$\mathcal{C}$-GRAPH AUTOMATIC GROUPS

MURRAY ELDER AND JENNIFER TABACK

Abstract. We generalize the notion of a graph automatic group introduced by Kharlampovich, Khoussainov and Miasnikov by replacing the regular languages in their definition with more powerful language classes. For a fixed language class $\mathcal{C}$, we call the resulting groups $\mathcal{C}$-graph automatic. We prove that the class of $\mathcal{C}$-graph automatic groups is closed under change of generating set, direct and free product for certain classes $\mathcal{C}$. We show that for quasi-realtime counter-graph automatic groups where normal forms have length that is linear in the geodesic length, there is an algorithm to compute normal forms (and therefore solve the word problem) in polynomial time. The class of quasi-realtime counter-graph automatic groups includes all Baumslag-Solitar groups, and the free group of countably infinite rank. Context-sensitive-graph automatic groups are shown to be a very large class, which encompasses, for example, groups with unsolvable conjugacy problem, the Grigorchuk group, and Thompson's groups F, T and V.

1. Introduction

In this article we consider extensions of the notion of a graph automatic group, introduced by Kharlampovich, Khoussainov and Miasnikov in [24], replacing the regular languages in their definition by more powerful language classes. Primarily we focus on the classes of context-free, counter, indexed and context-sensitive languages. We find that replacing regular languages with (quasi-realtime) counter languages preserves many of the desirable properties that graph automatic groups enjoy, including a polynomial time algorithm to compute normal forms. We prove that a finitely generated group is deterministic context-sensitive-graph automatic (with *quasigeodesic normal form* as defined below) precisely when its word problem is deterministic context-sensitive. It follows that the class of such groups is very large, and encompasses, for example, groups with unsolvable conjugacy problem, the Grigorchuk group, and Thompson's group V and all of its subgroups, which include Thompson's groups F and T. We present several examples of counter-graph automatic groups, including the non-solvable Baumslag-Solitar groups, which we show to be 3-counter-graph automatic. In [13] the authors and Sharif Younes prove that Thompson's group F is counter-graph automatic.

Several authors have considered generalized versions of automatic groups using different automata in place of finite state machines: Bridson and Gilman introduced a geometric version of *asynchronously* automatic groups using indexed languages

Date: June 4, 2014.

2010 *Mathematics Subject Classification.* 20F65; 68Q45.

Key words and phrases. automatic group; Cayley graph automatic group; counter language; context-sensitive language; word problem; polynomial time algorithm; Baumslag-Solitar group.

The first author is supported by Australian Research Council grant FT110100178, and the second author is partially supported by National Science Foundation grant DMS-1105407.

[3]; Baumslag, Shapiro and Short defined a class based on parallel computations by pushdown automata [1]; and Cho considered a version with counter languages in his PhD thesis [7]. Recent work of Brittenham and Hermiller [4] introduces the class of *autostackable groups* which also generalize the notion of automaticity.

The article is organized as follows. In Section 2 we define the key notions of counter languages and $\mathcal{C}$-graph automatic groups used in the paper. In Section 3 we give a polynomial time algorithm which computes normal forms in counter-graph automatic groups, and in Section 4 we examine the consequences of permitting context-sensitive languages in the definition of $\mathcal{C}$-graph automatic groups. In Section 5 we consider closure properties of $\mathcal{C}$-graph automatic groups, and in Section 6 we give examples of groups with counter-graph automatic structures.

Many of the ideas in this paper come from the paper by Olga Kharlampovich, Bakhadyr Khoussainov and Alexei Miasnikov [24], and we are grateful for their help with this project. We also thank Bob Gilman, Pascal Weil and especially Sharif Younes for helpful conversations about this paper. Lastly we thank the anonymous referee for helpful feedback and suggestions.

2. Background and Definitions

2.1. **Languages and automata.** For standard definitions of finite state, pushdown, nested stack, and linear bounded automata (accepting regular, context-free, indexed and context-sensitive languages respectively) see, for example, [21]. We begin by defining the particular types of counter automata we will use.

2.1.1. *Counter automata.* There are many variants of counter automata and languages in the literature, see for example [2, 8, 9, 11, 15, 17, 18, 22, 36]. In this article we define a counter automaton as follows.

Definition 2.1 (counter automaton)**.** A counter automaton can be defined with a variety of attribues:

(1) A *blind deterministic k-counter automaton* is a deterministic finite state automaton augmented with a finite number of integer counters: these are all initialized to zero, and can be incremented and decremented during operation, but not read; the automaton accepts a word exactly if it reaches an accepting state with the counters all returned to zero. [1]

(2) A *non-blind determistic k-counter automaton* is a deterministic finite state automaton augmented with a finite number of integer counters: these are all initialized to zero, and can be incremented, decremented, compared to zero and set to zero during operation; the automaton accepts a word exactly if it reaches an accepting state with the counters all returned to zero.

(3) A (blind or non-blind) k-counter automaton is *non-deterministic* if from each state there can be multiple transitions labeled by the same input letter, and transitions that read no input letter, labeled by ϵ. [2] Following Book and Ginsburg [2] we require these automata to run in *quasi-realtime*, meaning there is a bound on the number of consecutive ϵ transitions allowed.

[1] These are called $\mathbb{Z}^k$*-automata* in [11, 36, 23].

[2] These are called *multi-stack-counter automata* in [2].

Define $\mathscr{S}_k$ to be the class of languages accepted by a non-blind non-deterministic k-counter automata running in quasi-realtime, and $\mathscr{C}_k$ to be the class of languages accepted by a blind non-deterministic k-counter automata running in quasi-realtime.

It is well known ([35], see also [21] Theorem 7.9) that a non-blind non-deterministic k-counter automata with $k \geq 2$ and no time restriction can simulate a Turing machine, and so the class of languages accepted by such automata coincides with the class of recursively enumerable languages. Book and Ginsburg [2] prove that imposing the quasi-realtime requirement, the languages $\mathscr{C}_k$ and $\mathscr{S}_k$ form a strict hierarchy:

Theorem 2.2 (Book and Ginsburg [2])**.** *The language classes $\mathscr{C}_i$ and $\mathscr{S}_i$ satisfy the following inclusions.*

$$\mathscr{C}_1 \subsetneq \mathscr{S}_1 \subsetneq \mathscr{C}_2 \subsetneq \mathscr{S}_2 \subsetneq \mathscr{C}_3 \subsetneq \cdots \tag{1}$$

In this article all counter automata are assumed to run in quasi-realtime.

Lemma 2.3. *If $L \in \mathscr{S}_k$ then there is a constant F so that on reading a word of length n the absolute value of any counter is at most Fn.*

Proof. Let M be the non-deterministic k-counter automaton accepting L, and suppose the maximum amount any counter is changed by any transition is m. On input $u = u_1 \dots u_n$ consider all paths in M labeled $e_0 u_1 e_1 \dots e_{n-1} u_n e_n$ where e_i is a string of ϵ transitions, which by assumption has length at most some bound D. Then each subpath e_i can change the value of a counter by at most Dm, and so the entire path can change a counter by at most $Dm(n+1) + nm < 3Dmn$, so set $F = 3Dm$. □

Corollary 2.4. *The classes $\mathscr{C}_k, \mathscr{S}_k$ are strictly contained in the class of non-deterministic context-sensitive languages.*

Proof. A k-counter automaton can be simulated by a Turing machine, with each counter value stored on the tape. On input of length n, the amount of tape required to store the values of all counters is kFn by Lemma 2.3. The containment is strict by Theorem 2.2. □

In drawing k-counter automata (see examples in Section 6) we label transitions by the input letter to be read, with subscript a k-tuple from the following alphabet:

- $+, -$ to increase/decrease a counter by 1
- $+m, -m$ to increase/decrease a counter by $m \in \mathbb{N}$
- $=, \neq$ to compare a counter to zero
- $\downarrow$ to set a counter to zero.

For example, in a non-blind 4-counter automaton the label $1_{+,\neq\downarrow,,-3}$ means if the second counter is not 0, read input letter 1, add 1 to the first counter, set the second counter to 0, make no change to the third counter, and subtract 3 from the last counter; if the second counter was 0 then the transition is not followed.

2.2. **Closure properties of formal language classes.** We briefly outline some closure properties of the formal language classes we consider below.

Definition 2.5 (homomorphism of languages)**.** Let Λ, Σ be finite alphabets. For each $\lambda \in \Lambda$ let $r_\lambda \in \Sigma^*$ be a finite word, and let $L \subseteq \Lambda^*$. Then $\phi : L \to \Sigma^*$ defined by $\phi(\lambda_1 \dots \lambda_k) = r_{\lambda_1} \dots r_{\lambda_k}$ for $\lambda_i \in L$ is a *homomorphism of formal languages.* If r_{λ_i} is not the empty word for any λ_i then ϕ is an *ϵ-free homomorphism.*

A class $\mathcal{C}$ of formal languages is closed under (ϵ-free) homomorphism if $L \in \mathcal{C}$ is a language in the finite alphabet Λ and $\phi : \Lambda^* \to \Sigma^*$ is any homomorphism, then $\phi(L) \in \mathcal{C}$. The class $\mathcal{C}$ is closed under inverse homomorphism if for any $L \subset \Sigma^*$, where Σ is any finite alphabet, and any homomorphism $\phi : \Lambda^* \to \Sigma^*$, if $L \in \mathcal{C}$ then $\phi^{-1}(L) \in \mathcal{C}$.

Closure of a formal language class $\mathcal{C}$ under finite intersection varies widely with $\mathcal{C}$. The class of regular languages, for example, is closed under finite intersection, but the class of context-free languages is not, although the intersection of a context-free language and a regular language is again context-free. In her thesis, Brough introduces the following class of languages.

Definition 2.6 (poly-context-free; [5])**.** A language $L \subseteq \Sigma^*$ is *k-context-free* if it is the intersection of at most k context-free languages, and *poly-context-free* if it is the intersection of some finite number of context-free languages.

By design, the class of poly-context-free languages is closed under taking finite intersection, and intersection with regular languages.

The following lemma describes the closure of the class of counter languages under intersection.

Lemma 2.7. *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.*

Proof. Let M and N be counter automata with k and l counters respectively. Define a $(k+l)$-counter automaton with states $S \times T$ where S are the states of M and T are the states of N, as follows. Put a transition from (s,t) to (s',t') labeled by $\lambda_{\mathbf{x}}$ if

- there is a transition from s to s' in M labeled $\lambda_{(x_1,\dots,x_k)}$,
- there is a transition from t to t' in N labeled $\lambda_{(y_1,\dots,y_l)}$, and
- $\mathbf{x} = (x_1, \dots, x_k, y_1, \dots, y_l)$

where x_i, y_j are counter instructions.

If $l = 0$ then N is simply a finite state automaton and we recover the first statement. Note that the resulting automaton is blind and/or deterministic if and only if both M and N are. □

A linear bounded automaton is a Turing machine with memory linearly bounded by the size of the input, that is, there is a constant E so that on input a word of length n, the number of squares on the tape that can be used is En. See, for example, [21]. In this article a language is *(deterministic) context-sensitive* if it is the set of strings accepted by a (deterministic) linear bounded automaton. With this definition a context-sensitive language can contain the empty string. See [21] (pp. 225–226) and [38] for a discussion of this.

Lemma 2.8. *The classes of regular, counter, and poly-context-free languages are closed under homomorphism, inverse homomorphism, intersection with regular languages, and finite intersection.*

The class of context-sensitive languages is closed under ϵ-free homomorphism, inverse homomorphism, intersection with regular languages, and finite intersection.

