# Groups with ALOGTIME-hard word problems and PSPACE-complete compressed word problems

LAURENT BARTHOLDI, Saarland University, Germany
MICHAEL FIGELIUS*, Universität Siegen, Germany
MARKUS LOHREY†, Universität Siegen, Germany
ARMIN WEISS‡, Universität Stuttgart, Germany

We give lower bounds on the complexity of the word problem for a large class of non-solvable infinite groups that we call strongly efficiently non-solvable (SENS) groups. This class includes free groups, Grigorchuk's group and Thompson's groups. We prove that these groups have an $NC^1$-hard word problem and that for some of them (including Grigorchuk's group and Thompson's groups) the compressed word problem (which is equivalent to the circuit evaluation problem) is PSPACE-complete.

CCS Concepts: • **Theory of computation → Algebraic complexity theory**; **Circuit complexity**; • **Mathematics of computing → Combinatorics**.

Additional Key Words and Phrases: $NC^1$-hardness, word problem, $G$-programs, straight-line programs, non-solvable groups, self-similar groups, Thompson's groups, Grigorchuk's group

**ACM Reference Format:**
Laurent Bartholdi, Michael Figelius, Markus Lohrey, and Armin Weiß. 2018. Groups with ALOGTIME-hard word problems and PSPACE-complete compressed word problems. 1, 1 (October 2018), 41 pages. https://doi.org/XXXXXXX.XXXXXXX

## ACKNOWLEDGMENTS

The authors are grateful to Schloss Dagstuhl and the organizers of Seminar 19131 for the invitation, where this work began.

## 1 INTRODUCTION

The *word problem* of a finitely generated group $G$ is the most fundamental algorithmic problem in group theory: given a word over the generators of $G$, the question is whether this word represents the identity of $G$. The original motivation for the word problem came from topology and group theory [16], within Hilbert's "Entscheidungsproblem". Nevertheless, it also played a role in early computer science when Novikov and Boone constructed finitely presented groups with an undecidable word problem [11, 52]. Still, in many classes of groups it is (efficiently) decidable, a prominent example

*Funded by DFG project LO 748/12-1.
†Funded by DFG project LO 748/12-1.
‡Funded by DFG project DI 435/7-1 and WE 6835/1-2.

Authors' addresses: Laurent Bartholdi, laurent.bartholdi@gmail.com, Saarland University, Saarbrücken, Germany; Michael Figelius, Universität Siegen, Germany, figelius@eti.uni-siegen.de; Markus Lohrey, Universität Siegen, Germany, lohrey@eti.uni-siegen.de; Armin Weiß, Universität Stuttgart, Germany, armin.weiss@fmi.uni-stuttgart.de.

being the class of linear groups: Lipton and Zalcstein [43] (for linear groups over a field of characteristic zero) and Simon [57] (for linear groups over a field of prime characteristic) showed that their word problem is in LOGSPACE.

The class $\mathsf{NC}^1$ consists of those languages that are accepted by families of boolean circuits of logarithmic depth. When combined with appropriate uniformity conditions it yields the subclass ALOGTIME, which is contained in LOGSPACE – so it is a very small complexity class of problems efficiently solvable in parallel. A striking connection between the word problem for groups and complexity theory was established by Barrington [4]: for every finite non-solvable group $G$, the word problem of $G$ is $\mathsf{NC}^1$-complete. Moreover, the reduction is as simple as it could be: every output bit depends on only one input bit. Thus, one can say that $\mathsf{NC}^1$ is completely characterized via group theory. Moreover, this idea has been extended to characterize $\mathsf{ACC}^0$ by solvable monoids [5]. On the other hand, the word problem of a finite $p$-group is in $\mathsf{ACC}^0[p]$, so Smolensky's lower bound [58] implies that it is strictly easier than the word problem of a finite non-solvable group.

Barrington's construction is based on the observation that an and-gate can be simulated by a commutator. This explains the connection to non-solvability. In this light it seems natural that the word problem of finite $p$-groups is not $\mathsf{NC}^1$-hard: they are all nilpotent, so iterated commutators eventually become trivial. For infinite groups, a construction similar to Barrington's was used by Robinson [54] to show that the word problem of a non-abelian free group is $\mathsf{NC}^1$-hard. Since by [43] the word problem of a free group is in LOGSPACE, the complexity is narrowed down quite precisely (although no completeness result has been shown so far).

The first contribution of this paper is to identify the essence of Barrington's and Robinson's constructions. For this we introduce a strengthened condition of non-solvability, which we call *SENS (strongly efficiently non-solvable)*; see Definition 5.1. In a SENS group there are balanced nested commutators of arbitrary depth and whose word lengths grow at most exponentially. We also introduce *uniformly SENS* groups, where these balanced commutators are efficiently computable in a certain sense. We then follow Barrington's arguments and show that for every (uniformly) SENS group the word problem is hard for (uniform) $\mathsf{NC}^1$ (Theorems 6.1 and 6.3). This does not exclude the possibility that there is a non-solvable group $G$ whose word problem is not hard for (non-uniform) $\mathsf{NC}^1$, but it means that for such a group the word lengths of the $G$-elements witnessing the non-solvability must grow super-exponentially. We give in Example 5.11 a non-solvable group in which the latter happens.

Finite non-solvable groups and non-abelian free groups are easily seen to be uniformly SENS. We go beyond these classes and present a general criterion that implies the uniform SENS-condition. Using this criterion we show that *Thompson's groups* [13] and *weakly branched self-similar groups* [7, 51] are uniformly SENS. As a corollary we get:

**Corollary A.** *The word problems for the following groups are hard for* ALOGTIME*:*

- *the three Thompson's groups $F$, $T$, and $V$,*
- *weakly branched self-similar groups with a finitely generated branching subgroup.*

Thompson's groups $F < T < V$ (introduced in 1965) belong due to their unusual properties to the most intensively studied infinite groups. From a computational perspective it is interesting to note that all three Thompson's groups are co-context-free (i.e., the set of all non-trivial words over any set of generators is a context-free language) [40]. This implies that the word problems for Thompson's groups are in LOGCFL. To the best of our knowledge no better upper complexity bound is known. Weakly branched groups form an important subclass of the self-similar groups [51], containing several celebrated groups like the Grigorchuk group (the first example of a group with intermediate word growth) and the Gupta-Sidki groups. We also show that the word problem for contracting self-similar groups is in

LOGSPACE. This result is well-known, but to the best of our knowledge no explicit proof appears in the literature. The Grigorchuk group as well as the Gupta-Sidki groups are contracting and have finitely generated branching subgroups.

Another corollary of Theorem 6.3 is the following dichotomy result for finitely generated linear groups: for every finitely generated linear group the word problem is either in DLOGTIME-uniform $\mathsf{TC}^0$ or ALOGTIME-hard (Theorem 6.4). To prove this we use the Tits alternative (every finitely generated linear group either contains a free group of rank two or is virtually solvable) [59] together with a result from [38] stating that the word problem for a finitely generated solvable linear group is in DLOGTIME-uniform $\mathsf{TC}^0$.

In the second part of the paper we study the *compressed word problem* [46]. This is a succinct version of the word problem, where the input word is represented by a so-called straight-line program. A straight-line program is a context-free grammar that produces exactly one string. The length of this string can be exponentially larger than the size of the straight-line program. The compressed word problem for a finitely generated group $G$ is equivalent to the *circuit evaluation problem* for $G$. In the latter the input is a circuit where the input gates are labelled with generators of $G$ and the internal gates compute the product of their inputs. There is a distinguished output gate, and the question is whether this output gate evaluates to the group identity. For finite groups (and also monoids), the circuit evaluation problem has been studied in [9]. The circuit viewpoint also links the compressed word problem to the famous polynomial identity testing problem (the question whether an algebraic circuit over a polynomial ring evaluates to the zero-polynomial); see [56] for a survey: the compressed word problem for the group $\mathrm{SL}_3(\mathbb{Z})$ is equivalent to the polynomial identity testing problem with respect to polynomial time reductions [46, Theorem 4.16].

From a group theoretic viewpoint, the compressed word problem is interesting not only because group elements are naturally represented as straight line programs, but also because several classical (uncompressed) word problems reduce to compressed word problems. For instance, the word problem for a finitely generated subgroup of $\mathrm{Aut}(G)$ reduces to the compressed word problem for $G$ [46, Theorem 4.6]. Similar statements hold for certain group extensions [46, Theorems 4.8 and 4.9]. This motivates the search for groups in which the compressed word problem can be solved efficiently. For the following groups, the compressed word problem can be solved in polynomial time: finitely generated nilpotent groups [38] (for which the compressed word problem can be even solved in $\mathsf{NC}^2$), hyperbolic groups [34] (more generally, groups that are hyperbolic relative to a collection of free abelian subgroups [33]) and virtually special groups [46]. The latter are defined as finite extensions of subgroups of right-angled Artin groups and form a very rich class of groups containing for instance Coxeter groups [25], fully residually free groups [64] and fundamental groups of hyperbolic 3-manifolds [2]. Moreover, for finitely generated linear groups the compressed word problem belongs to coRP (complement of randomized polynomial time).

In this paper, we are mainly interested in lower bounds for compressed word problems. It is known that the compressed word problem for non-solvable finite groups and non-abelian free groups is P-complete [9, 44]. The proofs for these results use again the above mentioned constructions of Barrington and Robinson. For wreath products of the form $G \wr \mathbb{Z}$ with $G$ non-abelian the compressed word problem is coNP-hard [46, Theorem 4.21]. Moreover, recently, Wächter and the fourth author constructed an automaton group (a finitely generated group of tree automorphisms, where the action of generators is defined by a Mealy automaton) with a PSPACE-complete word problem and EXPSPACE-complete compressed word problem [62] – thus, the compressed word problem is provably more difficult than the word problem. The group arises from a quite technical construction; in particular, one cannot call this group natural. Here, we exhibit several natural groups (that were intensively studied in other parts of mathematics) with a PSPACE-complete compressed word problem and a word problem in LOGSPACE:

**Corollary B.** *The compressed word problem for the following groups is* PSPACE*-complete:*

- *wreath products $G \wr \mathbb{Z}$ where $G$ is finite non-solvable or free of rank at least two,*
- *Thompson's groups,*
- *the Grigorchuk group, and*
- *all Gupta-Sidki groups.*

To get the first point, we completely characterize the complexity of the compressed word problem for a wreath product $G \wr \mathbb{Z}$, where $G$ has a trivial center, in terms of the leaf language class defined by the word problem of $G$; see Theorem 8.2. This characterization implies that the compressed word problem for $G \wr \mathbb{Z}$ with $G$ a uniformly SENS group is PSPACE-hard. To get PSPACE-hardness of the compressed word problems for Thompson's groups, the Grigorchuk group, and the Gupta-Sidki groups we use a self-embedding property: each of these groups $G$ has the property that it contains a copy of a wreath product $G \wr A$ for some $A \neq 1$. Thompson's group $F$ has this property for $A = \mathbb{Z}$ [23]. For the Grigorchuk group, the Gupta-Sidki groups and more generally all weakly branched groups $G$ that satisfy an additional technical condition (the branching subgroup $K$ of $G$ is finitely generated and has elements of finite order) we show that one can take $A = \mathbb{Z}/p$ for some $p \geq 2$. Based on Theorem 8.2 we show that every group $G$ with the property that $G \wr A \leq G$ for some non-trivial $A$ has a PSPACE-hard compressed word problem (Theorem 9.6).

### 1.1 Related work.

Uniformly SENS groups were used in the recent paper [17] in the context of the power word problem and knapsack problem. In the power word problem for a finitely generated group $G$ [48], the input is an expression of the form $w_1^{z_1} w_2^{z_2} \cdots w_n^{z_n}$, where the $w_i$ are words over the generators of $G$ and the exponents $z_i$ are binary encoded integers, and it is asked whether $w_1^{z_1} w_2^{z_2} \cdots w_n^{z_n} = 1$ in $G$. It is shown in [17] that the power word problem for a wreath product $G \wr \mathbb{Z}$ with $G$ uniformly SENS is coNP-hard. It follows that Thompson's group $F$ has a coNP-complete power word problem. In addition it is shown in [17] that the so-called knapsack problem for a wreath product $G \wr \mathbb{Z}$ with $G$ uniformly SENS is hard for $\Sigma_2^p$ (second existential level of the polynomial time hierarchy). In the knapsack problem for a group $G$ the question is whether an equation $w_1^{x_1} w_2^{x_2} \cdots w_n^{x_n} = w$, where the $x_i$ are variables, has a solution in the natural numbers.

This work is the full version of the conference paper [6]. Here we give full proofs and some additional details; in particular, we present an example of a non-solvable group which is not SENS.

## 2 GENERAL NOTATIONS

For $a, b \in \mathbb{Z}$ we write $[a..b]$ for the interval $\{z \in \mathbb{Z} \mid a \leq z \leq b\}$. We use common notations from formal language theory. In particular, we use $\Sigma^*$ to denote the set of words over an alphabet $\Sigma$ including the *empty word* $\varepsilon$. Let $w = a_0 \cdots a_{n-1} \in \Sigma^*$ be a word over $\Sigma$ ($n \geq 0$, $a_0, \ldots, a_{n-1} \in \Sigma$). The *length* of $w$ is $|w| = n$. We write $\Sigma^{\leq d}$ for $\{w \in \Sigma^* \mid |w| \leq d\}$ and $\Sigma^{<d}$ for $\{w \in \Sigma^* \mid |w| < d\}$. For a letter $a \in \Sigma$ let $|w|_a = |\{i \mid a = a_i\}|$ be the number of occurrences of $a$ in $w$. For $0 \leq i < n$ let $w[i] = a_i$ and for $0 \leq i \leq j < n$ let $w[i : j] = a_i a_{i+1} \cdots a_j$. Moreover $w[: i] = w[0 : i]$. Note that in the notations $w[i]$ and $w[i : j]$ we take 0 as the first position in $w$. This will be convenient later.

The lexicographic order on $\mathbb{N}^*$ is defined as follows: a word $u \in \mathbb{N}^*$ is lexicographically smaller than a word $v \in \mathbb{N}^*$ if either $u$ is a prefix of $v$ or there exist $w, x, y \in \mathbb{N}^*$ and $i, j \in \mathbb{N}$ such that $u = wix$, $v = wjy$, and $i < j$.

A *finite ordered tree* is a finite set $T \subseteq \mathbb{N}^*$ such that for all $w \in \mathbb{N}^*$, $i \in \mathbb{N}$: if $wi \in T$, then $w, wj \in T$ for every $0 \leq j < i$. The set of *children* of $u \in T$ is $u\mathbb{N} \cap T$. A node $u \in T$ is a leaf of $T$ if it has no children. A *complete binary tree* is a subset $T \subseteq \{0,1\}^*$ such that $T = \{s \in \{0,1\}^* \mid |s| \leq k\}$ for some $k \geq 0$ where $k$ is called the *depth* of $T$.

The boolean function nand : $\{0,1\}^2 \to \{0,1\}$ (negated and) is defined by $\mathsf{nand}(0,0) = \mathsf{nand}(0,1) = \mathsf{nand}(1,0) = 1$ and $\mathsf{nand}(1,1) = 0$. Note that the standard boolean functions not and binary and and or can be expressed in terms of nand.

## 3 GROUPS

We assume that the reader is familiar with the basics of group theory, see e.g. [35, 55] for more details. Let $G$ be a group. We always write 1 for the group identity element. The group $G$ is called *finitely generated* if there exist a finite set $S$ and a surjective homomorphism of the free group over $S$ onto $G$. In this situation, the set $\Sigma = S \cup S^{-1} \cup \{1\}$ is our preferred generating set for $G$ and we have a surjective monoid homomorphism $\pi\colon \Sigma^* \to G$. The symbol 1 is useful for padding. We call the generating set $\Sigma$ *standard*. We have a natural involution on words over $\Sigma$ defined by $(a_1 \cdots a_n)^{-1} = a_n^{-1} \cdots a_1^{-1}$ for $a_i \in \Sigma$ (which is the same as forming inverses in the group). For words $u, v \in \Sigma^*$ we usually say that $u = v$ in $G$ or $u =_G v$ in case $\pi(u) = \pi(v)$. For group elements $g, h \in G$ or words $g, h \in \Sigma^*$ we write $g^h$ for the *conjugate* $h^{-1}gh$ and $[h,g]$ for the *commutator* $h^{-1}g^{-1}hg$. We call $g$ a *$d$-fold nested commutator*, if $d = 0$ or $g = [h_1, h_2]$ for $(d-1)$-fold nested commutators $h_1, h_2$.

A *subquotient* of $G$ is a quotient of a subgroup of $G$. The *center* of $G$, $Z(G)$ for short, is the set of all elements $g \in G$ that commute with every element from $G$. The center of $G$ is a normal subgroup of $G$.

The *word problem* for the finitely generated group $G$, $\mathrm{WP}(G)$ for short, is defined as follows:

**Input:** a word $w \in \Sigma^*$.
**Question:** does $w =_G 1$ hold?

We will also write $\mathrm{WP}(G, \Sigma)$ for the set $\{w \in \Sigma^* \mid w =_G 1\}$.

The word problem may be stated for any group whose elements may be written as words over a finite alphabet. This applies to subquotients $H/K$ of $G$ (also if $H$ is not finitely generated): given a word $w \in \Sigma^*$ with the guarantee that it belongs to $H$, does it actually belong to $K$? Note that the decidability of this problem depends on the actual choice of $H$ and $K$, not just on the isomorphism type of $H/K$.

We will consider groups $G$ that act on a set $X$ on the left or right. For $g \in G$ and $x \in X$ we write $x^g \in X$ (resp., ${}^g x$) for the result of a right (resp., left) action. A particularly important case arises when $G = \mathrm{Sym}(X)$ is the symmetric group on a set $X$, which acts on $X$ on the right.

### 3.1 Wreath products

A fundamental group construction that we shall use is the *wreath product*: given groups $G$ and $H$ acting on the right on sets $X$ and $Y$ respectively, their *wreath product* $G \wr H$ is a group acting on $X \times Y$. We start with the restricted direct product $G^{(Y)}$ (the base group) of all mappings $f : Y \to G$ having finite support $\mathrm{supp}(f) = \{y \mid f(y) \neq 1\}$ with the operation of pointwise multiplication. The group $H$ has a natural left action on $G^{(Y)}$: for $f \in G^{(Y)}$ and $h \in H$, we define ${}^h f \in G^{(Y)}$ by $({}^h f)(y) = f(y^h)$. The corresponding semidirect product $G^{(Y)} \rtimes H$ is the *wreath product* $G \wr H$. In other words:

- Elements of $G \wr H$ are pairs $(f, h) \in G^{(Y)} \times H$ and we simply write $fh$ for this pair.
- The multiplication in $G \wr H$ is defined as follows: Let $f_1 h_1, f_2 h_2 \in G \wr H$. Then $f_1 h_1 f_2 h_2 = f_1 {}^{h_1}\!f_2 h_1 h_2$, where the product $f_1 {}^{h_1}\!f_2 \colon y \mapsto f_1(y) f_2(y^{h_1})$ is the pointwise product.

The wreath product $G \wr H$ acts on $X \times Y$ by $(x,y)^{fh} = (x^{f(y)}, y^h)$. The wreath product defined above is also called the *(restricted) permutational wreath product*. There is also the variant where $G = X$ and $H = Y$ and both groups act on

themselves by right-multiplication, which is called the *(restricted) regular wreath product* (or *standard wreath product*). A subtle point is that the permutational wreath product is an associative operation whereas the regular wreath product is in general not. The term "restricted" refers to the fact that the base group is $G^{(Y)}$, i.e., only finitely supported mappings are taken into account. If $G^{(Y)}$ is replaced by $G^Y$ (i.e., the set of all mappings from $Y$ to $G$ with pointwise multiplication), then one speaks of an unrestricted wreath product. For $Y$ finite this makes of course no difference. There will be only two situations (Examples 5.11 and 5.12) where we need an unrestricted wreath product. The action of $G$ on $X$ in the permutational wreath product is usually not important for us, but it is nice to have an associative operation. For the right group $H$, we will only make use of the following cases:

- $H = \mathrm{Sym}(Y)$ acting on $Y$,
- $H$ a (finite or infinite) cyclic group acting on itself.

Thus, if $H$ is cyclic, the permutational wreath product and the regular wreath product (both denoted by $G \wr H$) coincide. Nevertheless, be aware that $G \wr (H \wr H) = (G \wr H) \wr H$ holds only for the permutational wreath product even if $H$ is cyclic. Note that if $G$ is generated by $\Sigma$ and $H$ is generated by $\Gamma$ then $G \wr H$ is generated by $\Sigma \cup \Gamma$.

### 3.2 Richard Thompson's groups

In 1965 Richard Thompson introduced three finitely presented groups $F < T < V$ acting on the unit-interval, the unit-circle and the Cantor set, respectively. Of these three groups, $F$ received most attention (the reader should not confuse $F$ with a free group). This is mainly due to the still open conjecture that $F$ is not amenable, which would imply that $F$ is another counterexample to a famous conjecture of von Neumann (a counterexample was found by Ol'shanskii). A standard reference of Thompson's groups is [13]. The group $F$ consists of all homeomorphisms of the unit interval that are piecewise affine, with slopes a power of 2 and dyadic breakpoints. Famously, $F$ is generated by two elements $x_0, x_1$ defined by

$$x_0(t) = \begin{cases} 2t & \text{if } 0 \le t \le \frac{1}{4}, \\ t + \frac{1}{4} & \text{if } \frac{1}{4} \le t \le \frac{1}{2}, \\ \frac{t}{2} + \frac{1}{2} & \text{if } \frac{1}{2} \le t \le 1, \end{cases} \qquad x_1(t) = \begin{cases} t & \text{if } 0 \le t \le \frac{1}{2}, \\ \frac{1}{2} + \frac{x_0(2t-1)}{2} & \text{if } \frac{1}{2} \le t \le 1. \end{cases}$$

The pattern repeats with $x_{n+1}$ acting trivially on the left subinterval and as $x_n$ on the right subinterval. We have $x_{k+1} = x_k^{x_i}$ for all $i < k$. In fact,

$$F = \langle x_0, x_1, x_2, \ldots \mid x_k^{x_i} = x_{k+1} (i < k) \rangle = \langle x_0, x_1 \mid [x_0 x_1^{-1}, x_0^{-1} x_1 x_0], [x_0 x_1^{-1}, x_0^{-2} x_1 x_0^2] \rangle. \tag{1}$$

The group $F$ is orderable (so in particular torsion-free), its derived subgroup $[F, F]$ is simple and the center of $F$ is trivial. Important for us is the following fact:

Lemma 3.1 ([23, Lemma 20]). *The group $F$ contains a subgroup isomorphic to $F \wr \mathbb{Z}$.*

Proof. The copy of $\mathbb{Z}$ is generated by $x_0$, and the copies of $F$ in $F^{(\mathbb{Z})}$ are the conjugates of $\langle x_1 x_2 x_1^{-2}, x_1^2 x_2 x_1^{-3} \rangle$ under powers of $x_0$. □

It follows, by iteration, that $F$ contains arbitrarily iterated wreath products $\mathbb{Z} \wr \cdots \wr \mathbb{Z}$, as well as the limit $((\cdots \wr \mathbb{Z}) \wr \mathbb{Z}) \wr \mathbb{Z}$.

### 3.3 Weakly branched groups

We continue our list of examples with an important class of groups acting on rooted trees. For more details, the monographs [7, 51] serve as good references.

Let $X$ be a finite set.[1] The free monoid $X^*$ serves as the vertex set of a regular rooted tree with an edge between $v$ and $vx$ for all $v \in X^*$ and all $x \in X$. The group $W$ of automorphisms of this tree naturally acts on the set $X$ of level-1 vertices, and permutes the subtrees hanging from them. Exploiting the bijection $X^+ = X^* \times X$, we thus have an isomorphism

$$\varphi \colon W \to W \wr \mathrm{Sym}(X) = W^X \rtimes \mathrm{Sym}(X), \tag{2}$$

mapping $g \in W$ to elements $f \in W^X$ and $\pi \in \mathrm{Sym}(X)$ as follows: $\pi$ is the restriction of $g$ to $X \subseteq X^*$, and $f$ is uniquely defined by $(xv)^g = x^\pi v^{f(x)}$. We always write $g@x$ for $f(x)$ and call it the *state (or coordinate) of $g$ at $x$*. If $X = [0..k]$, we write $g = \langle\!\langle g@0, \ldots, g@k \rangle\!\rangle \pi$.

*Definition 3.2.* A subgroup $G \leq W$ is *self-similar* if $\varphi(G) \leq G \wr \mathrm{Sym}(X)$. In other words: the actions on subtrees $xX^*$ are given by elements of $G$ itself. A self-similar group $G$ is *weakly branched* if there exists a non-trivial subgroup $K \leq G$ with $\varphi(K) \geq K^X$. In other words: for every $k \in K$ and every $x \in X$ the element acting as $k$ on the subtree $xX^*$ and trivially elsewhere belongs to $K$. A subgroup $K$ as above is called a *branching subgroup*.

