix} \begin{pmatrix} r \\ z_0 \end{pmatrix} \begin{pmatrix} \gamma_b \\ \chi_B \end{pmatrix} \begin{pmatrix} e \\ z_1 \end{pmatrix} \begin{pmatrix} \gamma_C \\ \chi_C \end{pmatrix}$,

where $z_0, z_1 \in \Sigma_T$ and $\gamma_i \in \Sigma_T^*$. Note that u and v may no longer lie in $\mathcal{F}$ or even L_T.

It follows from Lemma 2.6 that these three sub-languages are regular, and hence their union $\mathbb{N}_0$ is regular as well. The language L_{x_0} is then the intersection of $\mathbb{N}_0$ with the 2-counter language

$$\{\otimes(u, v) \mid u \in \mathcal{F}, v \in \Sigma_T^*\},$$

because the construction of $\mathbb{N}_0$ guarantees that if $u \in \mathcal{F}$ and $\otimes(u, v) \in \mathbb{N}_0$, then the string v lies in $\mathcal{F}$. Hence the language L_{x_0} can be recognized by a nonblind deterministic 2-counter automaton. □

5.2 The multiplier language L_{x_1}

If $g = (T, S) \in F$, then the process of multiplying g by x_1 is described in five cases which depend on the arrangement of carets in T and S.

In this section, if $g = (T, S)$ is as in Figure 6, we will use γ_A, γ_B, γ_C and γ_D to denote the strings in L_{int} that encode the subtrees A, B, C and D, respectively. Let χ_A, χ_B, χ_C and χ_D denote the strings in Σ_T^* that encode the carets of S which are paired, respectively, with subtrees A, B, C and D. If γ_R is empty for $R \in \{A, B, C, D\}$ then we say that χ_R is empty as well.

We now identify all pairs of strings in $\mathcal{F}$ which correspond to possible pairs g and gx_1. In Cases 1–4 below the difference between $v(g)$ and $v(gx_1)$ lies either at the end of the string or immediately after the unique caret of type "r" in the top line of the convolution $v(g)$. A finite state machine can easily detect this change. Then two counters are required to ensure that $v(g)$ lies in $\mathcal{F}$, and by the construction of the language, $v(gx_1)$ must lie in $\mathcal{F}$ as well. Thus the set of all convolutions described in Cases 1–4 will form a nonblind deterministic

2-counter language. The set of strings described in the final case of this section will require an additional counter to detect that the change between v g) and v gx_1) lies at the correct position in the interior of the convolution.

Case 1: Two carets must be added to $g = T, S$) in order to multiply by x_1, that is, the root caret of T has no right child. Then we must add a right child with a left child to the rightmost carets of T and S, creating an unreduced representative of g, before performing the multiplication. Let $g^* = T^*, S^*$) denote this unreduced representative for g.

Since the root caret of T has no right child, the string corresponding to T ends with "r". Then for some $z_0 \in \{e, r\}$, we have

- v g) $= \begin{pmatrix} \gamma_A \\ \chi_A \end{pmatrix} \begin{pmatrix} r \\ z_0 \end{pmatrix}$,

- v g^*) $= \begin{pmatrix} \gamma_A \\ \chi_A \end{pmatrix} \begin{pmatrix} r \\ z_0 \end{pmatrix} \begin{pmatrix} a \\ a \end{pmatrix} \begin{pmatrix} e \\ e \end{pmatrix}$ and

- v gx_1) $= \begin{pmatrix} \gamma_A \\ \chi_A \end{pmatrix} \begin{pmatrix} r \\ z_0 \end{pmatrix} \begin{pmatrix} e \\ a \end{pmatrix} \begin{pmatrix} e \\ e \end{pmatrix}$.

