Streaming in graph products

Markus Lohrey
Universität Siegen, Germany

Julio Xochitemol
Universität Siegen, Germany

Abstract

We investigate the streaming space complexity of word problems for groups. Using so-called distinguishers, we prove a transfer theorem for graph products of groups. Moreover, we use distinguishers to obtain a logspace streaming algorithm for the membership problem in a finitely generated subgroup of a free group.

2012 ACM Subject Classification CCS → Theory of computation → computational complexity and cryptography → problems, reductions and completeness

Keywords and phrases word problems for groups, streaming algorithms, graph products

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.63

Related Version *Full Version*: https://arxiv.org/abs/2202.04060

Funding *Julio Xochitemol*: Funded by DFG project LO 748/13-2.

1 Introduction

The word problem for a finitely generated (f.g. for short) group G is the following computational problem: Fix a finite set of generators Σ for G, which means that every element of G can be written as a finite product of elements from Σ. The input for the word problem is a finite word $a_1 a_2 \cdots a_n$ over the alphabet Σ and the question is whether this word evaluates to the group identity of G. The word problem was introduced by Dehn in 1911 [5]. It is arguably the most important computational problem in group theory and has been studied by group theorists as well as computer scientists. In general, the word problem is undecidable [4, 20], but for many classes of groups (e.g. linear groups, metabelian groups, hyperbolic groups) efficient algorithms exist; see e.g. [15] for an overview.

In [16, 17] we started to investigate streaming algorithms for the word problem of a group G. The input stream is a word $w \in \Sigma^*$ over the generators and the algorithm has to decide whether $w = 1$ holds in the group G. This can be viewed as a streaming algorithm for the formal language $\{w \in \Sigma^* : w = 1 \text{ in } G\}$. Streaming algorithms for formal languages (mainly subclasses of context-free languages) have been studied in [1, 2, 7, 18]. In [16, 17] we consider the space complexity of streaming algorithms. For deterministic streaming algorithms it turnes out that the space complexity of the word problem for a f.g. group G is tightly related to the growth of G: if $\gamma(n)$ is the growth function of G (which is the number of different group elements that can be represented by words of length at most n over the generating set), then the space complexity of the best deterministic streaming algorithm for the word problem of G is roughly $\log \gamma(n/2)$. This result basically reduces the study of deterministic streaming algorithms for word problems to the study of growth in groups, which is an important research area in geometric group theory with many deep results.

In [16, 17] we therefore mainly focus on randomized streaming algorithms for word problems. For this it turns out to be useful to consider so called distinguishers for groups. Roughly speaking, a distinguisher for a f.g. group G with finite generating set Σ is a randomized streaming algorithm $\mathcal{A}$ such that for all words $u, v \in \Sigma^*$ of length at most n we

49th International Symposium on Mathematical Foundations of Computer Science (MFCS 2024).
Editors: Rastislav Královič and Antonín Kučera; Article No. 63; pp. 63:1–63:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

have that: (i) if u and v evaluate to the same element of G then with high probability, u and v lead to the same memory state of $\mathcal{A}$, and (ii) if u and v evaluate to different elements of G then with high probability, u and v lead to different memory states of $\mathcal{A}$; see Section 4. It is easy to obtain from a distinguisher $\mathcal{R}$ for the group G a randomized streaming algorithm $\mathcal{S}$ for the word problem of G (with low error probability). Moreover, the space complexity of $\mathcal{S}$ is only twice the space complexity of $\mathcal{R}$; see Lemma 3.

We showed in [16, 17] that for many important f.g. groups G there exist logspace distinguishers with error probability $1/n^c$ for any constant $c > 1$, where n is the input length. This is in particular the case for f.g. linear groups (matrix groups). In general, the growth of a linear group can be $2^{\Theta(n)}$ (take for instance a free group of rank 2), and therefore its deterministic streaming space complexity can be $\Theta(n)$, which is the worst case (the streaming algorithms can always store the whole prefix of the input stream). We proved in [17] also the following transfer theorem for wreath products: If G is a f.g. group having a distinguisher with space complexity $s(n)$ and error probability $\epsilon(n)$ and A is a f.g. abelian group then there is a distinguisher for the wreath product $A \wr G$ having space complexity $\mathcal{O}(s(n) + \log n)$ and error probability roughly $n^2\epsilon(n)$. Interestingly, if H is any non-abelian group then any randomized streaming algorithm with error at most $1/2 - \epsilon$ for the word problem of $H \wr G$ must have the worst-case space complexity $\Theta(n)$; see [16, Theorem 21].

The first main result of the paper states a similar transfer theorem for *graph products*. This is an important construction in group theory that generalizes the direct product as well as the free product. It can be seen as a partially commutative version of the free product, where some of the factors G_i in a free product $G_1 * G_2 * \cdots * G_k$ are allowed to commute. Which of G_i commute is specified by a graph on the index set $\{1, \ldots, k\}$. We show that if every group G_i $(1 \leq i \leq k)$ has a distinguisher with space complexity at most $s(n)$ and error probability $\epsilon(n)$, then every graph product of the groups G_i has a distinguisher with space complexity $\mathcal{O}(s(n) + \log n)$ and error probability roughly $n^2\epsilon(n)$ (Theorem 9). As a corollary we obtain for instance a randomized streaming algorithm with logarithmic space complexity for the word problem of a graph product of linear groups. Theorem 9 is similar to the following result from [6]: If the word problem for every group G_i can be solved in deterministic logspace on a Turing machine then the same is true for every graph product of the G_i. Kausch in his thesis [13] strengthened this result by showing that the word problem of the graph product is AC_0-Turing-reducible to the word problems of the G_i $(1 \leq i \leq k)$ and the free group of rank two.

Our second main contribution deals with randomized streaming algorithms for subgroup membership problems. In a subgroup membership problem one has a subgroup H of a f.g. group G. For an input word $w \in \Sigma^*$ (Σ is again a finite set of generators for G) one has to determine whether w represents an element of H. The word problem is the special case where $H = 1$. We present a randomized streaming algorithm with logarithmic space complexity for the case where G is a f.g. free group and H is a f.g. subgroup of G (Theorem 14). Moreover, we show that this result extends neither to the case where H is not finitely generated (Theorem 15) nor the case where H is a finitely generated subgroup of a direct product of two free groups of rank two (Theorem 16).

2 Preliminaries

For integers $a < b$ let $[a, b]$ be the integer interval $\{a, a+1, \ldots, b\}$. We write $[0,1]_{\mathbb{R}}$ for the set $\{r \in \mathbb{R} \colon 0 \leq r \leq 1\}$ of all probabilities.

Let Σ be a finite alphabet. As usual we write Σ^* for the set of all finite words over the alphabet w. The empty word is denoted with ε. For a word $w = a_1 a_2 \cdots a_n$ ($a_1, a_2, \ldots, a_n \in \Sigma$) let $|w| = n$ be its length and $w[i] = a_i$ (for $1 \leq i \leq n$) the symbol at position i. A prefix of a word w is a word u such that $w = uv$ for some word v. We denote with $\mathcal{P}(w)$ the set of all prefixes of w. Let $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$ be the set of non-empty words and $\Sigma^{\leq n} = \{w \in \Sigma^* \colon |w| \leq n\}$ be the set of all words of length at most n. For a subalphabet $\Theta \subseteq \Sigma$ we denote with $\pi_\Theta : \Sigma^* \to \Theta^*$ the projection homomorphism that deletes all symbols from $\Sigma \setminus \Theta$ in a word: $\pi_\Theta(a) = a$ for $a \in \Theta$ and $\pi_\Theta(a) = \varepsilon$ for $a \in \Sigma \setminus \Theta$.

2.1 Sequential transducer

In Section 5 we make use of (left-)sequential transducers, see e.g. [3] for more details. A sequential transducer is a tuple $\mathcal{T} = (Q, \Sigma, \Gamma, q_0, \delta)$, where Q is a finite set of states, Σ is the input alphabet, Γ is the output alphabet, $q_0 \in Q$ is the initial state, and $\delta : Q \times \Sigma \to Q \times \Gamma^*$ is the transition function. The meaning of $\delta(q, a) = (p, u)$ is that if $\mathcal{T}$ is in state q and the next input symbol is a then it moves to state p and outputs the word u. We extend δ to a mapping $\delta : Q \times \Sigma^* \to Q \times \Gamma^*$ as follows, where $q \in Q$, $a \in \Sigma$ and $w \in \Sigma^*$:

- $\delta(q, \varepsilon) = (q, \varepsilon)$ for all $q \in Q$, and
- if $\delta(q, a) = (p, u)$ and $\delta(p, w) = (r, v)$ then $\delta(q, aw) = (r, uv)$.

We define the function $f_{\mathcal{T}} : \Sigma^* \to \Gamma^*$ computed by $\mathcal{T}$ by $f_{\mathcal{T}}(w) = x$ if and only if $\delta(q_0, w) = (q, x)$ for some $q \in Q$. Intuitively, in order compute $f_{\mathcal{T}}(w)$, $\mathcal{T}$ reads the word w starting in the initial state q_0 and thereby concatenates all the outputs produced in the transitions.

2.2 Probabilistic finite automata

In the following we introduce probabilistic finite automata [21, 22] as a model for randomized streaming algorithms. A *probabilistic finite automaton* (PFA) $\mathcal{A} = (Q, \Sigma, \iota, \rho, F)$ consists of a finite set of states Q, a finite alphabet Σ, an *initial state distribution* $\iota\colon Q \to [0,1]_{\mathbb{R}}$, a *transition probability function* $\rho\colon Q \times \Sigma \times Q \to [0,1]_{\mathbb{R}}$ and a set of final states $F \subseteq Q$ such that $\sum_{q \in Q} \iota(q) = 1$ and $\sum_{q \in Q} \rho(p, a, q) = 1$ for all $p \in Q$, $a \in \Sigma$. If ρ is required to map into $\{0, 1\}$, then $\mathcal{A}$ is a *semi-probabilitistic finite automaton* (semiPFA). This means that after choosing the initial state according to the distribution ι, $\mathcal{A}$ proceeds deterministically.