Note that we are weakening the usual definition of "weakly branched": indeed it is usually additionally required that $G$ act transitively on $X^n$ for all $n \in \mathbb{N}$. This extra property is not necessary for our purposes, so we elect to simply ignore it. In fact, all the results concerning branched groups that we shall use will be proven directly from Definition 3.2.

Note also that the join $\langle K_1 \cup K_2 \rangle$ of two branching subgroups $K_1$ and $K_2$ is again a branching subgroup. Hence, there exists a maximal branching subgroup. It immediately follows from the definition that, if $G$ is weakly branched, then for every $v \in X^*$ there is in $G$ a copy of its branching subgroup $K$ whose action is concentrated on the subtree $vX^*$. We denote this copy with $v * K$. With $v * k$ ($k \in K$) we denote the element of $K$ acting as $k$ on the subtree $vX^*$ and trivially elsewhere.

Our main focus is on finitely generated groups. We first note that the group $W$ itself is weakly branched. Here are countable weakly branched subgroups of $W$: For a subgroup $\Pi$ of $\mathrm{Sym}(X)$, define $\Pi_\infty \leq W$ as follows: set $\Pi_0 = 1 \leq W$ (the trivial subgroup) and $\Pi_{n+1} = \varphi^{-1}(\Pi_n \wr \Pi)$. We clearly have $\Pi_n \leq \Pi_{n+1}$, and we set $\Pi_\infty = \bigcup_{n \geq 0} \Pi_n$. In words, $\Pi_n$ consists of permutations of $X^*$ that may only modify the first $n$ symbols of strings, and $\Pi_\infty$ consists of permutations that may only modify a bounded-length prefix of strings. Clearly $\Pi_\infty$ is countable and $\varphi(\Pi_\infty) = \Pi_\infty \wr \Pi$.

Numerous properties are known to follow from the fact that a group is weakly branched. For example, it satisfies no group identity [1]. In fact, if $G$ is a weakly branched self-similar group and its branching subgroup $K$ contains an element of order $p$, then $K$ contains a copy of $(\mathbb{Z}/p)_\infty$, see [7, Theorem 6.9].

There exist important examples of finitely generated self-similar weakly branched groups, notably the *Grigorchuk group* $G$, see [21]. It may be described as a self-similar group in the following manner: it is a group generated by $\{a, b, c, d\}$, and acts on the rooted tree $X^*$ for $X = \{0, 1\}$. The action, and therefore the whole group, are defined by the restriction of $\varphi$ to $G$'s generators:

$$\varphi(a) = (0, 1), \quad \varphi(b) = \langle\!\langle a, c \rangle\!\rangle, \quad \varphi(c) = \langle\!\langle a, d \rangle\!\rangle, \quad \varphi(d) = \langle\!\langle 1, b \rangle\!\rangle,$$

where we use the notation $(0, 1)$ for the non-trivial element of $\mathrm{Sym}(X)$ (that permutes 0 and 1) and $\langle\!\langle w_0, w_1 \rangle\!\rangle$ for a tuple in $G^{\{0,1\}} \cong G \times G$. We record some classical facts:

Lemma 3.3. *The Grigorchuk group $G$ is infinite, torsion (i. e., all elements are of finite order), weakly branched, and all its finite subquotients are 2-groups (so in particular nilpotent). It has a branching subgroup $K$ of finite index, which is therefore finitely generated.*

[1] There will be one occasion (Proposition 5.13), where we will allow an infinite $X$.

(Recall that every weakly branched group is infinite and non-solvable, since it satisfies no identity. There are also easy direct proofs of these facts.)

Proof. That $G$ is an infinite torsion group is one of the *raison d'être* of $G$, see [21]. Let $K \leq G$ be the normal closure of $[b,a]$ in $G$. It is easy to see that it has index 16, and $\varphi(\big[[b,a],d\big]) = \langle\!\langle 1, [b,a]\rangle\!\rangle$ so $\varphi(K) \geq K \times K$ and $G$ is weakly branched; see also [7] for details. It is known that every element of $G$ has order a power of 2 [21], so the same holds for every subquotient of $G$. □

Other examples of finitely generated self-similar weakly branched groups with a f.g. branching subgroup include the Gupta-Sidki groups [24], the Hanoi tower groups [22], and all iterated monodromy groups of degree-2 complex polynomials [8] except $z^2$ and $z^2 - 2$.

### 3.4 Contracting self-similar groups

Recall the notation $g@x$ for the coordinates of $\varphi(g)$. We iteratively define $g@v = g@x_1 \cdots @x_n$ for any word $v = x_1 \cdots x_n \in X^*$.

*Definition 3.4 ([51, Definition 2.11.1]).* A self-similar group $G$ is called *contracting* if there is a finite subset $N \subseteq G$ (called the *nucleus*) such that, for all $g \in G$, we have $g@v \in N$ whenever $v$ is long enough (depending on $g$).

If $G$ is a finitely generated contracting group with word norm $\|\cdot\|$ (i.e., for $g \in G$, $\|g\|$ is the length of a shortest word over a fixed generating set of $G$ that represents $g$), then a more quantitative property holds: there are constants $0 < \lambda < 1$, $h \geq 1$ and $k \geq 0$ such that for all $g \in G$ we have

$$\|g@v\| \leq \lambda\|g\| + k \text{ for all } v \in X^h,$$

see e.g. [35, Proposition 9.3.11]. Then, for $c = -h/\log\lambda$ and a possibly larger $k$ we have $g@v \in N$ whenever $|v| \geq c\log\|g\| + k$. One of the cornerstones of Nekrashevych's theory of iterated monodromy groups is the construction of a contracting self-similar group that encodes a given expanding self-covering of a compact metric space. It is well-known and easy to check that the Grigorchuk group, the Gupta-Sidki groups and the Hanoi tower group for three pegs are contracting. The following result has been quoted numerous times, but has never appeared in print. A proof for the Grigorchuk group may be found in [20]:

Proposition 3.5. *Let $G$ be a finitely generated contracting self-similar group. Then* $\mathrm{WP}(G)$ *can be solved in* LOGSPACE *(deterministic logarithmic space).*

Proof. Fix a finite generating set $\Sigma$ for $G$ and assume that $G$ is contracting with $0 < \lambda < 1$, $h \geq 1$ and $k \geq 0$ as above. We can assume that $k \geq 1$. Let $N$ be the nucleus of $G$. By replacing the tree alphabet $X$ by $X^h$ we get $\|g@x\| \leq \lambda\|g\| + k$ for all $x \in X$. Hence, if $\|g\| \leq k/(1-\lambda)$ then also $\|g@x\| \leq k/(1-\lambda)$ for all $x \in X$. We now replace $\Sigma$ by the set of all $g \in G$ with $\|g\| \leq k/(1-\lambda)$ (note that $k/(1-\lambda) \geq 1$) and get $\varphi(\Sigma) \subseteq \Sigma^X \times \mathrm{Sym}(X)$. Furthermore, there exists $m$ such that every non-trivial element of $N$ acts non-trivially on $X^m$. Recall that for $c = -1/\log\lambda$ and a possibly larger $k$ we have $g@v \in N$ whenever $|v| \geq c\log\|g\| + k$. Hence, if $g$ is non-trivial then there must exist a $v \in X^*$ with $|v| = c\log\|g\| + k + m$ such that $g$ does not fix $v$.

The following algorithm solves $\mathrm{WP}(G)$: given $g \in \Sigma^*$, enumerate all vertices in $X^d$ for $d = c\log|g| + k + m$, and return "true" precisely when they are all fixed by $g$. The algorithm is correct by the previous remarks, and it remains to show that it requires logarithmic space. The vertices in $X^d$ are traversed by lexicographically enumerating them. They

can be stored explicitly since their length is bounded by $O(\log |g|)$. Now given a vertex $v \in X^d$, we apply the letters of $g$ to it one after the other. Again, this is done by a simple loop requiring $O(\log |g|)$ bits. Finally, to apply a generator to $v$, we use the property that all its states are generators ($\varphi(\Sigma) \subseteq \Sigma^X \times \mathrm{Sym}(X)$), and traverse $v$ by performing $|v|$ lookups in the table storing $(\varphi(a))_{a\in\Sigma}$. □

## 4 COMPLEXITY THEORY

We assume that the reader is familiar with the complexity classes LOGSPACE (deterministic logarithmic space), P (deterministic polynomial time), and PSPACE (polynomial space); see e.g. [3] for details. With polyL we denote that union of all classes NSPACE($\log^c n$) for a constant $c$. Since we also deal with sublinear time complexity classes, we use Turing machines with *random access* (this has no influence on the definition of the above classes). Such a machine has an additional index tape and some special query states. Whenever the Turing machine enters a query state, the following transition depends on the input symbol at the position which is currently written on the index tape in binary notation.

We use the abbreviations DTM (deterministic Turing machine), NTM (non-deterministic Turing machine) and ATM (alternating Turing machine). An ATM is an NTM together with a partition of the state set into existential and universal states. A configuration is called existential (resp., universal) if the current state in the configuration is existential (resp., universal). An existential configuration is accepting if there exists an accepting successor configuration, whereas a universal configuration is accepting if all successor configurations are accepting. Note that a universal configuration which does not have a successor configuration is accepting, whereas an existential configuration which does not have a successor configuration is non-accepting. Finally, an input word is accepted if the corresponding initial configuration is accepted. An ATM is in *input normal form* if its input alphabet is $\{0, 1\}$ and on any computation path it queries at most one input bit and halts immediately after returning the value of the input bit or its negation (depending on the current state of the Turing machine). We define the following complexity classes:

- DLINTIME: the class of languages that can be accepted by a DTM in linear time.
- DLOGTIME: the class of languages that can be accepted by a DTM in logarithmic time.
- ALOGTIME: the class of languages that can be accepted by an ATM in logarithmic time.
- APTIME: the class of languages that can be accepted by an ATM in polynomial time.

If $X$ is one of the above classes, we speak of an $X$-machine with the obvious meaning. It is well known that APTIME = PSPACE. Moreover, every language in ALOGTIME can be recognized by an ALOGTIME-machine in input normal form [60, Lemma 2.41].

A nand-machine is an NTM in which each configuration has either zero or two successor configurations and configurations are declared to be accepting, respectively non-accepting, according to the following rules, where $c$ is a configuration:

- If $c$ has no successor configurations and the state of $c$ is final (resp., non-final), then $c$ is accepting (resp., non-accepting).
- If $c$ has two successor configurations and both of them are accepting, then $c$ is not accepting.
- If $c$ has two successor configurations and at least one them is non-accepting, then $c$ is accepting.

Since the boolean functions and and or can be obtained with nand, it follows easily that PSPACE (resp., ALOGTIME) coincides with the class of all languages that can be accepted by a polynomially (resp., logarithmically) time-bounded nand-machine.

For a complexity class C we denote by ∀C the class of all languages $L$ such that there exists a polynomial $p(n)$ and a language $K \in \mathsf{C}$ such that $L = \{u \mid \forall v \in \{0,1\}^{p(|u|)} : u\#v \in K\}$. We have for instance ∀P = coNP and ∀PSPACE = PSPACE. Likewise we define the class $\mathsf{Mod}_m\mathsf{C}$ by $L \in \mathsf{Mod}_m\mathsf{C}$ if there exists a polynomial $p(n)$ and a language $K \in \mathsf{C}$ such that $L = \left\{u \mid |\{v \in \{0,1\}^{p(|u|)} : u\#v \in K\}| \not\equiv 0 \mod m\right\}$.

### 4.1 Efficiently computable functions

A function $f\colon \Gamma^* \to \Sigma^*$ is DLOGTIME-computable if there is some polynomial $p$ with $|f(x)| \leq p(|x|)$ for all $x \in \Gamma^*$ and the set $L_f = \{(x,a,i) \mid x \in \Gamma^*$ and the $i$-th letter of $f(x)$ is $a\}$ belongs to DLOGTIME. Here $i$ is a binary coded integer. Note that a DLOGTIME-machine for $L_f$ can first (using binary search) compute the binary coding of $|x|$ in time $O(\log|x|)$. Assume that the length of this binary coding is $\ell$. If $i$ has more than $\ell$ bits, the machine can reject immediately. As a consequence of this (and since $|\Sigma|$ is a constant), the running time of a DLOGTIME-machine for $L_f$ on input $(x,a,i)$ can be bounded by $O(\log|x|)$ (independently of the actual bit length of $i$). We can also assume that the DLOGTIME-machine outputs the letter $a$ on input of $x$ and $i$. In case $i > |x|$ we can assume that the machine outputs a distinguished letter. A DLOGTIME-reduction is a DLOGTIME-computable many-one reduction. We say that a DLOGTIME-machine *strongly* computes a function $f\colon \Sigma^* \to \Gamma^*$ with $|f(x)| \leq C\log(|x|)$ for all $x \in \Sigma^*$ and for some constant $C$ if it computes the function value by writing it sequentially on a separate output tape (be aware of the subtle difference and that strong DLOGTIME-computability is not a standard terminology, but it coincides with FDLOGTIME in [14].)

A PSPACE-transducer is a deterministic Turing-machine with a read-only input tape, a write-only output tape and a work tape, whose length is polynomially bounded in the input length $n$. The output is written sequentially on the output tape. Moreover, we assume that the transducer terminates for every input. This implies that a PSPACE-transducer computes a mapping $f : \Sigma^* \to \Gamma^*$, where $|f(x)|$ is bounded by $2^{|x|^{O(1)}}$. We call this mapping PSPACE-computable. We need the following simple lemma, see [47]:

Lemma 4.1. *Assume that the mapping $f : \Sigma^* \to \Gamma^*$ is* PSPACE*-computable and let $L \subseteq \Gamma^*$ be a language in* polyL*. Then $f^{-1}(L)$ belongs to* PSPACE*.*

### 4.2 Leaf languages

In the following, we introduce basic concepts related to leaf languages, more details can be found in [12, 28, 30, 31, 36]. An NTM $M$ with input alphabet $\Gamma$ is *adequate*, if (i) for every input $x \in \Gamma^*$, $M$ does not have an infinite computation on input $x$, (ii) the finite set of transition tuples of $M$ is linearly ordered, and (iii) when terminating $M$ prints a symbol $\alpha(q)$ from a finite alphabet $\Sigma$, where $q$ is the current state of $M$. For an input $x \in \Gamma^*$, we define the computation tree by unfolding the configuration graph of $M$ from the initial configuration. By condition (i) and (ii), the computation tree can be identified with a finite ordered tree $T(x) \subseteq \mathbb{N}^*$. For $u \in T(x)$ let $q(u)$ be the $M$-state of the configuration that is associated with the tree node $u$. Then, the leaf string $\mathrm{leaf}(M,x)$ is the string $\alpha(q(v_1)) \cdots \alpha(q(v_k)) \in \Sigma^+$, where $v_1, \ldots, v_k$ are all leaves of $T(x)$ listed in lexicographic order.

An adequate NTM $M$ is called *balanced*, if for every input $x \in \Gamma^*$, $T(x)$ is a complete binary tree. With a language $K \subseteq \Sigma^*$ we associate the language

$$\mathsf{LEAF}(M,K) = \{x \in \Gamma^* \mid \mathrm{leaf}(M,x) \in K\}.$$

Finally, we associate two complexity classes with $K \subseteq \Sigma^*$:

$$\begin{aligned} \mathsf{LEAF}(K) &= \{\mathsf{LEAF}(M,K) \mid M \text{ is an adequate polynomial time NTM}\} \\ \mathsf{bLEAF}(K) &= \{\mathsf{LEAF}(M,K) \mid M \text{ is a balanced polynomial time NTM}\} \end{aligned}$$

These classes are closed under polynomial time reductions. We clearly have $\mathsf{bLEAF}(K) \subseteq \mathsf{LEAF}(K)$. The following result was shown in [36] by padding computation trees to complete binary trees.

Lemma 4.2. *Assume that $K \subseteq \Sigma^*$ is a language such that $\Sigma$ contains a symbol $1$ with the following property: if $uv \in K$ for $u, v \in \Sigma^*$ then $u1v \in K$. Then $\mathsf{LEAF}(K) = \mathsf{bLEAF}(K)$.*

In particular, we obtain the following lemma:

Lemma 4.3. *Let $G$ be a finitely generated group and $\Sigma$ a finite standard generating set for $G$. Then $\mathsf{LEAF}(\mathrm{WP}(G,\Sigma)) = \mathsf{bLEAF}(\mathrm{WP}(G,\Sigma))$.*

Moreover, we have:

Lemma 4.4. *Let $G$ be finitely generated group and $\Sigma$, $\Gamma$ finite standard generating sets for $G$. Then $\mathsf{LEAF}(\mathrm{WP}(G,\Sigma)) = \mathsf{LEAF}(\mathrm{WP}(G,\Gamma))$.*

Proof. Consider a language $L \in \mathsf{LEAF}(\mathrm{WP}(G,\Sigma))$. Thus, there exists an adequate polynomial time NTM $M$ such that $L = \mathsf{LEAF}(M, \mathrm{WP}(G,\Sigma))$. We modify $M$ as follows: If $M$ terminates and prints the symbol $a \in \Sigma$, it enters a small nondeterministic subcomputation that produces the leaf string $w_a$, where $w_a \in \Gamma^*$ is a word that evaluates to the same group element as $a$. Let $M'$ be the resulting adequate polynomial time NTM. It follows that $\mathsf{LEAF}(M, \mathrm{WP}(G,\Sigma)) = \mathsf{LEAF}(M', \mathrm{WP}(G,\Gamma))$. □

Lemma 4.4 allows to omit the standard generating set $\Sigma$ in the notations $\mathsf{LEAF}(\mathrm{WP}(G,\Sigma))$ and $\mathsf{bLEAF}(\mathrm{WP}(G,\Sigma))$. We will always do that. In [30] it was shown that $\mathsf{PSPACE} = \mathsf{LEAF}(\mathrm{WP}(G))$ for every finite non-solvable group.

### 4.3 Circuit complexity

We define a *polynomial length projection* (or just *projection*) as a function $f\colon \{0,1\}^* \to \{0,1\}^*$ such that there is a function $d(n) \in O(\log n)$ with $|f(x)| = |f(y)| = 2^{d(n)}$ for all $x, y$ with $|x| = |y| = n$ and such that each output bit depends on at most one input bit in the following sense: For every $n \in \mathbb{N}$, there is a mapping $q_n\colon \{0,1\}^{d(n)} \to \{\langle j, a, b\rangle \mid j \in [1..n], a, b, \in \{0,1\}\}$, where $q_n(i) = \langle j, a, b\rangle$ means that for all $x \in \{0,1\}^n$ the $i$-th bit of $f(x)$ is $a$ if the $j$-th bit of $x$ is 1 and $b$ if it is 0. Here, we identify $i \in \{0,1\}^{d(n)}$ with a binary coded number from $[0..2^{d(n)} - 1]$ (so the first position in the output is zero). We also assume that the input position $j \in [1..n]$ is coded in binary, i.e., by a bit string of length $O(\log n)$. Note that the output length $2^{d(n)}$ is polynomial in $n$. Restricting the output length to a power of two (instead of an arbitrary polynomial) is convenient for our purpose but in no way crucial. Our definition of a projection is the same as in [14] except for our restriction on the output length. Moreover, in [14] projections were defined for arbitrary alphabets.

Let $q\colon \{1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\} \times \{0,1\}$ with $q(1^n, v) = q_n(v)$. We assume that $q(1^n, v)$ is a special dummy symbol if $|v| \neq d(n)$. We call $q$ the *query mapping* associated with the projection $f$. The projection $f$ is called *uniform* if (i) $1^{d(n)}$ is strongly computable in DLOGTIME from the string $1^n$, and (ii) $q$ is strongly DLOGTIME-computable. Notice that if a language $K$ is reducible to $L$ via a uniform projection, then $K$ is also DLOGTIME-reducible to $L$.

We are mainly interested in the circuit complexity class $\mathsf{NC}^1$. A language $L \subseteq \{0,1\}^*$ is in $\mathsf{NC}^1$ if it can be recognized by a family of logarithmic depth boolean circuits of bounded fan-in. More precisely, $L \subseteq \{0,1\}^*$ belongs to $\mathsf{NC}^1$ if there exists a family $(C_n)_{n\geq 0}$ of boolean circuits which, apart from the input gates $x_1, \ldots, x_n$, are built up from not-, and- and or-gates. In the following we also use nand-gates. All gates must have bounded fan-in, where the fan-in of a gate is the number of incoming edges of the gate. Without loss of generality, we assume that all and-, or- and nand-gates have fan-in two. The circuit $C_n$ must accept exactly the words from $L \cap \{0,1\}^n$, i.e., if each input gate $x_i$ receives the input $a_i \in \{0,1\}$, then a distinguished output gate evaluates to 1 if and only if $a_1 a_2 \cdots a_n \in L$. Finally, the depth (maximal length of a path from an input to the distinguished output) of $C_n$ must grow logarithmically in $n$. In the following, we also consider DLOGTIME-uniform $\mathsf{NC}^1$, which is well-known to coincide with ALOGTIME (see e. g. [60, Corollary 2.52]). DLOGTIME-uniform means that there is a DLOGTIME-machine which decides on input of two gate numbers $i$ and $j$ in $C_n$ (given in binary), a binary string $w$, and the string $1^n$ whether, when starting at gate $i$ in $C_n$ and following the path labelled by $w$, we reach gate $j$. Here, following the path labelled by $w$ means that we go to the left (right) input of $i$ if $w$ starts with a 0 (1) and so on. Moreover, we require that on input of $i$ in binary and the string $1^n$, the type of the gate $i$ in $C_n$ is computable in DLOGTIME. For more details on these definitions we refer to [60] (but we will not need the above definition of DLOGTIME-uniformity). For a language $L$ over a non-binary alphabet $\Sigma$, one first has to fix a binary encoding of the symbols in $\Sigma$. For membership in $\mathsf{NC}^1$ the concrete encoding is irrelevant. However, we still assume that all letters of $\Sigma$ are encoded using the same number of bits.

The class $\mathsf{AC}^0$ is defined as the class of languages (respectively functions) accepted (respectively computed) by circuits of constant depth and polynomial size with not-gates and unbounded fan-in and- and or-gates.

We will also work with a very restricted class of circuit families, where every circuit is a complete binary tree of nand-gates. For such a circuit, all the information is given by the labelling function for the input gates.

*Definition 4.5.* A *family of balanced* nand*-tree-circuits of logarithmic depth* $(C_n)_{n\in\mathbb{N}}$ is given by a mapping $d(n) \in O(\log n)$ and a query mapping $q\colon \{1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\} \times \{0,1\}$, which defines a projection $f$ mapping bit strings of length $n$ to bit strings of length $2^{d(n)}$. The corresponding circuit $C_n$ for input length $n$ is then obtained by taking $\{0,1\}^{\leq d(n)}$ as the set of gates. Every gate $v \in \{0,1\}^{<d(n)}$ computes the nand of $v0$ and $v1$. If $x \in \{0,1\}^n$ is the input string for $C_n$ and $f(x) = a_1 a_2 \cdots a_{2^{d(n)}}$, then the $i$-th leaf $v \in \{0,1\}^{d(n)}$ (in lexicographic order) is set to $a_i$.

Lemma 4.6. *For every $L$ in (non-uniform)* $\mathsf{NC}^1$ *there is a (non-uniform) family of balanced* nand*-tree-circuits of logarithmic depth.*

Proof. The proof is straightforward: clearly, or, and, and not gates can be simulated by nand gates. Now take the circuit $C_n$ for input length $n$. We first unfold $C_n$ into a tree by duplicating gates with multiple outputs. Since $C_n$ has constant fan-in and logarithmic depth, the resulting tree has still polynomial size (and logarithmic depth). To transform this tree into a complete binary tree, we replace leafs by complete binary subtrees. If we replace a leaf labelled with $x_i$ by a subtree of even (resp. odd) height, then we label all leafs of the subtree with $\langle i, 1, 0\rangle$ (resp., $\langle i, 0, 1\rangle$). This labelling defines the query mapping $q$ in the natural way. □

Lemma 4.7. *For every $L$ in* ALOGTIME *there is a family $C = (C_n)_{n\geq 0}$ of balanced* nand*-tree-circuits of logarithmic depth such that the mapping $1^n \mapsto 1^{d(n)}$ and the query mapping $q$ from Definition 4.5 can be strongly computed in* DLOGTIME.