Proof. See Chapter 11 of [21] for the cases of regular and context-sensitive languages, [5] for poly-context-free languages, and [16] for counter languages. □

2.3. **$\mathcal{C}$-graph automatic groups.** 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 2.9 (quasigeodesic normal form)**.** A *normal form for* (G, X, Λ) 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 for each $u \in L$, $|u|_\Lambda \leq D(||u||_X + 1)$ 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 $\overline{u}$.

Next we define the *convolution* of strings, which will be needed throughout the paper.

Definition 2.10 (convolution; Definition 2.3 of [24])**.** 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. Put $\Lambda_\diamond = \Lambda \cup \{\diamond\}$. 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 $i \leq |w_j|_\Lambda$ 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 [24] is recovered.

We begin with the definition of an automatic group, as introduced in [14].

Definition 2.11 (automatic group; [14])**.** Let (G, X) be a group and symmetric finite generating set. We say that (G, X) is *automatic* if there is a regular normal form $L \subseteq X^*$ 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 regular.

We remark that the usual definition of an automatic group requires a regular language L to be in surjection with G, rather that in bijection. Theorem 2.5.1 of [14] tells us that if a group has an automatic structure then there is an alternate automatic structure with a unique normal form word for each group element. Hence

there is no loss of generality in requiring a normal form to be in bijection with the group.

Kharlampovich, Khoussainov and Miasnikov extended this definition in [24] by allowing the language of normal forms to be defined over a finite alphabet other than a generating set for the group.

Definition 2.12 (graph automatic group; [24])**.** Let (G, X) be a group and symmetric generating set, and Λ a finite set of symbols. We say that (G, X, Λ) is *graph automatic* if there is a regular normal form $L \subseteq \Lambda^*$ 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 regular.

Note that unlike [24] we do not insist that the generating set X be finite; again our definition of a normal form requires a bijection between the group elements and the language of normal forms.

A useful first example to consider is the Heisenberg group (Example 6.6 of [24]), which is not automatic as it has a cubic Dehn function, but is graph automatic. To prove the latter statement, matrices are represented as the convolution of three binary integers.

The class of graph automatic groups includes the following groups which are known not to be automatic: the solvable Baumslag-Solitar groups, class 2 nilpotent groups, and non-finitely presented groups [24]. It also includes groups with unsolvable conjugacy problem [32]. It is not known if groups of intermediate growth belong to this class. Miasnikov and Savchuk [31] have shown that certain *graphs* of intermediate growth are graph automatic; see [24] for the definition of automatic structures on objects other than groups.

In this article we further extend the notion of a graph automatic group by replacing regular languages with other formal language classes.

Definition 2.13 ($\mathcal{C}$-graph automatic group)**.** Let $\mathcal{B}$ and $\mathcal{C}$ be formal language classes, (G, X) a group and symmetric generating set, and Λ a finite set of symbols.

(1) We say that (G, X, Λ) is $(\mathcal{B}, \mathcal{C})$-*graph automatic* if there is a normal form $L \subset \Lambda^*$ in the language class $\mathcal{B}$, 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 $\mathcal{C}$.
(2) If $\mathcal{B} = \mathcal{C}$ then we say that (G, X, Λ) is $\mathcal{C}$-*graph automatic*.
(3) If $\mathcal{B} = \mathcal{C}$ and $\Lambda = X$ then we say that (G, X) is $\mathcal{C}$-*automatic*.

For each $x \in X$ let M_x denote the automaton which accepts the language L_x.

In general we will restrict our attention to $\mathcal{C}$-graph automatic groups, where $\mathcal{C}$ is one of the following language classes: context-free; indexed; context-sensitive; poly-context-free; and (quasi-realtime) counter. As checking membership in L_x includes verifying that each of u, v in $\otimes(u, v)$ lie in L, the complexity of the class $\mathcal{C}$ is in general greater than or equal to that of $\mathcal{B}$. Precisely:

Lemma 2.14. *If $\mathcal{C}$ is closed under homomorphism, then a $(\mathcal{B}, \mathcal{C})$-graph automatic group is $\mathcal{C}$-graph automatic.*

Proof. Define a homomorphism from $\otimes(L, L)$ to L by a map that sends $\binom{\lambda_1}{\lambda_2}$ to λ_1 and $\binom{\diamond}{\lambda_1}$ to ϵ for all $\lambda_1 \in \Lambda$ and $\lambda_2 \in \Lambda_\diamond$. Then the language L is the image of L_x under this homomorphism restricted to L_x, so is in $\mathcal{C}$. □

Corollary 2.15. *If $\mathcal{B}$ and $\mathcal{C}$ are each one of the classes of regular, poly-context-free, quasi-realtime counter, or context-sensitive languages, then a $(\mathcal{B}, \mathcal{C})$-graph automatic group is $\mathcal{C}$-graph automatic.*

Proof. Since each class is contained within the class of context-sensitive languages, if $\mathcal{C}$ is context-sensitive then the result follows. Otherwise $\mathcal{C}$ is closed under homomorphism and the lemma applies. □

Definition 2.13 extends naturally to the context of biautomatic groups.

Definition 2.16 ($\mathcal{C}$-graph biautomatic group)**.** Let $\mathcal{C}$ be a formal language class, (G, X) a group and symmetric finite generating set, and Λ a finite set of symbols. We say that (G, X, Λ) is *$\mathcal{C}$-graph biautomatic* if there is a normal form $L \subset \Lambda^*$ in the language class $\mathcal{C}$, such that for each $x \in X$ the languages $\{\otimes(u, v) \mid u, v \in L, \overline{v} =_G \overline{u}x\}$ and $\{\otimes(u, v) \mid u, v \in L, \overline{v} =_G x\overline{u}\}$ are in the class $\mathcal{C}$. If $\Lambda = X$ we say that (G, X) is *$\mathcal{C}$-biautomatic*.

Miasnikov and Šunić [32] show that the classes of graph automatic and graph biautomatic groups are distinct. In Section 4 we show that when $\mathcal{C}$ denotes the class of determinisitic-context-sensitive languages, the classes of $\mathcal{C}$-graph automatic and $\mathcal{C}$-biautomatic groups coincide. In addition, there are deterministic context-sensitive-biautomatic groups with unsolvable conjugacy problem, in contrast to the cases of biautomatic and graph biautomatic groups.

In the proof of ([24], Lemma 8.2) is the following observation that graph automatic groups naturally possess a quasigeodesic normal form. For completeness we include a proof of this observation.

Lemma 2.17. *If (G, X, Λ) is graph automatic with respect to the regular normal form L, then L is a quasigeodesic normal form.*

Proof. Let C be an integer that is at least the length of the normal form for the identity, and at least the number of states in any of the finite state automata M_x, where $x \in X$.

Let $w = w_1 \dots w_n$ be a geodesic where $w_i \in X$, and let u_i be the normal form word for the prefix $w_1 \dots w_i$ of w, for $i = 0, \dots, n$, with u_0 representing the identity. By assumption u_0 has length at most C.

Assume for induction that the length of u_{i-1} is at most Ci.

The automaton M_{w_i} accepts the string labeled $\otimes(u_{i-1}, u_i)$. If u_i has length more than $C(i+1)$ then we have

$$\otimes(u_{i-1}, u_i) = \begin{pmatrix} y_1 \\ v_1 \end{pmatrix} \begin{pmatrix} y_2 \\ v_2 \end{pmatrix} \dots \begin{pmatrix} y_m \\ v_m \end{pmatrix} \begin{pmatrix} \diamond \\ v_{m+1} \end{pmatrix} \dots \begin{pmatrix} \diamond \\ v_n \end{pmatrix}$$

where $m \leq Ci$ and $n > C(i+1)$, so $n - m > C$ which is more than the number of states in M_{w_i}. If we apply the pumping lemma for regular languages to the suffix of $\otimes(u_{i-1}, u_i)$ beginning with $\begin{pmatrix} \diamond \\ v_{m+1} \end{pmatrix}$, we see that M_x accepts infinitely many normal form expressions for u_i, contradicting the uniqueness of the normal form. □

Note that when we generalize to $\mathcal{C}$-graph automatic groups, the lemma is no longer true — in Section 6 we give an example of a quasi-realtime 3-counter-graph automatic structure for the Baumslag-Solitar groups $BS(m,n)$ with non-quasigeodesic normal form.

Note that when proving a triple (G,X,Λ) is $\mathcal{C}$-graph automatic, the following observation shows that it suffices to check that just one of L_x or $L_{x^{-1}}$ lies in the class $\mathcal{C}$ for each $x \in X$.

Lemma 2.18. *If $\mathcal{C}$ is closed under ϵ-free homomorphism, then $L_x \in \mathcal{C}$ if and only if $L_{x^{-1}} \in \mathcal{C}$.*

Proof. The homomorphism that replaces each $\binom{\lambda_1}{\lambda_2}$ by $\binom{\lambda_2}{\lambda_1}$ for all $\lambda_i \in \Lambda_\diamond$ in L_x yields the language $L_{x^{-1}}$. □

2.4. **Remarks on the definition of graph automatic groups.** In [24] the authors implicitly assume that the normal form for the identity element is always the empty string — see, for example, the proof of Theorem 10.8 in [24]. In generalizing their definition and results, we realized this was a subtle issue. The definition of an automatic structure for a group (G,X) asserts the existence of a bijection (or surjection) from $L \subseteq X^*$ to G, together with a finite collection of regular languages which have finite descriptions either in terms of regular expressions, finite state automata, regular grammars or otherwise. In this definition there is no explicit information about the bijection from L to G, in particular the normal form word for the identity is not fixed by this. In Theorem 2.3.10 in [14], an algorithm is given that computes the normal form of any word in an automatic group, necessarily written in terms of the group generators, which runs in quadratic time. At the end of the proof of Theorem 2.3.10, it is explained how this algorithm can be used (in constant time) to find the normal form word for the identity, thus making this algorithm constructive. Hence in the case of automatic groups, the definition alone is enough to construct the bijection from L to G.

In the case of a graph automatic or $\mathcal{C}$-graph automatic group (G,X,Λ), many analogous results are not constructive unless one knows at least one pair $q \in L \subseteq \Lambda^*$ and $p \in G$ with $\overline{q} =_G p$. Hence this assumption is often included in the statement of the theorems in this paper.

We have modified the original definition of a graph automatic group by removing the requirement that G be finitely generated. In the case of $\mathcal{C}$-graph automatic groups, this allows us to capture groups such as F_∞ (see Section 6). Since Λ is finite, G must be countable. We were not able to find an example of a countably infinitely generated graph automatic group, so our evidence justifying this modification is perhaps less convincing. We add the hypothesis that G is finitely generated in several statements below on counter and context-sensitive-graph automatic groups.

Finally we remark that we know of no examples of $\mathcal{C}$-graph automatic groups which we can prove not to be graph automatic. This paper (and the examples we present in Section 6 and in [13]) grew out of an attempt to decide whether or not examples such as non-solvable Baumslag-Solitar groups and R. Thompson's group F are graph automatic.

3. Counter-graph automatic implies polynomial time algorithm to compute normal forms

In this section we extend the results of Epstein *et al.* ([14] Theorem 2.3.10) and Kharlampovich *et al.* ([24] Theorem 8.1) to show that for any finitely generated $\mathscr{S}_k$-graph automatic group there is an algorithm to compute normal forms for group elements that runs in polynomial time. Recall that $\mathscr{S}_k$ denotes the class of languages accepted by a non-deterministic, quasi-realtime, non-blind k-counter automaton; this class includes languages accepted by blind and/or deterministic k-counter languages.