Let $\mathcal{A} = (Q, \Sigma, \iota, \rho, F)$ be a PFA. For a random variable X with values from Q and $a \in \Sigma$ we define the random variable $X \cdot a$ (which also takes values from Q) by

$$\mathsf{Prob}[X \cdot a = q] = \sum_{p \in Q} \mathsf{Prob}[X = p] \cdot \rho(p, a, q).$$

For a word $w \in \Sigma^*$ we define a random variable $\mathcal{A}(w)$ with values from Q inductively as follows: the random variable $\mathcal{A}(\varepsilon)$ is defined such that $\mathsf{Prob}[\mathcal{A}(\varepsilon) = q] = \iota(q)$ for all $q \in Q$. Moreover, $\mathcal{A}(wa) = \mathcal{A}(w) \cdot a$ for all $w \in \Sigma^*$ and $a \in \Sigma$. Thus, $\mathsf{Prob}[\mathcal{A}(w) = q]$ is the probability that $\mathcal{A}$ is in state q after reading w. For a language $L \subseteq \Sigma^*$, the *error probability* of $\mathcal{A}$ on $w \in \Sigma^*$ for L is

$$\epsilon(\mathcal{A}, w, L) = \begin{cases} \mathsf{Prob}[\mathcal{A}(w) \notin F] & \text{if } w \in L, \\ \mathsf{Prob}[\mathcal{A}(w) \in F] & \text{if } w \notin L. \end{cases}$$

If $\mathcal{A}$ is a semiPFA then we can identify ρ with a mapping $\rho : Q \times \Sigma \to Q$, where $\rho(p, a)$ is the unique state q with $\rho(p, a, q) = 1$. This mapping ρ is extended to a mapping $\rho : Q \times \Sigma^* \to Q$ in the usual way: $\rho(p, \varepsilon) = p$ and $\rho(p, aw) = \rho(\rho(p, a), w)$. We then obtain

$$\mathsf{Prob}[\mathcal{A}(w) = q] = \sum \{\iota(p) : p \in Q, \rho(p, w) = q\}.$$

For a semiPFA $\mathcal{A} = (Q, \Sigma, \iota, \rho, F)$ and a boolean condition $\mathcal{E} : Q \to \{0, 1\}$ we define the probability $\mathsf{Prob}_{q \in Q}[\mathcal{E}(q)] = \sum\{\iota(q) : q \in Q, \mathcal{E}(q) = 1\}$. SemiPFAs are needed in Section 4 for the notion of a distinguisher.

2.3 Randomized streaming algorithms

In this section we define our model of randomized streaming algorithms. It is a non-uniform model in the sense that for every input length n we have a separate algorithm that handles inputs of length at most n. Formally, a (non-uniform) *randomized streaming algorithm* is a sequence $\mathcal{R} = (\mathcal{A}_n)_{n \geq 0}$ of PFA $\mathcal{A}_n$ over the same alphabet Σ. If every $\mathcal{A}_n$ is a semiPFA, we speak of a *semi-randomized streaming algorithm*.

Let $\epsilon_0, \epsilon_1 : \mathbb{N} \to [0, 1]_{\mathbb{R}}$ be monotonically decreasing functions. A randomized streaming algorithm $\mathcal{R} = (\mathcal{A}_n)_{n \geq 0}$ is (ϵ_0, ϵ_1)-*correct* for a language $L \subseteq \Sigma^*$ if for every large enough $n \geq 0$ and every word $w \in \Sigma^{\leq n}$ we have the following:

- if $w \in L$ then $\epsilon(\mathcal{A}_n, w, L) \leq \epsilon_1(n)$, and
- if $w \notin L$ then $\epsilon(\mathcal{A}_n, w, L) \leq \epsilon_0(n)$.

If $\epsilon_0 = \epsilon_1 =: \epsilon$ then we also say that $\mathcal{R}$ is ϵ-correct for L. We say that $\mathcal{R}$ is a randomized streaming algorithm for L if it is 1/3-correct for L. The choice of 1/3 for the error probability is not important. Using a standard application of the Chernoff bound, one can make the error probability an arbitrarily small constant; see [17, Theorem 4.1].

The *space complexity* of the randomized streaming algorithm $\mathcal{R} = (\mathcal{A}_n)_{n \geq 0}$ is the function $s(\mathcal{R}, n) = \lceil \log_2 |Q_n| \rceil$, where Q_n is the state set of $\mathcal{A}_n$. The motivation for this definition is that states of Q_n can be encoded by bit strings of length at most $\lceil \log_2 |Q_n| \rceil$. The *randomized streaming space complexity of the language* L is the smallest possible function $s(\mathcal{R}, n)$, where $\mathcal{R}$ is a randomized streaming algorithm for L. It is always bounded by $\mathcal{O}(n)$ since even a deterministic streaming algorithm can store the whole input word $w \in \Sigma^{\leq n}$ using $\mathcal{O}(n)$ bits.

As remarked before, our model of randomized streaming algorithms is non-uniform in the sense that for every input length n we have a separate streaming algorithm $\mathcal{A}_n$. This makes lower bounds of course stronger. On the other hand, the randomized streaming algorithms that we construct for concrete groups will be mostly uniform in the sense that there is an efficient algorithm that constructs from a given n the PFA $\mathcal{A}_n$. An exception is the following theorem (see [17, Theorem 4.3]), which uses non-uniformity in a crucial way.

▶ **Theorem 1.** *Let $\mathcal{R}$ be a randomized streaming algorithm such that $s(\mathcal{R}, n) \geq \Omega(\log n)$ and $\mathcal{R}$ is ϵ-correct for a language L. Then there exists a semi-randomized streaming algorithm $\mathcal{S}$ such that $s(\mathcal{S}, n) = \Theta(s(\mathcal{R}, n))$ and $\mathcal{S}$ is 2ϵ-correct for the language L.*

3 Groups and word problems

Let G be a group. The identity element will be always denoted with 1. For a subset $\Sigma \subseteq G$, we denote with $\langle \Sigma \rangle$ the subgroup of G generated by Σ. It is the set of all products of elements from $\Sigma \cup \Sigma^{-1}$. It can be also defined as the smallest (w.r.t. inclusion) subgroup of G that contains Σ. Similarly, the *normal closure* $N(\Sigma)$ of Σ is smallest normal subgroup of G that contains Σ. In this case, one gets the quotient group $G/N(\Sigma)$.

We only consider *finitely generated* (f.g.) groups G, for which there is a finite set $\Sigma \subseteq G$ with $G = \langle \Sigma \rangle$; such a set Σ is called a *finite generating set* for G. If $\Sigma = \Sigma^{-1}$ then we say that Σ is a *finite symmetric generating set* for G. In the following we assume that all finite generating sets are symmetric. Every word $w \in \Sigma^*$ evaluates to a group element $\pi_G(w)$ in the natural way; here $\pi_G : \Sigma^* \to G$ is the canonical morphism from the free monoid

Σ^* to G that is the identity on Σ. Instead of $\pi_G(u) = \pi_G(v)$ we also write $u \equiv_G v$. Let $\mathsf{WP}(G, \Sigma) = \{w \in \Sigma^* \mid \pi_G(w) = 1\}$ be the *word problem for* G with respect to the generating set Σ.

We are interested in streaming algorithms for words problems $\mathsf{WP}(G, \Sigma)$. By the following lemma (see [17, Lemma 5.1]) the randomized streaming space complexity for a word problem only changes by a constant when the generating set is changed.

▶ **Lemma 2.** *Let Σ_1 and Σ_2 be finite symmetric generating sets for the group G and let $s_i(n)$ be the randomized streaming space complexity of* $\mathsf{WP}(G, \Sigma_i)$. *Then there exists a constant c that depends on G, Σ_1 and Σ_2 such that $s_1(n) \le s_2(c \cdot n)$.*

4 Distinguishers for groups

Let G be a f.g. group G with the finite generating set Σ. Moreover, let $\epsilon_0, \epsilon_1 : \mathbb{N} \to [0,1]_{\mathbb{R}}$ be monotonically decreasing functions. A *semi-randomized* streaming algorithm $(\mathcal{A}_n)_{n\ge 0}$ with $\mathcal{A}_n = (Q_n, \Sigma, \iota_n, \rho_n, F_n)$ is called an (ϵ_0, ϵ_1)*-distinguisher* for G (with respect to Σ), if the following properties hold for all large enough $n \ge 0$ and all words $u, v \in \Sigma^{\le n}$:

- If $u \equiv_G v$ then $\mathsf{Prob}_{q\in Q_n}[\rho_n(q,u) = \rho_n(q,v)] \ge 1 - \epsilon_1(n)$. In other words: for a randomly chosen initial state, the semiPFA $\mathcal{A}_n$ arrives with probability at least $1 - \epsilon_1(n)$ in the same state after reading u and v.
- If $u \not\equiv_G v$ then $\mathsf{Prob}_{q\in Q_n}[\rho_n(q,u) \ne \rho_n(q,v)] \ge 1 - \epsilon_0(n)$. In other words: for a randomly chosen initial state, the semiPFA $\mathcal{A}_n$ arrives with probability at least $1 - \epsilon_0(n)$ in different states after reading u and v.

Note that the set F_n of final states of $\mathcal{A}_n$ is not important for a distinguisher and we will just write $\mathcal{A}_n = (Q_n, \Sigma, \iota_n, \rho_n)$ in the following if we talk about an (ϵ_0, ϵ_1)-distinguisher $(\mathcal{A}_n)_{n\ge 0}$. The following result is shown in [17].

▶ **Lemma 3.** *Let $\mathcal{R}$ be an (ϵ_0, ϵ_1)-distinguisher for G with respect to Σ. Then* $\mathsf{WP}(G, \Sigma)$ *has an (ϵ_0, ϵ_1)-correct semi-randomized streaming algorithm with space complexity $2 \cdot s(\mathcal{R}, n)$.*

Due to Lemma 3, our goal in the rest of the paper will be the construction of space efficient (ϵ_0, ϵ_1)-distinguishers for groups. In the next section we will need some further observations on (ϵ_0, ϵ_1)-distinguishers that we introduce in the rest of this section.

For equivalence relations $\equiv_1$ and $\equiv_2$ on a set A and a subset $S \subseteq A$ we say that:

- $\equiv_1$ refines $\equiv_2$ on S if for all $a, b \in S$ we have: if $a \equiv_1 b$ then $a \equiv_2 b$;
- $\equiv_1$ equals $\equiv_2$ on S if for all $a, b \in S$ we have: $a \equiv_1 b$ if and only if $a \equiv_2 b$.

For a semiPFA $\mathcal{A} = (Q, \Sigma, \iota, \rho)$ and a state $q \in Q$ we define the equivalence relation $\equiv_{\mathcal{A},q}$ on Σ^* as follows: $u \equiv_{\mathcal{A},q} v$ if and only if $\rho(q,u) = \rho(q,v)$. Whenever $\mathcal{A}$ is clear from the context, we just write $\equiv_q$ instead of $\equiv_{\mathcal{A},q}$. The statements from the following lemma can be shown by straightforward applications of the union bound; see [17].

▶ **Lemma 4.** *Let $(\mathcal{A}_n)_{n\ge 0}$ be an (ϵ_0, ϵ_1)-distinguisher for the f.g. group G with respect to the generating set Σ. Let $\mathcal{A}_n = (Q_n, \Sigma, \iota, \rho)$. Consider a set $S \subseteq \Sigma^{\le n}$. Then, the following statements hold, where $\equiv_q$ refers to $\mathcal{A}_n$:*

- $\mathsf{Prob}_{q\in Q_n}[\equiv_G$ *equals* $\equiv_q$ *on* $S] \ge 1 - \max\{\epsilon_0, \epsilon_1\}\binom{|S|}{2}$,
- $\mathsf{Prob}_{q\in Q_n}[\equiv_G$ *refines* $\equiv_q$ *on* $S] \ge 1 - \epsilon_1\binom{|S|}{2}$,
- $\mathsf{Prob}_{q\in Q_n}[\equiv_q$ *refines* $\equiv_G$ *on* $S] \ge 1 - \epsilon_0\binom{|S|}{2}$.