Proof sketch. We start with an ALOGTIME-machine $M$ for $L$ and construct a circuit family with the required properties. We can assume that $M$ works in two stages: first it computes the binary coding of the input length in

DLOGTIME (using binary search). The second stage performs the actual computation. We can assume that the second stage is in input normal form [60, Lemma 2.41] meaning that each computation path queries exactly one input position $i$ and halts immediately after querying that position (returning a bit that is determined by the $i$-th bit of the input). Furthermore, we can assume that the computation tree of the second stage of $M$ is a complete binary tree. For this we enforce all computation paths to be of the same length. Note that the running time of the second stage of $M$ can be bounded by $c \cdot |u|$, where $c$ is a fixed constant and $u$ is the binary coding of the input length which has been computed before. Hence, the second stage of the machine makes in parallel to the actual computation $c$ runs over $u$. Finally, we also assume that there is an alternation in every step (this can be ensured as in the transformation of an arbitrary $\mathsf{NC}^1$-circuit into a balanced nand-tree-circuit) and that the initial state is existential. The computation tree gives a tree-shaped circuit in a natural way (for details see [60, Theorem 2.48]). The depth of this tree is $d := c \cdot |u|$ (whose unary encoding is strongly computable in DLOGTIME by the above arguments). Since we start with an existential state and there is an alternation in every step, the resulting circuit uses only nand-gates (recall that $x$ nand $y$ = (not $x$) or (not $y$)). The fact that every computation path queries only one input position yields the query function $q$ from Definition 4.5. More precisely, let $v \in \{0,1\}^d$ be an input gate of the balanced nand-tree-circuit. Then $v$ determines a unique computation path of $M$. We simulate $M$ in DLOGTIME along this path and output the triple $\langle i, a, b\rangle$ if $M$ queries the $i$-th position of the input string (note that the binary coding of $i$ must be on the query tape of $M$) and outputs $a$ (resp., $b$) if the $i$-th input bit is 1 (resp., 0). □

*4.3.1 G-programs.* For infinite groups we have to adapt Barrington's notion of a $G$-program slightly. Our notation follows [60].

*Definition 4.8.* Let $G$ be a group with the finite standard generating set $\Sigma$. Recall our assumption that $1 \in \Sigma$. A $(G,\Sigma)$-program $P$ of length $m$ and input length $n$ is a sequence of *instructions* $\langle i_j, b_j, c_j\rangle$ for $0 \le j \le m-1$ where $i_j \in [1..n]$ and $b_j, c_j \in \Sigma$. On input of a word $x = a_1 \cdots a_n \in \{0,1\}^*$, an instruction $\langle i_j, b_j, c_j\rangle$ evaluates to $b_j$ if $a_{i_j} = 1$ and to $c_j$ otherwise. The evaluation of a $(G,\Sigma)$-program is the product (in the specified order) of the evaluations of its instructions, and is denoted with $P[x] \in \Sigma^*$.

A family $\mathcal{P} = (P_n)_{n\in\mathbb{N}}$ of $(G,\Sigma)$-programs, where $P_n$ has input length $n$, defines a function $f_{\mathcal{P}} \colon \{0,1\}^* \to G$: $f_{\mathcal{P}}(x)$ is the group element represented by $P_{|x|}[x]$. The language $L$ *accepted* by the family of $(G,\Sigma)$-programs is the set of words $x \in \{0,1\}^*$ such that $f_{\mathcal{P}}(x) = 1$ in $G$. For brevity, we also speak of a *family of $G$-programs* instead of $(G,\Sigma)$-programs with the understanding that there is some finite standard generating set $\Sigma$ which is shared by all programs of the family.

Notice two differences compared with the original definition: firstly, we fix the finite alphabet $\Sigma$, and secondly, for the accepted language we only take the preimage of 1 instead of a finite set of final states. The latter is more restrictive, but for the purpose of $\mathsf{NC}^1$-hardness causes no difference.

A family $\mathcal{P} = (P_n)_{n\in\mathbb{N}}$ of $(G,\Sigma)$-programs is called *uniform* if the length of $P_n$ is $2^{d(n)}$ for some function $d(n) \in O(\log n)$, the mapping $1^n \mapsto 1^{d(n)}$ is strongly computable in DLOGTIME, and the mapping that assigns to $1^n$ and $j \in \{0,1\}^{d(n)}$ (the latter is interpreted as a binary coded number) the instruction $\langle i_j, b_j, c_j\rangle$ of the $n$-input program $P_n$ is strongly computable in DLOGTIME. Notice that $i_j$ requires $\log n$ bits and $b_j$, $c_j$ require only a constant number of bits – thus, the tuple $\langle i_j, b_j, c_j\rangle$ can be written down in DLOGTIME. Be aware that here we slightly differ from [60, Definition 4.42, Definition 4.51] (which does not require *strong* DLOGTIME computability).

*Remark 4.9.* If a language $L$ is accepted by a family of polynomially length-bounded $(G,\Sigma)$-programs (by padding one can enforce the length to be of the form $2^{d(n)}$), then $L$ is reducible via projections to $\mathrm{WP}(G)$ – and, thus, also via

$\mathrm{AC}^0$-many-one reductions. This can be seen as follows: encode every letter in $\Sigma$ by a word over $\{0,1\}$ of some fixed constant length. Then the map assigning the evaluation of the $(G,\Sigma)$-program to an input word is a projection since the output at every position depends on only one input bit.

A similar statement holds in the uniform case (uniformity follows immediately from the definition): if $L$ is accepted by a uniform family of $(G,\Sigma)$-programs, then $L$ is reducible via uniform projections to $\mathrm{WP}(G)$.

## 5 EFFICIENTLY NON-SOLVABLE GROUPS

We now define the central group theoretic property that allows us to carry out a Barrington style construction:

*Definition 5.1.* We call a group $G$ with the finite standard generating set $\Sigma$ *strongly efficiently non-solvable (SENS)* if for every $d \in \mathbb{N}$ there is a collection of $2^{d+1}-1$ elements $g_{d,v} \in \Sigma^*$ for $v \in \{0,1\}^{\leq d}$ such that

(a) there is some constant $\mu \in \mathbb{N}$ with $\left|g_{d,v}\right| = 2^{\mu d}$ for all $v \in \{0,1\}^d$,
(b) $g_{d,v} = \left[g_{d,v0}, g_{d,v1}\right]$ for all $v \in \{0,1\}^{<d}$ (here we take the commutator of words),
(c) $g_{d,\varepsilon} \neq 1$ in $G$.

The group $G$ is called *uniformly strongly efficiently non-solvable* if, moreover,

(d) given $v \in \{0,1\}^d$, a number $i$ encoded in binary with $\mu d$ bits, and $a \in \Sigma$ one can decide in DLINTIME whether the $i$-th letter of $g_{d,v}$ is $a$.

If $Q = H/K$ is a subquotient of $G$, we call $Q$ *SENS in* $G$ if $G$ satisfies the conditions of a SENS group, all $g_{d,v}$ evaluate to elements of $H$, and $g_{d,\varepsilon} \notin K$. This definition is already interesting for $K = 1$.

Here are some simple observations:

- A strongly efficiently non-solvable group clearly cannot be solvable, so the above terminology makes sense.
- If one can find suitable $g_{d,v}$ of length at most $2^{\mu d}$, then these words can always be padded to length $2^{\mu d}$ thanks to the padding letter 1.
- It suffices to specify $g_{d,v}$ for $v \in \{0,1\}^d$; the other $g_{d,v}$ are then defined by Condition (b).
- We have $\left|g_{d,v}\right| = 2^{\mu d + 2(d-|v|)}$ for all $v \in \{0,1\}^{\leq d}$. Thus, all $g_{d,v}$ have length $2^{O(d)}$.
- Equivalently to Condition (d), we can require that given $v \in \{0,1\}^d$ and a binary encoded number $i$ with $\mu d$ bits, one can compute the $i$-th letter of $g_{d,v}$ in DLINTIME.

Henceforth, whenever $d$ is clear, we simply write $g_v$ instead of $g_{d,v}$.

The reason for the "strongly" in the name of SENS is that there is also a similar, but more general, property which we call ENS (see Remark 5.8 below). As there is little benefit from the extra definition, we only present it briefly in Remark 5.8 but refrain from proving any further results about it. We continue with some further observations about SENS groups.

Lemma 5.2. *The property of being SENS is independent of the choice of the standard generating set. The same applies to uniformly SENS.*

Proof. Let $\Sigma'$ be another standard generating set. Then, for some constant integer $k$, every element of $\Sigma$ may be written (thanks to the padding letter 1) as a word of length $2^k$ in $\Sigma'$. In particular, if $g_{d,v}$ has length $2^{\mu d}$ with respect to $\Sigma$, then it has length $2^{k+\mu d}$ with respect to $\Sigma'$. There is also a simple DLINTIME-algorithm for computing the $i$-th letter of $g_{d,v} \in (\Sigma')^*$: given $v$, and $i$, it runs the DLINTIME-algorithm for $\Sigma$ on input $v$ and $\left\lfloor i/2^k \right\rfloor$, obtaining a letter $\sigma \in \Sigma$. Then, it looks up the length-$2^k$ representation of $\sigma$ over $\Sigma'$, and extracts the $(i \bmod 2^k)$-th letter of that representation. □

Later (Example 5.12) we will give an example of a f.g. non-SENS group $H$ which is uniformly SENS in a group $G$.

Lemma 5.3. *If $Q = H/K$ is a finitely generated subquotient of a finitely generated group $G$ and $Q$ is SENS (resp. uniformly SENS), then $G$ is also SENS (resp. uniformly SENS).*

Proof. Let $\Gamma$ be a standard generating set of $Q$ and fix for every $a \in \Gamma$ an element $h_a \in H \leq G$ such that $h_a$ is mapped to $a$ under the canonical projection $\pi : H \to Q = H/K$. By Lemma 5.2 we can assume that all elements $h_a$ belong to the generating set of $G$. Let $h_{d,v} \in \Gamma^*$ be the words witnessing the fact that $Q$ is (uniformly) SENS (in Definition 5.1 they are denoted with $g_{d,v}$). We then define words $g_{d,v}$ by replacing every letter $a$ in $h_{d,v}$ by the letter $h_a$. Clearly, $\pi(g_{d,v}) = h_{d,v}$ holds. In particular, $g_{d,\varepsilon}$ is non-trivial, since $h_{d,\varepsilon}$ is non-trivial. □

Lemma 5.4. *If $G$ is SENS (resp. uniformly SENS) and $N$ a normal subgroup such that $G/N$ is solvable, then $N$ is SENS (resp. uniformly SENS) in $G$.*

Notice that Lemma 5.4, in particular, implies that, if $G$ is SENS (resp. uniformly SENS), then the commutator subgroup $G'$ is SENS (resp. uniformly SENS) in $G$.

Proof. Assume that $G/N$ is solvable of derived length $\delta$. Hence, any $\delta$-fold nested commutator of elements in $G$ is contained in $N$. Let $h_{d,v}$ be the elements witnessing that $G$ is (uniformly) SENS. Given $d$ and $v \in \{0,1\}^{\leq d}$ define $g_{d,v} = h_{d+\delta,v}$. Then all these elements are $\delta$-fold nested commutators and, hence, contained in $N$. The length bounds and uniformity condition are also clear. Thus, the elements $g_{d,v}$ witness that $N$ is (uniformly) SENS in $G$. □

Lemma 5.5. *If $G$ is SENS and $N$ a solvable normal subgroup of $G$, then $G/N$ is SENS.*

Be aware that we do not know whether there is a variant of Lemma 5.5 for uniformly SENS. The problem is to compute the word $u$ in the proof below.

Proof. Again, we only prove the statement for the case that $G$ is SENS. As in the proof of Lemma 5.4, let $h_{d,v}$ for $d \in \mathbb{N}$ and $v \in \{0,1\}^{\leq d}$ denote the elements witnessing that $G$ is SENS. Let $\delta$ denote the derived length of $N$. Assume for contradiction that all the elements $h_{d+\delta,v}$ for $v \in \{0,1\}^{\delta}$ are in $N$. Then, $h_{d+\delta,\varepsilon}$ would be trivial because it is a $\delta$-fold nested commutator of the $h_{d+\delta,v}$ for $v \in \{0,1\}^{\delta}$ and the derived length of $N$ is $\delta$. Thus, there exists some $u \in \{0,1\}^{\delta}$ such that $h_{d+\delta,u} \notin N$. We fix this $u$ and set $g_{d,v} = h_{d+\delta,uv}$ for $v \in \{0,1\}^{\leq d}$. Since $g_{d,\varepsilon} = h_{d+\delta,u} \notin N$, this shows that $G/N$ is SENS. □

Lemma 5.6. *If $G$ is SENS (resp. uniformly SENS), then $G/Z(G)$ is SENS (resp. uniformly SENS).*

Proof. As before, let $h_{d,v}$ for $d \in \mathbb{N}$ and $v \in \{0,1\}^{\leq d}$ denote the elements witnessing that $G$ is (uniformly) SENS. We set $g_{d,v} = h_{d+1,0v}$ for $v \in \{0,1\}^{\leq d}$. Then $g_{d,\varepsilon} = h_{d+1,0}$ cannot be in $Z(G)$ for otherwise $h_{d+1,\varepsilon} = [g_{d,\varepsilon}, h_{d+1,1}]$ would be trivial. This shows that $G/Z(G)$ is (uniformly) SENS. □

### 5.1 Finite and free groups and the ENS property

The following result is, for $G = A_5$, the heart of Barrington's argument:

Lemma 5.7. *If $G$ is a finite non-solvable group, then $G$ is uniformly SENS.*

Proof. Let us first show the statement for a non-abelian finite simple group $G$. By the proof of Ore's conjecture [41], every element of $G$ is a commutator. This means that we may choose $g_\varepsilon \neq 1$ at will, and given $g_v$ we define $g_{v0}, g_{v1}$ by

table lookup, having chosen once and for all for each element of $G$ a representation of it as a commutator. Computing $g_v$ requires $|v|$ steps and bounded memory.

If $G$ is finite non-solvable, then any composition series of $G$ contains a non-abelian simple composition factor $G_i/G_{i+1}$. Hence, we can apply Lemma 5.3. □

Notice that at the time of Barrington's original proof [4], Ore's conjecture was not known to hold. This explains that he used only what we call efficiently non-solvable (as defined in the following remark) in order to establish his result on $\mathsf{NC}^1$-hardness:

*Remark 5.8.* We can formulate a weaker condition than being strongly efficiently non-solvable which is closer to Barrington's original proof [4], but slightly more complicated to state: We call a group $G$ *efficiently non-solvable (ENS)* if there is an even constant $l$ such that for every $d \in \mathbb{N}$, there is a collection of elements $(g_{d,v})_{v\in[1..l]^{\leq d}}$ with

(a) $\left|g_{d,v}\right| \in 2^{O(d)}$ when $|v| = d$,
(b) $g_{d,v} = [g_{d,v1}, g_{d,v2}] \cdots [g_{d,v(l-1)}, g_{d,vl}]$ when $|v| < d$,
(c) $g_{d,\varepsilon} \neq 1$ in $G$.

Analogously to Definition 5.1, we can define *uniformly ENS* if the letters of $g_{d,v}$ for $|v| = d$ can be computed in DLINTIME.

To not overload the presentation, we prove our results only for SENS groups. Moreover, as the following lemma shows, in the non-uniform case the two definitions are indeed equivalent. Only in the uniform case, we do not know whether the two definitions agree – however, all examples of groups in our work are already uniformly SENS or not even ENS.

Lemma 5.9. *If $G$ is ENS, then $G$ is SENS.*

Proof. Let $(g_{d,v})_{v\in[1..l])^{\leq d}}$ be as in Remark 5.8, meaning that, in particular, $g_{d,\varepsilon} \neq 1$ in $G$. We can think of $g_{d,\varepsilon}$ as a word over the alphabet $\left\{g_{d,v}^{\pm 1} \;\middle|\; |v| = d\right\}$. Using the identity $[x, zy] = [x, y][x, z]^y$, we can rewrite $g_{d,\varepsilon}$ as a product $h_1 h_2 \cdots h_k$ of balanced nested commutators $h_i$ of depth $d$ (where no product appears inside any commutator). For this, observe that, since $[x, z]^y = [x^y, z^y]$, we can pull all conjugations inside the commutators. By doing so, we have to replace the $g_{d,v}$ for $|v| = d$ by conjugates of the original $g_{d,v}$. Notice that by Remark 5.8(a) and the recursive definition (b) we know that $\left|g_{d,v}\right| \in 2^{O(d)}$ for all $v \in [1..l]^{\leq d}$. Therefore the conjugates of the $g_{d,v}$ are of length $2^{O(d)}$ as well. Since $g_{d,\varepsilon} \neq 1$ in $G$, at least one of the balanced nested commutators $h_i$ is non-trivial. This $h_i$ is a non-trivial balanced nested commutator of depth $d$, hence, witnessing that $G$ is SENS. □

For the special case of finite non-solvable groups, the proof of Lemma 5.9 can be extended to actually show the *uniform* SENS property without using the deep result [41]:

Second proof of Lemma 5.7. First of all, by the very definition of non-solvability, every non-abelian finite simple $G$ group is ENS (as defined in Remark 5.8) by taking the full group as set of generators. Thus, Lemma 5.9 tells us that $G$ is SENS. Let $g_{d,v}$ be the words from Definition 5.1. It remains to show Condition (d) from Definition 5.1. Since $G$ is finite, we can find a subset $S$ of $G$ such that for each $g \in S$ there are $h_1, h_2 \in S$ with $g = [h_1, h_2]$. In order to find such $S$, take $d_0 = |G| + 1$. Then on any path from the root $\varepsilon$ to a leaf $v \in \{0,1\}^{d_0}$ in the complete binary tree of depth $d_0$, there must be vertices $u_v, t_v$ with $|u_v| < |t_v|$ (i. e., $u_v$ is a prefix of $t_v$ and the latter is a prefix of $v$) and $g_{d_0,u_v} = g_{d_0,t_v}$ in $G$. Let $S$ be just the union over all $g_{d_0,u}$ with $u$ being a prefix of some $t_v$. Using this $S$ (which can be hard-wired in our algorithm for

computing the $g_{d,v}$), the uniformity condition can be seen as follows: for each $g \in S$ fix $h_1, h_2 \in S$ with $g = [h_1, h_2]$ (also hard-wired in the algorithm). We can define new $g_{d,v}$ for the uniform SENS condition using this recursion by starting with some arbitrary fixed $g_\varepsilon = g_{d,\varepsilon} \in S$ for all $d$. Now it is clear that $g_{d,v}$ can be computed in DLINTIME from $v$ – actually by a finite state automaton (note that it is independent of $d$): the states are just the elements of $S$ with initial state $g_\varepsilon$. If the current state is $g = [h_1, h_2]$ and the next input bit from $v$ is 0, then the next state is $h_1$, otherwise, it is $h_2$. □

By Lemma 5.3 and Lemma 5.7, every group having a subgroup with a finite, non-solvable quotient is uniformly SENS. Since every free group projects to a finite simple group, we get:

Corollary 5.10. *If $F_n$ is a finitely generated free group of rank $n \geq 2$, then $F_n$ is uniformly SENS.*

This result was essentially shown by Robinson [54], who showed that the word problem of a free group of rank two is $\mathsf{NC}^1$-hard. He used a similar commutator approach as Barrington. One can prove Corollary 5.10 also directly by exhibiting a free subgroup of infinite rank whose generators are easily computable. For example, in $F_2 = \langle x_0, x_1 \rangle$ take $g_v = x_0^{-v} x_1 x_0^v$ for $v \in \{0,1\}^d$ viewing the string $v$ as a binary encoded number (the other $g_v$ for $v \in \{0,1\}^{<d}$ are then defined by the commutator identity in Definition 5.1), and appropriately padding with 1's. It is even possible to take the $g_v$ of constant length: consider a free group $F = \langle x_0, x_1, x_2 \rangle$, and the elements $g_v = x_{v \bmod 3}$ with $v$ read as the binary representation of an integer. It is easy to see that the nested commutator $g_\varepsilon$ is non-trivial.

### 5.2 Further examples of (not) SENS groups

*Example 5.11.* Here is a finitely generated group that is not solvable, has decidable word problem, but is not SENS. The construction is inspired from [63].

Start with the trivial group $H_0 = 1$ and set $H_{n+1} = H_n \wr \mathbb{Z}$. We have a natural embedding $H_0 \leq H_1$, which induces for all $n$ an embedding $H_n \leq H_{n+1}$. We set $H = \bigcup_{n \geq 0} H_n$, and denote by $x_0, x_1, \ldots$ the generators of $H$, starting with $\mathbb{Z} = \langle x_0 \rangle$. In particular, $H_d := \langle x_0, \ldots, x_d \rangle$ is solvable of class precisely $d$ whereas $H$ is non-solvable.

For an injective function $\tau \colon \mathbb{N} \to \mathbb{N}$ to be specified later, consider in the *unrestricted* wreath product $H^{\mathbb{Z}} \rtimes \mathbb{Z}$ the subgroup $G$ generated by the following two elements:

- the generator $t$ of $\mathbb{Z}$ and
- the function $f \colon \mathbb{Z} \to H$ defined by $f(\tau(n)) = x_n$ and all other values being 1.

We make the assumption that $\tau$ has the following property: For every integer $z \in \mathbb{Z} \setminus \{0\}$ there is at most one pair $(m, i) \in \mathbb{N} \times \mathbb{N}$ with $z = \tau(m) - \tau(i)$. For instance, the mapping $\tau(n) = 2^n$ has this property.

Let us define the conjugated mapping $f_i = t^{\tau(i)} f t^{-\tau(i)} \in G$. We have $f_i(0) = x_i$ and more generally $f_i(\tau(m) - \tau(i)) = x_m$ (and $f_i^{-1}(\tau(m) - \tau(i)) = x_m^{-1}$) for all $m$. Consider now a product $g = f_{i_1}^{\alpha_1} \cdots f_{i_k}^{\alpha_k}$ ($\alpha_1, \ldots, \alpha_k \in \{-1, 1\}$). We get $g(0) = x_{i_1}^{\alpha_1} \cdots x_{i_k}^{\alpha_k}$. For a position $z \in \mathbb{Z} \setminus \{0\}$ which is not a difference of two different $\tau$-values we have $g(z) = 1$. For all other non-zero positions $z$ there is a unique pair $(m, i)$ such that $z = \tau(m) - \tau(i)$, which yields $g(z) = x_m^e$, where $e$ is the sum of those $\alpha_j$ such that $i_j = i$. Hence, the commutator $[g, h]$ of two mappings $g = f_{i_1}^{\alpha_1} \cdots f_{i_k}^{\alpha_k}$ and $h = f_{j_1}^{\beta_1} \cdots f_{j_l}^{\beta_l}$ satisfies $[g, h](0) = [x_{i_1}^{\alpha_1} \cdots x_{i_k}^{\alpha_k}, x_{j_1}^{\beta_1} \cdots x_{j_l}^{\beta_l}]$ and $[g, h](z) = 0$ for all $z \in \mathbb{Z} \setminus \{0\}$. Hence, $G$ contains the restricted wreath product $[H, H] \wr \mathbb{Z}$, so in particular is infinite and non-solvable; and $G'$ contains the restricted direct product $[H, H]^{(\mathbb{Z})}$.

We now assume that $\tau$ grows superexponentially (take for instance $\tau(n) = 2^{n^2}$). Note that if $k \in \mathbb{Z}$ is not of the form $\tau(i) - \tau(j)$ for some $i, j \in \mathbb{N}$, then $t^k f t^{-k}$ and $f$ commute. It follows that the intersection of $G''$ with the ball of radius

$R$ in $G$ is contained in $[H_d, H_d]^{\mathbb{Z}}$ for $d$ growing sublogarithmically in $R$ (more precisely as $O(\sqrt{\log R})$), and in particular does not contain a nested non-trivial commutator of depth $\Omega(\log R)$. This implies that $G$ is not SENS.

Furthermore, if $\tau$ is computable, then $\mathrm{WP}(G)$ is decidable: given a word $w \in \{t^{\pm 1}, f^{\pm 1}\}^*$, compute its exponent sum in the letters $t^{\pm 1}$ and $f^{\pm 1}$ (which must both vanish if $w =_G 1$) and the coordinates $-|w|, \dots, |w|$ of its image in $H^{\mathbb{Z}}$. Each of these coordinates belongs to a finitely iterated wreath product $\mathbb{Z} \wr \cdots \wr \mathbb{Z}$, in which the word problem is decidable (again by counting exponents and computing coordinates).

*Example 5.12.* Here is an example of a f.g. non-SENS group which is uniformly SENS in a larger group. We continue on the notation of Example 5.11.

Consider the non-SENS group $G = \langle t, f \rangle$ from Example 5.11. The reason that $G$ fails to be SENS is the following: there are elements $y_i \in G$ ($i \geq 0$) such that a non-trivial depth-$d$ nested commutator may uniformly be constructed using $y_0, \dots, y_{d-1}$, but the $y_i$ have length growing superexponentially in $i$.