Notice that this case also includes the identity with $\gamma_A = \chi_A = \epsilon$ and $z_0 = r$. Then

$$v \ x_1) = \begin{pmatrix} r \\ r \end{pmatrix} \begin{pmatrix} e \\ a \end{pmatrix} \begin{pmatrix} e \\ e \end{pmatrix}.$$

Case 2: One caret must be added to $g = T, S$) in order to multiply by x_1, that is, the root caret of T has a right child which does not have a left child, so that the string in Σ_T^* corresponding to T is $\gamma_A r e \gamma_D$. To perform multiplication by x_1, we must add a left child to the right child of the root caret of T and to the corresponding leaf in S, creating an unreduced representative of g, which we denote by $g^* = T^*, S^*$). We consider three possibilities for this multiplication, based on the caret type z_0 of the caret in S paired with the root caret of T. Let n denote the infix number of the root caret in T and the corresponding caret of type z_0 in S.

In subcases a)– c) below, we always have

$$v \ g) = \begin{pmatrix} \gamma_A \\ \chi_A \end{pmatrix} \begin{pmatrix} r \\ z_0 \end{pmatrix} \begin{pmatrix} e \\ z_1 \end{pmatrix} \begin{pmatrix} \gamma_D \\ \chi_D \end{pmatrix},$$

where $z_0, z_1 \in \Sigma_T$.

a) First suppose $z_0 \in \{e, r, , b)\}$. Then the new caret is added to S as the left child of the caret of infix number $n + 1$ and will be type "a" in $S^* = S'$. Then we have

- v g^*) $= \begin{pmatrix} \gamma_A \\ \chi_A \end{pmatrix} \begin{pmatrix} r \\ z_0 \end{pmatrix} \begin{pmatrix} a \\ a \end{pmatrix} \begin{pmatrix} e \\ z_1 \end{pmatrix} \begin{pmatrix} \gamma_D \\ \chi_D \end{pmatrix}$ and

- v gx_1) $= \begin{pmatrix} \gamma_A \\ \chi_A \end{pmatrix} \begin{pmatrix} r \\ z_0 \end{pmatrix} \begin{pmatrix} e \\ a \end{pmatrix} \begin{pmatrix} e \\ z_1 \end{pmatrix} \begin{pmatrix} \gamma_D \\ \chi_D \end{pmatrix}$.

b) Next suppose z_0 is type "a". Then the new caret is added to S as the right child of caret n of type z_0), which changes the type of caret n in S to " ", and the new caret is of type "$)$". Then we have

- v g^*) $= \begin{pmatrix} \gamma_A \\ \chi_A \end{pmatrix} \begin{pmatrix} r \\ \end{pmatrix} \begin{pmatrix} a \\ ) \end{pmatrix} \begin{pmatrix} e \\ z_1 \end{pmatrix} \begin{pmatrix} \gamma_D \\ \chi_D \end{pmatrix}$ and

- v gx_1) $= \begin{pmatrix} \gamma_A \\ \chi_A \end{pmatrix} \begin{pmatrix} r \\ \end{pmatrix} \begin{pmatrix} e \\ ) \end{pmatrix} \begin{pmatrix} e \\ z_1 \end{pmatrix} \begin{pmatrix} \gamma_D \\ \chi_D \end{pmatrix}$.

c) Finally, suppose z_0 is type "$)$". As in b), the new caret is added to S as the right child of caret n, which changes to type "b", and the new caret is of type "$)$". Then we have

- v g^*) $= \begin{pmatrix} \gamma_A \\ \chi_A \end{pmatrix} \begin{pmatrix} r \\ b \end{pmatrix} \begin{pmatrix} a \\ ) \end{pmatrix} \begin{pmatrix} e \\ z_1 \end{pmatrix} \begin{pmatrix} \gamma_D \\ \chi_D \end{pmatrix}$ and

- v gx_1) $= \begin{pmatrix} \gamma_A \\ \chi_A \end{pmatrix} \begin{pmatrix} r \\ b \end{pmatrix} \begin{pmatrix} e \\ ) \end{pmatrix} \begin{pmatrix} e \\ z_1 \end{pmatrix} \begin{pmatrix} \gamma_D \\ \chi_D \end{pmatrix}$.

