The Word Problem for Finitary Automaton Groups

Maximilian Kotowsky

Insitut für Formale Methoden der Informatik (FMI)
Universität Stuttgart
Universitätsstraße 38
70569 Stuttgart, Germany

Jan Philipp Wächter *

Dipartimento di Matematica
Politecnico di Milano
Piazza Leonardo da Vinci, 32
20133 Milano, Italy

February 22, 2023

A finitary automaton group is a group generated by an invertible, deterministic finite-state letter-to-letter transducer whose only cycles are self-loops at an identity state. We show that, for this presentation of finite groups, the uniform word problem is coNP-complete. Here, the input consists of a finitary automaton together with a finite state sequence and the question is whether the sequence acts trivially on all input words. Additionally, we also show that the respective compressed word problem, where the state sequence is given as a straight-line program, is PSpace-complete. In both cases, we give a direct reduction from the satisfiablity problem for (quantified) boolean formulae.
Keywords. Automaton Group, Word Problem, Finitary, Activity.

*The second author is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 492814705.

1 Introduction

There a many connections between groups and automata (see e. g. [22]). In this article, we are mostly concerned with automaton groups, where the term automaton usually refers to an invertible, deterministic finite-state letter-to-letter transducer. In such an automaton, every state q induces a function mapping an input word u to the output word obtained by stating in q and following the path labeled by u in the input. Since the automaton is invertible, every such function is a bijection and the closure under composition of these functions (and their inverses) forms a group. This is the group generated by the automaton and any group arising in this way is an automaton group. Not every group is an automaton group but the class of automaton groups contains some very interesting examples (see e. g. [4]). The probably most famous one is Grigochuk's group, which – among other interesting properties – was the historically first group of intermediate growth (i. e. the numbers of elements that can be written as a word of length at most n over the generators grows slower than any exponential function but faster than any polynomial; see [12] for an introduction to this topic).

These interesting examples also led to an investigation of the algorithmic properties of automaton groups, where the presentation using automata is an alternative to the classical one using (typically finitely many) generators and relations. It turns out that this presentation is still quite powerful as many decision problems remain undecidable. For example, it is known that there is an automaton group with an undecidable conjugacy problem [24] (given two group elements, check whether they are conjugate) and one with an undecidable order problem [11, 3] (given a group element, check whether it has finite order). Decidability of the finiteness problem for automaton groups (given an automaton, check whether its generated group is finite) is still an open problem but the corresponding problem for semigroup has been shown to be undecidable [10].

The word problem (given a group element, check whether it is the neutral element), however, seems to have a special role for automaton groups. It is well known to be decidable and a guess and check approach also yields that the problem can be solved in non-deterministic linear space, even in the uniform case (where the generating automaton is also part of the input) [23, 9]. Regarding lower bounds, Armin Weiß and the second author proved that there is an automaton group with a PSpace-complete word problem [25].

In this work, we will apply similar ideas to investigate the complexity of the word problem for the lowest level of the activity hierarchy for automaton groups introduced by Sidki [21]. This hierarchy classifies automaton groups based on the structure of the cycles in the generating automaton. At the lowest level, which belongs to the class of finitary automata and finitary automaton groups, the only cycles are the self-loops at an identity state (i. e. a state where the output word is always the same as the input word). It turns out that this class coincides with the class of (all) finite groups.

On the next level, the class of bounded automata and bounded automaton groups, every path in the automaton may contain at most one cycle (not counting the self-loops at a possible identity state). This class still seems "finite enough" for many problems to be decidable. For example, the finiteness problem [7] as well as the order problem [8] are

decidable and there are positive results on the conjugacy problem [8]; the word problem of a bounded automaton group can be solved in deterministic logarithmic space [19, 2] and its complement is an ET0L language [6].