The following two simple lemmas are needed in Section 5; their proofs can be found in [17]. Recall that for a word w we write $\mathcal{P}(w)$ for the set of all prefixes of w.

▶ **Lemma 5.** *Let G be a f.g. group with the finite generating set Σ and let $\mathcal{A} = (Q, \Sigma, \iota, \rho)$ be a semiPFA with $q \in Q$. Consider $u, v \in \Sigma^*$ such that $\equiv_G$ refines $\equiv_q$ on $\mathcal{P}(u) \cup \mathcal{P}(v)$ and let $u = xyz$ with $y \equiv_G 1$. Then $\equiv_G$ refines $\equiv_q$ on $\mathcal{P}(xz) \cup \mathcal{P}(v)$.*

▶ **Lemma 6.** *Let G, $\mathcal{A}$, and q be as in Lemma 5. Consider $u, v \in \Sigma^*$ such that $\equiv_q$ refines $\equiv_G$ on $\mathcal{P}(u) \cup \mathcal{P}(v)$ and let $u = xyz$ with $\rho(q, x) = \rho(q, xy)$. Then $\equiv_q$ refines $\equiv_G$ on $\mathcal{P}(xz) \cup \mathcal{P}(v)$.*

Finally we state the following result from [17, Theorem 9.1]. Recall that a linear group is a group of matrices over some field.

▶ **Theorem 7.** *For every f.g. linear group G and every $c > 0$ there exists a $(1/n^c, 0)$-distinguisher with space complexity $\mathcal{O}(\log n)$.*

Theorem 7 can serve as a basis for the construction of further distinguishers. Note that in the positive case (where $u \equiv_G v$) the error probability is zero.

5 Randomized streaming algorithms for graph products

In this section we investigate a common generalization of the free product and direct product, which is known as the graph product of groups.

Let us first define the free product of two groups G and H. Let $A = G \backslash \{1\}$ and $B = H \backslash \{1\}$. W.l.o.g. we assume that $A \cap B = \emptyset$. The *free product* $G * H$ consists of all alternating sequences $a_1 a_2 \cdots a_n$ where $n \geq 0$, $a_i \in A \cup B$ for all $i \in [1, n]$ and $a_i \in A \Leftrightarrow a_{i+1} \in B$ for all $i \in [1, n-1]$. The identity element is of course the empty sequence ε. The product $u \cdot v$ of two elements $u, v \in G * H$ is obtained by concatenating u and v and then making the obvious simplifications according to the multiplication tables of G and H. More precisely, let $u = a_n a_{n-1} \cdots a_1$ and $v = b_1 b_2 \cdots b_m$. If $n = 0$ then $u \cdot v = v$ and if $m = 0$ then $u \cdot v = u$. Now assume that $n > 0$ and $m > 0$. If $a_1 \in A \Leftrightarrow b_1 \in B$ then $u \cdot v = a_n a_{n-1} \cdots a_1 b_1 b_2 \cdots b_m$. Otherwise choose $k \geq 0$ maximal such that $a_i = b_i^{-1}$ (in either G or H) holds for all $i \in [1, k]$. If $k = n$ then $u \cdot v = b_{k+1} \cdots b_m$ and if $k = m$ then $u \cdot v = a_n \cdots a_{k+1}$. Finally, if $k < n$ and $k < m$ then $u \cdot v = a_n \cdots a_{k+2}(a_{k+1} \cdot b_{k+1}) b_{k+2} \cdots b_m$. Note that $a_{k+1} \cdot b_{k+1}$ is a nontrivial element (either in G or H) by the choice of k. The free product of several groups $G_1, \ldots, G_c$ can be simply defined as $\ast_{i \in [1,c]} G_i = (\cdots((G_1 * G_2) * G_3) * \cdots * G_c)$.

A graph product is specified by a list of groups $G_1, \ldots, G_c$ and a symmetric and irreflexive relation $I \subseteq [1, c] \times [1, c]$. The corresponding *graph product* $G = \mathsf{GP}(G_1, \ldots, G_c, I)$ is the quotient $(\ast_{i \in [1,c]} G_i)/N$ of the free product $\ast_{i \in [1,c]} G_i$ modulo the normal closure N of all commutators $aba^{-1}b^{-1}$, where $a \in G_i$, $b \in G_j$ and $(i, j) \in I$. In other words, we take the free product $\ast_{i \in [1,c]} G_i$ but allow elements from groups G_i and G_j with $(i, j) \in I$ to commute. Graph products interpolate in a natural way between free products ($I = \emptyset$) and direct products ($I = \{(i, j) : i, j \in [1, c], i \neq j\}$). Graph products were introduced by Green in her thesis [8]. Graph products $\mathsf{GP}(G_1, \ldots, G_c, I)$, where every G_i is isomorphic to $\mathbb{Z}$, are also known as *graph groups* (or right-angled Artin groups). We will make use of the fact that every graph group is linear [11].

Let Σ_i be a finite symmetric generating set for G_i, where w.l.o.g. $1 \notin \Sigma_i$ and $\Sigma_i \cap \Sigma_j = \emptyset$ for $i \neq j$. Then, $\Sigma = \bigcup_{i=1}^{c} \Sigma_i$ generates G. For a word $u \in \Sigma^*$, the *block factorization* of u is the unique factorization $u = u_1 u_2 \cdots u_l$ such that $l \geq 0$, $u_1, \ldots, u_l \in \bigcup_{i \in [1,c]} \Sigma_i^+$ and $u_j u_{j+1} \notin \bigcup_{i \in [1,c]} \Sigma_i^+$ for all $j \in [1, l-1]$. The factors $u_1, u_2, \ldots, u_l$ are called the *blocks* of u.

We define several rewrite relations on words from Σ^* as follows: take $u, v \in \Sigma^*$ and let $u = u_1 u_2 \cdots u_l$ be the block factorization of u.

- We write $u \leftrightarrow_s v$ (s for swap) if there is $i \in [1, l-1]$ and $(j,k) \in I$ such that $u_i \in \Sigma_j^+$, $u_{i+1} \in \Sigma_k^+$ and $v = u_1 u_2 \cdots u_{i-1} u_{i+1} u_i u_{i+2} \cdots u_l$. In other words, we swap consecutive commuting blocks. Note that $\leftrightarrow_s$ is a symmetric relation.
- We write $u \to_d v$ (d for delete) if there is $i \in [1,l]$ and $j \in [1,c]$ such that $u_i \in \Sigma_j^+$, $u_i \equiv_{G_j} 1$ and $v = u_1 u_2 \cdots u_{i-1} u_{i+1} u_{i+2} \cdots u_l$. In other words, we delete a block that is trivial in its group.
- We write $u \leftrightarrow_r v$ (r for replace) if there is $i \in [1,l]$ and $j \in [1,c]$ such that $u_i, u_i' \in \Sigma_j^+$, $u_i \equiv_{G_j} u_i'$ and $v = u_1 u_2 \cdots u_{i-1} u_i' u_{i+1} u_{i+2} \cdots u_l$. In other words, we replace a block by an equivalent non-empty word. Note that $\leftrightarrow_r$ is a symmetric relation.

Clearly, in all three cases we have $u \equiv_G v$. If $u \to_d v$, then the number of blocks of v is smaller than the number of blocks of u and if $u \leftrightarrow_s v$ then the number of blocks of v can be smaller than the number of blocks of u (since two blocks can be merged into a single block). We write $u \leftrightarrow_{sr} v$ if $u \leftrightarrow_s v$ or $u \leftrightarrow_r v$ and we write $u \to_{sd} v$ if $u \leftrightarrow_s v$ or $u \to_d v$.

Let us say that a word $u \in \Sigma^*$ with l blocks is *reduced*, if there is no $v \in \Sigma^*$ such that $u \to_{sd}^* v$ and v has at most $l-1$ blocks. Clearly, for every word $u \in \Sigma^*$ there is a reduced word $u' \in \Sigma^*$ such that $u \to_{sd}^* u'$. The following result can be found in [8, Theorem 3.9] and [10] in slightly different notations.

▶ **Lemma 8.** *Let G be a graph product as above and $u, v \in \Sigma^*$. The following are equivalent:*

- *$u \equiv_G v$*
- *There are reduced words u', v' such that $u \to_{sd}^* u'$, $v \to_{sd}^* v'$, and $u' \leftrightarrow_r^* v'$.*

Consider a word $u \in \Sigma^*$ and its block factorization $u = u_1 u_2 \dots u_l$. A *pure prefix* of u is a word $u_{k_1} u_{k_2} \cdots u_{k_m}$ such that for some $i \in [1,c]$ we have

- $1 \le k_1 < k_2 < \cdots < k_m \le l$,
- $u_{k_1}, u_{k_2}, \dots, u_{k_m} \in \Sigma_i^+$ and
- if $k_i < j < k_{i+1}$ for some $i \in [1, m-1]$ or $j < k_1$ then $u_j \notin \Sigma_i^+$.

▶ **Theorem 9.** *Let $G = \mathsf{GP}(G_1, \dots, G_c, I)$ be a graph product as above and let $\mathcal{R}_i = (\mathcal{A}_{i,n})_{n \ge 0}$ be an (ϵ_0, ϵ_1)-distinguisher for G_i. Let $d \ge 1$ and define $\zeta_0 = 2\epsilon_0 c n^2 + 1/n^d$ and $\zeta_1 = 2\epsilon_1 c n^2$. Then, there exists a (ζ_0, ζ_1)-distinguisher for G with space complexity $\mathcal{O}(\sum_{i=1}^c s(\mathcal{R}_i, n) + \log n)$.*

Proof. Let us fix an input length n and let $\mathcal{A}_{i,n} = (Q_{i,n}, \Sigma_i, \iota_{i,n}, \rho_{i,n})$, where w.l.o.g. $Q_{i,n} = [0, |Q_{i,n}| - 1]$. To simplify the notation, we will omit the second subscript n in the following, i.e., we write $\mathcal{A}_i = (Q_i, \Sigma_i, \iota_i, \rho_i)$ with $Q_i = [0, |Q_i| - 1]$ for the semiPFA $\mathcal{A}_{i,n}$. For a state $q \in Q_i$, we will use the equivalence relation $\equiv_q \, = \, \equiv_{\mathcal{A}_i, q}$ defined in Section 4. For a word $w \in \Sigma^*$, we write $\pi_i(w)$ for the projection $\pi_{\Sigma_i}(w)$.