In all three cases, $v\ gx_1)$ is a reduced string. If χ_D is empty, then $z_1 = e$ because $v\ g)$ is itself a reduced string. Note that for a given string $v\ g)$ in Case 2, a finite state automaton can decide which subcase $v\ g)$ lies in.

Case 3: After multiplication by x_1, two carets must be removed to obtain the reduced tree pair diagram for the product. Suppose that the root caret of T has a right child which has a left child. If the following three conditions hold,

i)   the right child of the root caret of T does not have a right child;

ii) the left child of the right child of the root caret of T has no children;

iii) the final two carets of S are exterior carets with no interior children,

then multiplication yields an unreduced representative $gx_1^* = T^*, S^*)$. We must eliminate two redundant carets from each of T^* and S^* to obtain $gx_1 = T', S')$.

Given conditions i) and ii), the string corresponding to T ends with "rae". Condition iii) implies that the string corresponding to S ends with "$z_0 ee$" for $z_0 \in \{r, e\}$. Then we have

-   $v\ g) = \begin{pmatrix} \gamma_A \\ \chi_A \end{pmatrix} \begin{pmatrix} r \\ z_0 \end{pmatrix} \begin{pmatrix} a \\ e \end{pmatrix} \begin{pmatrix} e \\ e \end{pmatrix}$,

-   $v\ gx_1^*) = \begin{pmatrix} \gamma_A \\ \chi_A \end{pmatrix} \begin{pmatrix} r \\ z_0 \end{pmatrix} \begin{pmatrix} e \\ e \end{pmatrix} \begin{pmatrix} e \\ e \end{pmatrix}$ and

-   $v\ gx_1) = \begin{pmatrix} \gamma_A \\ \chi_A \end{pmatrix} \begin{pmatrix} r \\ z_0 \end{pmatrix}$.

Case 4: After multiplication by x_1, one caret must be removed to obtain the reduced tree pair diagram for the product. Suppose that the string w does not lie in Cases 1, 2 or 3. Suppose further that the root caret of T has a right child that has a left child. If the following three conditions hold,

i)   the right child of the root caret of T does not have a right child;

ii) the left child of the right child of the root caret of T does not have a right child but may have a left child);

iii) the final caret of S is an exterior caret with no left child,

then multiplication yields an unreduced representative $gx_1^* = T^*, S^*)$. We must eliminate a single redundant caret from each of T^* and S^* to obtain $gx_1 = T', S')$.

It follows from conditions i) and ii) that the string in Σ_T^* corresponding to T is $\gamma_A r \gamma_B ae$. Then for $z_0 \in \Sigma_T$ we have

-   $v\ g) = \begin{pmatrix} \gamma_A \\ \chi_A \end{pmatrix} \begin{pmatrix} r \\ z_0 \end{pmatrix} \begin{pmatrix} \gamma_B \\ \chi_B \end{pmatrix} \begin{pmatrix} a \\ e \end{pmatrix} \begin{pmatrix} e \\ e \end{pmatrix}$,

-   $v\ gx_1^*) = \begin{pmatrix} \gamma_A \\ \chi_A \end{pmatrix} \begin{pmatrix} r \\ z_0 \end{pmatrix} \begin{pmatrix} \gamma_B \\ \chi_B \end{pmatrix} \begin{pmatrix} e \\ e \end{pmatrix} \begin{pmatrix} e \\ e \end{pmatrix}$ and

-   $v\ gx_1) = \begin{pmatrix} \gamma_A \\ \chi_A \end{pmatrix} \begin{pmatrix} r \\ z_0 \end{pmatrix} \begin{pmatrix} \gamma_B \\ \chi_B \end{pmatrix} \begin{pmatrix} e \\ e \end{pmatrix}$.

To avoid overlap with Case 3, if $\chi_B = 0$ then $z_0 \notin \{e, r\}$ to ensure that there is only one redundant caret in $v\ gx_1^*)$.