Theorem 3.1. *Let (G, X) be a group with finite symmetric generating set, and Λ a finite set of symbols so that (G, X, Λ) is $\mathscr{S}_k$-graph automatic with quasigeodesic normal form L. Moreover, assume we are given $p \in X^*$ and $q \in L$ with $p =_G \overline{q}$. Then there is an algorithm that, on input a word $w = x_1 \dots x_n \in X^*$, computes $u \in L$ with $\overline{u} =_G w$, which runs in time $O(n^{2k+2})$.*

Proof. We will give an algorithm that on input $w = x_1 x_2 \cdots x_r \in X^*$ computes $u \in L$ where $\overline{u} =_G pw$, which runs in time $O(r^{2k+2})$. Running this algorithm on input p^{-1} gives a word $\mu \in L$ so that $\overline{\mu} =_G e$. The final algorithm is obtained with $q = \mu$ and $p = e$. Since p^{-1} has a fixed length, applying the algorithm to compute μ takes constant time.

For each $x \in X$ let M_x be the non-deterministic k-counter automaton accepting the language $\{\otimes(u, v) \mid u, v \in L, \overline{v} =_G \overline{u}x\}$ in quasi-real time. We begin with an enumeration of constants which appear in this argument.

(1) Let C be the quasigeodesic normal form constant for L.
(2) Let D be the maximum number of states in any M_x.
(3) Let E be the maximum over all M_x of the in-degree or out-degree of any vertex.
(4) Let F be the maximum over all M_x of the constant in Lemma 2.3; so on input of length n, the maximum absolute value of any counter in any machine M_x is Fn.
(5) Let $K-1$ be the maximum number of consecutive ϵ edges that can be read in any M_x.
(6) Let $P = |p|_X$ be the length of the word $p \in X^*$.

Note that we require finitely many generators to guarantee the existence of the constants D, E and F.

For each $i \in [1, n]$, let $u_i \in L$ be the string such that $\overline{u_i} =_G px_1 \dots x_i$, and set $u_0 = q$, so $\overline{u_0} =_G p$. Assume for induction that we have computed and stored u_i in time $O(i^{2k})$. Since $u_0 = q$ is constant length, the claim is true for $i = 0$. We find u_{i+1} in time $O((i+1)^{2k+1})$ as follows.

Write $u_i = \kappa_1 \dots \kappa_s \in L$ with $\kappa_j \in \Lambda$, and note that since $\overline{u_i} =_G px_1 \dots x_i$ we have $s \leq C(P + i + 1)$. Let $M = M_{x_{i+1}}$ be the non-deterministic k-counter automaton accepting $\otimes(u_i, u_{i+1})$.

Define a *configuration* of M to be a pair $(\tau, \mathbf{c})$ where τ is a state of M and $\mathbf{c} \in \mathbb{Z}^k$ represents the value of each counter. If τ_0 is the start state for M, then $(\tau_0, \mathbf{0})$ is the *start configuration* where $\mathbf{0} = (0, \dots, 0)$. Let $(\tau, \mathbf{c})_\diamond$ denote a configuration of M which is obtained by reading an input string of the form

$$\begin{pmatrix} \kappa_1 \\ \sigma_1 \end{pmatrix} \dots \begin{pmatrix} \kappa_l \\ \sigma_l \end{pmatrix} \begin{pmatrix} \kappa_{l+1} \\ \diamond \end{pmatrix} \dots \begin{pmatrix} \kappa_s \\ \diamond \end{pmatrix}$$

where $\sigma_t \in \Lambda$ and $l < s$, that is, the length of the string of symbols in the top coordinate is strictly longer than then length of the string of symbols in the bottom coordinate.

If $\mathbf{y}$ is a k-array of counter instructions and $\mathbf{c}$ is a k-tuple of counters, the notation $\mathbf{y}(\mathbf{c})$ means the k-tuple of counter values after $\mathbf{y}$ is applied to $\mathbf{c}$. If ω is a finite path in M let $[\omega]_{\mathbf{y}}$ denote the path with all the counter instructions collected together as $\mathbf{y}$.

We now build a directed graph $\mathcal{G}$ with vertices and edges defined recursively as follows. Vertices will be grouped together in sets S_j, and edges in sets T_j. The set S_j will consist of all configurations that can be obtained from the start configuration by following a path in M which contains exactly j edges not labeled by ϵ. For $j \leq s$, S_j is the set of configurations of M that can be obtained by reading

$$\begin{pmatrix} \kappa_1 \\ \sigma_1 \end{pmatrix} \dots \begin{pmatrix} \kappa_j \\ \sigma_j \end{pmatrix}$$

where $\sigma_t \in \Lambda_\diamond$.

The set S_0 consists of the configuration $(\tau_0, \mathbf{0})$, together with all configurations that can be reached by reading a path labeled ϵ^k from the start state in M. Recall that the number of consecutive ϵ transitions is bounded, so the set S_0 can be constructed by searching a bounded number of paths. Precisely, we must check at most

$$\sum_{k=1}^{K-1} E^k = O(E^K)$$

paths.

Given S_j with $j < s$, we construct S_{j+1} together with the set $T_{j+1} \subseteq S_j \times S_{j+1} \times \Lambda_\diamond$ of directed edges as follows.

(1) Initially set $S_{j+1} = T_{j+1} = \emptyset$.
(2) For each $(\tau, \mathbf{c}) \in S_j$ and each path from τ to τ' in M labeled $\left[\begin{pmatrix} \kappa_{j+1} \\ \sigma \end{pmatrix} \epsilon^r \right]_{\mathbf{y}}$ with $\sigma \in \Lambda$ and $\mathbf{y}$ a k-array of counter instructions, add $(\tau', \mathbf{y}(\mathbf{c}))$ to S_{j+1}, and add $((\tau, \mathbf{c}), (\tau', \mathbf{y}(\mathbf{c})), \sigma)$ to T_{j+1}.
(3) For each $(\tau, \mathbf{c}) \in S_j$ and each path from τ to τ' in M labeled $\left[\begin{pmatrix} \kappa_{j+1} \\ \diamond \end{pmatrix} \epsilon^r \right]_{\mathbf{y}}$, add $(\tau', \mathbf{y}(\mathbf{c}))_\diamond$ to S_{j+1}, and add $((\tau, \mathbf{c}), (\tau', \mathbf{y}(\mathbf{c}))_\diamond, \diamond)$ to T_{j+1}.
(4) For each $(\tau, \mathbf{c})_\diamond \in S_j$ and each path from τ to τ' in M labeled $\left[\begin{pmatrix} \kappa_{j+1} \\ \diamond \end{pmatrix} \epsilon^r \right]_{\mathbf{y}}$, add $(\tau', \mathbf{y}(\mathbf{c}))_\diamond$ to S_{j+1}, and add $((\tau, \mathbf{c})_\diamond, (\tau', \mathbf{y}(\mathbf{c}))_\diamond, \diamond)$ to T_{j+1}.

Since the number of consecutive ϵ transitions in M is at most $K - 1$, that is, $0 \leq r \leq K - 1$, the counter instructions $\mathbf{y}$ above are bounded.

Any configuration appearing in S_j and T_j is one that can be reached by reading $\otimes(\kappa_1 \dots \kappa_j, v)$ for some $v \in \Lambda_\diamond^*$. It follows that the set $S_s = S_{|u_i|_\Lambda}$ contains all possible configurations of M that can be reached by reading any string $\otimes(u_i, v)$ where $v \in \Lambda_\diamond^*$. If S_s does not contain a configuration $(\tau_a, \mathbf{0})$ or $(\tau_a, \mathbf{0})_\diamond$ where τ_a is an accept state of M, continue to construct sets S_{j+1} and T_{j+1} with $j \geq s$ as follows.

(1) Remove all elements of S_s of the form $(\tau, \mathbf{c})_\diamond$. A path to such a configuration cannot be extended to an accept configuration.

(2) Set $j = s$.
(3) While S_j does not contain a configuration $(\tau_a, \mathbf{0})$ where τ_a is an accept state of M:
   (a) For each $(\tau, \mathbf{c}) \in S_j$ and each path from τ to τ' in M labeled $\left[\epsilon^r \begin{pmatrix} \diamond \\ \sigma \end{pmatrix}\right]_{\mathbf{y}}$ with $\sigma \in \Lambda$ and $\mathbf{y}$ a k-array of counter instructions, add $(\tau', \mathbf{y}(\mathbf{c}))$ to S_{j+1}, and add $((\tau, \mathbf{c}), (\tau', \mathbf{y}(\mathbf{c})), \sigma)$ to T_{j+1}.
   (b) Increment j by 1.

Since L is a quasigeodesic normal form for G and $\overline{u_{i+1}} =_G px_1 \dots x_{i+1}$, the length of u_{i+1} is bounded by $C(P + i + 2)$. It follows that S_j will contain an accept configuration for some $j \leq C(P + i + 2)$, at which point the loop stops.

The time to construct and store the sets S_{j+1} and T_{j+1} is computed as follows. For each configuration in S_j we check at most E^K paths of length at most K in M, where $K - 1$ is the maximum number of consecutive ϵ edges that can be read, and E is the maximum out-degree. So to compute and store S_{j+1} and T_{j+1} takes time $O\left(|S_j|E^K\right)$.

Let $m \in \mathbb{N}$ be the minimal value so that $s \leq m \leq C(P + i + 2)$ and S_m contains an accept configuration $(\tau_a, \mathbf{0})$ or $(\tau_a, \mathbf{0})_\diamond$ (in which case $m = s$). As $\mathcal{G}$ is a directed graph, there is a directed labeled path $e_1 \dots e_m$ where $e_j \in T_j$ from $(\tau_0, \mathbf{0})$ to $(\tau_a, \mathbf{0})$ or $(\tau_a, \mathbf{0})_\diamond$, which can be found by backtracking through $\mathcal{G}$, scanning edges in T_j for $m \geq j \geq 0$. The time required to run this backtracking process is at most $O\left(\bigcup_{j=1}^m |T_j|\right)$.

The time required to construct and store the sets S_{j+1} and T_{j+1} for $0 \leq j < m$ is $O\left(\sum_{j=0}^{m-1} |S_j|E^K\right)$. It follows that the total time complexity for the algorithm is

$$O\left(\sum_{j=1}^{m} |T_j| + E^K \sum_{j=0}^{m-1} |S_j|\right) = O\left(\sum_{j=1}^{m} \left(|T_j| + E^K |S_{j-1}|\right)\right) = O\left(\sum_{j=1}^{m} |T_j|\right)$$

since $|S_{j-1}| \leq |T_j|$.

To complete the proof we compute $\sum_{j=1}^m |T_j|$. If $(\tau, \mathbf{c}) \in S_j$ then τ can be one of D states in M, and each counter has absolute value at most Fj (so has value c with $-Fj \leq c \leq F_j$), so the number of possible configurations is $D(2Fj + 1)^k$. We also have configurations of the form $(\tau, \mathbf{c})_\diamond$, so $|S_j| \leq 2D(2Fj + 1)^k$.

As $T_j \subseteq S_{j-1} \times S_j \times \Lambda_\diamond$ we have

$$|T_j| \leq 2D(2F(j-1) + 1)^k \cdot 2D(2Fj + 1)^k \cdot (|\Lambda| + 1) \leq Xj^{2k}$$

where $X = X(D, F, k, |\Lambda|)$ is a fixed constant. We also have $m \leq C(P+i+2) = Yi$ where $Y = Y(C, P)$ is a fixed constant. Thus

$$\sum_{j=1}^{m} |T_j| \leq \sum_{j=1}^{m} Xj^{2k} = X \sum_{j=1}^{m} j^{2k} \leq X \sum_{j=1}^{m} m^{2k} = Xm^{2k+1} \leq X(Yi)^{2k+1} = Zi^{2k+1}$$

where $Z = XY^{2k+1} = Z(C, D, F, P, k, |\Lambda|)$ is a fixed constant.