For every $i \in [1,c]$ we choose a new symbol a_i and consider the infinite cyclic group $\langle a_i \rangle \cong \mathbb{Z}$. Let $\Gamma = \{a_1, a_1^{-1}, \dots, a_c, a_c^{-1}\}$ and consider the graph group $H = \mathsf{GP}(\langle a_1 \rangle, \dots, \langle a_c \rangle, I)$. Since every graph group is linear, there is a $(1/m^d, 0)$-distinguisher $(\mathcal{B}_m)_{m \ge 0}$ with space complexity $\mathcal{O}(\log m)$ for H by Theorem 7. Let $\mathcal{B}_m = (R_m, \Gamma, \lambda_m, \sigma_m)$.

We build from the semiPFA $\mathcal{A}_i$ and a state $q \in Q_i$ a sequential transducer $\mathcal{T}_{i,q} = (Q_i, \Sigma_i, \{a_i, a_i^{-1}\}, q, \delta_i)$, where for all $a \in \Sigma_i$ and $p \in Q_i$ we define (recall that $Q_i \subseteq \mathbb{N}$):

$$\delta_i(p, a) = (\rho_i(p,a), a_i^{-p} a_i^{\rho_i(p,a)}).$$

Let $f_{i,q} := f_{\mathcal{T}_{i,q}} : \Sigma_i^* \to \{a_i, a_i^{-1}\}^*$ be the function computed by $\mathcal{T}_{i,q}$. For a tuple $\bar{q} = (q_1, \dots, q_c) \in \prod_{i \in [1,c]} Q_i$ of states from the semiPFAs $\mathcal{A}_i$ we define the sequential transducer $\mathcal{T}_{\bar{q}}$ by taking the direct product of the $\mathcal{T}_{i,q_i}$ ($i \in [1,c]$). Formally, it is defined as follows:

$$\mathcal{T}_{\bar{q}} = (\prod_{i \in [1,c]} Q_i, \Sigma, \Gamma, \bar{q}, \delta)$$

Algorithm 1 (ζ_0, ζ_1)-distinguisher for $\mathsf{GP}(G_1, \dots, G_c, I)$

```
global variables: $q_i \in Q_i$ for all $i \in [1,c]$, $r \in R_m$
initialization:
guess $q_i \in Q_i = [0, |Q_i| - 1]$ according to the input distribution $\iota_i$ of $\mathcal{A}_i$ ;
guess $r \in R_m$ according to the input distribution $\lambda_m$ of $\mathcal{B}_m$ ;
next input letter: $a \in \Sigma$
let $i \in [1,c]$ such that $a \in \Sigma_i$ ;
$r := \sigma_m(r, a_i^{-q_i} a_i^{\rho_i(q_i,a)})$ ; $q_i := \rho_i(q_i, a)$ ;
```

where for every $i \in [1,c]$, $a \in \Sigma_i$, and $(p_1, \dots, p_c) \in \prod_{i \in [1,c]} Q_i$ we have

$$\delta((p_1, \dots, p_c), a) = \big((p_1, \dots, p_{i-1}, \rho_i(p_i, a), p_{i+1}, \dots, p_c), a_i^{-p_i} a_i^{\rho_i(p_i,a)}\big).$$

Let $f_{\bar{q}} := f_{\mathcal{T}_{\bar{q}}} : \Sigma^* \to \Gamma^*$ be the function computed by $\mathcal{T}_{\bar{q}}$. Moreover, define

$$m = 2 \cdot n \cdot \max\{|Q_i| : i \in [1,c]\} \leq n \cdot 2^{1+\max\{s(\mathcal{R}_i,n) :\, i \in [1,c]\}}.$$

Note that $|f_{\bar{q}}(w)| \leq m$ if $|w| \leq n$.

Our randomized streaming algorithm for G and input length n uses the semiPFA $\mathcal{B}_m$. States of $\mathcal{B}_m$ can be stored with $\mathcal{O}(\log m) \leq \mathcal{O}(\max\{s(\mathcal{R}_i, n) \colon i \in [1,c]\} + \log n)$ bits. Basically, for an input word $w \in \Sigma^{\leq n}$ the algorithm simulates $\mathcal{A}_i$ ($i \in [1,c]$) on the projections $w_i = \pi_i(w)$ and feeds the word $f_{\bar{q}}(w)$ into the semiPFA $\mathcal{B}_m$. Here, the state tuple $\bar{q}$ is randomly guessed in the beginning according to the distributions ι_i. The complete streaming algorithm is Algorithm 1. It stores at most $\sum_{i=1}^{c} s(\mathcal{R}_i, n) + \mathcal{O}(\max\{s(\mathcal{R}_i, n) \colon i \in [1,c]\} + \log n)$ bits.

Before we analyze the error probability of the algorithm we need some preparation. For $i \in [1,c]$ and a word $w \in \Sigma^*$ let $\mathcal{P}_i(w) = \mathcal{P}(\pi_i(w))$ be the set of all prefixes of the projection $\pi_i(w)$. Assume that $y \in \Sigma_i^+$ is a block of w and write $w = xyz$. We then have $f_{\bar{q}} = f_{\bar{q}}(x) f_{\bar{r}}(y) f_{\bar{s}}(z)$, where $\delta(\bar{q}, x) = (\bar{r}, f_{\bar{q}}(x))$ and $\delta(\bar{r}, y) = (\bar{s}, f_{\bar{r}}(y))$. The word $f_{\bar{r}}(y)$ is also a block of $f_{\bar{q}}$ (for this it is important that the transducers $\mathcal{T}_{j,q}$ translate non-empty words into non-empty words). Since $y \in \Sigma_i^+$ we have $r_j = s_j$ for all $j \in [1,c] \setminus \{i\}$ and $f_{\bar{r}}(y) = f_{i,r_i}(y)$. In addition, the definition of the transducer $\mathcal{T}_{i,r_i}$ implies that $f_{\bar{r}}(y) \equiv_{\langle a_i \rangle} a_i^{-r_i} a_i^{s_i}$.

Consider now two input words $u, v \in \Sigma^{\leq n}$ and let $\mathcal{S}_i = \mathcal{P}_i(u) \cup \mathcal{P}_i(v)$ for $i \in [1,c]$, so that $|\mathcal{S}_i| \leq 2n$. By Lemma 4 we have for all $i \in [1,c]$:

- $\mathsf{Prob}_{q \in Q_i}[\equiv_q \text{ refines } \equiv_{G_i} \text{ on } \mathcal{S}_i] \geq 1 - \epsilon_0 \binom{|\mathcal{S}_i|}{2} \geq 1 - 2\epsilon_0 n^2$,
- $\mathsf{Prob}_{q \in Q_i}[\equiv_{G_i} \text{ refines } \equiv_q \text{ on } \mathcal{S}_i] \geq 1 - \epsilon_1 \binom{|\mathcal{S}_i|}{2} \geq 1 - 2\epsilon_1 n^2$.

Our error analysis of Algorithm 1 is based on the following two claims.

▷ Claim 10. Assume that $\bar{q} = (q_1, \dots, q_c)$ is such that $\equiv_{G_i}$ refines $\equiv_{q_i}$ on $\mathcal{S}_i$ for every $i \in [1,c]$. If $u \to_{sd}^* u'$ and $v \to_{sd}^* v'$, then $f_{\bar{q}}(u) \to_{sd}^* f_{\bar{q}}(u')$, $f_{\bar{q}}(v) \to_{sd}^* f_{\bar{q}}(v')$ and $\equiv_{G_i}$ refines $\equiv_{q_i}$ on $\mathcal{P}_i(u') \cup \mathcal{P}_i(v')$ for every $i \in [1,c]$.

Proof. It suffices to show the following: If $u \to_{sd} u'$ holds, then $f_{\bar{q}}(u) \to_{sd} f_{\bar{q}}(u')$ and $\equiv_{G_i}$ refines $\equiv_{q_i}$ on $\mathcal{P}_i(u') \cup \mathcal{P}_i(v)$ for every $i \in [1,c]$. From this (and the symmetric statement where $v \to_{sd} v'$ and $u = u'$) we obtain the general statement by induction on the number of $\to_{sd}$-steps. We distinguish two cases.

Case 1. $u \leftrightarrow_s u'$. We must have $u = xy_1y_2z$ and $u' = xy_2y_1z$ for blocks y_1, y_2 such that

$y_1 \in \Sigma_i^+$, $y_2 \in \Sigma_j^+$ and $(i,j) \in I$ (in particular $i \neq j$). We obtain

$$\begin{aligned} f_{\bar{q}}(u) &= f_{\bar{q}}(x) f_{\bar{p}}(y_1) f_{\bar{r}}(y_2) f_{\bar{s}}(z) \text{ and} \\ f_{\bar{q}}(u') &= f_{\bar{q}}(x) f_{\bar{p}}(y_2) f_{\bar{r}'}(y_1) f_{\bar{s}}(z), \end{aligned}$$

where $\delta(\bar{q}, x) = (\bar{p}, f_{\bar{q}}(x))$, $\delta(\bar{p}, y_1) = (\bar{r}, f_{\bar{p}}(y_1))$, $\delta(\bar{r}, y_2) = (\bar{s}, f_{\bar{r}}(y_2))$, $\delta(\bar{p}, y_2) = (\bar{r}', f_{\bar{p}}(y_2))$, and $\delta(\bar{r}', y_1) = (\bar{s}, f_{\bar{r}'}(y_1))$. If we write $\bar{p} = (p_1, \ldots, p_c)$, then there are states $r_i \in Q_i$ and $r_j \in Q_j$ such that

$$\begin{aligned} \bar{r} &= (p_1, \ldots, p_{i-1}, r_i, p_{i+1}, \ldots, p_c), && (1) \\ \bar{r}' &= (p_1, \ldots, p_{j-1}, r_j, p_{j+1}, \ldots, p_c), \text{ and} && (2) \\ \bar{s} &= (p_1, \ldots, p_{i-1}, r_i, p_{i+1}, \ldots, p_{j-1}, r_j, p_{j+1}, \ldots, p_c) && (3) \end{aligned}$$

(we assume w.l.o.g. that $i < j$). Moreover, $f_{\bar{p}}(y_1) = f_{i,p_i}(y_1) = f_{\bar{r}'}(y_1) \in \{a_i, a_i^{-1}\}^+$ and $f_{\bar{r}}(y_2) = f_{j,p_j}(y_2) = f_{\bar{p}}(y_2) \in \{a_j, a_j^{-1}\}^+$. Thus, we have

$$\begin{aligned} f_{\bar{q}}(u) &= f_{\bar{q}}(x) f_{\bar{p}}(y_1) f_{\bar{r}}(y_2) f_{\bar{s}}(z) \\ &= f_{\bar{q}}(x) f_{i,p_i}(y_1) f_{j,p_j}(y_2) f_{\bar{s}}(z) \\ &\leftrightarrow_s f_{\bar{q}}(x) f_{j,p_j}(y_2) f_{i,p_i}(y_1) f_{\bar{s}}(z) \\ &= f_{\bar{q}}(x) f_{\bar{p}}(y_2) f_{\bar{r}'}(y_1) f_{\bar{s}}(z) \\ &= f_{\bar{q}}(u'). \end{aligned}$$

Moreover, since $\mathcal{P}_i(u') = \mathcal{P}_i(u)$ and $\equiv_{G_i}$ refines $\equiv_{q_i}$ on $\mathcal{S}_i$ for all $i \in [1,c]$, it follows that $\equiv_{G_i}$ refines $\equiv_{q_i}$ on $\mathcal{P}_i(u') \cup \mathcal{P}_i(v)$ for all $i \in [1,c]$.