To summarize Cases 1–4, let $\mathcal{N}_1$ be the set of all $\otimes u, v)$ where $u = v\ g)$ and $v = v\ gx_1)$ are as listed in Cases 1–4, where γ_i and χ_i are now allowed to be any strings in Σ_T^*. As with the construction of L_{x_0}, we are not concerned with whether $u, v \in \mathcal{F}$. Each possible difference between the strings u and v in the previous four cases is easily detected by a finite state machine, and thus the language $\mathcal{N}_1$ is regular. Finally, intersect $\mathcal{N}_1$ with the language $\{\otimes x, y) \mid x \in \mathcal{F}, y \in \Sigma_T^*\}$ where the latter is a nonblind deterministic 2-counter language. Denote the intersection by $\mathcal{N}$; it is also a nonblind deterministic 2-counter language and consists of exactly those convolutions covered in Cases 1–4. Note that it is enough to check that one of u and v lies in $\mathcal{F}$ because the construction of $\mathcal{N}_1$ ensures that when one component lies in $\mathcal{F}$, the other one does as well.

Case 5: No carets must be added to multiply by x_1 and the resulting tree pair diagram is necessarily reduced. That is, suppose that $g = T, S)$ does not belong to any of the previous cases. Then multiplication by x_1 does not affect the range tree; if $g = T, S)$ and $gx_1 = T', S')$, then $S = S'$. The trees T and T' are depicted in Figure 6. We consider two cases according to whether the subtree C is empty, and note that a finite state

machine can detect which case a convolution lies in. Note that $\otimes \Sigma_T^*, \mathcal{F})$ is a 2-counter language, because Σ_T^* is regular and $\mathcal{F}$ requires two counters.

In Case 5, there are at most two carets which change type between T and T':

1) The parent caret of subtrees B and C; this caret is interior in T and exterior in T', and we denote it d. This change always occurs.

2) The root caret of subtree C, which changes type if and only if C is not empty. We denote this caret by f.

We show below that the x_1 multiplier language can be easily expressed using three counters. In this case, we switch our perspective and focus on the tree T' from the tree pair diagram for gx_1 rather than the tree T.

Case 5 a): Subtree C is empty. In this case,

$$u = v\ g) = \begin{pmatrix} \gamma_A \\ \chi_A \end{pmatrix} \begin{pmatrix} r \\ z_0 \end{pmatrix} \begin{pmatrix} \gamma_B \\ \chi_B \end{pmatrix} \begin{pmatrix} a \\ z_1 \end{pmatrix} \begin{pmatrix} e \\ z_2 \end{pmatrix} \begin{pmatrix} \gamma_D \\ \chi_D \end{pmatrix} \quad \text{and}$$

$$v = v\ gx_1) = \begin{pmatrix} \gamma_A \\ \chi_A \end{pmatrix} \begin{pmatrix} r \\ z_0 \end{pmatrix} \begin{pmatrix} \gamma_B \\ \chi_B \end{pmatrix} \begin{pmatrix} e \\ z_1 \end{pmatrix} \begin{pmatrix} e \\ z_2 \end{pmatrix} \begin{pmatrix} \gamma_D \\ \chi_D \end{pmatrix}$$

for some $z_0, z_1, z_2 \in \Sigma_T$. Let $\mathcal{K}_1$ be the regular language of all strings $\otimes u, v)$ where u and v have the above form with γ_B and γ_C replaced by arbitrary strings from Σ_{int}^* and γ_A, γ_D and χ_Y replaced by arbitrary strings from Σ_T^*, for $Y \in \{A, B, D\}$. As γ_B consists only of interior caret types, a finite state machine can easily check that after $\begin{pmatrix} r \\ z_0 \end{pmatrix}$ is read in v, the next symbol $\begin{pmatrix} x \\ y \end{pmatrix}$ where $x \notin \{a, b, \ ,)\}$ is $\begin{pmatrix} e \\ z_1 \end{pmatrix}$ and that it is paired with $\begin{pmatrix} a \\ z_1 \end{pmatrix}$ in u. Assume without loss of generality that this condition is also verified in $\mathcal{K}_1$. Then $\mathcal{L}_1 = \mathcal{K}_1 \cap \otimes \mathcal{F}, \Sigma_T^*)$ is a 2-counter language accepting exactly those convolutions $\otimes u, v)$ in Case 5 a), that is, with $C = \emptyset$.