To compute u_n which is the normal form for pw, we repeat this procedure for $i \in [1, n]$ so the total time complexity is $\sum_{i=1}^n Zi^{2k+1} \leq Zn^{2k+2}$. □

4. Context-sensitive-graph automatic groups

Recall that a linear bounded automaton is a Turing machine together with a constant D so that on input a word w of length n, the number of squares on the tape used for any operation involving w is Dn. The *read-head* of the Turing machine is a pointer to a particular square of the tape. A move of the Turing machine can involve reading the letter at the position of the read-head, writing to this position, or moving the read-head one square to the left or right. A letter written on the tape can be *marked* by overwriting it with an annotated version of the letter — for example the letter a can be replaced by $\hat{a}$.

A language is context-sensitive if it is accepted by a linear bounded automaton, and *deterministic context-sensitive*, or $\mathcal{DCS}$, if the linear bounded automaton is deterministic. Note that here we allow content-sensitive languages to include the empty string — in some usages context-sensitive languages are defined without this, in particular when defined via a grammar in which the right-hand sides of production rules are required to have positive length. Note also that is it not known if the class of deterministic and non-determistic linear space languages are distinct.

Shapiro [38] and Lakin and Thomas [25, 26] consider groups with context-sensitive word problem. Shapiro showed that any finitely generated subgroup of an automatic group has $\mathcal{DCS}$ word problem, and Lakin and Thomas proved several closure properties.

In this section we consider the class of $\mathcal{DCS}$-graph automatic groups. We show that if a finitely generated group G has a $\mathcal{DCS}$-graph automatic structure with quasigeodesic normal form, then its word problem in solvable in deterministic linear space. We also prove that if a finitely generated group G has deterministic linear space word problem then it has a $\mathcal{DCS}$-biautomatic structure (with no symbol alphabet needed) with geodesic normal form language.

We start with a simple subroutine to enumerate strings over an ordered alphabet in Shortlex order. Recall that for a finite totally ordered finite set Λ, the *Shortlex order* on Λ^* is defined as follows: for $u, v \in \Lambda^*$, $u <_{\mathrm{SL}} v$ if

- $|u|_\Lambda < |v|_\Lambda$, or
- $|u|_\Lambda = |v|_\Lambda$, $u = p\lambda_i u', v = p\lambda_j v'$ with $\lambda_i < \lambda_j$ and $p, u', v' \in \Lambda^*$.

Algorithm 4.1 (Shortlex subroutine)**.** *Let Σ be a finite totally ordered set, $\#, \$$ two symbols not in Σ, and $\sigma_0, \sigma_r \in \Sigma$ such that $\$ < \sigma_0 \leq \sigma \leq \sigma_r$ for all $\sigma \in \Sigma$. Let $v = v_1 \dots v_k \in \Sigma^*$, and assume $\#v\$$ is written on the tape of a linear bounded automaton. Then the next string in Shortlex order can be found and overwritten on the tape using space $k + 2$ as follows.*

(1) Move the read-head to the last letter of v (before the \$ symbol), and set a boolean variable `done` to be false.
(2) While not `done`:
   (a) If the letter at the read-head position is σ_r, move the read-head one position to the left.
   (b) If the read-head points to #, the contents of the tape must be $\#\sigma_r^k\$$. In this case overwrite the tape by $\#\sigma_0^{k+1}$ (consuming the \$ symbol) and set `done` to be true.
   (c) Else the letter at the read-head position is $v_i \in \Sigma$ with $v_i < \sigma_r$. The contents of tape are $\#v_1 \dots v_{i-1} v_i \sigma_r^{k-i}\$$. Let $v_i^* \in \Sigma$ be such that

$v_i < v_i^*$ and $\sigma \leq v_i^*$ implies $\sigma \leq v_i$. In this case overwrite the tape by $\#v_1 \dots v_{i-1} v_i^* \sigma_0^{k-i}\$$ and set `done` to be true.

Note that the subroutine writes either $\#v'\$$ or $\#v''$ to the tape, where $|v'|_\Sigma = |v|_\Sigma$ and $|v''|_\Sigma = |v|_\Sigma + 1$. If one ignores the $\#, \$$ symbols then the algorithm on input v returns the next string in Shortlex order in Σ^*.

Proposition 4.2. *Let G be a group and finite symmetric generating set X. If (G, X) has $\mathcal{DCS}$ word problem then (G, X) is $\mathcal{DCS}$-biautomatic, with normal form the set of Shortlex geodesics over X.*

Proof. Assume the word problem algorithm for (G, X) runs as follows. On input $u \in X^*$ written on a one-ended tape, the algorithm returns *yes* if u is trivial and *no* otherwise, and returns a blank tape, using at most $D|u|$ space.

Fix an order on the generators with x_0 the smallest and x_r the largest, and let L be the set of Shortlex geodesic words for G with respect to this order. By Definition 2.16 we must show that L and the languages $\{\otimes(u, v) \mid u, v \in L, v = xu\}$ and $\{\otimes(u, v) \mid u, v \in L, v = ux\}$ for each $x \in X$ are $\mathcal{DCS}$. Let \$ be a symbol not in X, and set $\$ < x_0$.

Define a deterministic linear bounded automaton to accept L as follows. Assume that $\%, \#, \$$ are distinct symbols not in X. On input $u \in X^*$ of length n:

(1) Write $\%u\#\,(\$)^{n+1}$ on the tape and set `done` to be false.
(2) While not `done`:
    (a) Set v to be the word on the tape between # and the first \$ symbol.
    (b) Scan the tape to check if u and v are identical as strings. If they are, accept u and set `done` to be true.
    (c) Else write uv^{-1} to the left of the % symbol. Call the word problem algorithm on the one-ended tape to the left of the % symbol. If it returns *yes*, reject u and set `done` to be true [3].
    (d) Else run the Shortlex subroutine (Algorithm 4.1) to overwrite v by the next word in Shortlex order.

The algorithm runs as follows. To start we have $v = \epsilon$. If $u = v$ then the empty string is accepted since it is the Shortlex geodesic for the identity. If not we overwrite v with the next word in Shortlex order, and compare to u. We iterate the loop until either the contents of the tape are $\%u\#u\$$, or we find a word v that equals u in the group and is shorter in Shortlex order. At any time the tape contains at most $4n + 3$ letters, and running the word problem algorithm takes space at most $D|uv^{-1}| \leq D(2n)$, so all together the space required is $2Dn + 4n + 3$.

The following algorithm accepts

$$\{\otimes(u, v) \mid u, v \in L, v = xu\} \text{ (respectively } \{\otimes(u, v) \mid u, v \in L, v = xu\})$$

for $x \in X$: On input $\otimes(u, v)$,

(1) run the preceding algorithm on u to check if $u \in L$;
(2) run the preceding algorithm on v to check if $v \in L$;
(3) call the linear space word problem algorithm on uxv^{-1} (respectively xuv^{-1}).

□

Note that there are subgroups of $F_2 \times F_2$ with unsolvable conjugacy problem [33, 34], which by [38] have $\mathcal{DCS}$ word problem and therefore are $\mathcal{DCS}$-biautomatic.

[3] The contents of the tape after this step are $\%u\#v\,(\$)^i$ with $|v|_\Lambda + i = n + 1$

It follows that $\mathcal{DCS}$-biautomatic does not imply solvable conjugacy problem, in contrast to the graph biautomatic case ([24] Theorem 8.5).

Next we show that $\mathcal{DCS}$-graph automatic groups with quasigeodesic normal form have deterministic linear space word problem.

Proposition 4.3. *Let (G, X) be a group with finite symmetric generating set, and Λ a finite set of symbols so that (G, X, Λ) is a $\mathcal{DCS}$-graph automatic group with quasigeodesic normal form $L \subset \Lambda^*$. Additionally, suppose we are given $p \in X^*$ and $q \in L$ with $p =_G \overline{q}$. Then there is an algorithm that on input a word $w = x_1 \dots x_n \in X^*$, computes $u \in L$ with $\overline{u} =_G w$ and runs in space $O(n)$.*

Proof. We first give the algorithm that on input $w \in X^*$ computes $u \in L$ where $\overline{u} =_G pw$. Running this algorithm on input p^{-1} gives a word $\mu \in L$ for the identity. The final algorithm is obtained with $q = \mu$ and $p = e$. Since p^{-1} has a fixed length the step to compute μ takes constant space.

For each $x \in X$ let L_x be the $\mathcal{DCS}$ language $\{\otimes(u, v) \mid u, v \in L, \overline{v} =_G \overline{u}x\}$. We begin with an enumeration of constants which appear in this argument.

(1) Let B be a constant so that for any $x \in X$ the space used by the linear bounded automaton accepting L_x on input of length n is Bn.
(2) Let C be the quasigeodesic normal form constant for L.
(3) Let $P = |p|_X$ be the length of the word $p \in X^*$.

Note that we require finitely many generators to guarantee the existence of the constant B.

Let $w = x_1 \dots x_n \in X^*$ be the input word, and define $w_0 = p$, $w_i = px_1 \dots x_i$ for $i \in [1, n]$, and let $u_i \in L$ be such that $\overline{u_i} =_G w_i$. Note that $u_0 = q$, and for each i the length of u_i is at most $C(P + i + 1)$. Let $\#$ be a symbol not in Λ. Define a total order on the (finite) set Λ.

We compute the normal form word representing w as follows. Write $w\#u_0\#$ on the tape, marking the first letter of w. This uses space at most $n + 2 + C(P + 1)$. Assume for induction that we have written $w\#u_i\#$ on the tape for $i < n$, and marked the letter at position $i + 1$ in w, using space at most $D(n) = n + 2 + (B + 2)C(P + n + 1)$.

Find u_{i+1} as follows.

(1) Set `done` to be false.
(2) Let v denote the string of symbols to the right of the last $\#$ on the tape. To begin we have $v = \epsilon$.
(3) While not `done`:
    (a) Run the deterministic linear space algorithm that accepts $L_{x_{i+1}}$ on $\otimes(u_i, v)$. Note that the length of the input to this subroutine is at most $C(P + n + 1)$ since L is quasigeodesic and u_i, u_{i+1} represent words of geodesic length at most n. It follows that the space needed for this step is at most $BC(P + n + 1)$.
        (i) If the subroutine returns true, then we have found $v = u_{i+1}$. Set `done` to be true.
        (ii) Else run the Shortlex subroutine (Algorithm 4.1) to overwrite v by the next word in Shortlex order.

If $i + 1 < n$, rewrite the tape as $w\#u_{i+1}\#$ and mark the letter at position $i + 2$ of w. If $i + 1 = n$, the word u_n is the required normal form word for w.

Since we know there is some string u_{i+1} of length at most $C(P+i+2)$ then this algorithm must terminate. Moreover, the amount of space used on the tape to store $w\#u_i\#v$ is bounded by $n+2+2C(P+n+1)$, as the length of $w\#\#$ is $n+2$, and u_i, v have length at most $C(P+n+1)$. The space used to run the subroutine on $\otimes(u_i, v)$ is bounded by $BC(P+n+1)$, so in total the amount of space required is at most $D(n)$. □

Combining these two propositions we obtain the following.