Case 2. $u \to_d u'$. Then we obtain a factorization $u = xyz$, where $y \in \Sigma_i^+$ is a block, $y \equiv_{G_i} 1$, and $u' = xz$. We obtain a factorization

$$f_{\bar{q}}(u) = f_{\bar{q}}(x) f_{\bar{r}}(y) f_{\bar{s}}(z),$$

where $\delta(\bar{q}, x) = (\bar{r}, f_{\bar{q}}(x))$ and $\delta(\bar{r}, y) = (\bar{s}, f_{\bar{r}}(y))$. The word $f_{\bar{r}}(y)$ is a block of $f_{\bar{q}}(u)$. For the projection $\pi_i(u)$ we have $\pi_i(u) = \pi_i(x) y \pi_i(z)$. Since $\equiv_{G_i}$ refines $\equiv_{q_i}$ on $\mathcal{S}_i$ and $\pi_i(x) \equiv_{G_i} \pi_i(x) y$, we obtain $\pi_i(x) \equiv_{q_i} \pi_i(x) y$. Since r_i (resp., s_i) is the state reached from q_i by the automaton $\mathcal{A}_i$ after reading $\pi_i(x)$ (resp., $\pi_i(x)y$), we obtain $r_i = s_i$ and hence $\bar{r} = \bar{s}$. This implies

$$f_{\bar{r}}(y) \equiv_{\langle a_i \rangle} a_i^{-r_i} a_i^{s_i} \equiv_{\langle a_i \rangle} 1.$$

Moreover, we have

$$f_{\bar{q}}(u') = f_{\bar{q}}(xz) = f_{\bar{q}}(x) f_{\bar{r}}(z) = f_{\bar{q}}(x) f_{\bar{s}}(z).$$

We therefore get $f_{\bar{q}}(u) \to_d f_{\bar{q}}(u')$.

It remains to show that $\equiv_{G_j}$ refines $\equiv_{q_j}$ on $\mathcal{P}_j(u') \cup \mathcal{P}_j(v)$ for every $j \in [1,c]$. For $j \neq i$ this is clear since $\mathcal{P}_j(u') \cup \mathcal{P}_j(v) = \mathcal{S}_j$. For $j = i$ we can use Lemma 5 for the words $\pi_i(u) = \pi_i(x) y \pi_i(z)$ and $\pi_i(v)$. ◁

▷ Claim 11. *Claim 2.* Assume that $\bar{q} = (q_1, \ldots, q_c)$ is such that $\equiv_{q_i}$ refines $\equiv_{G_i}$ on $\mathcal{S}_i$ for every $i \in [1,c]$. If $f_{\bar{q}}(u) \to_{sd}^* \tilde{u}$ and $f_{\bar{q}}(v) \to_{sd}^* \tilde{v}$, then there are $u', v' \in \Sigma^*$ such that $u \to_{sd}^* u'$, $v \to_{sd}^* v'$, $f_{\bar{q}}(u') = \tilde{u}$, $f_{\bar{q}}(v') = \tilde{v}$ and $\equiv_{q_i}$ refines $\equiv_{G_i}$ on $\mathcal{P}_i(u') \cup \mathcal{P}_i(v')$ for every $i \in [1,c]$.

Proof. The proof is very similar to the above proof of Claim 10. As in the above proof of Claim 10, it suffices to consider the case where $f_{\bar{q}}(u) \to_{sd} \tilde{u}$ and $\tilde{v} = f_{\bar{q}}(v)$.

Case 1. $f_{\bar{q}}(u) \leftrightarrow_s \tilde{u}$. Since the blocks of u are translated into the blocks of $f_{\bar{q}}(u)$ by the transducer $\mathcal{T}_{\bar{q}}$, we obtain a factorization $u = xy_1y_2z$ for blocks $y_1 \in \Sigma_i^+, y_2 \in \Sigma_j^+$ of u such that $(i, j) \in I$ (in particular $i \neq j$) and

$$\begin{aligned} f_{\bar{q}}(u) &= f_{\bar{q}}(x)f_{\bar{p}}(y_1)f_{\bar{r}}(y_2)f_{\bar{s}}(z), \\ \tilde{u} &= f_{\bar{q}}(x)f_{\bar{r}}(y_2)f_{\bar{p}}(y_1)f_{\bar{s}}(z). \end{aligned}$$

Here, the state tuples $\bar{p} = (p_1, \ldots, p_c)$, $\bar{r}$, and $\bar{s}$ are as in the above proof of Claim 10, see in particular (1) and (3). We can then define the tuple $\bar{r}'$ as in (2) and get $f_{\bar{p}}(y_1) = f_{i,p_i}(y_1) = f_{\bar{r}'}(y_1) \in \{a_i, a_i^{-1}\}^+$ and $f_{\bar{r}}(y_2) = f_{j,p_j}(y_2) = f_{\bar{p}}(y_2) \in \{a_j, a_j^{-1}\}^+$. We thus have

$$\tilde{u} = f_{\bar{q}}(x)f_{\bar{r}}(y_2)f_{\bar{p}}(y_1)f_{\bar{s}}(z) = f_{\bar{q}}(x)f_{\bar{p}}(y_2)f_{\bar{r}'}(y_1)f_{\bar{s}}(z) = f_{\bar{q}}(xy_2y_1z).$$

Clearly, we also have $u = xy_1y_2z \to_s xy_2y_1z$. So, we can set $u' = xy_2y_1z$. Since $\mathcal{P}_i(u') = \mathcal{P}_i(u)$ for all $i \in [1, c]$, it follows that $\equiv_{q_i}$ refines $\equiv_{G_i}$ on $\mathcal{P}_i(u') \cup \mathcal{P}_i(v)$ for all $i \in [1, c]$.

Case 2. $f_{\bar{q}}(u) \to_d \tilde{u}$. Then we obtain a factorization $u = xyz$, where $y \in \Sigma_i^+$ is a block of u,

$$\begin{aligned} f_{\bar{q}}(u) &= f_{\bar{q}}(x)f_{\bar{r}}(y)f_{\bar{s}}(z), \text{ and} \\ \tilde{u} &= f_{\bar{q}}(x)f_{\bar{s}}(z). \end{aligned}$$

The state tuples $\bar{r}$ and $\bar{s}$ are such that $\delta(\bar{q}, x) = (\bar{r}, f_{\bar{q}}(x))$ and $\delta(\bar{r}, y) = (\bar{s}, f_{\bar{r}}(y))$. Moreover, the word $f_{\bar{r}}(y)$ is a block of $f_{\bar{q}}(u)$ with

$$a_i^{-r_i}a_i^{s_i} \equiv_{\langle a_i \rangle} f_{\bar{r}}(y) \equiv_{\langle a_i \rangle} 1.$$

This implies that $r_i = s_i$ and hence $\bar{r} = \bar{s}$. We therefore have

$$\rho_i(q_i, \pi_i(x)) = r_i = s_i = \rho_i(q_i, \pi_i(x)y).$$

Since $\equiv_{q_i}$ refines $\equiv_{G_i}$ on $\mathcal{S}_i$ and $\pi_i(x), \pi_i(x)y \in \mathcal{S}_i$, we get $\pi_i(x) \equiv_{G_i} \pi_i(x)y$, i.e., $y \equiv_{G_i} 1$. If we set $u' = xz$ we get $u \to_d u'$ and

$$\tilde{u} = f_{\bar{q}}(x)f_{\bar{s}}(z) = f_{\bar{q}}(x)f_{\bar{r}}(z) = f_{\bar{q}}(xz) = f_{\bar{q}}(u').$$

It remains to show that $\equiv_{q_j}$ refines $\equiv_{G_j}$ on $\mathcal{P}_j(u') \cup \mathcal{P}_j(v)$ for every $j \in [1, c]$. For $j \neq i$ this is clear since $\mathcal{P}_j(u') \cup \mathcal{P}_j(v) = \mathcal{S}_j$. For $j = i$ we can use Lemma 6 for the words $\pi_i(u) = \pi_i(x)y\pi_i(z)$ and $\pi_i(v)$. ◁

We now estimate the error for the input words u and v. There are two cases to consider:

Case 1. $u \equiv_G v$. We will show that Algorithm 1 reaches with probability at least $1 - 2\epsilon_1 cn^2$ the same state when running on u and v, respectively. For this, assume that the randomly selected initial states $q_i \in Q_i$ are such that $\equiv_{G_i}$ refines $\equiv_{q_i}$ on $\mathcal{S}_i$ for all $i \in [1, c]$. This happens with probability at least $1 - 2\epsilon_1 cn^2$.

First note that $u \equiv_G v$ implies $\pi_i(u) \equiv_{G_i} \pi_i(v)$ for all $i \in [1, c]$. Since $\equiv_{G_i}$ refines $\equiv_{q_i}$ on $\mathcal{S}_i$, we obtain $\rho_i(q_i, \pi_i(u)) = \rho_i(q_i, \pi_i(v))$. It remains to show that after reading u and v, also the states of $\mathcal{B}_m$ are the same. For this we show that $f_{\bar{q}}(u) \equiv_H f_{\bar{q}}(v)$ in the graph group H.

From Lemma 8 it follows that there are reduced words $u', v' \in \Sigma^*$ such that $u \to_{sd}^* u'$, $v \to_{sd}^* v'$, and $u' \leftrightarrow_r^* v'$. Claim 10 implies $f_{\bar{q}}(u) \to_{sd}^* f_{\bar{q}}(u')$, $f_{\bar{q}}(v) \to_{sd}^* f_{\bar{q}}(v')$, and $\equiv_{G_i}$ refines $\equiv_{q_i}$ on $\mathcal{P}_i(u') \cup \mathcal{P}_i(v')$ for every $i \in [1, c]$. Since $u' \leftrightarrow_r^* v'$ we can write the block factorizations of u' and v' as $u' = u_1 u_2 \cdots u_l$ and $v' = v_1 v_2 \cdots v_l$ with $u_i, v_i \in \Sigma_{j_i}^+$ for some $j_i \in [1,c]$ and $u_i \equiv_{G_{j_i}} v_i$ for all $i \in [1,l]$. The block factorizations of $f_{\bar{q}}(u')$ and $f_{\bar{q}}(v')$ can be written as $f_{\bar{q}}(u') = \tilde{u}_1 \tilde{u}_2 \cdots \tilde{u}_l$ and $f_{\bar{q}}(v') = \tilde{v}_1 \tilde{v}_2 \cdots \tilde{v}_l$ with $\tilde{u}_i, \tilde{v}_i \in \{a_{j_i}, a_{j_i}^{-1}\}^+$.