Case 5 b): Subtree C is not empty. In this case, in $v\ gx_1)$ write $\gamma_C = \eta_1 f \eta_2$ and in $v\ g)$ write $\gamma_C = \eta_1 \overline{f} \eta_2$ where f, resp. $\overline{f}$, corresponds to the root caret of the subtree C. Here $\eta_i \in L_{\text{int}}^*$, and we let η_1' and η_2' in Σ_T^* denote the strings of caret types paired with the caret types in η_1 and η_2, respectively. Then

$$u = v\ g) = \begin{pmatrix} \gamma_A \\ \chi_A \end{pmatrix} \begin{pmatrix} r \\ z_0 \end{pmatrix} \begin{pmatrix} \gamma_B \\ \chi_B \end{pmatrix} \begin{pmatrix} \ \\ z_1 \end{pmatrix} \begin{pmatrix} \eta_1 \\ \eta_1' \end{pmatrix} \begin{pmatrix} \overline{f} \\ z_3 \end{pmatrix} \begin{pmatrix} \eta_2 \\ \eta_2' \end{pmatrix} \begin{pmatrix} e \\ z_2 \end{pmatrix} \begin{pmatrix} \gamma_D \\ \chi_D \end{pmatrix} \quad \text{and}$$

$$v = v\ gx_1) = \begin{pmatrix} \gamma_A \\ \chi_A \end{pmatrix} \begin{pmatrix} r \\ z_0 \end{pmatrix} \begin{pmatrix} \gamma_B \\ \chi_B \end{pmatrix} \begin{pmatrix} e \\ z_1 \end{pmatrix} \begin{pmatrix} \eta_1 \\ \eta_1' \end{pmatrix} \begin{pmatrix} f \\ z_3 \end{pmatrix} \begin{pmatrix} \eta_2 \\ \eta_2' \end{pmatrix} \begin{pmatrix} e \\ z_2 \end{pmatrix} \begin{pmatrix} \gamma_D \\ \chi_D \end{pmatrix},$$

where $z_0, z_1, z_2, z_3 \in \Sigma_T$ and

1) if the right subtree of the root caret of C is empty then $\overline{f} =)$ and $f = a$, or

2) if the right subtree of the root caret of C is not empty then $\overline{f} = b$ and $f = \ $.

Let $\mathcal{K}_2$ be the regular language of all strings $\otimes u, v)$ where u and v have the above form with all γ_B, η_1 and η_2 replaced by arbitrary strings from Σ_{int}^* and $\gamma_A, \gamma_D, \chi_Y, \eta_1'$ and η_2' replaced by arbitrary strings from Σ_T^*, for $Y \in \{A, B, D\}$. As in Case 5 a), assume that the finite state machine recognizing $\mathcal{K}_2$ verifies that the change in the caret type of the parent caret of subtree B occurs in the appropriate position that is, the first "e" following the root caret in the string representing T').

To detect that the change in type from caret $\overline{f}$ to caret f occurs in the correct position in the string we will use a single counter. The counter instructions used to show that a string from Σ_{int}^* forms a valid interior subtree were as follows:

1) when the machine reads " " the counter is incremented by 1;

2) the machine may read ")" only when the value of the counter is positive, and then the value of the counter is decremented by 1;

3) if the value of the counter is 0 and the letter "b" is read, the machine proceeds to a fail state.