Theorem 4.4. *The following classes of groups coincide:*

(1) *finitely generated $\mathcal{DCS}$-graph automatic groups with quasigeodesic normal form;*
(2) *finitely generated $\mathcal{DCS}$-biautomatic groups with Shortlex geodesic normal form;*
(3) *finitely generated groups with $\mathcal{DCS}$ word problem.*

The class of such groups is very large — groups with $\mathcal{DCS}$ word problem include all linear groups [28], *logspace embeddable* groups studied by the first author, Elston and Ostheimer [10], and all finitely generated subgroups of automatic groups [38]. It also includes the co-indexed and co-context free groups as described in [19, 20, 27]. These groups have co-word problems accepted by non-deterministic pushdown or nested-stack automata, which can be simulated by deterministic linear bounded automata since as described in these articles, the non-determinism is confined to an initial guessing step. It follows that the word problem for these groups is accepted by the same deterministic linear bounded automata. These classes include the Higman-Thompson groups, Thompson's group V, Houghton's groups, and the Grigorchuk group.

Note that the number of configurations of a linear bounded automaton is exponential in the length of the input string, so the time complexity of computing the normal form of a word in a $\mathcal{DCS}$-biautomatic group is at most exponential. The next example shows that a polynomial time algorithm to compute normal forms of $\mathcal{DCS}$-biautomatic structures seems unlikely to exist.

Let $G = \mathbb{Z}_2 \wr \mathbb{Z}^2$. By ([10] Theorem 14) the word problem for G is in deterministic logspace and therefore deterministic linear space, so it follows from Proposition 4.2 that (G, X) is $\mathcal{DCS}$-biautomatic with Shortlex geodesic normal form, where X is the standard generating set. The *bounded geodesic length problem* (see [12, 30]) for a group G with finite generating set X is the following:

Problem 4.5 (Bounded geodesic length problem)**.** On input an integer k and a string $w \in X^*$, decide if the geodesic length of w is less than k.

Suppose one could prove that a $\mathcal{DCS}$-graph automatic structure with quasigeodesic normal form for a finitely generated group implied a polynomial time algorithm that on input a string of generators computes the normal form. Then by Proposition 4.2 we may assume the group has a $\mathcal{DCS}$-biautomatic structure with normal form the set of all Shortlex geodesics. Parry [37] proved that the bounded geodesic length problem for $\mathbb{Z}_2 \wr \mathbb{Z}^2$ is NP-complete. So if such an algorithm could be constructed to run in polynomial time, we would have P=NP.

A second example is the class of free metabelian groups — Svetla Vassileva has shown they have normal forms (and hence word problem) computable in logspace

[39], and Miasnikov *et al.* [30] proved the bounded geodesic length problem for these groups is NP-complete.

5. Closure properties

In this section we show that under certain conditions $\mathcal{C}$-graph automaticity is preserved under change of group generating set, direct and free product. Recall that by Lemma 2.8 the following classes are closed under intersection with regular languages, finite intersection, ϵ-free homomorphism, and inverse homomorphism: regular languages, $\mathscr{C}_k$, $\mathscr{S}_k$, poly-context free languages, context-sensitive languages. Moreover these classes all contain the class of regular languages.

Lemma 5.1 (Change of generators)**.** *Let G be a group with two symmetric generating sets X and Y, Λ a finite alphabet, and let $\mathcal{C}$ be a class of formal languages that is closed under finite intersection and inverse homomorphism, and contains the class of regular languages. If (G, X, Λ) is $\mathcal{C}$-graph automatic, then (G, Y, Λ) is $\mathcal{C}$-graph automatic.*

Proof. Since we can use the same language $L \subseteq \Lambda^*$ for (G, Y, Λ) as for (G, X, Λ), it suffices to show that each language L_y lies in the class $\mathcal{C}$.

Let $Y_1 \subseteq Y$ be the set of generators that do not equal the identity in G. For each $y \in Y_1$, choose $u_y \in X^+$ such that $u_y =_G y$. Fix $y \in Y_1$ and suppose $u_y = x_1 \dots x_k$ with $x_i \in X$. Consider convolutions of $k+1$ strings $v_i \in L$

$$\otimes(v_0, v_1, v_2, \dots, v_k)$$

so that $\overline{v_i} =_G \overline{v_{i-1}} x_i$ for $1 \leq i \leq k$. Let P be the language of all such convolutions.

For each x_i appearing in u_y define a language A_i of convolutions of $k+1$ strings over Λ where rows i and $i+1$ correspond to the language L_{x_i}, and all other rows can be any words in Λ^*. Then A_i is the inverse image of L_{x_i} under the homomorphism which sends $\otimes(v_0, \dots, v_k)$ to $\otimes(v_{i-1}, v_i)$.

Then $\bigcap_{i=1}^{k} A_i$ is in $\mathcal{C}$ since the class is closed under finite intersection.

Finally consider the ϵ-free homomorphism from $\bigcap_{i=1}^{k} A_i$ to $\otimes(L, L)$ defined by

$$\otimes(v_0, v_1, v_2, \dots, v_k) \mapsto \otimes(v_0, v_k).$$

Since y is assumed to be non-trivial, the image this map is guaranteed to be ϵ-free. The language L_y is the image of $\bigcap_{i=1}^{k} A_i$ under this homomorphism, so is in $\mathcal{C}$.

To complete the proof, we must consider $y \in Y \setminus Y_1$, that is, y equals the identity element. In this case $L_y = \{\otimes(u, u) \mid u \in L\}$ which is regular, and so by assumption in $\mathcal{C}$. □

Note that the lemma holds when one or both of X and Y are countably infinite, since for each $y \in Y$ the word u_y is a finite string of letters in X.

Lemma 5.2 (Direct product)**.** *Let G and H be groups with symmetric generating sets X and Y respectively, Λ and Γ finite alphabets, and let $\mathcal{C}$ be a class of formal languages that is closed under intersection with regular languages, finite intersection and inverse homomorphism. If (G, X, Λ) and (H, Y, Γ) are $\mathcal{C}$-graph automatic, then the group $G \times H$ is $\mathcal{C}$-graph automatic.*

Proof. Assume Λ and Γ are disjoint. Let $L_G \subset \Lambda^*$ and $L_H \subset \Gamma^*$ denote the languages of normal forms for each group, and $Z = \{(x, 1_H), (1_G, y) \mid x \in X, y \in Y\}$ a generating set for $G \times H$. Define a normal form $L = \otimes(L_G, L_H)$ for $G \times H$.

The language $\otimes(L_G, \Gamma^*)$ is the inverse image of the homomorphism from $\otimes(L_G, \Gamma^*)$ to L_G which sends $\otimes(u, v)$ to u, and similarly for $\otimes(\Lambda^*, L_H)$. Then L is the intersection of these languages and hence lies in the class $\mathcal{C}$.

For each $x \in X$ let L_x be the multiplier language for the $\mathcal{C}$-graph automatic structure on G. Define

$$L_1 = \{\otimes(\otimes(u, w), \otimes(v, w)) \mid u, v \in \Lambda^*, w \in \Gamma^*\},$$

$$L_2 = \{\otimes(\otimes(u, w), \otimes(v, z)) \mid u, v \in \Lambda^*, w \in L_H, z \in \Gamma^*\},$$

and

$$L_3 = \{\otimes(\otimes(u, w), \otimes(v, z)) \mid u, v \in L_G, \overline{v} =_G \overline{u}x, w, z \in \Gamma^*\}.$$

Then L_1 is regular, L_2 is the inverse image of the homomorphism

$$\phi : \otimes(\otimes(\Lambda^*, \Gamma^*), \otimes(\Lambda^*, \Gamma^*)) \to L_H$$

given by $\otimes(\otimes(a, b), \otimes(c, d)) = b$, and L_3 is the inverse image of the homomorphism

$$\phi : \otimes(\otimes(\Lambda^*, \Gamma^*), \otimes(\Lambda^*, \Gamma^*)) \to L_x$$

given by $\otimes(\otimes(a, b), \otimes(c, d)) = \otimes(a, c)$, so L_2 and L_3 lie in $\mathcal{C}$.

It follows that

$$L_{(x,1_H)} = \{\otimes(\otimes(u, w), \otimes(v, w)) \mid u, v \in L_G, \overline{v} =_G \overline{u}x, w \in L_H\}$$

is in $\mathcal{C}$ since it is the intersection $L_1 \cap L_2 \cap L_3$.

A similar argument applies to multiplier languages $L_{(1_G,y)}$. □

For certain language classes $\mathcal{C}$ we prove that $\mathcal{C}$-graph automatic groups are closed under free product. The following argument is specific to the class of non-blind counter languages, and can be modified to apply to poly-context-free, and context-sensitive languages.

Lemma 5.3 (Free product)**.** *Let G and H be groups with symmetric generating sets X and Y respectively, and Λ and Γ finite alphabets. If (G, X, Λ) is $\mathscr{S}_k$-graph automatic and (H, Y, Γ) is $\mathscr{S}_l$-graph automatic, then $G * H$ is $\mathscr{S}_{\max\{k,l\}}$-graph automatic.*

Proof. Assume that Λ and Γ are distinct sets of symbols, and let $L_G \subset \Lambda^*, L_H \subset \Gamma^*$ be the normal form languages for G, H respectively, and $\lambda_0 \in L_G$ and $\gamma_0 \in L_H$ the normal form words for the identity in each language.

Define $L_1 = L_G \setminus \{\lambda_0\}$; this is a k-counter language as it is the intersection of L_G with the regular language $\Lambda^* \setminus \{\lambda_0\}$, and similarly $L_2 = L_H \setminus \{\gamma_0\}$ is an l-counter language. If L_1 contains the empty string, choose $u \in \Lambda^* \setminus L_G$ and replace L_1 by its image under the homomorphism from L_1 to Λ^* which sends ϵ to u and is the identity on all other strings. Then L_1 remains a k-counter language. Similarly if L_2 contains the empty string, it can be replaced. Define

$$L = \left\{ \begin{array}{l} \epsilon, \\ \#u_1\#v_1\#\ldots\#u_s\#v_s, \\ \#u_1\#v_1\#\ldots\#v_{s-1}\#u_s, \\ \#v_1\#u_2\#\ldots\#u_s\#v_s, \\ \#v_1\#u_2\#\ldots\#v_{s-1}\#u_s \end{array} \middle| \; s > 0, u_i \in L_1, v_i \in L_2 \right\}$$

over the alphabet $\{\#\} \cup \Lambda \cup \Gamma$. There is an obvious bijection from L to the free product, namely the map that deletes all $\#$, sends u_i to $\overline{u_i}$ and v_i to $\overline{v_i}$.

Let M_1 be the k-counter automaton accepting L_1, with start state τ_1; analogously let M_2 be the l-counter automaton with start state τ_2 accepting L_2. Assume the sets of states of M_1 and M_2 are distinct. Define a nondeterministic, non-blind $\max\{k,l\}$-counter automaton M as follows. The states of M are the states of M_1 and M_2 together with three new states $\kappa_0, \kappa_1, \kappa_2$. The start state for M is κ_0, and accepting states are κ_1 and κ_2. The edges in M are as follows:

(1) Every edge in M_1 is again an edge in M, where the first k counters correspond to the k counters in M_1.
(2) Every edge in M_2 is again an edge in M, where the first l counters correspond to the l counters in M_2.
(3) For each accept state τ_a in M_1, put an edge from τ_a to κ_2 labeled $\epsilon_{=,\dots,=}$. Note that this transition is allowed only when all counters are zero.
(4) For each accept state τ'_a in M_2, put an edge from τ'_a to κ_1 labeled $\epsilon_{=,\dots,=}$. Again, this edge is followed only when all counters are zero.
(5) Put an edge labeled ϵ from κ_0 to κ_1, and an edge labeled ϵ from κ_0 to κ_2.
(6) Put an edge labeled # from κ_1 to τ_1, and an edge labeled # from κ_2 to τ_2.