We will be interested in the finitary level. As we have discussed, studying the word problem of these groups is the same as studying the word problem of arbitrary finite groups. It is well known that a group is finite if and only if its word problem (i. e. the formal language of words over the generators representing the neutral element) is regular. While this does not settle the precise complexity for the individual groups entirely, we will approach this setting from a different perspective. We will consider the uniform word problem, where the group is part of the input in a suitable presentation. Typical such presentations include, for example, the classical one with generators and relations, Caley graphs and tables or presenting the elements as matrices or permutations. Our presentation of choice is that of using an automaton (in the way described above). Here, we will show that the uniform word problem is coNP-complete by giving a direct reduction from the satisfiability problem for boolean formulae. Then, we will show that the uniform compressed word problem, where the input state sequence is not given directly but only compressed in the form of a context-free grammar (or, more precisely, a straight-line program), is PSpace-complete and, thus, exponentially harder (under common complexity theoretic assumptions). This reflects a similar (potentially) exponential gap in the general case [25]. We prove this latter result by giving a direct reduction from the satisfiability problem for quantified boolean formulae. This approach of simulating logical formulae in automata is similar to the techniques used in [25] and we hope that the general idea can be extended to further settings, for example, to obtain lower bound results for further levels of the activity hierarchy. The underlying idea is to use certain commutators for simulating logical conjunctions. This is often attributed to Barrington, who showed [1] that non-solvable finite groups have ALogTime-complete word problems (see [2] for more results in that direction). However, there are also similar ideas predating Barrington [16, 17, 18, 14].

2 Preliminaries

Logic. For this paper, we will require some basic knowledge about propositional and first-order logic. We use $\bot$ to denote a *false* truth value and $\top$ to denote the truth value *true*. We let $\mathbb{B} = \{\bot, \top\}$ and may evaluate the truth value $\mathcal{A}(\varphi)$ of a formula φ over the *variables* $\mathbb{X}$ under an *assignment* $\mathcal{A} : \mathbb{X} \to \mathbb{B}$ in the usual way. If this evaluates to $\top$, we say that $\mathcal{A}$ *satisfies* φ and φ is *satisfiable* if it is satisfied by some assignment. A *literal* is either a variable x or the negation $\neg x$ of a variable. In the first case, the literal is *positive* and, in the second case, it is *negative*. A *clause* is a disjunction $\bigvee_{i=1}^{n} L_i$ of literals L_i. A conjunction $\bigwedge_{k=1}^{K} C_k$ of clauses C_k is a formula in *conjunctive normal form*. If all the clauses contain exactly 3 distinct literals, we say that the formula is in 3-*conjunctive normal form*.

Words and Group Operations. An alphabet is a non-empty, finite set Σ. A finite sequence $w = a_1 \dots a_\ell$ of elements $a_1, \dots, a_\ell \in \Sigma$ is a *word* and its *length* is $|w| = \ell$. The unique word of length 0 is denoted by ε and the set of all words over Σ is Σ^*, which forms a monoid whose operation is the concatenation of words (and whose neutral element is ε).

We will often work with words in the context of generating a group. In this case, we assume that, for an alphabet Q, we have a disjoint copy $Q^{-1} = \{q^{-1} \mid q \in Q\}$ of formal inverse letters. For the set of words over such positive and negative letters, we write $Q^{\pm *} = (Q \cup Q^{-1})^*$ and we may extend the notation q^{-1} to words by letting $(q_1 \dots q_\ell)^{-1} = q_\ell^{-1} \dots q_1^{-1}$ where we additionally also use the convention $(q^{-1})^{-1} = q$. We say a group G is *generated* by Q if there is a monoid homomorphism $\pi : Q^{\pm *} \to G$ with $\pi(q^{-1}) = \pi(q)^{-1}$. In this context, we write $\boldsymbol{p} = \boldsymbol{q}$ *in* G for $\pi(\boldsymbol{p}) = \pi(\boldsymbol{q})$ (where $\boldsymbol{p}, \boldsymbol{q} \in Q^{\pm *}$) and also $\boldsymbol{p} = g$ *in* G if $\pi(\boldsymbol{p}) = g$. So, for example, we write $\boldsymbol{p} = \mathbb{1}$ in G if $\pi(\boldsymbol{p})$ is the neutral element of the group G, which we usually denote by $\mathbb{1}$.