When these counter instructions are applied to a machine reading $\tau_{\text{tree}}^{-1}\ T')$, the value of the counter is 0 before each substring γ_Y is read, for $Y \in \{A, B, C, D\}$. Moreover, after the final letter of η_1 is read, the value of the counter is again 0. If η_2 is nonempty, then upon reading " " corresponding to the root caret of subtree C caret f) the value of the counter is set to 1 and will not equal zero again until after the entire string η_2 has been read by the machine. Hence in Figure 7 we check that the value of the counter is strictly positive as η_2 is read.
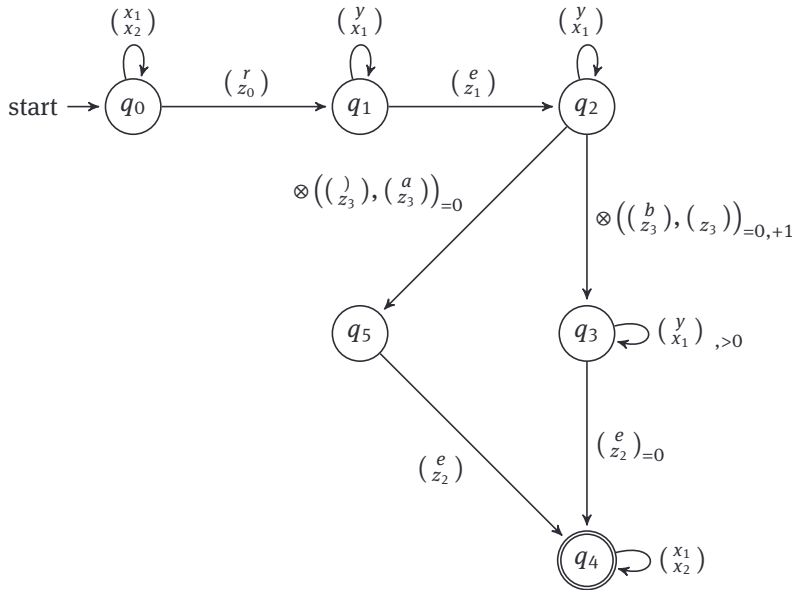
Figure 7. For $\otimes u, v)$ as in Case 5, this machine checks that the change in the root caret of subtree C from T to T' occurs in the correct spot in the string and has the correct label change. The edge labels correspond to the letters in the string $v = v\ gx_1) \in \mathcal{F}$ and the corresponding letter from $u = v\ g)$ is ignored, except at the root caret of subtree C, where both the letter from u and from v are considered. The letters z_0, z_1, z_2 are as in the expressions for u and v above, $y \in \Sigma_{\text{int}}$ and $x_i \in \Sigma_T$. The start state is q_0 and the accept state is q_4. The letter $\mathfrak{I}$ denotes the set of counter instructions used in the construction of $_{\text{int}}$.

The machine in Figure 7 reads convolutions $\otimes u, v) \in \mathcal{K}_2$. The state transitions are determined solely by the letter from v, with one exception: the carets corresponding to the root caret of the subtree C in T' or T). In Figure 7 we label the state transitions only by the letter read in v except at this position. When $v \in \mathcal{F}$ as well, this machine works as follows. Let $y \in \Sigma_{\text{int}}$ and $x \in \Sigma_T$ be arbitrary letters. Let $\mathfrak{I}$ denote the set of counter instructions given above. The label $\left(\begin{smallmatrix} \alpha \\ \beta \end{smallmatrix}\right)$ denotes the appropriate counter rules from $\mathfrak{I}$ which apply to the letter α from $\left(\begin{smallmatrix} \alpha \\ \beta \end{smallmatrix}\right)$, for $\alpha, \beta \in \Sigma_T$.

1) The loop at state q_0 reads the string γ_A corresponding to subtree A in T' and the corresponding string χ_A from $v\ gx_1)$. The edge labeled $\left(\begin{smallmatrix} r \\ z_0 \end{smallmatrix}\right)$ can only be read when this subtree is completed.

2) The loop at state q_1 reads the string γ_B corresponding to subtree B in T' and the corresponding string χ_B from $v\ gx_1)$. The edge labeled $\left(\begin{smallmatrix} e \\ z_1 \end{smallmatrix}\right)$ can only be read when this subtree is completed.