Essentially by the same construction as in Example 5.11 one can embed $G$ as a heavily distorted subgroup in a finitely generated subgroup $\widetilde{G} := \langle t, f, \tilde{t}, \tilde{f} \rangle$ of the unrestricted wreath product $G^{\mathbb{Z}} \rtimes \mathbb{Z}$, thereby bringing the $y_i$ back to exponential length: the elements $t, f$ are the generators of $G$, seen as elements of $G^{\mathbb{Z}}$ supported at 0; $\tilde{t}$ is the generator of $\mathbb{Z}$; and $\tilde{f} \in G^{\mathbb{Z}}$ takes value $y_i$ at $2^i$. Then $G$ is uniformly SENS in $\widetilde{G}$, since the $[y_i, y_j]$ are expressible as words of length $2^{O(i+j)}$ in $\tilde{f}, \tilde{t}$, and their inverses.

The following technical result will be used to prove that weakly branched groups and Thompson's group $F$ are uniformly SENS.

Proposition 5.13. *Let $G$ be a finitely generated group with the standard generating set $\Sigma$. Moreover, let $h_d$ ($d \in \mathbb{N}$) be words over $\Sigma$ with $|h_d| \in 2^{O(d)}$ and such that given $1^d$ and a binary coded number $i$ with $O(d)$ bits one can compute in* DLINTIME *the $i$-th letter of $h_d$. Assume that $H = \langle h_0, h_1, \dots \rangle$ acts on a tree of words $X^*$ (where $X$ is not necessarily finite), and that $X$ contains pairwise distinct elements $v_{-1}, v, v_1$ such that*

- *$h_d$ fixes all of $X^* \setminus v^d X^*$, and*
- *$(v^d v_{-1})^{h_d} = v^{d+1}$ and $(v^{d+1})^{h_d} = v^d v_1$.*

*Then $H$ is uniformly SENS in $G$, so in particular $G$ is uniformly SENS. Moreover, if $H$ is finitely generated and the $h_d$ are words over the generators of $H$, then $H$ is uniformly SENS.*

Proof. For non-negative integers $d, q$ and $s \in \{-1, 1\}$, consider the following elements $g_{d,s,q}$, defined inductively:

$$g_{0,s,q} = h_q, \qquad g_{d,s,q} = [g_{d-1,-1,0}^{s},\, g_{d-1,1,q+1}] \text{ if } d > 0.$$

We claim that $g_{d,1,0} \neq_G 1$. This implies the proposition: By definition $g_{d,1,0}$ is a $d$-fold nested commutator of words of the form $h_r^{\pm 1}$ for various $r \leq d$. It is easy to see that given $v \in \{0,1\}^d$, the index $r_v$ corresponding to the leaf of the commutator tree that is indexed by $v$ is computable in DLINTIME and by the hypothesis of the proposition $h_{r_v}$ is DLINTIME-computable.

Thus, it remains to show that $g_{d,1,0}$ is non-trivial. Indeed, we claim that, for $d > 0$, the element $g_{d,s,q}$ acts only on the subtrees below $v^{d+q}$ and $v^{d-1}v_s$, and furthermore acts as $h_{d+q}$ on the subtree below $v^{d+q}$.

We prove this claim by induction on $d$. Recall that for $g \in \mathrm{Aut}(X^*)$ and a node $w \in X^*$ we write $w * g$ for the element of $\mathrm{Aut}(X^*)$ that acts as $g$ on the subtree $wX^*$ and trivially elsewhere. Note that a conjugate $(w * g)^h$ with $h \in \mathrm{Aut}(X^*)$ can be written as $(w * g)^h = w^h * g'$ for some $g' \in \mathrm{Aut}(X^*)$. With this notation, we may write $h_r = v^r * k_r$

for $k_r = h_r@v^r \in \mathrm{Aut}(X^*)$. Our claim becomes ($\Box$ represents an arbitrary element of $\mathrm{Aut}(X^*)$ that is not important)

$$g_{d,s,q} = (v^{d+q} * k_{d+q})(v^{d-1}v_s * \Box).$$

For $d = 1$ we have

$$g_{1,s,q} = [h_0^s, h_{1+q}] = \left(h_{1+q}^{h_0^s}\right)^{-1} h_{1+q} = \left((v^{1+q} * k_{1+q})^{h_0^s}\right)^{-1} (v^{1+q} * k_{1+q}).$$

Moreover, the conjugate $(v^{1+q} * k_{1+q})^{h_0^s}$ is of the form $(v^{1+q})^{h_0^s} * \Box = v_s * \Box$ and we get $g_{1,s,q} = (v_s * \Box)^{-1}(v^{1+q} * k_{1+q}) = (v^{1+q} * k_{1+q})(v_s * \Box)$.

Consider now $d > 1$. By induction, $g_{d-1,-1,0} = (v^{d-1} * k_{d-1})(v^{d-2}v_{-1} * \Box)$ and $g_{d-1,1,q+1} = (v^{d+q} * k_{d+q})(v^{d-2}v_1 * \Box)$. Now $v^{d-2}v_{-1} * f$, $v^{d-1} * g$, and $v^{d-2}v_1 * h$ commute for all $f, g, h \in \mathrm{Aut}(X^*)$ since they act non-trivially on disjoint subtrees. We get

$$g_{d,s,q} = [g_{d-1,-1,0}^s, g_{d-1,1,q+1}] = [v^{d-1} * k_{d-1}^s, v^{d+q} * k_{d+q}] = (v^{d-1}v_s * \Box)(v^{d+q} * k_{d+q})$$

using arguments as for the case $d = 1$. $\Box$

Theorem 5.14. *Let $G$ be a finitely generated group with $G \wr H \leq G$ for some non-trivial group $H$. Then $G$ is uniformly SENS.*

Proof. By possibly replacing $H$ with a cyclic subgroup, we can assume that $H = \mathbb{Z}$ or $H = \mathbb{Z}/p$ for some $p \in \mathbb{Z}$. Moreover, we can assume that $p \geq 3$: if $p = 2$, we can use the associativity of the permutational wreath product: $G \wr (\mathbb{Z}/2 \wr \mathbb{Z}/2) = (G \wr \mathbb{Z}/2) \wr \mathbb{Z}/2 \leq G \wr \mathbb{Z}/2 \leq G$. Thus, since $\mathbb{Z}/2 \wr \mathbb{Z}/2$ contains an element of order 4, we have $G \wr \mathbb{Z}/4 \leq G$. Hence, we have $G \wr H \leq G$ for $H = \mathbb{Z}$ or $H = \mathbb{Z}/p$ with $p \geq 3$. Let $t$ be a generator of $H$ and $\Sigma$ be a standard generating set for $G$. W. l. o. g. we can assume that $t \in \Sigma$.

Now, consider the endomorphism $\sigma : G \to G$ given by the embedding $G \wr H \leq G$. After padding with the appropriate number of 1's, we can view $\sigma$ as a substitution $\sigma : \Sigma \to \Sigma^{2^\lambda}$ for some constant $\lambda$. We then define words $h_d = \sigma^d(t)$ for all $d \in \mathbb{N}$, and note that $|h_d| = 2^{\lambda d}$. It is straightforward to see that on input of $1^d$ and a binary coded number $i$ one can compute in DLINTIME the $i$-th letter of $h_d$. Moreover, it follows that $\langle h_0, \ldots, h_k \rangle$ is the $k$-fold iterated wreath product of cyclic groups and so $\langle h_0, h_1, \ldots \rangle \cong (\cdots \wr \mathbb{Z}) \wr \mathbb{Z}$ or $\langle h_0, h_1, \ldots \rangle \cong (\cdots \wr (\mathbb{Z}/p)) \wr (\mathbb{Z}/p)$, which acts on the rooted tree $X^*$ with $X = H$ in the canonical way. We then apply Proposition 5.13 with $(v_{-1}, v, v_1) = (-1, 0, 1)$ (or $(v_{-1}, v, v_1) = (p-1, 0, 1)$). $\Box$

As an immediate consequence of Theorem 5.14 and Lemma 3.1, we obtain:

Corollary 5.15. *Thompson's groups $F < T < V$ are uniformly SENS.*

One can also show Corollary 5.15 directly without using Proposition 5.13. Consider the infinite presentation (1). From the relations $x_i^{-1}x_k x_i = x_{k+1}$ $(i < k)$ the reader can easily check that $g = x_3 x_2^{-1}$ satisfies the identity

$$g = [g, g^{x_0^{-1}}]^{x_1} = [g^{x_1}, g^{x_0^{-1}x_1}].$$

Nesting this identity $d$ times and pushing conjugations to the leaf level of the resulting tree yields the words $g_{d,v}$. More precisely, let us define words $c_v$ ($v \in \{0,1\}^*$) by $c_\varepsilon = \varepsilon$, $c_{v0} = x_1 c_v$, and $c_{v1} = x_0^{-1}x_1 c_v$. We then define $g_{d,v} = g^{c_v}$ for $v \in \{0,1\}^{\leq d}$ and immediately get $g_{d,v} = [g_{d,v0}, g_{d,v1}]$ in $F$. Clearly, the word $c_v$ can be computed in $\mathsf{DTIME}(O(|v|))$. Hence, $g_{d,v}$ can be computed in $\mathsf{DTIME}(O(d))$.

Theorem 5.16. *Let $G$ be a weakly branched self-similar group, and assume that it admits a finitely generated branching subgroup $K$. Then $K$ and hence $G$ are uniformly SENS.*

Proof. Let $K$ be a finitely generated branching subgroup of $G$ and let $X^*$ be the tree on which $G$ acts. Let $\varphi$ as in (2). First, we may find an element $k \in K$ and a vertex $v \in X^*$ such that $v, v_{-1} \coloneqq v^{k^{-1}}$, and $v_1 \coloneqq v^k$ are pairwise distinct. Indeed $K$ contains an element $k \neq 1$. If $k$ has order $> 2$ (possibly $\infty$), then there is a vertex $v$ on which it acts as a cycle of length $> 2$. If $k^2 = 1$, then take a vertex $v$ with $v^k \neq v$. Then the orbit of $vv$ under $k \cdot (v * k)$ has length four, so we only have to replace $k$ by $k \cdot (v * k)$ and $v$ by $vv$. After replacing $X$ by $X^{|v|}$, we can assume that $v_{-1}, v, v_1 \in X$.

Since $\varphi(K)$ contains $K^X$, there exists an endomorphism $\sigma$ of $K$, given on generators of $K$ by $\sigma(g) = \varphi^{-1}(1, \dots 1, g, 1, \dots, 1)$ with the unique $g$ in position $v$. We fix a standard generating set $\Sigma$ for $K$ and express $\sigma$ as a substitution $\sigma\colon \Sigma \to \Sigma^*$. By padding its images with 1's, we may assume that $\sigma$ maps every generator to a word of length $2^\mu$ for some fixed $\mu$. Also without loss of generality, we may assume that the $k$ from the previous paragraph is a generator. In particular, the words $h_d = \sigma^d(k) \in \Sigma^*$ have length $2^{\mu d}$, and the letter at a given position of $h_d$ can be computed in $\mathsf{DTIME}(O(d))$. We then apply Proposition 5.13. □

By Lemma 3.3 and Theorem 5.16 the Grigorchuk group is uniformly SENS. For this special case we want to explore an alternative (and simpler) proof: indeed, we show that there exist non-trivial nested commutators of arbitrary depth with individual entries of bounded (and not merely exponentially-growing) length and computable in DLINTIME:

Proposition 5.17. *Consider in the Grigorchuk group $G = \langle a, b, c, d\rangle$ the elements*

$$x = (abad)^2 \quad \text{and} \quad y = x^b = babadabac.$$

*Define recursively elements $z_v \in \{x, y, x^{-1}, y^{-1}\}$ for all $v \in \{0,1\}^*$ as follows:*

- *$z_\varepsilon = x$;*
- *if $z_v$ is defined, then we define $z_{v0}$ and $z_{v1}$ according to the following table:*

| $z_v$ | $z_{v0}$ | $z_{v1}$ |
|---|---|---|
| $x$ | $x^{-1}$ | $y^{-1}$ |
| $x^{-1}$ | $y^{-1}$ | $x^{-1}$ |
| $y$ | $y$ | $x$ |
| $y^{-1}$ | $x$ | $y$ |

*For every $d \in \mathbb{N}$ and $v \in \{0,1\}^{\leq d}$ let $g_{d,v} = z_v$ for $|v| = d$ and $g_{d,v} = [g_{v0}, g_{v1}]$ if $|v| < d$. We then have $g_{d,\varepsilon} \neq 1$ in $G$. In particular, $G$ is uniformly SENS.*

Proof. That $x \neq 1 \neq y$ is easy to check by computing their action on the third level of the tree. Now the following equations are easy to check in $G$:

$$\begin{aligned}
[x, y] &= \langle\!\langle 1, \langle\!\langle 1, y^{-1}\rangle\!\rangle\rangle\!\rangle,\\
[x^{-1}, y^{-1}] &= \langle\!\langle 1, \langle\!\langle 1, x\rangle\!\rangle\rangle\!\rangle,\\
[y, x] &= \langle\!\langle 1, \langle\!\langle 1, y\rangle\!\rangle\rangle\!\rangle,\\
[y^{-1}, x^{-1}] &= \langle\!\langle 1, \langle\!\langle 1, x^{-1}\rangle\!\rangle\rangle\!\rangle.
\end{aligned}$$

In other words: $[z_{v0}, z_{v1}] = \langle\!\langle 1, \langle\!\langle 1, z_v\rangle\!\rangle\rangle\!\rangle$. The checks are tedious to compute by hand, but easy in the GAP package FR (note that vertices are numbered from 1 in GAP and from 0 here):

```
gap> LoadPackage("fr");
gap> AssignGeneratorVariables(GrigorchukGroup);
gap> x := (a*b*a*d)^2; y := x^b;
gap> Assert(0,Comm(x,y) = VertexElement([2,2],y^-1));
gap> Assert(0,Comm(x^-1,y^-1) = VertexElement([2,2],x));
gap> Assert(0,Comm(y,x) = VertexElement([2,2],y));
gap> Assert(0,Comm(y^-1,x^-1) = VertexElement([2,2],x^-1));
```

We wish to prove that $g_{d,\varepsilon} \neq 1$ in $G$. Now the equation $[z_{v0}, z_{v1}] = \langle\!\langle 1, \langle\!\langle 1, z_v \rangle\!\rangle \rangle\!\rangle$ immediately implies that $g_{d,v}$ acts as $z_v$ on the subtree below vertex $1^{2(d-|v|)}$ and trivially elsewhere. In particular, $g_{d,\varepsilon}$ acts as $z_\varepsilon = x \neq 1$ on the subtree below vertex $1^{2d}$ and is non-trivial.

With this definition, the $g_{d,v}$ satisfy the definition of a SENS group. Moreover, given some $v \in \{0,1\}^d$, $g_{d,v}$ can be computed in time $O(d)$ by a deterministic finite state automaton with state set $\{x^{\pm 1}, y^{\pm 1}\}$. □

## 6 EFFICIENTLY NON-SOLVABLE GROUPS HAVE $\mathsf{NC}^1$-HARD WORD PROBLEM

We are ready to state and prove our generalization of Barrington's theorem, namely that SENS groups have $\mathsf{NC}^1$-hard word problems, both in the non-uniform and uniform setting. We start with the non-uniform setting.

Theorem 6.1. *Let $G$ be strongly efficiently non-solvable and let $\Sigma$ be a finite standard generating set for $G$. Then every language in $\mathsf{NC}^1$ can be recognized by a family of $(G,\Sigma)$-programs of polynomial length. In particular, $\mathrm{WP}(G)$ is hard for $\mathsf{NC}^1$ under projection reductions as well as $\mathsf{AC}^0$-many-one-reductions.*

Note that for the second statement we need the padding letter 1 in the generating set for $G$; otherwise, we get a $\mathsf{TC}^0$-many-one reduction.

The proof of Theorem 6.1 essentially follows Barrington's proof that the word problem of finite non-solvable groups is $\mathsf{NC}^1$-hard [4]. The crucial observation here is that it suffices to construct for every gate $v$ only one $G$-program (plus one for the inverse) which evaluates to $g_{d,v}$ or to 1 depending on the truth value $v$ evaluates to, where $g_{d,v}$ is from Definition 5.1.

Also note that Barrington uses conjugates of commutators in his proof and iterates this process. However, since $z^{-1}[x,y]z = [z^{-1}xz, z^{-1}yz]$ in every group, the conjugating elements can be pushed through to the inner-most level.

Proof. Given a language $L$ in $\mathsf{NC}^1$, we start by constructing a family of $G$-programs for $L$. For this let $(C_n)_{n\in\mathbb{N}}$ be an $\mathsf{NC}^1$ circuit family for $L$. Let us fix an input length $n$ and write $C = C_n$. Since $\mathsf{NC}^1$ is closed under complementation, we can assume that for every input word $x \in \{0,1\}^n$, we have $x \in L$ if and only if the output gate of the circuit $C$ evaluates to 0 on input $x$. By Lemma 4.6 we may assume that $C$ is a balanced nand-tree-circuit of depth $d \in O(\log n)$ with each leaf labelled by a possibly negated input variable or constant via the input mapping $q_n\colon \{0,1\}^d \to [1..n] \times \{0,1\} \times \{0,1\}$. All non-leaf gates are nand-gates.

For each gate $v \in \{0,1\}^{\leq d}$ let $g_v = g_{d,v}$ as in Definition 5.1. We construct two $G$-programs $P_v$ and $P_v^{-1}$ (both of input length $n$) such that for every input $x \in \{0,1\}^n$ ($x$ is taken as the input for $C$, $P_v$, and $P_v^{-1}$) we have

$$P_v[x] =_G \begin{cases} g_v & \text{if } v \text{ evaluates to 1,} \\ 1 & \text{if } v \text{ evaluates to 0,} \end{cases} \tag{3}$$

and $P_v^{-1}[x] = P_v[x]^{-1}$ in $G$. Notice that we have $g_v P_v^{-1}[x] = g_v$ if $v$ evaluates to 0 and $g_v P_v^{-1}[x] = 1$, otherwise. Thus, $g_v P_v^{-1}$ is a $G$-program for the "negation" of $P_v$. Moreover, by Eq. (3), $P_\varepsilon$ evaluates to 1 on input $x$ if and only if the output gate evaluates to 0 which by our assumption was the case if and only if $x \in L$.

The construction of the $P_v$ and $P_v^{-1}$ is straightforward: For an input gate $v \in \{0,1\}^d$ we simply define $P_v$ to be a $G$-program evaluating to $g_v$ or 1 – in which case it evaluates to which element depends on $q_n(v)$. More precisely, write $g_v = a_1 \cdots a_m$ with $a_i \in \Sigma$. If $q_n(v) = \langle i, a, b\rangle$ for $i \in [1..n]$ and $a, b \in \{0,1\}$, we set $P_v = \langle i, a_1^a, a_1^b\rangle \cdots \langle i, a_m^a, a_m^b\rangle$ and $P_v^{-1} = \langle i, a_m^{-a}, a_m^{-b}\rangle \cdots \langle i, a_1^{-a}, a_1^{-b}\rangle$. For a nand-gate $v$ with inputs from $v0$ and $v1$, we define

$$P_v = g_v[P_{v1}, P_{v0}] = g_v P_{v1}^{-1} P_{v0}^{-1} P_{v1} P_{v0},$$
$$P_v^{-1} = [P_{v0}, P_{v1}] g_v^{-1} = P_{v0}^{-1} P_{v1}^{-1} P_{v1} P_{v1} g_v^{-1},$$

where the $g_v$ and $g_v^{-1}$ represent constant $G$-programs evaluating to $g_v$ and $g_v^{-1}$, respectively, irrespective of the actual input (such constant $G$-programs consist of triples of the form $\langle 1, a, a\rangle$ for $a \in \Sigma$). These constant $G$-programs are defined via the commutator identities $g_v = \left[g_{v0},\, g_{v1}\right]$ for $v \in \{0,1\}^{<d}$ in Definition 5.1.

Clearly, by induction we have $P_v[x]^{-1} = P_v^{-1}[x]$ in $G$ (for every input $x$). Let us show that Eq. (3) holds: For an input gate $v \in \{0,1\}^d$, Eq. (3) holds by definition. Now, let $v \in \{0,1\}^{<d}$. Then, by induction, we have the following equalities in $G$:

$$P_v[x] = g_v[P_{v1}[x], P_{v0}[x]] = \begin{cases} g_v & \text{if } v0 \text{ or } v1 \text{ evaluates to 0,} \\ g_v[g_{v1},\, g_{v0}] & \text{if } v0 \text{ and } v1 \text{ evaluate to 1,} \end{cases}$$
$$= \begin{cases} g_v & \text{if } v \text{ evaluates to 1,} \\ 1 & \text{if } v \text{ evaluates to 0.} \end{cases}$$

Note that $[g_{v1},\, g_{v0}] = [g_{v0},\, g_{v1}]^{-1} = g_v^{-1}$ for the last equality. Thus, $P_v$ satisfies Eq. (3). For $P_v^{-1}$ the analogous statement can be shown with the same calculation. For a leaf $v \in \{0,1\}^d$, we have $|g_v| \in 2^{O(d)} = n^{O(1)}$ by Condition (a) from Definition 5.1 (recall that $d \in O(\log n)$). Hence, $P_v^{-1}$ and $P_v$ have polynomial length in $n$. Finally, also $P_\varepsilon$ has polynomial length in $n$ (with the same argument as for $g_\varepsilon$; see the remark after Definition 5.1).

The fact that $\mathrm{WP}(G)$ is $\mathsf{NC}^1$-hard under projection reductions as well as $\mathsf{AC}^0$-many-one-reductions follows now form Remark 4.9. □

*Remark 6.2.* The above construction also shows that from a given Boolean formula (i.e., a tree-like circuit that is given as an expression) $F$ with variables $x_1, \ldots, x_n$ one can compute in LOGSPACE a $G$-program $P$ with input length $n$ such that for every $x = a_1 \cdots a_n \in \{0,1\}^n$ we have $P[x] = 1$ if and only if $F$ evaluates to true when variable $x_i$ receives the truth value $a_i$ for $1 \le i \le n$. To show this, one first has to balance $F$ in the sense that $F$ is transformed into an equivalent Boolean formula of depth $O(\log |F|)$. This can be done even in $\mathsf{TC}^0$ [18].

Theorem 6.3. *Let $G$ be uniformly strongly efficiently non-solvable and $\Sigma$ be a finite standard generating set of $G$. Then every language in* ALOGTIME *can be recognized by a uniform family of polynomial length $(G, \Sigma)$-programs. In particular,* $\mathrm{WP}(G)$ *is hard for* ALOGTIME *under uniform projection reductions (thus, also under* DLOGTIME*-reductions).*

Notice that again for this theorem we need the padding letter 1 in $\Sigma$ and that all letters of $\Sigma$ are encoded using the same number of bits; otherwise, we get a $\mathsf{TC}^0$-many-one reduction.

The proof of Theorem 6.3 is conceptually simple, but the details are quite technical: We know that ALOGTIME is the same as DLOGTIME-uniform $\mathsf{NC}^1$, so we apply the construction of Theorem 6.1. By a careful padding with trivial

$G$-programs, we can ensure that from the binary representation of some index $i$, we can read in DLOGTIME the input gate of the $\mathsf{NC}^1$-circuit on which the $i$-th instruction in the $G$-program depends (this is the main technical part of the proof). Then the theorem follows easily from the requirements of being uniformly SENS and from the special type of DLOGTIME-uniformity of the circuit shown in Lemma 4.7.

Proof. By Theorem 6.1, we know that every language $L$ in ALOGTIME can be recognized by a family of polynomial length $(G, \Sigma)$-programs. It remains to show that the construction of the $G$-programs is uniform. In order to do so, we refine the construction of Theorem 6.1.

Fix a constant $\mu$ such that for all $v \in \{0,1\}^d$ the word $g_v = g_{d,v}$ has length $2^{\mu d}$. We start with an ALOGTIME-machine $M$. By Lemma 4.7, we can assume that the balanced nand-tree-circuit family $(C_n)_{n\in\mathbb{N}}$ in the proof of Theorem 6.1 is DLOGTIME-uniform in the sense that the depth function $1^n \mapsto 1^{d(n)}$ as well as the input mapping $q$ from Definition 4.5 can be strongly computed in DLOGTIME. Fix an input length $n$ and let $d = d(n)$ be the depth of the circuit $C = C_n$. From $1^n$ we can strongly compute $1^d$ in DLOGTIME by the above assumptions.

We now follow the recursive definition of the $G$-programs $P_v$ and $P_v^{-1}$ from the proof of Theorem 6.1. In order to have a nicer presentation, we wish that all $G$-programs corresponding to one layer of the circuit have the same length. To achieve this, we also define the constant $G$-programs $g_v$ and $g_v^{-1}$ precisely (which evaluate to the recursive commutators from Definition 5.1). Moreover, for each $v \in \{0,1\}^{\leq d}$ we introduce a new constant $G$-program $1_v$ of the same length as $g_v$ which evaluates to 1 in $G$. For $v \in \{0,1\}^d$ the program $1_v$ is the instruction $\langle 1,1,1\rangle$ repeated $2^{\mu d}$ times. The programs $1_v$ are only there for padding reasons and $1_u$ and $1_v$ are the same for $|u| = |v|$.