See Figure 1. Then M is a non-blind non-deterministic $\max\{k,l\}$-counter automaton which accepts the language L.

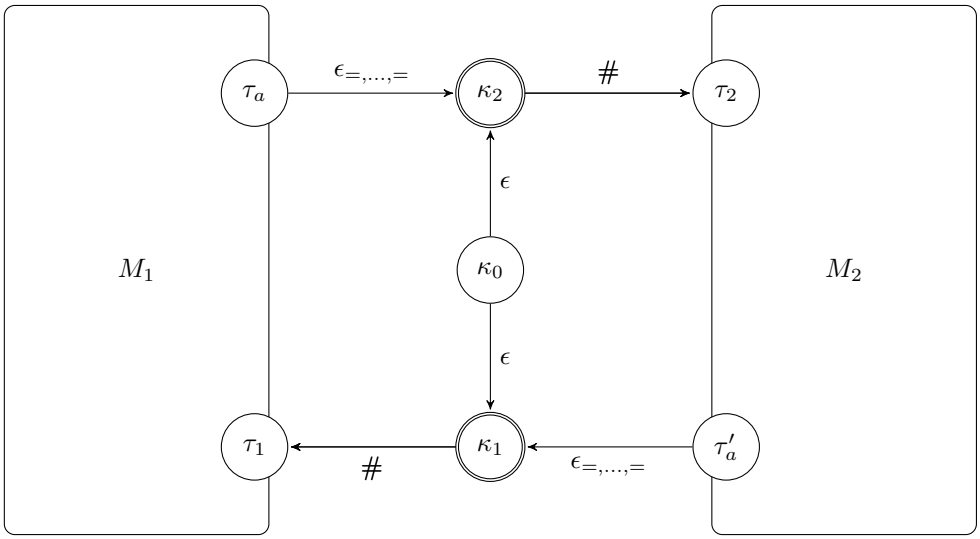

FIGURE 1. Construction of the automaton M in the proof of Lemma 5.3. Start state is κ_0 and accept states are κ_1 and κ_2.

Let $x \in X$, and let $L_{G,x}$ be the multiplier language for the counter-graph automatic structure on G. Analogously, for $y \in Y$ let $L_{H,y}$ be the multiplier language for the counter-graph automatic structure on H.

We will describe the multiplier language in the case of multiplication by $x \in X$ and leave the analogous case of $y \in Y$ to the reader.

The multiplier language $L_x = \{\otimes(p,q) \mid p,q \in L, \overline{q} =_{G*H} \overline{p}x\} \subseteq \otimes(L,L)$ for $G * H$ is accepted by a modified version of M which we denote M_x, constructed as follows.

(1) Let M_x initially have the same states and transitions as M, with none labeled as accept states. Replace each edge label $\alpha \neq \epsilon$ by $\begin{pmatrix} \alpha \\ \alpha \end{pmatrix}$.
(2) Let $\lambda_1\lambda_2\cdots\lambda_s \in L_H$ be the normal form word for x. Add a new state χ_1 and a path from κ_1 to χ_1 labeled $\binom{\diamond}{\#}\binom{\diamond}{\lambda_1}\ldots\binom{\diamond}{\lambda_s}$. Declare χ_1 to be an accept state. This ensures that if p is empty, or p ends with a subword from H, that $\otimes(p,q)$ is accepted, where $\overline{q} =_G \overline{p}x$.
(3) From κ_1 add an edge to a copy of the machine $L_{G,x}$ labeled ϵ. Declare all previous accept states of this machine to be accept states of M_x. If p ends with a subword from G, say $p = \beta\gamma$ where γ is the maximal suffix from G, then β corresponds to a path through M with an epsilon edge leading to τ_1. At that point, $L_{G,x}$ checks that the two suffix strings differ by x in G.

□

6. Examples

6.1. **Infinitely generated groups.** The purpose of this example is to show that non-finitely generated groups are captured by the class of $\mathcal{C}$-graph automatic groups for appropriate $\mathcal{C}$.

Proposition 6.1. *The free group $F_\infty = \langle x_1, x_2, x_3, \cdots \mid - \rangle$ on the countable set of generators $Y = \{x_i \mid i \in \mathbb{Z}_+\}$ is deterministic non-blind 2-counter-graph automatic.*

Proof. The idea is to represent generators and their inverses as positive or negative unary integers. Let $X = Y \cup Y^{-1}$, $\Lambda = \{p, n, 1\}$, and define a homomorphism $\phi : X^* \to \Lambda^*$ by $\phi(x_i) = p1^i$ and $\phi(x_i^{-1}) = n1^i$. For example, $x_2^3x_5^{-1}$ is mapped to $p11p11p11n11111$. The set of freely reduced finite strings of generators is a normal form for F_∞, so define a normal form $L \subseteq \Lambda^*$ to be the image of this set under ϕ. Note that the identity corresponds to the empty string ϵ.

Let $L_1 \subseteq \Lambda^*$ be the set of strings of the form $r_1 1^{\eta_1} \ldots r_k 1^{\eta_k}$ where $r_i \in \{p, n\}$ and $\eta_i \in \mathbb{Z}^+$. Let L_2 be the set of strings in L_1 where $r_{2i-1} \neq r_{2i}$ implies $\eta_{2i-1} \neq \eta_{2i}$, and L_3 the strings in L_1 where $r_{2i} \neq r_{2i+1}$ implies $\eta_{2i} \neq \eta_{2i+1}$, for $i \geq 1$. That is, in L_2 substrings $r_{2i-1}1^{\eta_{2i-1}}r_{2i}1^{\eta_{2i}}$ represent a freely reduced pair, and in L_3 substrings $r_{2i}1^{\eta_{2i}}r_{2i+1}1^{\eta_{2i+1}}$ represent a freely reduced pair. For example, $n1p11n11p1$ is in L_2 but not L_3. The intersection $L_2 \cap L_3$ is then the normal form language L.

A deterministic non-blind 1-counter automaton accepting L_3 is shown in Figure 2. The automaton accepting L_2 is obtained from this by setting s_2 to be the start state. Recall that the notation $1_{\neq\downarrow}$ means if the counter is nonzero, read 1 and set the counter to 0.

Then $L = L_2 \cap L_3$ is deterministic non-blind 2-counter by Lemma 2.7.

The multiplier language L_{x_i} for the generator x_i is the set of strings in $\otimes(L, L)$ of the form

$$\binom{r_1}{r_1}\binom{1}{1}^{\eta_1}\binom{r_2}{r_2}\binom{1}{1}^{\eta_2}\cdots\binom{r_k}{r_k}\binom{1}{1}^{\eta_k}\binom{\diamond}{p}\binom{\diamond}{1}^{i}$$

if $r_k = p$ or $\eta_k \neq i$, and otherwise if $r_k = n$ and $\eta_k = i$

$$\binom{r_1}{r_1}\binom{1}{1}^{\eta_1}\binom{r_2}{r_2}\binom{1}{1}^{\eta_2}\cdots\binom{r_{k-1}}{r_{k-1}}\binom{1}{1}^{\eta_{k-1}}\binom{n}{\diamond}\binom{1}{\diamond}^{i}.$$

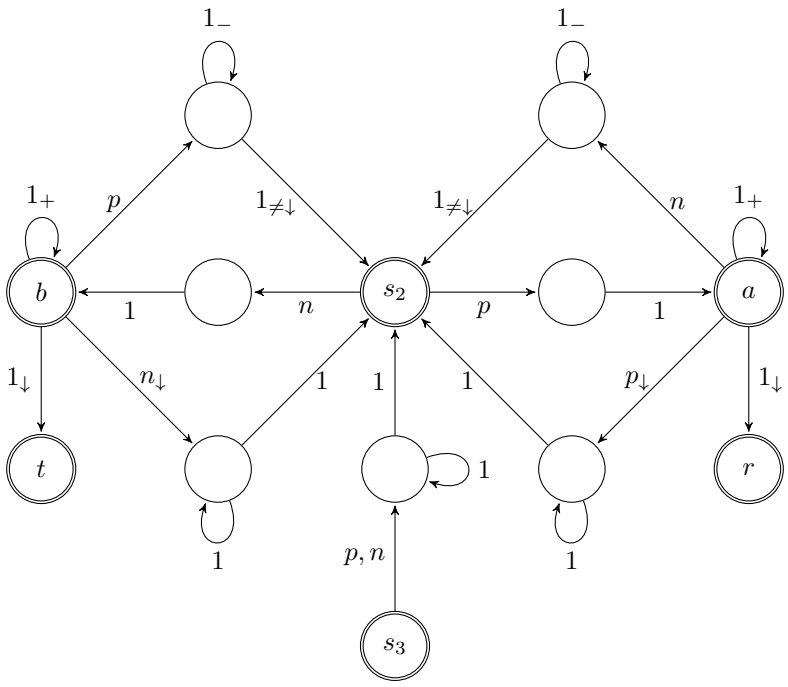

FIGURE 2. Deterministic non-blind 1-counter automaton accepting the language L_3 in the proof of Proposition 6.1. The start state is s_3. Accept states are s_2, s_3, a, b, r, t. The automaton for L_2 is identical with start state s_2.

Define $L^+_{x_i}$ to be the regular language is given by the regular expression

$$\left\{\binom{1}{1}, \binom{p}{p}, \binom{n}{n}\right\}^* \left\{\binom{\diamond}{p}\binom{\diamond}{1}^i\right\},$$

and $L^-_{x_i}$ the language given by the regular expression

$$\left\{\binom{1}{1}, \binom{p}{p}, \binom{n}{n}\right\}^* \left\{\binom{n}{\diamond}\binom{1}{\diamond}^i\right\}.$$

Next consider the language $\otimes(\Lambda^*, L)$. Modify the automaton in Figure 2 by replacing edges labeled $x_\#$ (where $x \in \{p, n, 1\}$ and $\#$ denotes some counter instructions) by four edges labeled $\binom{p}{x}_\#, \binom{n}{x}_\#, \binom{1}{x}_\#, \binom{\diamond}{x}_\#$. The intersection of the two languages of strings accepted by this automaton with start state either s_2 or s_3 is the language $\otimes(\Lambda^*, L)$, and is deterministic non-blind 2-counter.

A similar argument shows that $\otimes(L, \Lambda^*)$ is deterministic non-blind 2-counter. Then L_{x_i} is the union of $L^+_{x_i} \cap \otimes(\Lambda^*, L)$ and $L^-_{x_i} \cap \otimes(L, \Lambda^*)$, and so is deterministic non-blind 2-counter. □

6.2. **Baumslag-Solitar groups.** In [24] the solvable Baumslag-Solitar groups are shown to be graph automatic. Here we show that the non-solvable Baumslag-Solitar groups are blind deterministic 3-counter-graph automatic.

Proposition 6.2. *Let $2 \leq m < n$. Then $BS(m,n) = \langle a, t \mid ta^m t^{-1} = a^n \rangle$ is blind deterministic 3-counter-graph automatic.*

Proof. Any word in $\{a^{\pm 1}, b^{\pm 1}\}^*$ can be transformed into a normal form for the corresponding group element by "pushing" each a and a^{-1} in the word as far to the right as possible and freely reducing using the identities

$$\begin{array}{lll} a^{\pm 1}a^{\mp 1} = 1, & a^{\pm n}t = ta^{\pm m}, & a^{-i}t = a^{n-i}ta^{-m}, \\ t^{\pm 1}t^{\mp 1} = 1, & a^{\pm m}t^{-1} = t^{-1}a^{\pm n}, & a^{-j}t^{-1} = a^{m-j}t^{-1}a^{-n}. \end{array}$$

where $0 < i < n$ and $0 < j < m$, so that only positive powers of a appear before a $t^{\pm 1}$ letter. The resulting word can be written as Pa^N, where P is a freely reduced word in the alphabet $\Pi = \{t, at, \dots a^{n-1}t, t^{-1}, at^{-1}, \dots a^{m-1}t^{-1}\}$ (see for example [29] p.181). Let $\Gamma \subseteq \Pi^*$ be the set of freely reduced words in Π^*.