We claim that $\tilde{u}_i \equiv_{\langle a_{j_i} \rangle} \tilde{v}_i$ for all $i \in [1,l]$, which implies $f_{\bar{q}}(u') \equiv_H f_{\bar{q}}(v')$. Since $u_i \equiv_{G_{j_i}} v_i$ for all $i \in [1,l]$, we get the following: if $u'' = u_{k_1} u_{k_2} \cdots u_{k_e} \in \Sigma_j^*$ is a pure prefix of u' for some $j \in [1,c]$ then $v'' = v_{k_1} v_{k_2} \cdots v_{k_e} \in \Sigma_j^*$ is a pure prefix of v' such that $u'' \equiv_{G_j} v''$. Since $\equiv_{G_j}$ refines $\equiv_{q_j}$ on $\mathcal{P}_j(u') \cup \mathcal{P}_j(v')$ and $u'', v'' \in \mathcal{P}_j(u') \cup \mathcal{P}_j(v')$, we obtain $\rho_j(q_j, u'') = \rho_j(q_j, v'')$. This implies $\tilde{u}_i \equiv_{\langle a_{j_i} \rangle} \tilde{v}_i$ for all $i \in [1,l]$ and hence $f_{\bar{q}}(u') \equiv_H f_{\bar{q}}(v')$. From this, we finally get $f_{\bar{q}}(u) \equiv_H f_{\bar{q}}(u') \equiv_H f_{\bar{q}}(v') \equiv_H f_{\bar{q}}(v)$.

Recall that $f_{\bar{q}}(u)$ (resp., $f_{\bar{q}}(v)$ is the word fed into the semiPFA $\mathcal{B}_m$ on input u (resp., v). Since $(\mathcal{B}_n)_{n \geq 0}$ is a $(1/n^d, 0)$-distinguisher for H, it follows that $f_{\bar{q}}(u)$ and $f_{\bar{q}}(v)$ lead in $\mathcal{B}_m$ with probability one to the same state. Hence, Algorithm 1 reaches with probability at least $1 - 2\epsilon_1 c n^2$ the same state when running on u and v, respectively.

Case 2. $u \not\equiv_G v$. We will show that Algorithm 1 reaches with probability at least $1-(2\epsilon_0 c n^2 + 1/n^d)$ different states when running on u and v, respectively. To show this, assume that the randomly selected initial states $q_i \in Q_i$ are such that $\equiv_{q_i}$ refines $\equiv_{G_i}$ on $\mathcal{S}_i$ for all $i \in [1,c]$. This happens with probability at least $1 - 2\epsilon_0 c n^2$.

We claim that $f_{\bar{q}}(u) \not\equiv_H f_{\bar{q}}(v)$. In order to get a contradiction, assume that $f_{\bar{q}}(u) \equiv_H f_{\bar{q}}(v)$. From Lemma 8 it follows that there are reduced words $\tilde{u}, \tilde{v} \in \Gamma^*$ such that $f_{\bar{q}}(u) \to_{sd}^* \tilde{u}$, $f_{\bar{q}}(v) \to_{sd} \tilde{v}$ and $\tilde{u} \leftrightarrow_r \tilde{v}$. By Claim 11 there are $u', v' \in \Sigma^*$ such that $u \to_{sd}^* u'$, $v \to_{sd}^* v'$, $f_{\bar{q}}(u') = \tilde{u}$, $f_{\bar{q}}(v') = \tilde{v}$ and $\equiv_{q_i}$ refines $\equiv_{G_i}$ on $\mathcal{P}_i(u') \cup \mathcal{P}_i(v')$ for every $i \in [1,c]$.

Since $f_{\bar{q}}(u') \leftrightarrow_r^* f_{\bar{q}}(v')$ we can write the block factorizations of $f_{\bar{q}}(u')$ and $f_{\bar{q}}(v')$ as $f_{\bar{q}}(u') = \tilde{u}_1 \tilde{u}_2 \cdots \tilde{u}_l$ and $f_{\bar{q}}(v') = \tilde{v}_1 \tilde{v}_2 \cdots \tilde{v}_l$ with $\tilde{u}_i, \tilde{v}_i \in \{a_{j_i}, a_{j_i}^{-1}\}^+$ for some $j_i \in [1,c]$ and $\tilde{u}_i \equiv_{\langle a_{j_i} \rangle} \tilde{v}_i$ for all $i \in [1,l]$. Clearly, the block factorizations of u' and v' can then be written as $u' = u_1 u_2 \cdots u_l$ and $v' = v_1 v_2 \cdots v_l$, where the block $u_i \in \Sigma_{j_i}^+$ (resp., $v_i \in \Sigma_{j_i}^+$) is translated into the block $\tilde{u}_i$ (resp., $\tilde{v}_i$) by the sequential transducer $\mathcal{T}_{\bar{q}}$.

We claim that $u_i \equiv_{G_{j_i}} v_i$ for all $i \in [1,l]$. Since $\tilde{u}_i \equiv_{\langle a_{j_i} \rangle} \tilde{v}_i$ for all $i \in [1,l]$ we have the following: if $\tilde{u}'' = \tilde{u}_{k_1} \tilde{u}_{k_2} \cdots \tilde{u}_{k_e} \in \{a_j, a_j^{-1}\}^*$ is a pure prefix of $f_{\bar{q}}(u')$ for some $j \in [1,c]$ then $\tilde{v}'' = \tilde{v}_{k_1} \tilde{v}_{k_2} \cdots \tilde{v}_{k_e} \in \{a_j, a_j^{-1}\}^*$ is a pure prefix of $f_{\bar{q}}(v')$ and $\tilde{u}'' \equiv_{\langle a_j \rangle} \tilde{v}''$. Let $p_j = \rho_j(q_j, u_{k_1} u_{k_2} \cdots u_{k_e})$ and $r_j = \rho_j(q_j, v_{k_1} v_{k_2} \cdots v_{k_e})$. We therefore have

$$a_j^{-q_j} a_j^{p_j} \equiv_{\langle a_j \rangle} \tilde{u}'' \equiv_{\langle a_j \rangle} \tilde{v}'' \equiv_{\langle a_j \rangle} a_j^{-q_j} a_j^{r_j},$$

i.e., $p_j = r_j$. Since $\equiv_{q_j}$ refines $\equiv_{G_j}$ on $\mathcal{P}_j(u') \cup \mathcal{P}_j(v')$ and $u_{k_1} u_{k_2} \cdots u_{k_e}$ as well as $v_{k_1} v_{k_2} \cdots v_{k_e}$ belong to $\mathcal{P}_j(u') \cup \mathcal{P}_j(v')$, we obtain $u_{k_1} u_{k_2} \cdots u_{k_e} \equiv_{G_j} v_{k_1} v_{k_2} \cdots v_{k_e}$. This holds for all pure prefixes of u'. We therefore have $u_i \equiv_{G_{j_i}} v_i$ for all $i \in [1,l]$, which implies $u' \equiv_G v'$. Finally, we get $u \equiv_G u' \equiv_G v' \equiv_G v$, which is a contradiction. Hence, we must have $f_{\bar{q}}(u) \not\equiv_H f_{\bar{q}}(v)$.

Since the algorithm feeds $f_{\bar{q}}(u)$ (resp., $f_{\bar{q}}(v)$) into the semiPFA $\mathcal{B}_m$, the latter reaches different states with probability at least $1 - 1/m^d \geq 1 - 1/n^d$ (under the assumption that $\equiv_{q_i}$ refines $\equiv_{G_i}$ on $\mathcal{S}_i$ for all $i \in [1,c]$). Hence, the probability that Algorithm 1 reaches different states when running on u and v is at least $(1 - 2\epsilon_0 c n^2)(1 - 1/n^d) \geq 1 - (2\epsilon_0 c n^2 + 1/n^d)$. ◀

Theorem 9 only makes sense if $\epsilon_0(n), \epsilon_1(n) < 1/2cn^2$. Most of the distinguishers from [17] have an error probability of $1/n^c$ for any chosen $c > 0$; see Theorem 7 for the case of f.g. linear groups. Moreover, notice that if $\epsilon_1 = 0$ then also $\zeta_1 = 0$ in Theorem 9. Combined with Theorem 7, Lemma 3, and the transfer theorems from [17] mentioned in Section 1, we get a logspace randomized streaming algorithm with a one-sided error (no error if $w \equiv_G 1$) for the word problem of any f.g. group G that can be constructed from f.g. linear groups using finite extensions, wreath products with a f.g. abelian left factor, and graph products. These groups are in general not linear; see the characterization of linear wreath products in [23].

6 Randomized streaming algorithms for subgroup membership

Let G be a f.g. group with a finite symmetric generating set Σ and let H be a subgroup of G. As before, $\pi_G : \Sigma^* \to G$ is the morphism that maps a word $w \in \Sigma^*$ to the group element represented by w. We define the language $\mathsf{GWP}(G, H, \Sigma) = \{w \in \Sigma^* : \pi_G(w) \in H\}$. GWP stands for generalized word problem which is another common name for the subgroup membership problem. Note that $\mathsf{GWP}(G, 1, \Sigma) = \mathsf{WP}(G, \Sigma)$. In the following we are interested in randomized streaming algorithms for $\mathsf{GWP}(G, H, \Sigma)$. One can easily show a statement for $\mathsf{GWP}(G, H, \Sigma)$ analogously to Lemma 2, which allows us to skip the generating set Σ in combination with the $\mathcal{O}$-notation and just write $\mathsf{GWP}(G, H)$ in the following.