Now the $G$-programs $P_v$, $P_v^{-1}$, $g_v$, $g_v^{-1}$, and $1_v$ corresponding to a gate $v \in \{0,1\}^{<d}$ are defined as follows (note that each of these programs consists of 8 blocks):

$$P_v = g_{v0}^{-1} g_{v1}^{-1} g_{v0}\, g_{v1}\;\; P_{v1}^{-1} P_{v0}^{-1} P_{v1}\, P_{v0} \tag{4}$$

$$P_v^{-1} = P_{v0}^{-1} P_{v1}^{-1} P_{v0}\, P_{v1}\;\; g_{v1}^{-1} g_{v0}^{-1} g_{v1}\, g_{v0} \tag{5}$$

$$g_v = g_{v0}^{-1} g_{v1}^{-1} g_{v0}\, g_{v1}\;\; 1_{v0}\, 1_{v0}\, 1_{v0}\, 1_{v0} \tag{6}$$

$$g_v^{-1} = g_{v1}^{-1} g_{v0}^{-1} g_{v1}\, g_{v0}\;\; 1_{v0}\, 1_{v0}\, 1_{v0}\, 1_{v0} \tag{7}$$

$$1_v = 1_{v0}\, 1_{v0}\, 1_{v0}\, 1_{v0}\;\; 1_{v0}\, 1_{v0}\, 1_{v0}\, 1_{v0}. \tag{8}$$

Clearly, these $G$-programs all evaluate as described in the proof of Theorem 6.1 and all programs corresponding to one layer have the same length. Moreover, for $v \in \{0,1\}^{<d}$ with $|v| = c$ the length of the $G$-program $g_v$ is exactly $2^{\mu d+3(d-c)}$ and, thus, also the length of $P_v$ and $P_v^{-1}$ is exactly $2^{\mu d+3(d-c)}$.

For the $G$-program $P_\varepsilon$ (which has length $2^{(\mu+3)d}$) we can prove the uniformity condition: Given the string $1^n$ and a binary coded integer $i \in [0..2^{(\mu+3)d} - 1]$ with $(\mu+3)d \in O(\log n)$ bits, we want to compute in DLOGTIME the $i$-th instruction in $P_\varepsilon$, where $P_\varepsilon$ is the $G$-program assigned to the $n$-input circuit. Note that DLOGTIME means time $O(\log n)$ (due to the input $1^n$). Since we have computed $1^d$ already in DLOGTIME, we can check in DLOGTIME whether $i$ has indeed $(\mu+3)d$ bits.

Next, given $i$ and $1^n$, the DLOGTIME-machine goes over the first $3d$ bits of $i$ and thereby computes an input gate $v \in \{0,1\}^d$ of $C$ bit by bit together with one of the five symbols $\sigma \in \{P_*, P_*^{-1}, g_*, g_*^{-1}, 1_*\}$. The meaning of $v$ and $\sigma$ is that $\sigma[* \to v]$ (which is obtained by replacing $*$ by $v \in \{0,1\}^d$ in $\sigma$) is the $G$-program to which the $i$-th instruction in $P_\varepsilon$ belongs to. The approach is similar to [60, Theorem 4.52]. We basically run a deterministic finite state transducer with states $P_*, P_*^{-1}, g_*, g_*^{-1}, 1_*$ that reads three bits of $i$ and thereby outputs one bit of $v$. We start with $\sigma = P_*$. Note

each of the $G$-programs $P_v$, $P_v^{-1}$, $g_v$, $g_v^{-1}$, $1_v$ for $|v| < d$ consists of $8 = 2^3$ blocks of equal length. The next three bits in $i$ determine to which block we have to descend. Moreover, the block determines the next bit of $v$ and the next state. Let us give an example: assume that the current state $\sigma$ is $P_*$ and $b \in \{0,1\}^3$ is the next 3-bit block of $i$. Recall that $P_v = g_{v0}^{-1} g_{v1}^{-1} g_{v0} g_{v1} P_{v1}^{-1} P_{v0}^{-1} P_{v1} P_{v0}$ for $|v| < d$. The following operations are done:

- If $b = 000$, then print 0 and set $\sigma := g_*^{-1}$ (descend to block $g_{v0}^{-1}$).
- If $b = 001$, then print 1 and set $\sigma := g_*^{-1}$ (descend to block $g_{v1}^{-1}$).
- If $b = 010$, then print 0 and set $\sigma := g_*$ (descend to block $g_{v0}$).
- If $b = 011$, then print 1 and set $\sigma := g_*$ (descend to block $g_{v1}$).
- If $b = 100$, then print 1 and set $\sigma := P_*^{-1}$ (descend to block $P_{v1}^{-1}$).
- If $b = 101$, then print 0 and set $\sigma := P_*^{-1}$ (descend to block $P_{v0}^{-1}$).
- If $b = 110$, then print 1 and set $\sigma := P_*$ (descend to block $P_{v1}$).
- If $b = 111$, then print 0 and set $\sigma := P_*$ (descend to block $P_{v0}$).

For other values of $\sigma$ the behavior of the machine is similar and implements the definitions for $P_v^{-1}$, $g_v$, $g_v^{-1}$, and $1_v$ in (5)–(8).

Assume now that the above DLOGTIME-machine has computed $v \in \{0,1\}^d$ and $\sigma \in \{P_*, P_*^{-1}, g_*, g_*^{-1}, 1_*\}$. If $\sigma = 1_*$, then the $i$-th instruction of $P_\varepsilon$ is the padding instruction $\langle 1,1,1\rangle$. If $\sigma \in \{P_*, P_*^{-1}, g_*, g_*^{-1}\}$, then the machine reads the last $\mu d$ bits of the binary encoding of $i$. These $\mu d$ bits are interpreted as a binary coded position $j$ in $g_{d,v}$ or $g_{d,v}^{-1}$. Assume that $\sigma \in \{P_*, g_*\}$. The machine then computes the $j$-th symbol $a \in \Sigma$ of $g_{d,v}$ in $\mathsf{DTIME}(O(d))$ according to Definition 5.1 (and, thus, in DLOGTIME as $d \in O(\log n)$ and $1^n$ is part of the input) and outputs the instruction $\langle 1, a, a\rangle$ in case $\sigma = g_*$. If $\sigma = P_*$, then $q(1^n, v)$ has to be computed, which can be done in DLOGTIME by Lemma 4.7. If $q(1^n, v) = \langle k, b, c\rangle$ with $k \in [1..n]$ and $b, c \in \{0,1\}$, the machine then outputs the instruction $\langle k, a^b, a^c\rangle$. If $\sigma = \{P_*^{-1}, g_*^{-1}\}$, then we proceed in a similar fashion. Instead of the $j$-th letter of $g_v$ we have to compute the $j$-the letter of $g_v^{-1}$, which is the inverse of the $(2^{\mu d} - j + 1)$-th letter of $g_v$. The binary coding of $2^{\mu d} - j + 1$ can be computed in time $O(\log n)$ (and hence DLOGTIME) since subtraction can be done in linear time. Thus, we have obtained a DLOGTIME-uniform family of $G$-programs for $L$.

The second part of the theorem (that $\mathrm{WP}(G)$ is hard for ALOGTIME under uniform projection reductions) follows again from Remark 4.9. □

Recall that Corollary A from the introduction states that the word problems for the three Thompson's groups $F$, $T$, and $V$ as well as for weakly branched self-similar groups with a finitely generated branching subgroup are hard for ALOGTIME. Now, this is a direct consequence of Corollary 5.15, Theorem 5.16, and Theorem 6.3.

### 6.1 Consequences for linear groups

Here is another application of Theorem 6.3: in [38] it was shown that for every f.g. linear solvable group the word problem belongs to DLOGTIME-uniform $\mathsf{TC}^0$. It was also asked whether for every f.g. linear group the word problem is in DLOGTIME-uniform $\mathsf{TC}^0$ or ALOGTIME-hard (be aware that it might be the case that DLOGTIME-uniform $\mathsf{TC}^0$ = ALOGTIME). We can confirm this. Recall that a group $G$ is called $\mathsf{C}_1$-by-$\mathsf{C}_2$ for group classes $\mathsf{C}_1$ and $\mathsf{C}_2$ if $G$ has a normal subgroup $K \in \mathsf{C}_1$ such that $G/K \in \mathsf{C}_2$.

Theorem 6.4. *For every f.g. linear group the word problem is in* DLOGTIME*-uniform* $\mathsf{TC}^0$ *or* ALOGTIME*-hard. More precisely: let $G$ be a f.g. linear group.*

- *If $G$ is finite solvable, then* $\mathrm{WP}(G)$ *belongs to* DLOGTIME*-uniform* $\mathsf{ACC}^0$.

- *If $G$ is infinite solvable, then* $\mathrm{WP}(G)$ *is complete for* DLOGTIME*-uniform* $\mathsf{TC}^0$ *(via uniform* $\mathsf{AC}^0$ *Turing reductions).*
- *If $G$ is solvable-by-(finite non-solvable), then* $\mathrm{WP}(G)$ *is complete for* ALOGTIME *(via* DLOGTIME *or uniform projection reductions).*
- *In all other cases,* $\mathrm{WP}(G)$ *is* ALOGTIME*-hard and in* LOGSPACE.

Note that we can obtain a similar dichotomy for hyperbolic groups: they are either virtually abelian or contain a non-abelian free subgroup. In the first case, the word problem is in DLOGTIME-uniform $\mathsf{TC}^0$, in the second case it is ALOGTIME-hard.

Proof. Let $G$ be f.g. linear. First of all, by [43, 57], $\mathrm{WP}(G)$ belongs to LOGSPACE. By Tits' alternative [59], $G$ either contains a free subgroup of rank 2 or is virtually solvable (meaning that it has a solvable subgroup of finite index). In the former case, $\mathrm{WP}(G)$ is ALOGTIME-hard by Corollary 5.10 and Theorem 6.3. Let us now assume that $G$ is virtually solvable. Let $K$ be a solvable subgroup of $G$ of finite index. By taking the intersection of all conjugates of $K$ in $G$, we can assume that $K$ is a normal subgroup of $G$. If also $G/K$ is solvable, then $G$ is solvable. Hence, $\mathrm{WP}(G)$ is in DLOGTIME-uniform $\mathsf{ACC}^0$ (if $G$ is finite) or, by [38], complete for DLOGTIME-uniform $\mathsf{TC}^0$ (if $G$ is infinite). Finally, assume that the finite group $G/K$ is non-solvable (thus, $G$ is solvable-by-(finite non-solvable). By Lemmas 5.3 and 5.7, $G$ is uniformly SENS, and Theorem 6.3 implies that $\mathrm{WP}(G)$ is ALOGTIME-hard. Moreover, by [54, Theorem 5.2], $\mathrm{WP}(G)$ is $\mathsf{AC}^0$-reducible to $\mathrm{WP}(K)$ and $\mathrm{WP}(G/K)$. The latter belongs to ALOGTIME and $\mathrm{WP}(K)$ belongs to DLOGTIME-uniform $\mathsf{ACC}^0$ if $K$ is finite and to DLOGTIME-uniform $\mathsf{TC}^0$ if $K$ is infinite (note that $K$ as a finite index subgroup of $G$ is f.g. linear too). In all cases, $\mathrm{WP}(G)$ belongs to ALOGTIME. □

## 7 COMPRESSED WORDS AND THE COMPRESSED WORD PROBLEM

In the rest of the paper we deal with the compressed word problem, which is a succinct version of the word problem, where the input word is given in a compressed form by a so-called straight-line program. In this section, we introduce straight-line programs and the compressed word problem and state a few simple facts. For more details on the compressed word problem see [46].

A *straight-line program* (SLP for short) over the alphabet $\Sigma$ is a triple $\mathcal{G} = (V, \rho, S)$, where $V$ is a finite set of variables such that $V \cap \Sigma = \emptyset$, $S \in V$ is the start variable, and $\rho : V \to (V \cup \Sigma)^*$ is a mapping such that the relation $\{(A, B) \in V \times V : B \text{ occurs in } \rho(A)\}$ is acyclic. For the reader familiar with context free grammars, it might be helpful to view the SLP $\mathcal{G} = (V, \rho, S)$ as the context-free grammar $(V, \Sigma, P, S)$, where $P$ contains all productions $A \to \rho(A)$ for $A \in V$. The definition of an SLP implies that this context-free grammar derives exactly one terminal word, which will be denoted by $\mathrm{val}(\mathcal{G})$. Formally, one can extend $\rho$ to a morphism $\rho : (V \cup \Sigma)^* \to (V \cup \Sigma)^*$ by setting $\rho(a) = a$ for all $a \in \Sigma$. The above acyclicity condition on $\rho$ implies that for $m = |V|$ we have $\rho^m(w) \in \Sigma^*$ for all $w \in (V \cup \Sigma)^*$. We then define $\mathrm{val}_{\mathcal{G}}(w) = \rho^m(w)$ (the string derived from the sentential form $w$) and $\mathrm{val}(\mathcal{G}) = \mathrm{val}_{\mathcal{G}}(S)$.

The word $\rho(A)$ is also called the *right-hand side* of $A$. Quite often, it is convenient to assume that all right-hand sides are of the form $a \in \Sigma$ or $BC$ with $B, C \in V$. This corresponds to the well-known Chomsky normal form for context-free grammars. There is a simple linear time algorithm that transforms an SLP $\mathcal{G}$ with $\mathrm{val}(\mathcal{G}) \neq \varepsilon$ into an SLP $\mathcal{G}'$ in Chomsky normal form with $\mathrm{val}(\mathcal{G}) = \mathrm{val}(\mathcal{G}')$, see e.g. [46, Proposition 3.8].

We define the size of the SLP $\mathcal{G} = (V, \rho, S)$ as the total length of all right-hand sides: $|\mathcal{G}| = \sum_{A \in V} |\rho(A)|$. SLPs offer a succinct representation of words that contain many repeated substrings. For instance, the word $(ab)^{2^n}$ can be produced by the SLP $\mathcal{G} = (\{A_0, \ldots, A_n\}, \rho, A_n)$ with $\rho(A_0) = ab$ and $\rho(A_{i+1}) = A_i A_i$ for $0 \leq i \leq n-1$. It was shown

independently in [32, 50, 53] that one can check in polynomial time whether two given SLPs produce the same string. We need the following upper bound on the length of the word $\mathrm{val}(\mathcal{G})$:

Lemma 7.1 (c.f. [15]). *For every SLP $\mathcal{G}$ we have $|\mathrm{val}(\mathcal{G})| \leq 3^{|\mathcal{G}|/3}$.*

We also need polynomial time algorithms for a few algorithmic problems for SLPs:

Lemma 7.2 ([46, Chapter 3]). *The following problems can be solved in polynomial time, where $\mathcal{G}$ is an SLP over a terminal alphabet $\Sigma$, $a \in \Sigma$, and $p, q \in \mathbb{N}$ (the latter are given in binary notation):*

- *Given $\mathcal{G}$, compute the length $|\mathrm{val}(\mathcal{G})|$.*
- *Given $\mathcal{G}$ and $a$, compute the number $|\mathrm{val}(\mathcal{G})|_a$ of occurrences of $a$.*
- *Given $\mathcal{G}$ and $p$, compute the symbol $\mathrm{val}(\mathcal{G})[p] \in \Sigma$ (in case $0 \leq p < |\mathrm{val}(\mathcal{G})|$ does not hold, the algorithm outputs a special symbol).*
- *Given $\mathcal{G}$ and $p, q$, compute an SLP for the string $\mathrm{val}(\mathcal{G})[p : q]$ (in case $0 \leq p \leq q < |\mathrm{val}(\mathcal{G})|$ does not hold, the algorithm outputs a special symbol).*

Lemma 7.3 (c.f. [46, Lemma 3.12]). *Given a symbol $a_0 \in \Sigma$ and a sequence of morphisms $\varphi_1, \ldots, \varphi_n : \Sigma^* \to \Sigma^*$, where every $\varphi_i$ is given by a list of the words $\varphi_i(a)$ for $a \in \Sigma$, one can compute in* LOGSPACE *an SLP for the word $\varphi_1(\varphi_2(\cdots \varphi_n(a_0) \cdots))$.*

The *compressed word problem* for a finitely generated group $G$ with the finite standard generating set $\Sigma$, CompressedWP$(G, \Sigma)$ for short, is the following decision problem:

**Input:** an SLP $\mathcal{G}$ over the alphabet $\Sigma$.
**Question:** does $\mathrm{val}(\mathcal{G}) = 1$ hold in $G$?

It is an easy observation that the computational complexity of the compressed word problem for $G$ does not depend on the chosen generating set $\Sigma$ in the sense that if $\Sigma'$ is another finite standard generating set for $G$, then CompressedWP$(G, \Sigma)$ is LOGSPACE-reducible to CompressedWP$(G, \Sigma')$ [46, Lemma 4.2]. Therefore we do not have to specify the generating set and we just write CompressedWP$(G)$.

The compressed word problem for $G$ is equivalent to the problem whether a given circuit over the group $G$ evaluates to 1: Take an SLP $\mathcal{G} = (V, \rho, S)$ in Chomsky normal form and built a circuit by taking $V$ is the set of gates. If $\rho(A) = a \in \Sigma$ then $A$ is an input gate that is labelled with the group generator $a$. If $\rho(A) = BC$ with $B, C \in V$ then $B$ is left input gate for $A$ and $C$ is the right input gate for $A$. Such a circuit can be evaluated in the natural way (every internal gate computes the product of its input values) and the circuit output is the value at gate $S$.

From a given SLP $\mathcal{G}$ a PSPACE-transducer can compute the word $\mathrm{val}(\mathcal{G})$. With Lemma 4.1 we get:

Lemma 7.4. *If $G$ is a finitely generated group such that* WP$(G)$ *belongs to* polyL*, then* CompressedWP$(G)$ *belongs to* PSPACE.

## 8 COMPRESSED WORD PROBLEMS FOR WREATH PRODUCTS

In this section we consider regular wreath products of the form $G \wr \mathbb{Z}$. The following result was shown in [46] (for $G$ non-abelian) and [39] (for $G$ abelian). In this section all hardness results are with respect to LOGSPACE reductions.

Theorem 8.1 (c.f. [39, 46]). *If $G$ is a finitely generated group, then*

- *CompressedWP$(G \wr \mathbb{Z})$ is* coNP*-hard if $G$ is non-abelian and*

- *CompressedWP($G \wr \mathbb{Z}$) belongs to* coRP *(complement of randomized polynomial time) if $G$ is abelian.*

In this section, we prove the following result, which improves upon the first statement of Theorem 8.1.

Theorem 8.2. *Let $G$ be a finitely generated non-trivial group.*

- *CompressedWP($G \wr \mathbb{Z}$) belongs to* $\forall\mathsf{LEAF}(\mathrm{WP}(G))$.
- *CompressedWP($G \wr \mathbb{Z}$) is hard for the class* $\forall\mathsf{LEAF}(\mathrm{WP}(G/Z(G)))$.

*In particular, if $Z(G) = 1$, then CompressedWP($G \wr \mathbb{Z}$) is complete for* $\forall\mathsf{LEAF}(\mathrm{WP}(G))$.

Be aware that in the case that $G$ is abelian, $\mathrm{WP}(G/Z(G))$ is the set of all words over the generators, and so $\forall\mathsf{LEAF}(\mathrm{WP}(G/Z(G)))$ consists of only the universal language. Therefore, for abelian $G$, the hardness statement in Theorem 8.2 is trivial.

The proof of the lower bound uses some of the techniques from the paper [45], where a connection between leaf strings and SLPs was established. In Sections 8.1–8.3 we will introduce these techniques. The proof of Theorem 8.2 will be given in Section 8.4.

*Remark 8.3.* Let $G$ be a finite solvable group with composition series $1 = G_0 \leq G_1 \leq \cdots \leq G_r = G$ meaning that $G_{i-1}$ is normal in $G_i$ and $G_i/G_{i-1}$ is cyclic of prime order $p_i$ for $i \in \{1, \ldots, r\}$. In this case, [27, Satz 4.32] implies that $\mathsf{LEAF}(\mathrm{WP}(G)) \subseteq \mathsf{Mod}_{p_1} \cdots \mathsf{Mod}_{p_r}\mathsf{P}$. Thus, using Theorem 8.2 we obtain that CompressedWP($G \wr \mathbb{Z}$) belongs to $\forall\mathsf{Mod}_{p_1} \cdots \mathsf{Mod}_{p_r}\mathsf{P}$. On the other hand, [29, Theorem 2.2] states that $\mathsf{coMod}_m\mathsf{P} \subseteq \mathsf{LEAF}(\mathrm{WP}(G/Z(G)))$ for $m = |G/Z(G)|$; thus, it follows that CompressedWP($G \wr \mathbb{Z}$) is hard for $\forall\mathsf{coMod}_m\mathsf{P}$. Moreover, by [26, Theorem 2.6], $\mathsf{coMod}_m\mathsf{P} = \mathsf{coMod}_k\mathsf{P}$ for $k = \prod_{p|m} p$ where the product runs over all prime divisors of $m$. As the next examples show, there are the extreme cases that CompressedWP($G \wr \mathbb{Z}$) actually belongs to $\forall\mathsf{coMod}_m\mathsf{P}$ and also that it is hard for $\forall\mathsf{Mod}_{p_1} \cdots \mathsf{Mod}_{p_r}\mathsf{P}$ (at least, we give an example for $r = 2$):

- If $G$ is a finite, non-abelian $p$-group (i. e., $p_i = p$ for all $i$), then

$$\mathsf{LEAF}(\mathrm{WP}(G)) \subseteq \mathsf{Mod}_p \cdots \mathsf{Mod}_p\mathsf{P} = \mathsf{Mod}_p\mathsf{P} \subseteq \mathsf{LEAF}(\mathrm{WP}(G))$$

  by [10, Theorem 6.7] and likewise $\mathsf{LEAF}(\mathrm{WP}(G/Z(G))) = \mathsf{Mod}_p\mathsf{P}$. Hence, in this case CompressedWP($G \wr \mathbb{Z}$) is complete for $\forall\mathsf{Mod}_p\mathsf{P}$. More generally, for a finite non-abelian nilpotent group $G$ (i. e., a direct product of $p$-groups) and $m = |G/Z(G)|$, it follows that CompressedWP($G \wr \mathbb{Z}$) is complete for $\forall\mathsf{coMod}_m\mathsf{P}$. This is because by [26, Lemma 2.4] a language $L$ is in $\mathsf{coMod}_m\mathsf{P}$ if and only if it can be written as an intersection $\bigcap_{p|m} L_p$ for languages $L_p \in \mathsf{Mod}_p\mathsf{P}$ for $p|m$.
- Finally, consider the symmetric group on three elements $S_3$. By [29, Example 2.5] we have $\mathsf{LEAF}(\mathrm{WP}(S_3)) = \mathsf{Mod}_3\mathsf{Mod}_2\mathsf{P}$ (also written as $\mathsf{Mod}_3{\oplus}\mathsf{P}$). Since $S_3$ has trivial center, it follows that CompressedWP($S_3 \wr \mathbb{Z}$) is complete for $\forall\mathsf{Mod}_3{\oplus}\mathsf{P}$.

### 8.1 Subsetsum problems

In the following, we will identify a bit string $\alpha = a_1a_2\cdots a_n$ ($a_1, \ldots, a_n \in \{0, 1\}$) with the vector $(a_1, a_2, \ldots, a_n)$. In particular, for another vector $\overline{s} = (s_1, s_2, \ldots, s_n) \in \mathbb{N}^n$ we will write $\alpha \cdot \overline{s} = \sum_{i=1}^{n} a_i \cdot s_i$ for the scalar product. Moreover, we write $\sum \overline{s}$ for the sum $s_1 + s_2 + \cdots + s_n$.

A sequence $(s_1, \ldots, s_n)$ of natural numbers is *super-decreasing* if $s_i > s_{i+1} + \cdots + s_n$ for all $i \in [1..n]$. For example, $(s_1, \ldots, s_n)$ with $s_i = 2^{n-i}$ is super-decreasing. An instance of the *subsetsum problem* is a tuple $(t, s_1, \ldots, s_k)$ of binary

coded natural numbers. It is a positive instance if there are $a_1, \ldots, a_k \in \{0,1\}$ such that $t = a_1 s_1 + \cdots + a_k s_k$. Subsetsum is a classical NP-complete problem, see e.g. [19]. The *super-decreasing subsetsum* problem is the restriction of subsetsum to instances $(t, s_1, \ldots, s_k)$, where $(s_1, \ldots, s_k)$ is super-decreasing. In [37] it was shown that super-decreasing subsetsum is P-complete.[2] We need a slightly generalized version of the construction showing P-hardness that we discuss in Section 8.2 below. Whereas in [37], the authors only have to deal with a nand-circuit with a single output gate and a fixed bit-assignment for the input gates, we have to deal with nand-circuits with several output gates and have to consider all possible bit-assignments to the input gates. The latter is needed to prove the lower bound in the second point of Theorem 8.2.