It is clear that the language of the words of the form Pa^N with $P \in \Gamma, N \in \mathbb{N}$ is regular, and in bijection with the group. The idea for the counter-graph automatic structure is to represent the integer N in two different ways, so that multiplication by the generator t can be easily recognised.

For $N \in \mathbb{Z}$, if N is positive write $N = pm + r = qn + s$ with $0 \leq r < m$ and $0 \leq s < n$; if N is negative write $N = -(pm + r) = -(qn + s)$; and otherwise write $N = 0$. Define L to be the language

$$L = \left\{ \begin{array}{l|l} P\#1^r\#1^p\#1^s\#1^q, & P \in \Gamma, \\ P\#(-1)^r\#(-1)^p\#(-1)^s\#(-1)^q, & r \in [0,m), s \in [0,n), \\ P\#\#\#\# & r + pm = s + qn, \\ & r + pm > 0 \end{array} \right\}.$$

Then L is in bijection with words of the form Pa^N for N positive, negative and zero, so is a normal form for $BS(m,n)$ over the alphabet $\Lambda = \Pi \cup \{1, -1, \#\}$.

For example, in $BS(4,7)$:

- the string $at\#111\#1\#\#1$ represents the word ata^7;
- the string $at\#11111\#1\#\#1$ is rejected since $r = 5$ is not less than $m = 4$;
- the string $at\#11\#11\#1\#1$ is rejected since $r + pm = 10$ whereas $s + qn = 8$.

Let L_1 be the language

$$L_1 = \left\{ \begin{array}{l|l} P\#1^r\#1^p\#1^s\#1^q, & P \in \{a, t^{\pm 1}\}^*, \\ P\#(-1)^r\#(-1)^p\#(-1)^s\#(-1)^q, & r, p, s, q \in \mathbb{N}, \\ P\#\#\#\# & r + pm = s + qn > 0 \end{array} \right\}.$$

Then L_1 is accepted by the blind deterministic 1-counter automaton shown in Figure 3.

Let L_2 be the regular language of strings

$$L_2 = \left\{ \begin{array}{l|l} P\#1^r\#1^p\#1^s\#1^q, & P \in \Gamma, \\ P\#(-1)^r\#(-1)^p\#(-1)^s\#(-1)^q, & r, s, p, q \in \mathbb{N}, \\ P\#\#\#\# & r < m, s < n \end{array} \right\}.$$

Then $L = L_1 \cap L_2$ is a blind-1-counter language.

Now we turn to the multiplier languages L_a and L_t.

First observe that the languages $\otimes(L, \Lambda^*)$ and $\otimes(\Lambda^*, L)$ are blind-1-counter, and so $\otimes(L, L) = \otimes(L, \Lambda^*) \cap \otimes(\Lambda^*, L)$ is a blind-2-counter language by Lemma 2.7.

We will describe L_a as the union of a set of languages intersected with $\otimes(L, L)$. Note that L_a is the set of strings $\otimes(u, v)$ where $\overline{u} = Pa^N, \overline{v} = Pa^{N+1}$. Recall that

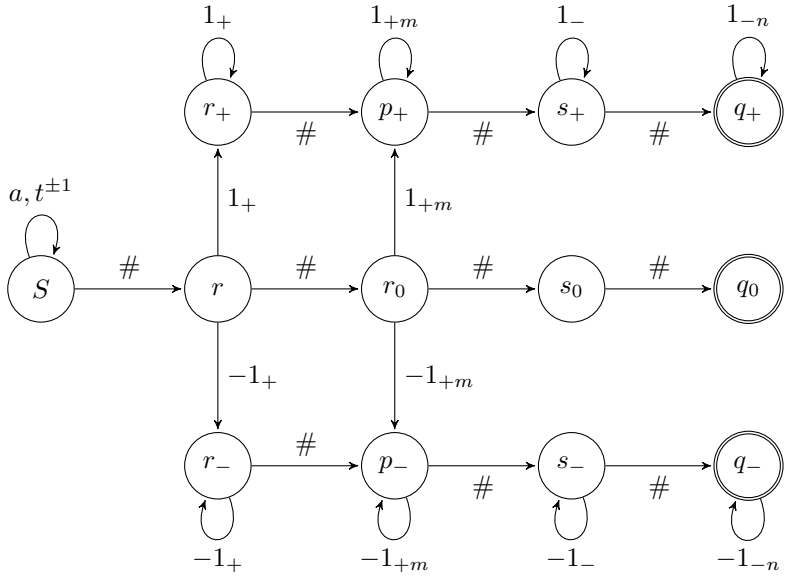

FIGURE 3. 1-counter automaton accepting the language L_1 for $BS(m, n)$. Accept states are q_+, q_0 and q_-. The counter checks the equation $r + pm = s + qn$ is satisfied.

the regular expression $\{1\}^*\#\{1\}^*$ denotes the set of strings in $\{1, \#\}^*$ with exactly one # letter. The languages are as follows, for $0 \leq r \leq m-2$:

- $\mathcal{L}_r = \left\{ \otimes \begin{pmatrix} P\#1^r\#1^p\#Q, \\ P\#1^{r+1}\#1^p\#R \end{pmatrix} \;\middle|\; \begin{array}{l} P \in \{a, t^{\pm 1}\}^*, \\ p \in \mathbb{N}, \\ Q, R \in \{1\}^*\#\{1\}^* \end{array} \right\}$;
- $\mathcal{L}_{m-1} = \left\{ \otimes \begin{pmatrix} P\#1^{m-1}\#1^p\#Q, \\ P\#\#1^{p+1}\#R \end{pmatrix} \;\middle|\; \begin{array}{l} P \in \{a, t^{\pm 1}\}^*, \\ p \in \mathbb{N}, \\ Q, R \in \{1\}^*\#\{1\}^* \end{array} \right\}$;
- $\mathcal{K}_{r+1} = \left\{ \otimes \begin{pmatrix} P\#(-1)^{r+1}\#(-1)^p\#Q, \\ P\#(-1)^r\#(-1)^p\#R \end{pmatrix} \;\middle|\; \begin{array}{l} P \in \{a, t^{\pm 1}\}^*, \\ p \in \mathbb{N}, \\ Q, R \in \{-1\}^*\#\{-1\}^* \end{array} \right\}$;
- $\mathcal{K}_0 = \left\{ \otimes \begin{pmatrix} P\#\#(-1)^{p+1}\#Q, \\ P\#(-1)^{m-1}\#R \end{pmatrix} \;\middle|\; \begin{array}{l} P \in \{a, t^{\pm 1}\}^*, \\ p \in \mathbb{N}, \\ Q, R \in \{-1\}^*\#\{-1\}^* \end{array} \right\}$.

These languages are designed simply to check the condition that $\overline{u} = Pa^N, \overline{v} = Pa^{N+1}$. Each language is regular, so its intersection with $\otimes(L, L)$ is a blind 2-counter language. It follows that L_a is blind 2-counter.

Now we come to the language L_t. We will again intersect with the blind 2-counter language $\otimes(L, L)$. We must accept strings $\otimes(u, v)$ for words $u, v \in L$ with $\overline{u} = Pa^N$ and $\overline{v} = Pa^N t$. We consider the following cases, which depend on whether or not P ends in t^{-1}, and whether or not n divides N.

Case 1 P ends in t or is empty:

For $N \geq 0$, write $N = qn + s$ with $0 \leq s < n$. Then $a^N t = a^s t a^{qm}$. This gives strings of the form

$$\otimes(P\#1^\alpha\#1^\beta\#1^s\#1^q, Pa^s t\#\#1^q\#1^\gamma\#1^\delta)$$

where $\alpha, \beta, \gamma, \delta$ are the appropriate integers. Note that there is no cancelation between P and the letters added, since P is either empty or ends in t.

For $N < 0$, write $N = -(qn + s)$ with $0 \leq s < n$. Then $a^N t = a^{-s} t a^{-qm}$. If $s = 0$ then this gives the set of strings

$$\otimes(P\#(-1)^\alpha\#(-1)^\beta\#\#(-1)^q, Pt\#\#(-1)^q\#(-1)^\gamma\#(-1)^\delta).$$

If $s > 0$ then $a^N t = a^{-s} t a^{-qm} = a^{n-s} t a^{-m-qm}$ which gives the set of strings

$$\otimes(P\#(-1)^\alpha\#(-1)^\beta\#(-1)^s\#(-1)^q, Pa^{n-s} t\#\#(-1)^{q+1}\#(-1)^\gamma\#(-1)^\delta).$$

Again there is no cancelation between P and the letters added, since P is either empty or ends in t.

These strings can be obtained by intersecting the following languages with $\otimes(L, L)$:

- $\mathcal{U}_s = \left\{ \otimes\begin{pmatrix} P\#Q\#1^s\#1^q, \\ Pa^s t\#\#1^q\#R \end{pmatrix} \;\middle|\; \begin{array}{l} P \in \{\epsilon, wt \;:\; w \in \{a, t^{\pm 1}\}^*\}, \\ q \in \mathbb{N}, \\ Q, R \in \{1\}^*\#\{1\}^* \end{array} \right\}$
  for $0 \leq s \leq n-1$,
- $\mathcal{V}_s = \left\{ \otimes\begin{pmatrix} P\#Q\#(-1)^s\#(-1)^q, \\ Pa^{n-s} t\#\#(-1)^{q+1}\#R \end{pmatrix} \;\middle|\; \begin{array}{l} P \in \{\epsilon, wt \;:\; w \in \{a, t^{\pm 1}\}^*\}, \\ q \in \mathbb{N}, \\ Q, R \in \{1\}^*\#\{1\}^* \end{array} \right\}$
  for $1 \leq s \leq n-1$.
- $\mathcal{V}_0 = \left\{ \otimes\begin{pmatrix} P\#Q\#\#(-1)^q, \\ Pt\#\#(-1)^q\#R \end{pmatrix} \;\middle|\; \begin{array}{l} P \in \{\epsilon, wt \;:\; w \in \{a, t^{\pm 1}\}^*\}, \\ q \in \mathbb{N}, \\ Q, R \in \{1\}^*\#\{1\}^* \end{array} \right\}$.

The languages $\mathcal{U}_s, V_s$ for $0 \leq s \leq n-1$ are blind 1-counter — the counter is used to check the entries $(\pm 1)^q$ are the same in each component of the convoluted string.

Case 2 P ends in t^{-1}, and n does not divide N.

In this case $N = qn + s$ with $0 < |s| < n$.