For this section we fix a finite alphabet Σ and take a copy $\Sigma^{-1} = \{a^{-1} : a \in \Sigma\}$ of formal inverses. Let $\Gamma = \Sigma \cup \Sigma^{-1}$. We extend the mapping $a \mapsto a^{-1}$ ($a \in \Sigma$) to the whole alphabet Γ by setting $(a^{-1})^{-1} = a$. Moreover, for a word $w = a_1 a_2 \cdots a_n$ with $a_i \in \Gamma$ we define $w^{-1} = a_n^{-1} \cdots a_2^{-1} a_1^{-1}$. A word $w \in \Gamma^*$ is called reduced if it contains no factor of the form aa^{-1} for $a \in \Gamma$. Let $\mathsf{Red}(\Gamma) \subseteq \Gamma^*$ be the set of reduced words. It is convenient to identify the free group $F(\Sigma)$ with the set $\mathsf{Red}(\Gamma)$ of reduced words and the following multiplication operation: Let $u, v \in \mathsf{Red}(\Gamma)$. Then one can uniquely write u and v as $u = xy$ and $v = y^{-1}z$ such that $xz \in \mathsf{Red}(\Gamma)$ and define the product of u and v in the free group $F(\Sigma)$ as xz. For every word $w \in \Gamma^*$ we can define a unique reduced word $\mathsf{red}(w)$ as follows: if $w \in \mathsf{Red}(\Gamma)$ then $\mathsf{red}(w) = w$ and if $w = uaa^{-1}v$ for $u, v \in \Gamma^*$ and $a \in \Gamma$ then $\mathsf{red}(w) = \mathsf{red}(uv)$. It is important that this definition does not depend on which factor aa^{-1} is deleted in w. The reduction relation $uaa^{-1}v \to uv$ for all $u, v \in \Gamma^*$ and $a \in \Gamma$ is a so-called confluent relation. The reduction mapping $w \mapsto \mathsf{red}(w)$ then becomes the canonical morphism mapping a word $w \in \Gamma^*$ to the element of the free group represented by w.

In the following we have to deal with a special class of finite automata over the alphabet Γ. A partial deterministic finite automaton (partial DFA) is defined as an ordinary DFA except that the transition function $\delta : Q \times \Gamma \to Q$ is only partially defined. As for (total) DFAs we extend the partial transition function $\delta : Q \times \Gamma \to Q$ to a partial function $\delta : Q \times \Gamma^* \to Q$. For $q \in Q$ and $w \in \Gamma^*$ we write $\delta(q, w) = \bot$ if $\delta(q, w)$ is undefined, which means that one cannot read the word w into the automaton $\mathcal{A}$ starting from state q. A *partial inverse automaton* $\mathcal{A} = (Q, \Gamma, q_0, \delta, q_f)$ over the alphabet Γ is a partial DFA with a single final state q_f and such that for all $p, q \in Q$ and $a \in \Gamma$, $\delta(p, a) = q$ implies $\delta(q, a^{-1}) = p$.

The main technique to deal with f.g. subgroups of a free group is Stallings folding [12]. For our purpose it suffices to know that for every f.g. subgroup $G \leq F(\Sigma)$ there exists a partial inverse automaton $\mathcal{A}_G$ over the alphabet Γ such that for every $w \in \mathsf{Red}(\Gamma)$, $w \in G$ if and only if $w \in L(\mathcal{A}_G)$. We call $\mathcal{A}_G$ the Stallings automaton for G. It has the additional property that the unique final state is the initial state. The Stallings automaton for G can be constructed quite efficiently from a given set of generators for G, but we do not need this fact since G will be fixed and not considered to be part of the input in our main result, Theorem 14 below.

Let G be a fixed f.g. subgroup of $F(\Sigma)$ and let $\mathcal{A}_G = (Q, \Gamma, q_0, \delta, q_0)$ be its Stallings automaton in the following. An important property of $\mathcal{A}_G$ is the following: If $q, q' \in Q$ and $u \in \Gamma^*$ (u is not necessarily reduced) are such that $\delta(q, u) = q'$ then also $\delta(q, \mathsf{red}(u)) = q'$. This follows from the fact that $\delta(q, aa^{-1}) = q$ for every $q \in Q$ and $a \in \Gamma$. In particular, if $\delta(q_0, u) \neq \bot$, then $u \in L(\mathcal{A}_G)$ if and only if $\mathsf{red}(u) \in L(\mathcal{A}_G)$ if and only if $\mathsf{red}(u) \in G$.

▶ **Definition 12.** *For a word $w \in \Gamma^*$ we define the $\mathcal{A}_G$-factorization of w uniquely as either*

(i) $w = w_0 a_1 u_1 \; w_1 a_2 u_2 \cdots w_{k-1} a_k u_k \; w_k$ *or*

(ii) $w = w_0 a_1 u_1 \; w_1 a_2 u_2 \cdots w_{k-1} a_k u_k \; w_k a_{k+1} v$

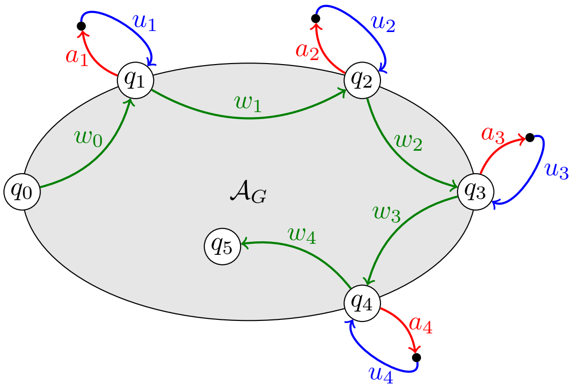

Figure 1 An $\mathcal{A}_G$-factorization of type (i) for $k = 4$. The red-blue loops outside of $\mathcal{A}_G$ are loops in the Cayley-graph of the free group $F(\Sigma)$.

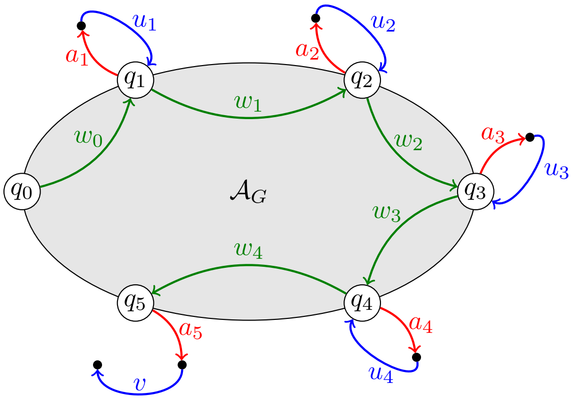

Figure 2 An $\mathcal{A}_G$-factorization of type (ii) for $k = 4$. The red-blue loops outside of $\mathcal{A}_G$ are loops in the Cayley-graph of the free group $F(\Sigma)$.

such that the following properties hold, where $\ell = k$ in case (i) and $\ell = k + 1$ in case (ii):

- $w_0, \ldots, w_k, u_1, \ldots, u_k, v \in \Gamma^*$, $a_1, \ldots, a_\ell \in \Gamma$,
- *there are states $q_1, \ldots, q_{k+1} \in Q$ such that $\delta(q_i, w_i) = q_{i+1}$ for all $i \in [0, k]$ (recall that q_0 is the initial state of $\mathcal{A}_G$),*
- $\delta(q_i, a_i) = \bot$ *for all* $i \in [1, \ell]$,
- *for all $i \in [1, k]$, $\mathsf{red}(a_i u_i) = \varepsilon$ but there is no prefix $u \neq u_i$ of u_i with $\mathsf{red}(a_i u) = \varepsilon$, and*
- *in case (ii), v has no prefix x with $\mathsf{red}(a_{k+1} x) = \varepsilon$.*

Depending on which of the two cases (i) and (ii) in Definition 12 holds, we say that w has an $\mathcal{A}_G$-factorization of type (i) or type (ii).

Let us explain the intuition of the $\mathcal{A}_G$-factorization of w; see also Figures 1 and 2. We start reading the word w into the automaton $\mathcal{A}_G$, beginning at q_0, as long as possible. If it turns out that $\delta(q_0, w)$ is defined, then the $\mathcal{A}_G$-factorization of w consists of the single factor $w_0 = w$ and we obtain type (i). Otherwise, there is a shortest prefix w_0 of w (the first factor of the $\mathcal{A}_G$-factorization) such that after reading w_0 we reach the state $\delta(q_0, w_0) = q_1$ of $\mathcal{A}_G$ and $\delta(q_1, a_1) = \bot$, where a_1 is the symbol following w_0 in w. In other words, when trying

Algorithm 2 $1/n^c$-correct randomized streaming algorithm for $\mathsf{GWP}(F(\Sigma), G)$

```
global variables: $q \in Q$, $p, r \in R_n$, $\beta \in \{0,1\}$
initialization:
$q := q_0$ ; $\beta := 1$ ;
guess $r \in R_n$ according to the initial state distribution $\lambda_n$ of $\mathcal{B}_n$ ;
next input letter: $a \in \Gamma$
if $\beta = 1$ and $\delta(q,a) = \perp$ then
    $\beta := 0$ ; $p := r$
end
if $\beta = 1$ and $\delta(q,a) \neq \perp$ then
    $q := \delta(q,a)$
end
$r := \sigma_n(r,a)$ ;
if $\beta = 0$ and $r = p$ then
    $\beta := 1$
end
accept if $\beta = 1$ and $q = q_0$
```

to read a_1, we escape the automaton $\mathcal{A}_G$ for the first time. At this point let $w = w_0 a_1 x$. We then take the shortest prefix u_1 of x such that $a_1 u_1 = 1$ in $F(\Sigma)$ (if such a prefix does not exist, we terminate in case (ii)). This yields a new factorization $w = w_0 a_1 u_1 y$. We then repeat this process with the word y starting from the state q_1 as long as possible. There are two possible terminations of the process: starting from state q_k we can read the whole remaining suffix into $\mathcal{A}_G$ (and arrive in state q_{k+1}). This suffix then yields the last factor w_k and we obtain (i). In the other case, we leave the automaton $\mathcal{A}_G$ with the symbol a_{k+1} from state q_{k+1} ($\delta(q_{k+1}, a_{k+1}) = \perp$) and the remaining suffix has no prefix x such that $a_{k+1}x$ evaluates to the identity in $F(\Sigma)$. The remaining suffix then yields the last factor v and we obtain (ii). The following lemma is shown in [17].

▶ **Lemma 13.** *Let $w \in \Gamma^*$ and assume that the $\mathcal{A}_G$-factorization of w and the states $q_1, \ldots, q_{k+1}$ are as in Definition 12.*

- *If the $\mathcal{A}_G$-factorization of w is of type (i) then $\mathsf{red}(w) \in G$ if and only if $q_{k+1} = q_0$.*
- *If the $\mathcal{A}_G$-factorization of w is of type (ii) then $\mathsf{red}(w) \notin G$.*

▶ **Theorem 14.** *Let G a fixed f.g. subgroup of $F(\Sigma)$. For every $c > 0$ there is a $1/n^c$-correct randomized streaming algorithm for $\mathsf{GWP}(F(\Sigma), G)$ with space complexity $\mathcal{O}(\log n)$.*

Proof. We would like to use the Stallings automaton $\mathcal{A}_G = (Q, \Gamma, q_0, \delta, q_0)$ as a streaming algorithm for $\mathsf{GWP}(F(\Sigma), G)$ (note that $|Q|$ is a constant since G is fixed). The problem is that the input word is not necessarily reduced. We solve this problem by using a $(1/n^{c+2}, 0)$-distinguisher $(\mathcal{B}_n)_{n \geq 0}$ for $F(\Sigma)$ with space complexity $\mathcal{O}(\log n)$. It exists by Theorem 7 since f.g. free groups are linear. Fix an input length n and let $\mathcal{B}_n = (R_n, \Gamma, \lambda_n, \sigma_n)$. Consider an input word $w \in \Gamma^{\leq n}$. Our randomized streaming algorithm for $\mathsf{GWP}(F(\Sigma), G)$ is Algorithm 2.