### 8.2 From boolean circuits to super-decreasing subsetsum

For this section, we have to fix some more details on boolean circuits. Let us consider a boolean circuit $C$ with input gates $x_1, \ldots, x_m$ and output gates $y_0, \ldots, y_{n-1}$.[3] We view $C$ as a directed acyclic graph with multi-edges (there can be two edges between two nodes); the nodes are the gates of the circuit. The number of incoming edges of a gate is called its *fan-in* and the number of outgoing edges is the *fan-out*. Every input gate $x_i$ has fan-in zero and every output gate $y_i$ has fan-out zero. Besides the input gates there are two more gates $c_0$ and $c_1$ of fan-in zero, where $c_i$ carries the constant truth value $i \in \{0,1\}$. Besides $x_1, \ldots, x_m, c_0, c_1$ every other gate has fan-in two and computes the nand of its two input gates. Moreover, we assume that every output gate $y_i$ is a nand-gate. For a bit string $\alpha = b_1 \cdots b_m$ ($b_1, \ldots, b_m \in \{0,1\}$) and $0 \leq i \leq n-1$ we denote with $C(\alpha)_i$ the value of the output gate $y_i$ when every input gate $x_j$ ($1 \leq j \leq m$) is set to $b_j$. Thus, $C$ defines a map $\{0,1\}^m \to \{0,1\}^n$.

We assume now that $C$ is a boolean circuit as above with the following additional property that will be satisfied later in the proof of Theorem 8.2 (see Step 1 in the proof): For all input bit strings $\alpha \in \{0,1\}^m$ there is exactly one $i \in [0..n-1]$ such that $C(\alpha)_i = 1$. Using a refinement of the construction from [37] we compute in LOGSPACE $q_0, \ldots, q_{n-1} \in \mathbb{N}$ and two super-decreasing sequences $\overline{r} = (r_1, \ldots r_m)$ and $\overline{s} = (s_1, \ldots, s_k)$ for some $k$ (all numbers are represented in binary notation) with the following properties:

- The $r_1, \ldots, r_m$ are pairwise distinct powers of 4.
- For all $0 \leq i \leq n-1$ and all $\alpha \in \{0,1\}^m$: $C(\alpha)_i = 1$ if and only if there exists $\delta \in \{0,1\}^k$ such that $\delta \cdot \overline{s} = q_i + \alpha \cdot \overline{r}$.

Let us first add for every input gate $x_i$ two new nand-gates $\bar{x}_i$ and $\bar{\bar{x}}_i$, where $\bar{\bar{x}}_i$ has the same outgoing edges as $x_i$. Moreover we remove the old outgoing edges of $x_i$ and replace them by the edges $(x_i, \bar{x}_i)$, $(c_1, \bar{x}_i)$ and two edges from $\bar{x}_i$ to $\bar{\bar{x}}_i$. This has the effect that every input gate $x_i$ has a unique outgoing edge. Clearly, the new circuit computes the same boolean function (basically, we introduce two negation gates for every input gate). Let $g_1, \ldots, g_p$ be the nand-gates of the circuit enumerated in reverse topological order, i.e., if there is an edge from gate $g_i$ to gate $g_j$ then $i > j$. We denote the two edges entering gate $g_i$ with $e_{2i+n-2}$ and $e_{2i+n-1}$. Moreover, we write $e_i$ ($0 \leq i \leq n-1$) for an imaginary edge that leaves the output gate $y_i$ and whose target gate is unspecified. Thus, the edges of the circuit are $e_0, \ldots, e_{2p+n-1}$. We now define the natural numbers $q_0, \ldots, q_{n-1}, r_1, \ldots r_m, s_1, \ldots, s_k$ with $k = 3p$:

- Let $I = \{j \mid e_j \text{ is an outgoing edge of the constant gate } c_1 \text{ or a nand-gate}\}$. For $0 \leq i \leq n-1$ we define the number $q_i$ as

$$q_i = \sum_{j \in I \setminus \{i\}} 4^j. \tag{9}$$

[2] In fact, [37] deals with the *super-increasing* subsetsum problem. But this is only a nonessential detail. For our purpose, super-decreasing sequences are more convenient.

[3] It will be convenient for us to number the input gates from 1 and the output gates from 0.

Recall that $e_i$ is the unique outgoing edge of the output gate $y_i$.

- If $e_j$ is the unique outgoing edge of the input gate $x_i$ then we set $r_i = 4^j$. We can choose the reverse topological sorting of the nand-gates in such a way that $r_1 > r_2 > \cdots > r_m$ (we only have to ensure that the target gates $\overline{x}_1, \ldots, \overline{x}_m$ of the input gates appear in the order $\overline{x}_m, \ldots, \overline{x}_1$ in the reverse topological sorting of the nand-gates).
- To define the numbers $s_1, \ldots, s_k$ we first define for every nand-gate $g_i$ three numbers $t_{3i}$, $t_{3i-1}$ and $t_{3i-2}$ as follows, where $I_i = \{j \mid e_j \text{ is an outgoing edge of gate } g_i\}$:

$$\begin{aligned} t_{3i} &= 4^{2i+n-1} + 4^{2i+n-2} + \sum_{j \in I_i} 4^j \\ t_{3i-1} &= 4^{2i+n-1} - 4^{2i+n-2} = 3 \cdot 4^{2i+n-2} \\ t_{3i-2} &= 4^{2i+n-2} \end{aligned}$$

Then, the tuple $(s_1, \ldots, s_k)$ is $(t_{3p}, t_{3p-1}, t_{3p-2}, \ldots, t_3, t_2, t_1)$, which is indeed super-decreasing (see also [37]). In fact, we have $s_i - (s_{i+1} + \cdots + s_k) \geq 4^{n-1}$ for all $i \in [1..k]$. To see this, note that the sets $I_{i+1}, \ldots, I_k$ are pairwise disjoint. This implies that the $n$ low-order digits (corresponding to the edges $e_0, \ldots, e_{n-1}$) in the base-4 expansion of $s_{i+1} + \cdots + s_k$ are zero or one.

In order to understand this construction, one should think of the edges of the circuit carrying truth values. Recall that there are $2p + n$ edges in the circuit (including the imaginary outgoing edges of the output gates $y_0, \ldots, y_{n-1}$). A number in base-4 representation with $2p + n$ digits that are either 0 or 1 represents a truth assignment to the $2p + n$ edges, where a 1-digit represents the truth value 1 and a 0-digit represents the truth value 0. Consider an input string $\alpha = b_1 \cdots b_m \in \{0,1\}^m$. Then the number

$$N(\alpha) := \sum_{j \in I} 4^j + b_1 r_1 + \cdots + b_m r_m = \sum_{j \in I} 4^j + \alpha \cdot \overline{r}$$

encodes the truth assignment for the circuit edges, where:

- all outgoing edges of the constant gate $c_1$ carry the truth value 1,
- all outgoing edges of the constant gate $c_0$ carry the truth value 0,
- the unique outgoing edge of an input gate $x_i$ carries the truth value $b_i$,
- all outgoing edges of nand-gates carry the truth value 1.

Claim 1. *$C(\alpha)_i = 1$ if and only if there exists $\delta \in \{0,1\}^k$ such that $\delta \cdot \overline{s} = N(\alpha) - 4^i$.*

Note that $N(\alpha) - 4^i = q_i + \alpha \cdot \overline{r}$, where $q_i$ is from (9). To prove Claim 1 we apply the canonical algorithm for super-decreasing subsetsum with input $(N(\alpha), \overline{s})$. This algorithm initializes a counter $A$ to $N(\alpha)$ and then goes over the sequence $s_1, \ldots, s_k$ in that order. In the $j$-th step ($1 \leq j \leq k$) we set $A$ to $A - s_j$ if $A \geq s_j$. If $A < s_j$, we do not modify $A$. After that we proceed with $s_{j+1}$. After processing $s_k$ the algorithm terminates with a certain counter value $A$. Clearly, this final value $A$ has the property that there is $\delta \in \{0,1\}^k$ such that $\delta \cdot \overline{s} + A = N(\alpha)$. In order to prove Claim 1, let us first show the following statement.

Claim 2. *Assume that $i \in [0..n-1]$ is the unique index such that $C(\alpha)_i = 1$. Then the canonical algorithm with input $(N(\alpha), \overline{s})$ terminates with the counter value $A = 4^i$.*

Proof of Claim 2. To prove this, we show that running the canonical algorithm with input $(N(\alpha), \overline{s})$ exactly corresponds to evaluating the circuit $C$ with input $\alpha$. Thereby the nand-gates are evaluated in the topological order

$g_p, g_{p-1}, \ldots, g_1$. Assume that $g_j$ is the gate that we want to evaluate next. In the canonical algorithm with input $(N(\alpha), \overline{s})$ the evaluation of $g_j$ is simulated by the three numbers $t_{3j}$, $t_{3j-1}$, and $t_{3j-2}$. At the point where the algorithm checks whether $t_{3j}$ can be subtracted from the current $A$, the base-4 digits at positions $2j+n, \ldots, 2p+n-1$ in the counter value $A$ have been already set to zero. If the digits at the next two high-order positions $2j+n-1$ and $2j+n-2$ are still 1 (i.e., the input edges $e_{2j+n-2}$ and $e_{2j+n-1}$ for gate $g_j$ carry the truth value 1), then we can subtract $t_{3j}$ from $A$. Thereby we subtract all powers $4^{2j+n-1}$, $4^{2j+n-2}$ and $4^h$, where $e_h$ is an outgoing edge for gate $g_j$. Since gate $g_j$ evaluates to zero (both input edges carry 1), this subtraction correctly simulates the evaluation of gate $g_j$: all outgoing edges $e_h$ of $g_j$ (that were initially set to 1) are set to 0. On the other hand, if one of the two digits at positions $2j+n-1$ and $2j+n-2$ in $A$ is 0 (which means that gate $g_j$ evaluates to 1), then we cannot subtract $t_{3j}$ from $A$. If both digits at positions $2j+n-1$ and $2j+n-2$ in $A$ are 0, then also $t_{3j-1}$ and $t_{3j-2}$ cannot be subtracted. On the other hand, if exactly one of the two digits at positions $2j+n-1$ and $2j+n-2$ is 1, then with $t_{3j-1}$ and $t_{3j-2}$ we can set these two digits to 0 (thereby digits at positions $< 2j+n-2$ are not modified). After processing the final weight $s_k = t_1$ all digits in $A$ are set to zero except for the digit at the unique position $i$ that corresponds to the output edge of gate $y_i$ (the unique output gate that evaluates to zero). Initially, in $N(\alpha)$ this digit was set to 1 and it remains 1. Hence, the final counter value is $4^i$. □

Proof of Claim 1. We are now in the position to prove Claim 1. Let $y_j$ ($j \in [0..n-1]$) be the unique output gate that evaluates to 1, i.e., all output gates $y_{j'}$ with $j' \neq j$ evaluate to zero. Then, by Claim 2, the canonical algorithm terminates with the counter value $A = 4^j$. Therefore there exists $\delta \in \{0,1\}^k$ such that $\delta \cdot \overline{s} + 4^j = N(\alpha)$. Hence, if $i = j$ (i.e., $C(\alpha)_i = 1$) then $\delta \cdot \overline{s} = N(\alpha) - 4^i = q_i + \alpha \cdot \overline{r}$.

Now assume that $C(\alpha)_i \neq 1$, i.e., $j \neq i$. We claim that there is no $\delta' \in \{0,1\}^k$ such that $\delta' \cdot \overline{s} + 4^i = N(\alpha)$. In order to get a contradiction, assume that such a $\delta' \in \{0,1\}^k$ exists. We get $\delta \cdot \overline{s} + 4^j = \delta' \cdot \overline{s} + 4^i$, i.e., $\delta \cdot \overline{s} - \delta' \cdot \overline{s} = 4^i - 4^j$. Clearly, $\delta \neq \delta'$. Since $i, j \in [0..n-1]$ we have $|\delta \cdot \overline{s} - \delta' \cdot \overline{s}| < 4^{n-1}$. But $s_i - (s_{i+1} + \cdots + s_k) \geq 4^{n-1}$ for all $i \in [1..k]$ implies that $|\delta \cdot \overline{s} - \delta' \cdot \overline{s}| \geq 4^{n-1}$ – a contradiction. □

### 8.3 From super-decreasing subsetsum to straight-line programs

In [42] a super-decreasing sequence $\overline{t} = (t_1, \ldots, t_k)$ of natural numbers is encoded by the string $S(\overline{t}) \in \{0,1\}^*$ of length $\sum \overline{t} + 1$ such that for all $0 \leq p \leq \sum \overline{t}$:

$$S(\overline{t})[p] = \begin{cases} 1 & \text{if } p = \alpha \cdot \overline{t} \text{ for some } \alpha \in \{0,1\}^k, \\ 0 & \text{otherwise.} \end{cases} \tag{10}$$

Note that in the first case, $\alpha$ is unique. Since $\overline{t}$ is a super-decreasing sequence, the number of 1's in the string $S(\overline{t})$ is $2^k$. Also note that $S(\overline{t})$ starts and ends with 1. In [42] it was shown that from a super-decreasing sequence $\overline{t}$ of binary encoded numbers one can construct in LOGSPACE an SLP for the word $S(\overline{t})$.

### 8.4 Proof of Theorem 8.2

Let us fix a regular wreath product of the form $G \wr \mathbb{Z}$ for a finitely generated group $G$. Such groups are also known as generalized lamplighter groups (the lamplighter group arises for $G = \mathbb{Z}_2$). Throughout this section, we fix a set of standard generators $\Sigma$ for $G$ and let $\tau = 1$ be the generator for $\mathbb{Z}$. Then $\Sigma \cup \{\tau, \tau^{-1}\}$ is a standard generating set for the wreath product $G \wr \mathbb{Z}$. In $G \wr \mathbb{Z}$ the $G$-generator $a \in \Sigma$ represents the mapping $f_a \in G^{(\mathbb{Z})}$ with $f_a(0) = a$ and $f_a(z) = 1$ for $z \neq 0$. For a word $w \in (\Sigma \cup \{\tau, \tau^{-1}\})^*$ we define $\eta(w) := |w|_\tau - |w|_{\tau^{-1}}$. Thus, the element of $G \wr \mathbb{Z}$ represented by $w$ is of the form $f\tau^{\eta(w)}$ for some $f \in G^{(\mathbb{Z})}$. Recall the definition of the left action of $\mathbb{Z}$ on $G^{(\mathbb{Z})}$ from Section 3.1 (where

we take $H = Y = \mathbb{Z}$). For better readability, we write $c \circ f$ for ${}^c f$ ($c \in \mathbb{Z}$, $f \in G^{(\mathbb{Z})}$). Hence, we have $(c \circ f)(z) = f(z + c)$. If one thinks of $f$ as a bi-infinite word over the alphabet $G$, then $c \circ f$ is the same word but shifted by $-c$.

The following intuition might be helpful: Consider a word $w \in (\Sigma \cup \{\tau, \tau^{-1}\})^*$. In $G \wr \mathbb{Z}$ we can simplify $w$ to a word of the form $\tau^{z_0} a_1 \tau^{z_1} a_2 \cdots \tau^{z_{k-1}} a_k \tau^{z_k}$ (with $z_j \in \mathbb{Z}$, $a_j \in \Sigma$), which in $G \wr \mathbb{Z}$ can be rewritten as

$$\tau^{z_0} a_1 \tau^{z_1} a_2 \cdots \tau^{z_{k-1}} a_k \tau^{z_k} = \Big( \prod_{j=1}^{k} \tau^{z_0+\cdots+z_{j-1}} a_j \tau^{-(z_0+\cdots+z_{j-1})} \Big)\, \tau^{z_0+\cdots+z_k}.$$

Hence, the word $w$ represents the group element

$$\Big( \prod_{j=1}^{k} (z_0 + \cdots + z_{j-1}) \circ f_{a_j} \Big)\, \tau^{z_0+\cdots+z_k}.$$

This gives the following intuition for evaluating $\tau^{z_0} a_1 \tau^{z_1} a_2 \cdots \tau^{z_{k-1}} a_k \tau^{z_k}$: In the beginning, every $\mathbb{Z}$-position carries the $G$-value 1. First, go to the $\mathbb{Z}$-position $-z_0$ and multiply the $G$-element at this position with $a_1$ (on the right), then go to the $\mathbb{Z}$-position $-z_0 - z_1$ and multiply the $G$-element at this position with $a_2$, and so on.

Proof of Theorem 8.2. The easy part is to show that the compressed word problem for $G \wr \mathbb{Z}$ belongs to $\forall\mathsf{LEAF}(\mathrm{WP}(G))$. In the following, we make use of the statements from Lemma 7.2. Let $\mathcal{G}$ be an SLP over the alphabet $\Sigma \cup \{\tau, \tau^{-1}\}$ and let $f\tau^{\eta(\mathrm{val}(\mathcal{G}))} \in G \wr \mathbb{Z}$ be the group element represented by $\mathrm{val}(\mathcal{G})$. By Lemma 7.2 we can compute $\eta(\mathrm{val}(\mathcal{G}))$ in polynomial time. If $\eta(\mathrm{val}(\mathcal{G})) \neq 0$, then the Turing-machine rejects by printing a non-trivial generator of $G$ (here we need the assumption that $G$ is non-trivial). So, let us assume that $\eta(\mathrm{val}(\mathcal{G})) = 0$. We can also compute in polynomial time two integers $b, c \in \mathbb{Z}$ such that $\mathrm{supp}(f) \subseteq [b..c]$. We can take for instance $b = -|\mathrm{val}(\mathcal{G})|$ and $c = |\mathrm{val}(\mathcal{G})|$. It suffices to check whether for all $x \in [b..c]$ we have $f(x) = 1$. For this, the Turing-machine branches universally to all binary coded integers $x \in [b..c]$ (this yields the $\forall$-part in $\forall\mathsf{LEAF}(\mathrm{WP}(G))$). Consider a specific branch that leads to the integer $x \in [b..c]$. From $x$ and the input SLP $\mathcal{G}$ the Turing-machine then produces a leaf string over the standard generating set $\Sigma$ of $G$ such that this leaf string represents the group element $f(x) \in G$. For this, the machine branches to all positions $p \in [0..|\mathrm{val}(\mathcal{G})| - 1]$ (if $p < q < |\mathrm{val}(\mathcal{G})|$ then the branch for $p$ is to the left of the branch for $q$). For a specific position $p$, the machine computes in polynomial time the symbol $a = \mathrm{val}(\mathcal{G})[p]$. If $a$ is $\tau$ or $\tau^{-1}$ then the machine prints $1 \in \Sigma$. On the other hand, if $a \in \Sigma$ then the machine computes in polynomial time $d = \eta(\mathrm{val}(\mathcal{G})[: p])$. This is possible by first computing an SLP for the prefix $\mathrm{val}(\mathcal{G})[: p]$. If $d = -x$ then the machine prints the symbol $a$, otherwise the machine prints the trivial generator 1. It is easy to observe that the leaf string produced in this way represents the group element $f(x)$.

We now show the hardness statement from Theorem 8.2. By Lemma 4.3 it suffices to show that $\textsc{CompressedWP}(G \wr \mathbb{Z})$ is hard for $\forall\mathsf{bLEAF}(\mathrm{WP}(G/Z(G)))$ with respect to LOGSPACE-reductions. Let $a_0, \ldots, a_{n-1}$ be an arbitrary enumeration of the standard generators in $\Sigma$. Fix a language $L \in \forall\mathsf{bLEAF}(\mathrm{WP}(G/Z(G)))$. From the definition of the class $\forall\mathsf{bLEAF}(\mathrm{WP}(G/Z(G)))$ it follows that there exist two polynomials $p_1$ and $p_2$ and a balanced polynomial time NTM $M$ running in time $p_1 + p_2$ that outputs a symbol from $\Sigma$ after termination and such that the following holds: Consider an input word $z$ and let $T(z)$ be the corresponding computation tree of $M$. Let $m_1 = p_1(|z|)$, $m_2 = p_2(|z|)$, and $m = m_1 + m_2$. Note that the nodes of $T(z)$ are the bit strings of length at most $m$. For every leaf $\alpha \in \{0,1\}^m$ let us denote with $\lambda(\alpha)$ the symbol from $\Sigma$ that $M$ prints when reaching the leaf $\alpha$. Then $z \in L$ if and only if for all $\beta \in \{0,1\}^{m_1}$ the string

$$\lambda_\beta := \prod_{\gamma \in \{0,1\}^{m_2}} \lambda(\beta\gamma) \tag{11}$$

represents a group element from the center $Z(G)$. Here (and in the following), the product in the right-hand side of (11) goes over all bit strings of length $m_2$ in lexicographic order. Our construction consists of five steps:

*Step 1.* Note that given a bit string $\alpha \in \{0,1\}^m$, we can compute in polynomial time the symbol $\lambda(\alpha) \in \Sigma$ by following the computation path specified by $\alpha$. Using the classical Cook-Levin construction (see e.g. [3]), we can compute from the input $z$ and $a \in \Sigma$ in LOGSPACE a boolean circuit $C_{z,a}$ with $m$ input gates $x_1, \ldots, x_m$ and a single output gate $y_0$ such that for all $\alpha \in \{0,1\}^m$: $C_{z,a}(\alpha)_0 = 1$ if and only if $\lambda(\alpha) = a$. By taking the disjoint union of these circuits and merging the input gates, we can build a single circuit $C_z$ with $m$ input gates $x_1, \ldots, x_m$ and $n = |\Sigma|$ output gates $y_0, \ldots, y_{n-1}$. For every $\alpha \in \{0,1\}^m$ and every $0 \leq i \leq n-1$ the following holds: $C_z(\alpha)_i = 1$ if and only if $\lambda(\alpha) = a_i$.

*Step 2.* Using the construction from Section 8.2 we can compute from the circuit $C_z$ in LOGSPACE numbers $q_0, \ldots, q_{n-1} \in \mathbb{N}$ and two super-decreasing sequences $\overline{r} = (r_1, \ldots, r_m)$ and $\overline{s} = (s_1, \ldots, s_k)$ with the following properties:

- The $r_1, \ldots, r_m$ are pairwise distinct powers of 4.
- For all $0 \leq i \leq n-1$ and all $\alpha \in \{0,1\}^m$ we have: $\lambda(\alpha) = a_i$ if and only if $C_z(\alpha)_i = 1$ if and only if there is $\delta \in \{0,1\}^k$ such that $\delta \cdot \overline{s} = q_i + \alpha \cdot \overline{r}$.

Note that for all $\alpha \in \{0,1\}^m$ there is a unique $i$ such that $C_z(\alpha)_i = 1$. Hence, for all $\alpha \in \{0,1\}^m$ there is a unique $i$ such that $q_i + \alpha \cdot \overline{r}$ is of the form $\delta \cdot \overline{s}$ for some $\delta \in \{0,1\}^k$. For this unique $i$ we have $\lambda(\alpha) = a_i$.

We split the super-decreasing sequence $\overline{r} = (r_1, \ldots, r_m)$ into the two sequences $\overline{r}_1 = (r_1, \ldots, r_{m_1})$ and $\overline{r}_2 = (r_{m_1+1}, \ldots, r_m)$. For the following consideration, we need the following numbers:

$$\ell = \max\left\{ \sum \overline{r}_1 + \max\{q_0, \ldots, q_{n-1}\} + 1,\ \sum \overline{s} - \sum \overline{r}_2 - \min\{q_0, \ldots, q_{n-1}\} + 1 \right\} \tag{12}$$

$$\pi = \ell + \sum \overline{r}_2 \tag{13}$$

The binary codings of these numbers can be computed in LOGSPACE (since iterated addition, max, and min can be computed in LOGSPACE). The precise value of $\ell$ will be only relevant at the end of step 4.

*Step 3.* By the result from [42] (see Section 8.3) we can construct in LOGSPACE from the three super-decreasing sequences $\overline{r}_1, \overline{r}_2$ and $\overline{s}$ three SLPs $\mathcal{G}_1$, $\mathcal{G}_2$ and $\mathcal{H}$ over the alphabet $\{0,1\}$ such that $\mathrm{val}(\mathcal{G}_1) = S(\overline{r}_1)$, $\mathrm{val}(\mathcal{G}_2) = S(\overline{r}_2)$ and $\mathrm{val}(\mathcal{H}) = S(\overline{s})$ (see (10)). For all positions $p \geq 0$ (in the suitable range) we have:

$$\begin{aligned}
\mathrm{val}(\mathcal{G}_1)[p] = 1 &\iff \exists \beta \in \{0,1\}^{m_1} : p = \beta \cdot \overline{r}_1 \\
\mathrm{val}(\mathcal{G}_2)[p] = 1 &\iff \exists \gamma \in \{0,1\}^{m_2} : p = \gamma \cdot \overline{r}_2 \\
\mathrm{val}(\mathcal{H})[p] = 1 &\iff \exists \delta \in \{0,1\}^{k} : p = \delta \cdot \overline{s}
\end{aligned}$$

Note that $|\mathrm{val}(\mathcal{G}_1)| = \sum \overline{r}_1 + 1$, $|\mathrm{val}(\mathcal{G}_2)| = \sum \overline{r}_2 + 1$, and $|\mathrm{val}(\mathcal{H})| = \sum \overline{s} + 1$.