For $N \geq 0$ write $N = qn + s$ with $s > 0$. Then $Pa^N t = Pa^s t a^{qm}$ where $Pa^s t$ has no cancelation so is in normal form. This gives the set of strings

$$\otimes(P\#1^\alpha\#1^\beta\#1^s\#1^q, Pa^s t\#\#1^q\#1^\gamma\#1^\delta).$$

For $N < 0$, write $N = -(qn + s)$ with $s > 0$. Then

$$a^N t = a^{-s} t a^{-qm} = a^{n-s} t a^{-m-qm}$$

and so $Pa^N t = Pa^{n-s} t a^{-m-qm}$ and P does not cancel, so this gives the set of strings

$$\otimes(P\#(-1)^\alpha\#(-1)^\beta\#(-1)^s\#(-1)^q, Pa^{n-s} t\#\#(-1)^{q+1}\#(-1)^\gamma\#(-1)^\delta).$$

These strings can be obtained by intersecting the following languages with $\otimes(L, L)$:

- $\mathcal{W}_s = \left\{ \otimes\begin{pmatrix} P\#Q\#1^s\#1^q, \\ Pa^s t\#\#1^q\#R \end{pmatrix} \;\middle|\; \begin{array}{l} P \in \{\epsilon, wt^{-1} \;:\; w \in \{a, t^{\pm 1}\}^*\}, \\ q \in \mathbb{N}, \\ Q, R \in \{1\}^*\#\{1\}^* \end{array} \right\}$
  for $1 \leq s \leq n-1$,

- $\mathcal{X}_s = \left\{ \otimes \left(\begin{array}{l} P\#Q\#(-1)^s\#(-1)^q, \\ Pa^{n-s}t\#\#(-1)^{q+1}\#R \end{array} \right) \;\middle|\; \begin{array}{l} P \in \{\epsilon, wt^{-1} \;:\; w \in \{a, t^{\pm 1}\}^*\}, \\ q \in \mathbb{N}, \\ Q, R \in \{-1\}^*\#\{-1\}^* \end{array} \right\}$
  for $1 \leq s \leq n-1$.

Again the languages $\mathcal{W}_s, X_s$ for $1 \leq s \leq n-1$ are blind 1-counter — the counter is used to check the entries $(\pm 1)^q$ are the same in each component of the convoluted string.

Case 3 P ends in t^{-1}, and n divides N.

Put $P = Ta^c t^{-1}$, where $c \in [0, m)$ and T is empty or ends in $t^{\pm 1}$. Since we will intersect with $\otimes(L, L)$ we don't care whether $Ta^c t^{-1}$ is freely reduced or not.

For $N \geq 0$ write $N = qn$ so

$$Pa^N t = Pta^{qm} = Ta^c t^{-1} t a^{qm} = Ta^{c+qm}.$$

This gives the set of strings

$$\otimes(Ta^c t^{-1}\#1^\alpha\#1^\beta\#\#1^q, T\#1^c\#1^q\#1^\gamma\#1^\delta).$$

For $N < 0$ write $N = -(qn)$ and

$$Pa^N t = Pta^{-qm} = Ta^c t^{-1} t a^{-qm} = Ta^{c-qm} = Ta^{c-m}a^{-(q-1)m}$$

This gives the set of strings

$$\otimes(Ta^c t^{-1}\#(-1)^\alpha\#(-1)^\beta\#\#(-1)^q, T\#(-1)^{m-c}\#(-1)^{q-1}\#(-1)^\gamma\#(-1)^\delta).$$

These strings can be obtained by intersecting the following languages with $\otimes(L, L)$:

- $\mathcal{Y}_c = \left\{ \otimes \left(\begin{array}{l} Ta^c t^{-1}\#Q\#\#1^q, \\ T\#1^c\#1^q\#R \end{array} \right) \;\middle|\; \begin{array}{l} T \in \{a, t^{\pm 1}\}^*, \\ q \in \mathbb{N}, \\ Q, R \in \{1\}^*\#\{1\}^* \end{array} \right\}$
  for $0 \leq c \leq n-1$,
- $\mathcal{Z}_c = \left\{ \otimes \left(\begin{array}{l} Ta^c t^{\pm 1}\#Q\#\#(-1)^q, \\ T\#(-1)^c\#(-1)^q\#R \end{array} \right) \;\middle|\; \begin{array}{l} T \in \{a, t^{\pm 1}\}^*, \\ c \in [0, n), \\ q \in \mathbb{N}, \\ Q, R \in \{-1\}^*\#\{-1\}^* \end{array} \right\}$
  for $0 \leq c \leq n-1$,

Once again the languages $\mathcal{Y}_c, \mathcal{Z}_c$ for $0 \leq c \leq n-1$ are blind 1-counter — the counter is used to check the entries $(\pm 1)^q$ are the same in each component of the convoluted string.

It follows that the language L_t is the union of the languages $\mathcal{U}_i, \mathcal{V}_i, \mathcal{W}_i, \mathcal{X}_i, \mathcal{Y}_i, \mathcal{Z}_i$ each intersected with $\otimes(L, L)$ and is therefore blind deterministic 3-counter. □

We remark that the above normal form language is not quasigeodesic. In [6] Burillo and the first author find a metric estimate for $BS(m, n)$. It is shown that the geodesic length of the element equal to a^N is $O(\log N)$, while the normal form representative given above has length $O(N/m + N/n) = O(N)$.

References

[1] Gilbert Baumslag, Michael Shapiro, and Hamish Short. Parallel poly-pushdown groups. *J. Pure Appl. Algebra*, 140(3):209–227, 1999.

[2] Ronald Book and Seymour Ginsburg. Multi-stack-counter languages. *Math. Systems Theory*, 6:37–48, 1972.

[3] Martin R. Bridson and Robert H. Gilman. Formal language theory and the geometry of 3-manifolds. *Comment. Math. Helv.*, 71(4):525–555, 1996.

[4] Mark Brittenham and Susan Hermiller. Stackable groups, tame filling invariants, and algorithmic properties of groups, 2011. http://arxiv.org/abs/1109.6309.
[5] Tara Brough. Groups with poly-context-free word problem. *Groups Complex. Cryptol.*, 6(1):9–29, 2014.
[6] José Burillo and Murray Elder. Metric properties of Baumslag-Solitar groups, 2014. http://arxiv.org/abs/1402.3859.
[7] Hong Ray Cho. *An Introduction to Counter Groups*. Doctoral Thesis, Stevens Institute of Technology, 2006.
[8] Murray Elder. G-automata, counter languages and the Chomsky hierarchy. In *Groups St. Andrews 2005. Vol. 1*, volume 339 of *London Math. Soc. Lecture Note Ser.*, pages 313–318. Cambridge Univ. Press, Cambridge, 2007.
[9] Murray Elder. A linear-time algorithm to compute geodesics in solvable Baumslag-Solitar groups. *Illinois J. Math.*, 54(1):109–128, 2010.
[10] Murray Elder, Gillian Elston, and Gretchen Ostheimer. On groups that have normal forms computable in logspace. *J. Algebra*, 381:260–281, 2013.
[11] Murray Elder, Mark Kambites, and Gretchen Ostheimer. On groups and counter automata. *Internat. J. Algebra Comput.*, 18(8):1345–1364, 2008.
[12] Murray Elder and Andrew Rechnitzer. Some geodesic problems in groups. *Groups Complex. Cryptol.*, 2(2):223–229, 2010.
[13] Murray Elder, Jennifer Taback, and Sharif Younes. Language complexity of Thompson's group F, 2014. In preparation.
[14] David B. A. Epstein, James W. Cannon, Derek F. Holt, Silvio V. F. Levy, Michael S. Paterson, and William P. Thurston. *Word processing in groups*. Jones and Bartlett Publishers, Boston, MA, 1992.
[15] Patrick C. Fischer, A. R. Meyer, and Arnold L. Rosenberg. Counter machines and counter languages. *Math. Systems Theory*, 2:265–283, 1968.
[16] S. A. Greibach and Seymour Ginsburg. Multitape afa. *J. Assoc. Comput. Mach.*, 19:193–221, 1972.
[17] Sheila A. Greibach. Remarks on the complexity of nondeterministic counter languages. *Theoret. Comput. Sci.*, 1(4):269–288, 1975/76.
[18] Sheila A. Greibach. Remarks on blind and partially blind one-way multicounter machines. *Theoret. Comput. Sci.*, 7(3):311–324, 1978.
[19] Derek F. Holt, Sarah Rees, Claas E. Röver, and Richard M. Thomas. Groups with context-free co-word problem. *J. London Math. Soc. (2)*, 71(3):643–657, 2005.
[20] Derek F. Holt and Claas E. Röver. Groups with indexed co-word problem. *Internat. J. Algebra Comput.*, 16(5):985–1014, 2006.
[21] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation*. Addison-Wesley Publishing Co., Reading, Mass., 1979. Addison-Wesley Series in Computer Science.
[22] Mark Kambites. Word problems recognisable by deterministic blind monoid automata. *Theoret. Comput. Sci.*, 362(1-3):232–237, 2006.
[23] Mark Kambites. Formal languages and groups as memory. *Comm. Algebra*, 37(1):193–208, 2009.
[24] Olga Kharlampovich, Bakhadyr Khoussainov, and Alexei G. Miasnikov. From automatic structures to automatic groups, 2011. http://arxiv.org/abs/1107.3645.
[25] Stephen R. Lakin and Richard M. Thomas. Context-sensitive decision problems in groups. In *Developments in language theory*, volume 3340 of *Lecture Notes in Comput. Sci.*, pages 296–307. Springer, Berlin, 2004.
[26] Stephen R. Lakin and Richard M. Thomas. Space complexity and word problems of groups. *Groups Complex. Cryptol.*, 1(2):261–273, 2009.
[27] Jörg Lehnert and Pascal Schweitzer. The co-word problem for the Higman-Thompson group is context-free. *Bull. Lond. Math. Soc.*, 39(2):235–241, 2007.
[28] Richard J. Lipton and Yechezkel Zalcstein. Word problems solvable in logspace. *J. Assoc. Comput. Mach.*, 24(3):522–526, 1977.
[29] Roger C. Lyndon and Paul E. Schupp. *Combinatorial group theory*. Springer-Verlag, Berlin, 1977. Ergebnisse der Mathematik und ihrer Grenzgebiete, Band 89.

[30] Alexei G. Miasnikov, Vitalii Roman'kov, Alexander Ushakov, and Anatoly Vershik. The word and geodesic problems in free solvable groups. *Trans. Amer. Math. Soc.*, 362(9):4655–4682, 2010.
[31] Alexei G. Miasnikov and Dmytro Savchuk. An example of an automatic graph of intermediate growth, 2013. http://arxiv.org/abs/1312.3710.
[32] Alexei G. Miasnikov and Zoran Šunić. Cayley graph automatic groups are not necessarily Cayley graph biautomatic. In *Language and automata theory and applications*, volume 7183 of *Lecture Notes in Comput. Sci.*, pages 401–407. Springer, Heidelberg, 2012.
[33] K. A. Mihaĭlova. The occurrence problem for free products of groups. *Mat. Sb. (N.S.)*, 75 (117):199–210, 1968.
[34] Charles F. Miller, III. *On group-theoretic decision problems and their classification*. Princeton University Press, Princeton, N.J., 1971. Annals of Mathematics Studies, No. 68.
[35] Marvin L. Minsky. Recursive unsolvability of Post's problem of "tag" and other topics in theory of Turing machines. *Ann. of Math. (2)*, 74:437–455, 1961.
[36] Victor Mitrana and Ralf Stiebe. The accepting power of finite automata over groups. In *New trends in formal languages*, volume 1218 of *Lecture Notes in Comput. Sci.*, pages 39–48. Springer, Berlin, 1997.
[37] Walter Parry. Growth series of some wreath products. *Trans. Amer. Math. Soc.*, 331(2):751–759, 1992.
[38] Michael Shapiro. A note on context-sensitive languages and word problems. *Internat. J. Algebra Comput.*, 4(4):493–497, 1994.
[39] Svetla Vassileva. *Space and time complexity of algorithmic problems in groups*. PhD thesis, McGill University, 2013.

School of Mathematical and Physical Sciences, The University of Newcastle, Callaghan NSW 2308, Australia
E-mail address: murrayelder@gmail.com

Department of Mathematics, Bowdoin College, Brunswick, ME 04011, USA
E-mail address: jtaback@bowdoin.edu