3) The loop at state q_2 reads the string η_1 corresponding to the left subtree of the root caret of subtree C in T' and the corresponding string η_1'. We apply the counter instructions $\mathfrak{I}$ to the top line of $v\ gx_1)$. After this string has been read, the value of the counter is 0.

4) If the right subtree of the root caret of subtree C is empty, then the root caret of C in $v\ gx_1)$ has type ")" in T' and the machine transitions to state q_5 verifying the counter value. As this corresponds to the root caret of C the machine simultaneously checks that the corresponding symbol from $u = v\ g)$ is $\left(\begin{smallmatrix} a \\ z_3 \end{smallmatrix}\right)$ before transitioning to state q_5.

5) If the right subtree of the root caret of subtree C in $v\ gx_1)$ is not empty, then the root caret of C has type " " in T' and the machine transitions to state q_3 verifying the counter value. The machine simultaneously checks that the corresponding symbol from $u = v\ g)$ is $\left(\begin{smallmatrix} b \\ z_3 \end{smallmatrix}\right)$. The loop at state q_3 reads the string η_2 corresponding to the right subtree of the root caret of C, using the counter instructions $\mathfrak{I}$ for the top line of $v\ gx_1)$ but additionally verifying that the value of the counter is never 0 while this string is read. After the entire string η_2 is read the counter value is 0, and this is verified along the edge leading to state q_4.

6) The loop at state q_4 reads the string γ_D corresponding to subtree D in T' and the corresponding string χ_D.

Let $\mathcal{L}_2$ be the intersection of the language accepted by the machine in Figure 7 with $\otimes \Sigma_T^*, \mathcal{F})$; the latter verifies that $v \in \mathcal{F}$ and hence $u \in \mathcal{F})$ as well. Then $\mathcal{L}_2$ is the language of all convolutions accepted in Case 5 b);

as it is the intersection of a 1-counter language with a 2-counter language, we conclude that $\mathcal{L}_2$ is a 3-counter language. It follows that L_{x_1} is the union of $\mathcal{L}_1$ and $\mathcal{L}_2$, hence a nonblind deterministic 3-counter language. This completes the proof of Theorem 4.1.

References

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

[2] K. S. Brown and R. Geoghegan, An infinite-dimensional torsion-free FP_∞ group, *Invent. Math.* **77** 1984), no. 2, 367–381.

[3] J. W. Cannon, W. J. Floyd and W. R. Parry, Introductory notes on Richard Thompson's groups, *Enseign. Math. 2)* **42** 1996), no. 3–4, 215–256.

[4] S. Cleary and J. Taback, Combinatorial properties of Thompson's group F, *Trans. Amer. Math. Soc.* **356** 2004), no. 7, 2825–2849.

[5] S. Cleary and J. Taback, Seesaw words in Thompson's group F, in: *Geometric Methods in Group Theory*, Contemp. Math. 372, American Mathematical Society, Providence 2005), 147–159.

[6] M. Elder, É. Fusy and A. Rechnitzer, Counting elements and geodesics in Thompson's group F, *J. Algebra* **324** 2010), no. 1, 102–121.

[7] M. Elder and J. Taback, -graph automatic groups, *J. Algebra* **413** 2014), 289–319.

[8] M. Elder and J. Taback, Thompson's group F is 1-counter graph automatic, preprint 2015), http://arxiv.org/abs/1501.04313.

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

[10] V. S. Guba and M. V. Sapir, The Dehn function and a regular set of normal forms for R. Thompson's group F, *J. Austral. Math. Soc. Ser. A* **62** 1997), no. 3, 315–328.

[11] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, 1979.

[12] O. Kharlampovich, B. Khoussainov and A. Miasnikov, From automatic structures to automatic groups, *Groups Geom. Dyn.* **8** 2014), no. 1, 157–198.

[13] V. Shpilrain and A. Ushakov, Thompson's group and public key cryptography, in: *Applied Cryptography and Network Security*, Lecture Notes in Comput. Sci. 3531, Springer, Berlin 2005), 151–163.