*Step 4.* We build in LOGSPACE for every $i \in [0..n-1]$ an SLP $\mathcal{H}_i$ from the SLP $\mathcal{H}$ by replacing in every right-hand side of $\mathcal{H}$ every occurrence of 0 by $\tau^{-1}$ and every occurrence of 1 by $a_i\tau^{-1}$. Let $T_i$ be the start variable of $\mathcal{H}_i$, let $S_1$ be the start variable of $\mathcal{G}_1$, and let $S_2$ be the start variable of $\mathcal{G}_2$. We can assume that the variable sets of the SLPs $\mathcal{G}_1, \mathcal{G}_2, \mathcal{H}_0, \ldots, \mathcal{H}_{n-1}$ are pairwise disjoint. We next combine these SLPs into a single SLP $\mathcal{I}$. The variables of $\mathcal{I}$ are the variables of the SLPs $\mathcal{G}_1, \mathcal{G}_2, \mathcal{H}_0, \ldots, \mathcal{H}_{n-1}$ plus a fresh variable $S$ which is the start variable of $\mathcal{I}$. The right-hand sides for the variables are defined below. In the right-hand sides we write powers $\tau^p$ for integers $p$ whose binary codings can be computed in LOGSPACE. Such powers can be produced by small subSLPs that can be constructed in LOGSPACE too.

- In all right-hand sides of $\mathcal{G}_1$ and $\mathcal{G}_2$ we replace all occurrences of the terminal symbol 0 by the $\mathbb{Z}$-generator $\tau$.

- We replace every occurrence of the terminal symbol 1 in a right-hand side of $\mathcal{G}_1$ by $S_2\tau^\ell$, where $\ell$ is from (12).
- We replace every occurrence of the terminal symbol 1 in a right-hand side of $\mathcal{G}_2$ by $\sigma\tau$, where

$$\sigma = \tau^{q_0} T_0 \tau^{h-q_0} \tau^{q_1} T_1 \tau^{h-q_1} \cdots \tau^{q_{n-1}} T_{n-1} \tau^{h-q_{n-1}} \tag{14}$$

  and $h = \sum \overline{s} + 1$ is the length of the word $\mathrm{val}(\mathcal{H})$ (which is $-\eta(\mathrm{val}_{\mathcal{I}}(T_i))$ for every $i \in [0..n-1]$). Note that $\eta(\mathrm{val}_{\mathcal{I}}(\sigma)) = 0$.
- Finally, the right-hand side of the start variable $S$ is $S_1\tau^{-d}$ where $d := \sum \overline{r}_1 + 1 + 2^{m_1} \cdot \pi$. (note that $d = \eta(\mathrm{val}_{\mathcal{I}}(S_1))$).

Before we explain this construction, let us first introduce some notations.

- Let $u := \mathrm{val}_{\mathcal{I}}(S_2)$. We have $\eta(u) = |\mathrm{val}(\mathcal{G}_2)|$. Hence, the group element represented by $u$ can be written as $f_u\tau^{|\mathrm{val}(\mathcal{G}_2)|}$ for a mapping $f_u \in G^{(\mathbb{Z})}$.
- Let $v := \mathrm{val}_{\mathcal{I}}(\sigma)$ where $\sigma$ is from (14). Note that $\eta(v) = 0$. Hence, the group element represented by $v$ is a mapping $f_v \in G^{(\mathbb{Z})}$. Its support is a subset of the interval from position $-\max\{q_0, \ldots, q_{n-1}\}$ to position $\sum \overline{s} - \min\{q_0, \ldots, q_{n-1}\}$.
- For $\beta \in \{0,1\}^{m_1}$ let $\mathsf{bin}(\beta)$ be the number represented by $\beta$ in binary notation (thus, $\mathsf{bin}(0^{m_1}) = 0$, $\mathsf{bin}(0^{m_1-1}1) = 1, \ldots, \mathsf{bin}(1^{m_1}) = 2^{m_1} - 1$). Moreover, let

$$p_\beta := -\mathsf{bin}(\beta) \cdot \pi.$$

First, note that $\eta(\mathrm{val}(\mathcal{I})) = 0$. This is due to the factor $\tau^{-d}$ in the right-hand side of the start variable $S$ of $\mathcal{I}$. Hence, the group element represented by $\mathrm{val}(\mathcal{I})$ is a mapping $f \in G^{(\mathbb{Z})}$. The crucial claim is the following:

Claim 1. *For every $\beta \in \{0,1\}^{m_1}$, $f(p_\beta)$ is the group element represented by the leaf string $\lambda_\beta$ from (11).*

*Proof of the claim.* In the following, we compute in the restricted direct product $G^{(\mathbb{Z})}$. Recall that the multiplication in this group is defined by the pointwise multiplication of mappings.

Since we replaced in $\mathcal{G}_1$ every 1 in a right-hand side by $S_2\tau^\ell$, which produces $u\tau^\ell$ in $\mathcal{I}$ (which evaluates to $f_u\tau^{\pi+1}$) the mapping $f$ is a product (in the restricted direct product $G^{(\mathbb{Z})}$) of shifted copies of $f_u$. More precisely, for every $\beta' \in \{0,1\}^{m_1}$ we get the shifted copy

$$\left(\beta' \cdot \overline{r}_1 + \mathsf{bin}(\beta') \cdot \pi\right) \circ f_u \tag{15}$$

of $f_u$. The shift distance $\beta' \cdot \overline{r}_1 + \mathsf{bin}(\beta') \cdot \pi$ can be explained as follows: The 1 in $\mathrm{val}(\mathcal{G}_1)$ that corresponds to $\beta' \in \{0,1\}^{m_1}$ occurs at position $\beta' \cdot \overline{r}_1$ (the first position is 0) and to the left of this position we find $\mathsf{bin}(\beta')$ many 1's and $\beta' \cdot \overline{r}_1 - \mathsf{bin}(\beta')$ many 0's in $\mathrm{val}(\mathcal{G}_1)$. Moreover, every 0 in $\mathrm{val}(\mathcal{G}_1)$ was replaced by $\tau$ (shift by 1) and every 1 in $\mathrm{val}(\mathcal{G}_1)$ was replaced by $u\tau^\ell$ (shift by $\ell + |\mathrm{val}(\mathcal{G}_2)| = \pi + 1$). Hence, the total shift distance is indeed (15). Also note that if $\beta' \in \{0,1\}^{m_1}$ is lexicographically smaller than $\beta'' \in \{0,1\}^{m_1}$ then $\beta' \cdot \overline{r}_1 < \beta'' \cdot \overline{r}_1$. This implies that

$$f = \prod_{\beta' \in \{0,1\}^{m_1}} \left(\beta' \cdot \overline{r}_1 + \mathsf{bin}(\beta') \cdot \pi\right) \circ f_u = \prod_{\beta' \in \{0,1\}^{m_1}} \left(\beta' \cdot \overline{r}_1 - p_{\beta'}\right) \circ f_u.$$

Let us now compute the mapping $f_u$. Recall that we replaced in $\mathcal{G}_2$ every occurrence of 1 by $\sigma\tau$, where $\sigma$ is from (14) and derives to $v$. The 1's in $\mathrm{val}(\mathcal{G}_2)$ occur at positions of the form $\gamma \cdot \overline{r}_2$ for $\gamma \in \{0,1\}^{m_2}$ and if $\gamma \in \{0,1\}^{m_2}$ is lexicographically smaller than $\gamma' \in \{0,1\}^{m_2}$ then $\gamma \cdot \overline{r}_2 < \gamma' \cdot \overline{r}_2$. We therefore get

$$f_u = \prod_{\gamma \in \{0,1\}^{m_2}} (\gamma \cdot \overline{r}_2) \circ f_v.$$

We obtain

$$\begin{aligned} f &= \prod_{\beta' \in \{0,1\}^{m_1}} (\beta' \cdot \overline{r}_1 - p_{\beta'}) \circ f_u \\ &= \prod_{\beta' \in \{0,1\}^{m_1}} (\beta' \cdot \overline{r}_1 - p_{\beta'}) \circ \prod_{\gamma \in \{0,1\}^{m_2}} (\gamma \cdot \overline{r}_2 \circ f_v) \\ &= \prod_{\beta' \in \{0,1\}^{m_1}} \prod_{\gamma \in \{0,1\}^{m_2}} (\beta' \cdot \overline{r}_1 + \gamma \cdot \overline{r}_2 - p_{\beta'}) \circ f_v \end{aligned}$$

and hence

$$f(p_\beta) = \prod_{\beta' \in \{0,1\}^{m_1}} \prod_{\gamma \in \{0,1\}^{m_2}} f_v(p_\beta - p_{\beta'} + \beta' \cdot \overline{r}_1 + \gamma \cdot \overline{r}_2).$$

We claim that for all $\beta \neq \beta'$ and all $\gamma \in \{0,1\}^{m_2}$ we have

$$f_v(p_\beta - p_{\beta'} + \beta' \cdot \overline{r}_1 + \gamma \cdot \overline{r}_2) = 1. \tag{16}$$

Let us postpone the proof of this for a moment. From (16) we get

$$f(p_\beta) = \prod_{\gamma \in \{0,1\}^{m_2}} f_v(\beta \cdot \overline{r}_1 + \gamma \cdot \overline{r}_2).$$

Consider a specific $\gamma \in \{0,1\}^{m_2}$ and let $\alpha = \beta\gamma$ and $p = \beta \cdot \overline{r}_1 + \gamma \cdot \overline{r}_2 = \alpha \cdot \overline{r}$. From the definition of $v = \mathrm{val}_{\mathcal{I}}(\sigma)$ it follows that for all $x \in \mathbb{Z}$, $f_v(x)$ is a product of those group generators $a_i$ such that $x = -q_i + \delta \cdot \overline{s}$ for some $\delta \in \{0,1\}^k$. For the position $p$ this means that $q_i + \alpha \cdot \overline{r} = \delta \cdot \overline{s}$. By our previous remarks, there is a unique such $i \in [0..n-1]$ and for this $i$ we have $\lambda(\alpha) = a_i$. Hence, we obtain $f_v(p) = \lambda(\alpha) = \lambda(\beta\gamma)$ and thus

$$f(p_\beta) = \prod_{\gamma \in \{0,1\}^{m_2}} \lambda(\beta\gamma) = \lambda_\beta.$$

It remains to show (16). To get this identity, we need the precise value of $\ell$ from (12) (so far, the value of $\ell$ was not relevant). Assume now that $\beta \neq \beta'$, which implies

$$|p_\beta - p_{\beta'}| \geq \pi = \ell + \sum \overline{r}_2.$$

Hence, we either have

$$\begin{aligned} p_\beta - p_{\beta'} + \beta' \cdot \overline{r}_1 + \gamma \cdot \overline{r}_2 &\geq \ell + \sum \overline{r}_2 + \beta' \cdot \overline{r}_1 + \gamma \cdot \overline{r}_2 \\ &\geq \ell + \sum \overline{r}_2 \\ &> \sum \overline{s} - \min\{q_0, \ldots, q_{n-1}\} \end{aligned}$$

or

$$\begin{aligned} p_\beta - p_{\beta'} + \beta' \cdot \overline{r}_1 + \gamma \cdot \overline{r}_2 &\leq -\ell - \sum \overline{r}_2 + \beta' \cdot \overline{r}_1 + \gamma \cdot \overline{r}_2 \\ &\leq -\ell + \sum \overline{r}_1 \\ &< -\max\{q_0, \ldots, q_{n-1}\}, \end{aligned}$$

where the strict inequalities follow from our choice of $\ell$. Recall that the support of the mapping $f_v$ is contained in $[-\max\{q_0, \ldots, q_{n-1}\} .. \sum \overline{s} - \min\{q_0, \ldots, q_{n-1}\}]$. This shows (16) and hence the claim.

*Step 5.* By the above claim, we have $f(p_\beta) \in Z(G)$ for all $\beta \in \{0,1\}^{m_1}$ if and only if $\lambda_\beta \in Z(G)$ for all $\beta \in \{0,1\}^{m_1}$, which is equivalent to $z \in L$. The only remaining problem is that the word $\mathrm{val}(\mathcal{I})$ produces some "garbage" group elements $f(x)$ on positions $x$ that are not of the form $p_\beta$. Note that for every $g \in G \setminus Z(G)$, there is a generator $a_i \in \Sigma$ such that the commutator $[g, a_i]$ is non-trivial. We now produce from $\mathcal{I}$ an SLP $\mathcal{I}^{-1}$ such that $\mathrm{val}(\mathcal{I}^{-1})$ represents the inverse element of $f \in G^{(\mathbb{Z})}$, which is the mapping $g$ with $g(x) = f(x)^{-1}$ for all $x \in \mathbb{Z}$. To construct $\mathcal{I}^{-1}$, we have to reverse every right-hand side of $\mathcal{I}$ and replace every occurrence of a symbol $a_0, \ldots, a_{n-1}, \tau, \tau^{-1}$ by its inverse.

It is easy to compute in LOGSPACE for every $i \in [0..n-1]$ an SLP for the word

$$w_i := \left(a_i \tau^\pi\right)^{2^{m_1}} \tau^{-2^{m_1} \cdot \pi}.$$

Then the group element represented by $w_i$ is the mapping $f_i \in G^{(\mathbb{Z})}$ whose support is the set of positions $p_\beta$ for $\beta \in \{0,1\}^{m_1}$ and $f_i(p_\beta) = a_i$ for all $\beta \in \{0,1\}^{m_1}$. We can also compute in LOGSPACE an SLP for the word $w_i^{-1}$. We then built in LOGSPACE SLPs $\mathcal{J}_0, \ldots, \mathcal{J}_{n-1}$ such that $\mathrm{val}(\mathcal{J}_i) = \mathrm{val}(\mathcal{I}^{-1}) w_i^{-1} \mathrm{val}(\mathcal{I}) w_i$. Hence, the word $\mathrm{val}(\mathcal{J}_i)$ represents the group element $g_i \in G^{(\mathbb{Z})}$, where $g_i(x) = 1$ for all $x \in \mathbb{Z} \setminus \{p_\beta \mid \beta \in \{0,1\}^{m_1}\}$ and $g_i(p_\beta) = f(p_\beta)^{-1} a_i^{-1} f(p_\beta) a_i = [f(p_\beta), a_i]$.

Finally, we construct in LOGSPACE an SLP $\mathcal{J}$ such that

$$\mathrm{val}(\mathcal{J}) = \mathrm{val}(\mathcal{J}_0)\, \tau\, \mathrm{val}(\mathcal{J}_1)\, \tau\, \mathrm{val}(\mathcal{J}_2) \cdots \tau\, \mathrm{val}(\mathcal{J}_{n-1})\, \tau^{-n+1}.$$

We can assume that $n \leq \ell + \sum \overline{r}_2 = \pi$ ($n$ is a constant and we can always make $\ell$ bigger). Then $\mathrm{val}(\mathcal{J})$ evaluates to the group element $g \in G^{(\mathbb{Z})}$ with $g(x) = 1$ for $x \in \mathbb{Z} \setminus \{p_\beta - i \mid \beta \in \{0,1\}^{m_1}, 0 \leq i \leq n-1\}$ and $g(p_\beta - i) = g_i(p_\beta) = [f(p_\beta), a_i]$ for $0 \leq i \leq n-1$. Hence, if $f(p_\beta) \in Z(G)$ for all $\beta \in \{0,1\}^{m_1}$ then $\mathrm{val}(\mathcal{J}) = 1$ in $G \wr \mathbb{Z}$. On the other hand, if there is a $\beta \in \{0,1\}^{m_1}$ such that $f(p_\beta) \in G \setminus Z(G)$ then there is an $a_i$ such that $[f(p_\beta), a_i] \neq 1$. Hence $g(p_\beta - i) \neq 1$ and $\mathrm{val}(\mathcal{J}) \neq 1$ in $G \wr \mathbb{Z}$. This proves the theorem. □

The following remark will be needed in the next section.

*Remark 8.4.* Consider the SLP $\mathrm{val}(\mathcal{J})$ computed in the previous proof from the machine input $z$. We showed that $z \in L$ if and only if $\mathrm{val}(\mathcal{J}) = 1$ in $G \wr \mathbb{Z}$. Let $s = |\mathrm{val}(\mathcal{J})|$; it is a number that grows exponentially with $|z|$. The binary expansion of $s$ can be computed from $z$ in LOGSPACE using simple arithmetics. Let $t$ be any positive integer with $t \geq 2s + 1$. Then $\mathrm{val}(\mathcal{J}) = 1$ in $G \wr \mathbb{Z}$ if and only if $\mathrm{val}(\mathcal{J}) = 1$ in $G \wr (\mathbb{Z}/t)$ where in the latter equality $\tau$ is taken for the generator of $\mathbb{Z}/t$. To see this, note that during the evaluation of $\mathrm{val}(\mathcal{J})$ in $G \wr \mathbb{Z}$ only the $G$-elements at positions in the interval $[-s..s]$ (whose size is at most $t$) can be multiplied with a generator of $G$. Intuitively, $\mathrm{val}(\mathcal{J})$ evaluates in $G \wr \mathbb{Z}$ in the same way as in $G \wr (\mathbb{Z}/t)$.

## 9 PSPACE-COMPLETE COMPRESSED WORD PROBLEMS

In this section, we will use Theorem 8.2 (and Remark 8.4) to show PSPACE-completeness of the compressed word problem for several groups. For upper upper bounds, we will make use of the following simple lemma:

Lemma 9.1. *If* $\mathrm{WP}(G)$ *belongs to* polyL, *then* $\mathrm{CompressedWP}(G \wr \mathbb{Z})$ *belongs to* PSPACE.

Proof. We use a result of Waack [61] according to which the word problem for a wreath product $G_1 \wr G_2$ is uniformly $\mathsf{NC}^1$-reducible (and hence LOGSPACE-reducible) to the word problems for $G_1$ and $G_2$. Since $\mathrm{WP}(G)$ belongs to polyL and $\mathrm{WP}(\mathbb{Z})$ belongs to LOGSPACE, it follows that $\mathrm{WP}(G \wr \mathbb{Z})$ belongs to polyL (polyL is closed under LOGSPACE-reductions). Hence, by Lemma 7.4 the compressed word problem for $G \wr \mathbb{Z}$ belongs to PSPACE. □

The following lemma generalizes the inclusion PSPACE $\subseteq$ LEAF(WP($G$)) for $G$ finite non-solvable (where in fact equality holds) from [30]. It can be proved directly using the same idea based on commutators as Theorem 6.3. Here we follow a different approach and derive it by a padding argument from Theorem 6.3.

Lemma 9.2. *If the finitely generated group $G$ is uniformly SENS, then* PSPACE $\subseteq$ LEAF(WP($G/Z(G)$)).

Proof. Let $L \subseteq \Gamma^*$ belong to PSPACE. Recall that PSPACE = APTIME. Hence, there is an ATM for $L$ with running time bounded by a polynomial $p(n)$. We can assume that $p(n) \geq n$ for all $n$. Now, consider the language

$$\mathsf{Pad}_{2^{p(n)}}(L) = \left\{ v\$^{2^{p(|w|)}-|v|} \;\middle|\; v \in L \right\},$$

where \$ is some fresh letter. Then $\mathsf{Pad}_{2^{p(n)}}(L)$ is in ALOGTIME: Let $w$ be the input word and let $n = |w|$ be the input length. First, we check whether $w \in \Gamma^*\$^*$ (the latter regular language even belongs to uniform $\mathsf{AC}^0$). If not, we reject, otherwise we can write $w = v\$^k$ for some $k \in \mathbb{N}$ and $v \in \Gamma^*$. Let $m = n - k = |v|$. We next have to verify that $n = 2^{p(m)}$. Using binary search, we compute in DLOGTIME the binary representation of the input length $n$. If $n$ is not a power of two (which is easy to check from the binary representation of $n$), then we reject. Otherwise, let $l = \log_2 n$. The unary representations of $l$ can be obtained from the binary representation of $n$. It remains to check $l = p(m)$. Using $1^l$ we can check whether $|v| = m \leq l$. If not, we reject. Otherwise, we can produce $1^m$. Since polynomials are time constructible we can simply run a clock for $p(m)$ steps, and stop if the number of steps exceeds $l$. Finally, we check whether $v \in L$ (by assumption this can be done in ATIME($p(|v|)$), which is contained in ALOGTIME because of the increased input length). Thus, $\mathsf{Pad}_{2^{p(n)}}(L)$ is in ALOGTIME.

Since we aim for applying Theorem 6.3, we have to encode every symbol $c \in \Gamma \cup \{\$\}$ by a bit string $\gamma(c)$ of length $2^\mu$ for some fixed constant $\mu$. Hence, we consider the language $\gamma(\mathsf{Pad}_{2^{p(n)}}(L))$, which belongs to ALOGTIME as well. Observe that by Lemma 5.6, also $G/Z(G)$ is uniformly SENS. Thus, we can apply Theorem 6.3, which states that there is a uniform family $(P_n)_{n\in\mathbb{N}}$ of $(G/Z(G), \Sigma)$-programs of polynomial length recognizing $\gamma(\mathsf{Pad}_{2^{p(n)}}(L))$. Be aware, however, that "polynomial" here means polynomial in the input length for $\gamma(\mathsf{Pad}_{2^{p(n)}}(L))$. Let $Q_n = P_{2^{p(n)+\mu}}$, which has length $2^{d(n)}$ for some function $d(n) \in O(p(n))$. By the uniformity of $(P_n)_{n\in\mathbb{N}}$ we can compute $1^{d(n)}$ from $1^{2^{p(n)+\mu}}$ in $\mathsf{DTIME}(O(\log(2^{p(n)+\mu}))) = \mathsf{DTIME}(O(p(n)))$. Here we do not have to construct the unary representation of $2^{p(n)+\mu}$: recall that we have a random access Turing machine for the computation. One can easily check whether the content of the address tape (a binary coded number) is at most $2^{p(n)+\mu}$.

Now, we construct an adequate NTM $M$ with $L = \mathsf{LEAF}(M, \mathrm{WP}(G/Z(G)))$: on input $z \in \Gamma^*$ of length $n$ the machine $M$ produces a full binary tree of depth $d(n)$. In the $i$-th leaf ($i \in [0..2^{d(n)} - 1]$) it computes the $i$-th instruction of $Q_n$. By the uniformity of $(P_n)_{n\in\mathbb{N}}$ this can be done in $\mathsf{DTIME}(O(p(n)))$, so $M$ respects a polynomial time bound. Let $\langle j, a, b\rangle$ be the computed instruction. Here $j \in [1..2^{p(n)+\mu}]$ is a position in $\gamma(z\$^{2^{p(n)}-n})$. Depending on the input bit at position $j$ in $\gamma(z\$^{2^{p(n)}-n})$ (which can be easily computed from $z$ and $j$ in polynomial time), the machine then outputs either $a$ or $b$. We then have $\mathrm{leaf}(M, z) = Q_n[\gamma(z\$^{2^{p(n)}-n})]$. Thus, $z \in L$ iff $\gamma(z\$^{2^{p(n)}-n}) \in \gamma(\mathsf{Pad}_{2^{p(n)}}(L))$ iff $Q_n[\gamma(z\$^{2^{p(n)}-n})] \in \mathrm{WP}(G/Z(G))$ iff $\mathrm{leaf}(M, z) \in \mathrm{WP}(G/Z(G))$. □

From Theorem 8.2 and Lemma 9.2 we get:

Corollary 9.3. *If $G$ is uniformly SENS, then* Compressed*WP($G \wr \mathbb{Z}$) is* PSPACE*-hard.*

Since finite non-solvable groups and finitely generated free group of rank at least two are uniformly SENS and their word problems can be solved in LOGSPACE (see [43] for the free group case), we obtain the following from Lemma 9.1 and Corollary 9.3:

Corollary 9.4. *If $G$ is a finite non-solvable group or a finitely generated free group of rank at least two, then* CompressedWP$(G \wr \mathbb{Z})$ *is* PSPACE*-complete.*

We now consider groups $G$ with a self-embedding property: $G \wr H \leq G$ for a non-trival group $H$. For the case that $H$ is a torsion group, we need the following lemma.