The space needed by Algorithm 2 is $\mathcal{O}(\log n)$: The variables q and β need constant space and p and r both need $\mathcal{O}(\log n)$ bits. Let us now show that the error probability of Algorithm 2 is bounded by $1/n^c$. For this let $S = \mathcal{P}(w)$ be the set of all prefixes of w. For the initially guessed state $r_0 \in R_n$ (line 3) we have by Lemma 4:

$$\mathop{\mathsf{Prob}}_{r_0 \in R_n}[\equiv_{F(\Sigma)} \text{ equals } \equiv_{r_0} \text{ on } S] \geq 1 - 1/n^{c+2}\binom{|S|}{2} \geq 1 - 1/n^c.$$

Assume for the further consideration that the guessed state r_0 is such that $\equiv_{F(\Sigma)}$ and $\equiv_{r_0}$ are equal on S. We claim that under this assumption, Algorithm 2 accepts in line 13 after reading w if and only if $\mathsf{red}(w) \in G$. For this, assume that the $\mathcal{A}_G$-factorization of w and the states $q_1, \ldots, q_{k+1} \in Q$ are as in Definition 12. By Lemma 13 it suffices to show the following:

(a) If the $\mathcal{A}_G$-factorization of w is of type (i) then after reading w we have $\beta = 1$ and $q = q_{k+1}$ in Algorithm 2.

(b) If the $\mathcal{A}_G$-factorization of w is of type (ii) then $\beta = 0$ after reading w in Algorithm 2.

To see this, observe that Algorithm 2 simulates $\mathcal{B}_n$ on w starting from r_0 (line 9). Moreover, initially we have $\beta = 1$ (line 1). This implies that Algorithm 2 simulates the Stallings automaton $\mathcal{A}_G$ on w as long as possible (line 7). If this possible for the whole input w (i.e., $\delta(q_0, w) \neq \bot$) then w has an $\mathcal{A}_G$-factorization of type (i) consisting of the single factor w (i.e., $k = 0$). Moreover, after processing w by Algorithm 2, we have $\beta = 1$ and the program variable q holds $\delta(q_0, w) = q_1 = q_{k+1}$. We obtain the above case (a).

Assume now that $k > 0$. The $\mathcal{A}_G$-factorization of w starts with $w_0 a_1$, where $\delta(q_0, w_0) = q_1$ and $\delta(q_1, a_1) = \bot$. After processing w_0 by Algorithm 2 we have $q = q_1$ and $r = \sigma_n(r_0, w_0)$. While processing the next letter a_1, Algorithm 2 sets β to 0 and saves the current state $r = \sigma_n(r_0, w_0)$ of $\mathcal{B}_n$ in the variable p (line 4). Let us write $w = w_0 a_1 v$. Since the flag β was set to 0, Algorithm 2 only continues the simulation of $\mathcal{B}_n$ on input $a_1 v$ starting from state $\sigma_n(r_0, w_0) = p$. Our assumption that $\equiv_{F(\Sigma)}$ and $\equiv_{r_0}$ are equal on the set S implies that for every prefix x of v we have: $\mathsf{red}(a_1 x) = \varepsilon$ if and only if $p = \sigma_n(r_0, w_0) = \sigma_n(r_0, w_0 a_1 x)$. In line 10, the algorithm checks the latter equality in each step (as long as $\beta = 0$). If there is no prefix x of v with $\mathsf{red}(a_1 x) = \varepsilon$ then the $\mathcal{A}_G$-factorization of w is of type (ii) (it is $w_0 a_1 v$) and the flag β is 0 after reading w. We then obtain the above case (b). Otherwise, u_1 is the shortest prefix of v with $\mathsf{red}(a_1 u_1) = \varepsilon$. Moreover, after processing u_1, the if-condition in line 10 is true for the first time. The algorithm then sets the flag β back to 1 (line 11) and resumes the simulation of the automaton $\mathcal{A}_G$ in state q_1 (which is still stored in the program variable q). This process now repeats and we see that the algorithm correctly locates the factors of the $\mathcal{A}_G$-factorization of w. This shows the above points (a) and (b). ◀

It is not possible to generalize Theorem 14 to subgroups of $F(\Sigma)$ that are not finitely generated. The proof of the following theorem (see [17]) uses the fact there is a finitely presented group whose word problem has randomized streaming space complexity $\Omega(n)$. A concrete example is Thompson's group F; see [16, Corollary 22].

▶ **Theorem 15.** *The free group F_2 of rank two has a normal subgroup N such that the randomized streaming space complexity of the language* $\mathsf{GWP}(F_2, N)$ *is in $\Theta(n)$.*

Applying Mihaĭlova's construction [19] to the normal subgroup N from Theorem 15 yields:

▶ **Theorem 16.** *There is a f.g. subgroup G of $F_2 \times F_2$ such that the randomized streaming space complexity of* $\mathsf{GWP}(F_2 \times F_2, G)$ *is in $\Theta(n)$.*

7 Open problems

A very important class of groups in geometric group theory is the class of hyperbolic groups; see [9] for background. Free groups are the simplest hyperbolic groups. Hyperbolic groups have some good algorithmic properties. For instance, their word problems can be decided in linear time by Dehn's algorithm. It would be interesting to know whether every hyperbolic group has an ϵ-distinguisher (say for $\epsilon = 1/3$) with space complexity $\mathcal{O}(\log n)$, which would imply that every hyperbolic group has randomized streaming space complexity $\mathcal{O}(\log n)$. One should remark that it is also open whether the word problem for a hyperbolic group belongs to randomized logspace (RL). The best known space upper bound for the word problem of a hyperbolic group is $\mathsf{DPSPACE}(\log^2 n)$. This follows from the fact that the word problem of a hyperbolic group belongs to LogCFL [14].

It would be also interesting to see whether a transfer theorem similar to Theorem 9 can be shown for certain fundamental groups of graphs of groups, e.g. when all edge groups are finite.

References

1 Ajesh Babu, Nutan Limaye, Jaikumar Radhakrishnan, and Girish Varma. Streaming algorithms for language recognition problems. *Theoretical Computer Science*, 494:13–23, 2013. doi:10.1016/j.tcs.2012.12.028.

2 Gabriel Bathie and Tatiana Starikovskaya. Property testing of regular languages with applications to streaming property testing of visibly pushdown languages. In *Proceedings of the 48th International Colloquium on Automata, Languages, and Programming, ICALP 2021*, volume 198 of *LIPIcs*, pages 119:1–119:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.119.

3 J. Berstel. *Transductions and context–free languages.* Teubner Studienbücher, Stuttgart, 1979. doi:10.1007/978-3-663-09367-1.

4 William W. Boone. The word problem. *Annals of Mathematics. Second Series*, 70:207–265, 1959. doi:10.2307/1970103.

5 Max Dehn. Über unendliche diskontinuierliche Gruppen. *Mathematische Annalen*, 71:116–144, 1911. In German. doi:10.1007/BF01456932.

6 Volker Diekert and Jonathan Kausch. Logspace computations in graph products. *Journal of Symbolic Computation*, 75:94–109, 2016. doi:10.1016/j.jsc.2015.11.009.

7 Nathanaël François, Frédéric Magniez, Michel de Rougemont, and Olivier Serre. Streaming property testing of visibly pushdown languages. In *Proceedings of the 24th Annual European Symposium on Algorithms, ESA 2016*, volume 57 of *LIPIcs*, pages 43:1–43:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ESA.2016.43.

8 Elisabeth R. Green. *Graph Products of Groups.* PhD thesis, The University of Leeds, 1990.

9 Derek F. Holt, Sarah Rees, and Claas E. Röver. *Groups, Languages and Automata*, volume 88 of *London Mathematical Society Student Texts.* Cambridge University Press, 2017. doi:10.1017/9781316588246.

10 Tim Hsu and Daniel T. Wise. On linear and residual properties of graph products. *Michigan Mathematical Journal*, 46(2):251–259, 1999. doi:10.1307/mmj/1030132408.

11 Stephen P. Humphries. On representations of Artin groups and the Tits conjecture. *Journal of Algebra*, 169(3):847–862, 1994. doi:10.1006/jabr.1994.1312.

12 Ilya Kapovich and Alexei Myasnikov. Stallings foldings and subgroups of free groups. *Journal of Algebra*, 248(2):608–668, 2002. doi:10.1006/jabr.2001.9033.

13 Jonathan Kausch. *The parallel complexity of certain algorithmic problems in group theory.* PhD thesis, University of Stuttgart, 2017. URL: 10.18419/opus-9152.

14 Markus Lohrey. Decidability and complexity in automatic monoids. *International Journal of Foundations of Computer Science*, 16(4):707–722, 2005. doi:10.1142/S0129054105003248.

15 Markus Lohrey. *The Compressed Word Problem for Groups.* SpringerBriefs in Mathematics. Springer, 2014. doi:10.1007/978-1-4939-0748-9.

16 Markus Lohrey and Lukas Lück. Streaming word problems. In *Proceedings of the 47th International Symposium on Mathematical Foundations of Computer Science, MFCS 2022*, volume 241 of *LIPIcs*, pages 72:1–72:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.MFCS.2022.72.

17 Markus Lohrey, Lukas Lück, and Julio Xochitemol. Streaming word problems. *CoRR*, abs/2202.04060, 2022. URL: https://arxiv.org/abs/2202.04060.

18 Frédéric Magniez, Claire Mathieu, and Ashwin Nayak. Recognizing well-parenthesized expressions in the streaming model. *SIAM Journal on Computing*, 43(6):1880–1905, 2014. doi:10.1137/130926122.

19 K. A. Mihaĭlova. The occurrence problem for direct products of groups. *Math. USSR Sbornik*, 70:241–251, 1966. English translation.

20 Pjotr S. Novikov. On the algorithmic unsolvability of the word problem in group theory. *American Mathematical Society, Translations, II. Series*, 9:1–122, 1958. doi:10.1090/trans2/009.

21 Azaria Paz. *Introduction to Probabilistic Automata*. Academic Press, 1971. doi:10.1016/C2013-0-11297-4.

22 Michael O. Rabin. Probabilistic automata. *Information and Control*, 6(3):230–245, 1963. doi:10.1016/S0019-9958(63)90290-0.

23 B. A. F. Wehrfritz. Wreath products and chief factors of linear groups. *Journal of the London Mathematical Society (2)*, 4:671–681, 1972. doi:10.1112/jlms/s2-4.4.671.