Lemma 9.5. *Let $G$ be a finitely generated group with the standard generating set $\Sigma$ such that $G \wr (\mathbb{Z}/p) \leq G$ for some $p \geq 2$. Let $\tau_n$ be a generator for the cyclic group $\mathbb{Z}/p^n$ for $n \geq 1$. Then $G \wr (\mathbb{Z}/p^n) \leq G$ for every $n \geq 1$, and given $n$ in unary encoding and $a \in \Sigma \cup \{\tau_n, \tau_n^{-1}\}$ one can compute in* LOGSPACE *an SLP $\mathcal{G}_{n,a}$ over the terminal alphabet $\Sigma$ such that the mapping $a \mapsto \mathrm{val}(\mathcal{G}_{n,a})$ $(a \in \Sigma \cup \{\tau_n, \tau_n^{-1}\})$ induces an embedding of $G \wr (\mathbb{Z}/p^n)$ into $G$.*

Proof. We fix an embedding $\varphi_1 : G \wr (\mathbb{Z}/p) \to G$. We prove the lemma by induction on $n$. The case $n = 1$ is clear. Consider $n \geq 2$ and assume that we have the embedding $\varphi_{n-1} : G \wr (\mathbb{Z}/p^{n-1}) \to G$. We show that

$$G \wr (\mathbb{Z}/p^n) = G \wr \langle \tau_n \rangle \leq (G \wr \langle \tau_{n-1} \rangle) \wr \langle \tau_1 \rangle = (G \wr (\mathbb{Z}/p^{n-1})) \wr (\mathbb{Z}/p)$$

via an embedding $\psi_n$. For this we define $\psi_n(g) = g \in G \leq G \wr (\mathbb{Z}/p^{n-1})$ for $g \in G$ and $\psi_n(\tau_n) = \tau_{n-1}\tau_1$. It is easy to see that this defines indeed an embedding. The element $\tau_{n-1}\tau_1$ generates a copy of $\mathbb{Z}/p^n$ by cycling through $p$ copies of $\mathbb{Z}/p^{n-1}$ and incrementing mod $p^{n-1}$ the current $\mathbb{Z}/p^{n-1}$-value.

We extend the embedding $\varphi_{n-1} : G \wr (\mathbb{Z}/p^{n-1}) \to G$ to an embedding

$$\varphi_{n-1} : (G \wr (\mathbb{Z}/p^{n-1})) \wr (\mathbb{Z}/p) \to G \wr (\mathbb{Z}/p)$$

by letting $\varphi_{n-1}$ operate as the identity mapping on the right factor $\mathbb{Z}/p$. Finally, we can define $\varphi_n : G \wr (\mathbb{Z}/p^n) \to G$ by $\varphi_n = \psi_n \circ \varphi_{n-1} \circ \varphi_1$, where composition is executed from left to right. We get

$$\varphi_n(\tau_n) = \varphi_1(\varphi_{n-1}(\psi_n(\tau_n))) = \varphi_1(\varphi_{n-1}(\tau_{n-1}\tau_1)) = \varphi_1(\varphi_{n-1}(\tau_{n-1}))\varphi_1(\tau_1).$$

and $\varphi_n(g) = \varphi_1(\varphi_{n-1}(\psi_n(g))) = \varphi_1(\varphi_{n-1}(g))$. By induction on $n$ we get

$$\varphi_n(\tau_n) = \varphi_1^n(\tau_1)\varphi_1^{n-1}(\tau_1) \cdots \varphi_1^2(\tau_1)\varphi_1(\tau_1).$$

and $\varphi_n(g) = \varphi_1^n(g)$ for $g \in G$. Lemma 7.3 implies that given $n$ in unary encoding we can compute in LOGSPACE SLPs for $\varphi_n(\tau_n)$ and all $\varphi_n(g)$ $(g \in G)$. □

Using Lemma 9.5 we can show:

Theorem 9.6. *Let $G$ be a finitely generated group such that $G \wr H \leq G$ for some non-trival group $H$. Then* CompressedWP$(G)$ *is* PSPACE*-hard.*

Proof. Assume that $G \wr H \leq G$ for some $H \neq 1$. We can assume that $H$ is a cyclic group. By Theorem 5.14, $G$ is uniformly SENS. For the case that $H = \mathbb{Z}$ we can directly use Theorem 9.3.

Let us now assume that $H = \mathbb{Z}/p$ for some $p \geq 2$. Since $G$ is uniformly SENS, Lemma 9.2 yields PSPACE $\subseteq$ LEAF(WP$(G/Z(G))$). It therefore suffices to show that CompressedWP$(G)$ is hard for the complexity class $\forall$LEAF(WP$(G/Z(G))$).

Consider a language $L \in \forall$LEAF(WP$(G/Z(G))$) and an input word $z$ of length $n$. Let $\mathcal{J}$ be the SLP that we computed in the proof of Theorem 8.2 in LOGSPACE from $z$. We showed that $z \in L$ if and only if $\mathrm{val}(\mathcal{J}) = 1$ in $G \wr \mathbb{Z}$. Let $s = |\mathrm{val}(\mathcal{J})|$; it is a number in $2^{n^{O(1)}}$. Hence, we can choose a fixed polynomial $q$ such that $p^{q(n)} \geq 2s + 1$ for all input lengths $n$. Let $m = q(n)$. By Remark 8.4 we have $z \in L$ if and only if $\mathrm{val}(\mathcal{J}) = 1$ in $G \wr (\mathbb{Z}/p^m)$.

From $1^m = 1^{q(n)}$ (which can be constructed in LOGSPACE) we can compute by Lemma 9.5 for every $a \in \Sigma \cup \{\tau_m, \tau_m^{-1}\}$ an SLP $\mathcal{G}_{m,a}$ over the terminal alphabet $\Sigma$ such that the mapping $a \mapsto \mathrm{val}(\mathcal{G}_{m,a})$ $(a \in \Sigma \cup \{\tau_m, \tau_m^{-1}\})$ induces an embedding of the wreath product $G \wr (\mathbb{Z}/p^m)$ into $G$. Note that $\log m \in O(\log n)$. Hence, the space needed for the construction of the $\mathcal{G}_{m,a}$ is also logarithmic in the input length $n$. We can assume that the variable sets of the SLPs $\mathcal{G}_{m,a}$ $(a \in \Sigma \cup \{\tau_m, \tau_m^{-1}\})$ and $\mathcal{J}$ are pairwise disjoint. Let $S_{m,a}$ be the start variable of $\mathcal{G}_{m,a}$. We construct an SLP $\mathcal{G}$ by taking the union of the SLPs $\mathcal{G}_{m,a}$ $(a \in \Sigma \cup \{\tau_m, \tau_m^{-1}\})$ and $\mathcal{J}$ and replacing in every right-hand side of $\mathcal{J}$ every occurrence of a terminal symbol $a$ by $S_{m,a}$. We have $\mathrm{val}(\mathcal{G}) = 1$ in $G$ if and only if $\mathrm{val}(\mathcal{J}) = 1$ in $G \wr (\mathbb{Z}/p^m)$ if and only if $z \in L$. □

For Thompson's group $F$ we have $F \wr \mathbb{Z} \leq F$ (Lemma 3.1). Moreover, Lehnert and Schweitzer have shown that $F$ is co-context-free, i.e., the complement of the word problem of $F$ (with respect to any finite generating set) is a context-free language [40]. This implies that the word problem for $F$ belongs to the complexity class LogCFL (the closure of the context-free languages under LOGSPACE-reductions). It is known that $\mathsf{LogCFL} \subseteq \mathsf{DSPACE}(\log^2 n)$ [49]. Therefore, Lemma 7.4 and Theorem 9.6 yield:

Corollary 9.7. *The compressed word problem for Thompson's group $F$ is* PSPACE*-complete.*

In rest of the section we prove that the compressed word problem for some weakly branched groups (including the Grigorchuk group and the Gupta-Sidki groups) is PSPACE-complete as well. We restrict ourselves to weakly branched groups $G$ whose branching subgroup $K$ is not torsion-free.

Lemma 9.8. *Let $G$ be a weakly branched group whose branching subgroup $K$ contains elements of finite order. Then $K$ contains $K \wr (\mathbb{Z}/p)$ for some $p \geq 2$.*

Proof. Let $k \in K$ be an element of finite order. Up to replacing $k$ by a power of itself, we may assume $k$ has prime order $p$. In particular, there exists a vertex $v \in X^*$ whose orbit under $k$ has size $p$. Then $\langle v * K, k \rangle \cong K \wr (\mathbb{Z}/p)$ is the desired subgroup. □

The following result applies in particular to the Grigorchuk group and the Gupta-Sidki groups, showing both their compressed word problems to be PSPACE-complete.

Corollary 9.9. *Let $G$ be a weakly branched group whose branching subgroup is finitely generated and contains elements of finite order.*

- *CompressedWP$(G)$ is* PSPACE*-hard.*
- *If $G$ is also contracting, then CompressedWP$(G)$ is* PSPACE*-complete.*

Proof. By Lemma 9.8 the branching subgroup $K$ of $G$ satisfies the hypotheses of Theorem 9.6, so the compressed word problem for $K$ (and hence $G$) is PSPACE-hard.

If $G$ is also contracting, then the word problem of $G$ is in LOGSPACE by Proposition 3.5, so Lemma 7.4 implies that CompressedWP$(G)$ belongs to PSPACE. □

Corollaries 9.4, 9.7, and 9.9 give new (and natural) examples for groups where the compressed word problem is provably more difficult than the word problem (since polyL is a proper subset of PSPACE). The first example for such a group was provided in [62]: it is an automaton group where the word problem is PSPACE-complete and the compressed word problem is EXPSPACE-complete. Let us also remark, that the Grigorchuk group is an example of a group where the compressed word problem is even more difficult than the power word problem. For the power word problem [48]

the input consists of a word $w_1^{z_1} w_2^{z_2} \cdots w_n^{z_n}$, where the exponents $z_i$ are given in binary representation and the $w_i$ are explicitly given words over the group generators. In terms of complexity, the power word problem lies between the word problem and the compressed word problem. It is shown in [48] that the power word problem for the Grigorchuk group belongs to LOGSPACE, whereas by Corollary 9.9 the compressed word problem is PSPACE-complete.

## 10 CONCLUSION AND OPEN PROBLEMS

We have added an algorithmic constraint (uniformly SENS) to the algebraic notion of being a non-solvable group which implies that the word problem is $\mathsf{NC}^1$-hard (resp. ALOGTIME-hard). Using this, we produced several new examples of non-solvable groups with an ALOGTIME-hard word problem. However, the question remains open whether all non-solvable groups have ALOGTIME-hard word problem, even if they are not SENS. For every contracting self-similar group the word problem belongs LOGSPACE. Here, the question remains whether there exists a contracting self-similar group with a LOGSPACE-complete word problem. In particular, is the word problem for the Grigorchuk group LOGSPACE-complete? (we proved that it is ALOGTIME-hard). Also the precise complexity of the word problem for Thompson's group $F$ is open. It is ALOGTIME-hard and belongs to LOGCFL; the latter follows from [40]. In fact, from the proof in [40] one can deduce that the word problem for $F$ belongs to LOGDCFL (the closure of the deterministic context-free languages with respect to LOGSPACE-reductions).

Finally, we showed that the compressed word problem is PSPACE-hard for every weakly branched group whose branching subgroup is finitely generated and contains elements of finite order. It remains open whether this result holds for all weakly branched groups.

## REFERENCES


[1] Miklós Abért. 2005. Group laws and free subgroups in topological groups. *Bull. London Math. Soc.* 37, 4 (2005), 525–534. https://doi.org/10.1112/S002460930500425X

[2] Ian Agol. 2013. The virtual Haken conjecture. *Documenta Mathematica* 18 (2013), 1045–1087. With an appendix by Ian Agol, Daniel Groves, and Jason Manning.

[3] Sanjeev Arora and Boaz Barak. 2009. *Computational Complexity - A Modern Approach.* Cambridge University Press.

[4] David A. Mix Barrington. 1989. Bounded-Width Polynomial-Size Branching Programs Recognize Exactly Those Languages in $NC^1$. *J. Comput. Syst. Sci.* 38, 1 (1989), 150–164. https://doi.org/10.1016/0022-0000(89)90037-8

[5] David A. Mix Barrington and Denis Thérien. 1988. Finite Monoids and the Fine Structure of $NC^1$. *J. ACM* 35 (1988), 941–952.

[6] Laurent Bartholdi, Michael Figelius, Markus Lohrey, and Armin Weiß. 2020. Groups with ALOGTIME-Hard Word Problems and PSPACE-Complete Circuit Value Problems. In *35th Computational Complexity Conference, CCC 2020, July 28-31, 2020, Saarbrücken, Germany (Virtual Conference) (LIPIcs, Vol. 169)*, Shubhangi Saraf (Ed.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 29:1–29:29. https://doi.org/10.4230/LIPIcs.CCC.2020.29

[7] Laurent Bartholdi, Rostislav I. Grigorchuk, and Zoran Šunik. 2003. Branch groups. In *Handbook of algebra, Vol. 3.* Handb. Algebr., Vol. 3. Elsevier/North-Holland, Amsterdam, 989–1112. https://doi.org/10.1016/S1570-7954(03)80078-5

[8] Laurent Bartholdi and Volodymyr V. Nekrashevych. 2008. Iterated monodromy groups of quadratic polynomials. I. *Groups Geom. Dyn.* 2, 3 (2008), 309–336. https://doi.org/10.4171/GGD/42

[9] Martin Beaudry, Pierre McKenzie, Pierre Péladeau, and Denis Thérien. 1997. Finite Monoids: From Word to Circuit Evaluation. *SIAM J. Comput.* 26, 1 (1997), 138–152.

[10] Richard Beigel and John Gill. 1992. Counting Classes: Thresholds, Parity, Mods, and Fewness. *Theor. Comput. Sci.* 103, 1 (1992), 3–23. https://doi.org/10.1016/0304-3975(92)90084-S

[11] William W. Boone. 1959. The Word Problem. *Ann. of Math.* 70, 2 (1959), 207–265.

[12] Daniel P. Bovet, Pierluigi Crescenzi, and Riccardo Silvestri. 1992. A uniform approach to define complexity classes. *Theoretical Computer Science* 104, 2 (1992), 263–283.

[13] John W. Cannon, William J. Floyd, and Walter R. Parry. 1996. Introductory notes on Richard Thompson's groups. *L'Enseignement Mathématique* 42, 3 (1996), 215–256.

[14] Hervé Caussinus, Pierre McKenzie, Denis Thérien, and Heribert Vollmer. 1998. Nondeterministic $NC^1$ Computation. *J. Comput. Syst. Sci.* 57, 2 (1998), 200–212. https://doi.org/10.1006/jcss.1998.1588

[15] Moses Charikar, Eric Lehman, Ding Liu, Rina Panigrahy, Manoj Prabhakaran, Amit Sahai, and Abhi Shelat. 2005. The smallest grammar problem. *IEEE Transactions on Information Theory* 51, 7 (2005), 2554–2576.

[16] Max Dehn. 1911. Über unendliche diskontinuierliche Gruppen. *Math. Ann.* 71, 1 (1911), 116–144. https://doi.org/10.1007/BF01456932

[17] Michael Figelius, Moses Ganardi, Markus Lohrey, and Georg Zetzsche. 2020. The Complexity of Knapsack Problems in Wreath Products. In *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*. 126:1–126:18. https://doi.org/10.4230/LIPIcs.ICALP.2020.126

[18] Moses Ganardi and Markus Lohrey. 2019. A Universal Tree Balancing Theorem. *ACM Transactions on Computation Theory* 11, 1 (2019), 1:1–1:25.

[19] Michael R. Garey and David S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP–completeness*. Freeman.

[20] Max Garzon and Yechezkel Zalcstein. 1991. The complexity of Grigorchuk groups with application to cryptography. *Theoretical Computer Science* 88, 1 (1991), 83–98.

[21] Rostislav I. Grigorchuk. 1980. On Burnside's problem on periodic groups. *Funktsional. Anal. i Prilozhen.* 14, 1 (1980), 53–54.

[22] Rostislav I. Grigorchuk and Zoran Šunić. 2006. Asymptotic aspects of Schreier graphs and Hanoi Towers groups. *C. R. Math. Acad. Sci. Paris* 342, 8 (2006), 545–550. https://doi.org/10.1016/j.crma.2006.02.001

[23] Victor S. Guba and Mark V. Sapir. 1999. On subgroups of the R. Thompson group $F$ and other diagram groups. *Mat. Sb.* 190, 8 (1999), 3–60. https://doi.org/10.1070/SM1999v190n08ABEH000419

[24] Narain Gupta and Saïd Sidki. 1983. On the Burnside problem for periodic groups. *Math. Z.* 182, 3 (1983), 385–388. https://doi.org/10.1007/BF01179757

[25] Frédéric Haglund and Daniel T. Wise. 2010. Coxeter groups are virtually special. *Advances in Mathematics* 224, 5 (2010), 1890–1903.

[26] Ulrich Hertrampf. 1990. Relations among MOD-classes. *Theoretical Computer Science* 74, 3 (1990), 325–328. https://doi.org/10.1016/0304-3975(90)90081-R

[27] Ulrich Hertrampf. 1994. *Über Komplexitätsklassen, die mit Hilfe von k-wertigen Funktionen definiert werden.* Habilitationsschrift. Universität Würzburg.

[28] Ulrich Hertrampf. 1997. The shapes of trees. In *Proceedings of the 3rd Annual International Conference on Computing and combinatorics (COCOON 1997), Shanghai (China)*. Lecture Notes in Computer Science, Vol. 1276. Springer, 412–421.

[29] Ulrich Hertrampf. 2000. Algebraic acceptance mechanisms for polynomial time machines. *SIGACT News* 31, 2 (2000), 22–33. https://doi.org/10.1145/348210.348215

[30] Ulrich Hertrampf, Clemens Lautemann, Thomas Schwentick, Heribert Vollmer, and Klaus W. Wagner. 1993. On the power of polynomial time bit-reductions. In *Proceedings of the Eighth Annual Structure in Complexity Theory Conference (San Diego, CA, 1993)*. IEEE Computer Society Press, 200–207.

[31] Ulrich Hertrampf, Heribert Vollmer, and Klaus Wagner. 1996. On balanced versus unbalanced computation trees. *Mathematical Systems Theory* 29, 4 (1996), 411–421.

[32] Yoram Hirshfeld, Mark Jerrum, and Faron Moller. 1994. A Polynomial-time Algorithm for Deciding Equivalence of Normed Context-free Processes. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science, FOCS 1994*. IEEE Computer Society, 623–631. https://doi.org/10.1109/SFCS.1994.365729

[33] Derek Holt and Sarah Rees. 2020. *The compressed word problem in relatively hyperbolic groups.* Technical Report. arXiv.org. https://arxiv.org/abs/2005.13917.

[34] Derek F. Holt, Markus Lohrey, and Saul Schleimer. 2019. Compressed Decision Problems in Hyperbolic Groups. In *Proceedings of the 36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019 (LIPIcs, Vol. 126)*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 37:1–37:16. http://www.dagstuhl.de/dagpub/978-3-95977-100-9

[35] Derek F. Holt, Sarah Rees, and Claas E. Röver. 2017. *Groups, Languages and Automata.* London Mathematical Society Student Texts, Vol. 88. Cambridge University Press. https://doi.org/10.1017/9781316588246

[36] Birgit Jenner, Pierre McKenzie, and Denis Thérien. 1996. Logspace and logtime leaf languages. *Information and Computation* 129, 1 (1996), 21–33.

[37] Howard J. Karloff and Walter L. Ruzzo. 1989. The Iterated Mod Problem. *Information and Computation* 80, 3 (1989), 193–204.

[38] Daniel König and Markus Lohrey. 2018. Evaluation of circuits over nilpotent and polycyclic groups. *Algorithmica* 80, 5 (2018), 1459–1492.

[39] Daniel König and Markus Lohrey. 2018. Parallel identity testing for skew circuits with big powers and applications. *IJAC* 28, 6 (2018), 979–1004.

[40] Jörg Lehnert and Pascal Schweitzer. 2007. The co-word problem for the Higman-Thompson group is context-free. *Bulletin of the London Mathematical Society* 39, 2 (02 2007), 235–241. https://doi.org/10.1112/blms/bdl043

[41] Martin W. Liebeck, Eamonn A. O'Brien, Aner Shalev, and Pham Huu Tiep. 2010. The Ore conjecture. *J. Eur. Math. Soc. (JEMS)* 12, 4 (2010), 939–1008. https://doi.org/10.4171/JEMS/220

[42] Yury Lifshits and Markus Lohrey. 2006. Querying and Embedding Compressed Texts. In *Proceedings of the 31th International Symposium on Mathematical Foundations of Computer Science, MFCS 2006 (Lecture Notes in Computer Science, Vol. 4162)*. Springer, 681–692.

[43] Richard J. Lipton and Yechezkel Zalcstein. 1977. Word Problems Solvable in Logspace. *Journal of the Association for Computing Machinery* 24, 3 (1977), 522–526.

[44] Markus Lohrey. 2006. Word problems and membership problems on compressed words. *SIAM J. Comput.* 35, 5 (2006), 1210 – 1240.

[45] Markus Lohrey. 2011. Leaf languages and string compression. *Information and Computation* 209, 6 (2011), 951–965.

[46] Markus Lohrey. 2014. *The Compressed Word Problem for Groups.* Springer. https://doi.org/10.1007/978-1-4939-0748-9

[47] Markus Lohrey and Christian Mathissen. 2013. Isomorphism of regular trees and words. *Information and Computation* 224 (2013), 71–105.

[48] Markus Lohrey and Armin Weiß. 2019. The power word problem. In *Proceedings of MFCS 2019 (LIPIcs, Vol. 138)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 43:1–43:15.

[49] Philip M. Lewis II, Richard Edwin Stearns, and Juris Hartmanis. 1965. Memory bounds for recognition of context-free and context-sensitive languages. In *Proceedings of the 6th Annual Symposium on Switching Circuit Theory and Logical Design*. IEEE Computer Society, 191–202.

[50] Kurt Mehlhorn, R. Sundar, and Christian Uhrig. 1994. Maintaining Dynamic Sequences Under Equality-Tests in Polylogarithmic Time. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 1994*. ACM/SIAM, 213–222. http://dl.acm.org/citation.cfm?id=314464.314496

[51] Volodymyr Nekrashevych. 2005. *Self-similar groups*. Mathematical Surveys and Monographs, Vol. 117. American Mathematical Society, Providence, RI. xii+231 pages. https://doi.org/10.1090/surv/117

[52] Piotr S. Novikov. 1955. On the algorithmic unsolvability of the word problem in group theory. *Trudy Mat. Inst. Steklov* (1955), 1–143. In Russian.

[53] Wojciech Plandowski. 1994. Testing Equivalence of Morphisms on Context-Free Languages. In *Proceedings of the Second Annual European Symposium on Algorithms, ESA 1994 (Lecture Notes in Computer Science, Vol. 855)*. Springer, 460–470. https://doi.org/10.1007/BFb0049431

[54] David Robinson. 1993. *Parallel Algorithms for Group Word Problems*. Ph. D. Dissertation. University of California, San Diego.

[55] Joseph J. Rotman. 1995. *An Introduction to the Theory of Groups (fourth edition)*. Springer.

[56] Amir Shpilka and Amir Yehudayoff. 2010. Arithmetic Circuits: A survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science* 5, 3-4 (2010), 207–388. https://doi.org/10.1561/0400000039

[57] Hans-Ulrich Simon. 1979. Word problems for groups and contextfree recognition. In *Proceedings of Fundamentals of Computation Theory, FCT 1979*. Akademie-Verlag, 417–422.

[58] Roman Smolensky. 1987. Algebraic Methods in the Theory of Lower Bounds for Boolean Circuit Complexity. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*. 77–82. https://doi.org/10.1145/28395.28404

[59] Jacques Tits. 1972. Free subgroups in linear groups. *Journal of Algebra* 20, 2 (1972), 250–270.

[60] Heribert Vollmer. 1999. *Introduction to Circuit Complexity*. Springer, Berlin.

[61] Stephan Waack. 1990. The Parallel Complexity of Some Constructions in Combinatorial Group Theory. *Journal of Information Processing and Cybernetics EIK* 26 (1990), 265–281.

[62] Jan Philipp Wächter and Armin Weiß. 2020. An Automaton Group with PSPACE-Complete Word Problem. In *37th International Symposium on Theoretical Aspects of Computer Science, STACS 2020, March 10-13, 2020, Montpellier, France*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 6:1–6:17. https://doi.org/10.4230/LIPIcs.STACS.2020.6

[63] John S. Wilson. 1980. Embedding theorems for residually finite groups. *Math. Z.* 174, 2 (1980), 149–157. https://doi.org/10.1007/BF01293535

[64] Daniel T. Wise. 2009. Research announcement: the structure of groups with a quasiconvex hierarchy. *Electronic Research Announcements in Mathematical Sciences* 16 (2009), 44–55.