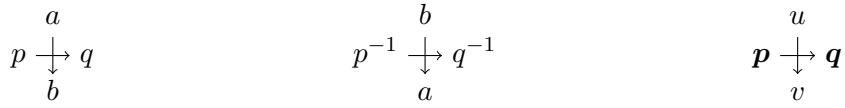
In addition to taking the inverse, we lift further group operations to words. In analogy to the *conjugation* $g^k = k^{-1}gk$ of some group element $g \in G$ by another one $k \in G$, we also write $\boldsymbol{q}^{\boldsymbol{p}}$ for the word $\boldsymbol{q}^{\boldsymbol{p}} = \boldsymbol{p}^{-1}\boldsymbol{q}\boldsymbol{p}$ (where $\boldsymbol{p}, \boldsymbol{q} \in Q^{\pm *}$). Note that this notation is compatible with the conjugation as we have $\pi(\boldsymbol{q}^{\boldsymbol{p}}) = \pi(\boldsymbol{q})^{\pi(\boldsymbol{p})}$. We also do the same for the *commutator* $[h, g] = h^{-1}g^{-1}hg$ of two group elements $g, h \in G$ and write $[\boldsymbol{q}, \boldsymbol{p}]$ for the word $[\boldsymbol{q}, \boldsymbol{p}] = \boldsymbol{q}^{-1}\boldsymbol{p}^{-1}\boldsymbol{q}\boldsymbol{p}$. Again, this is compatible with the projection π: $\pi([\boldsymbol{q}, \boldsymbol{p}]) = [\pi(\boldsymbol{q}), \pi(\boldsymbol{p})]$.

Automata and Automaton Groups. In the context of this paper, an automaton is a finite state letter-to-letter transducer. Formally, an *automaton* $\mathcal{T}$ is a triple (Q, Σ, δ) where Q is a finite, non-empty set of *states*, Σ is the (input and output) *alphabet* of $\mathcal{T}$ and $\delta \subseteq Q \times \Sigma \times \Sigma \times Q$ is the *transition* relation. In this context, we usually write $p \xrightarrow{a/b} q$ for the tuple $(p, a, b, q) \in Q \times \Sigma \times \Sigma \times Q$. This is a transition *starting* in p, *ending* in q with *input* a and *output* b.

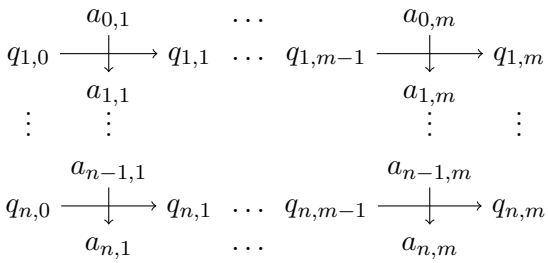
An automaton $\mathcal{T} = (Q, \Sigma, \delta)$ is *deterministic* and *complete* if we have $d_{p,a} = |\{p \xrightarrow{a/b} q \mid b \in \Sigma, q \in Q\}| = 1$ for all $p \in Q$ and $a \in \Sigma$. It is additionally *invertible* if we also have $d'_{p,b} = |\{p \xrightarrow{a/b} q \mid a \in \Sigma, q \in Q\}| = 1$ for all $p \in Q$ and $b \in \Sigma$. We will call a deterministic, complete and invertible automaton a *$\mathscr{G}$-automaton*.

Another way of indicating that we have a transition $p \xrightarrow{a/b} q \in \delta$ is to use the cross diagram in Figure 1a. Multiple cross diagrams may be combined into a larger one. For example, the cross diagram in Figure 1d indicates that we have $q_{i,j-1} \xrightarrow{a_{i-1,j}/a_{i,j}} q_{i,j} \in \delta$ for all $1 \le i \le n$ and $1 \le j \le m$. Typically, we will omit unnecessary intermediate states if we do not need to name them. Additionally, we also allow abbreviations in the form of words (instead of only single letters) in the input and output and state sequences (i. e. words over Q) on the left and the right. Note, however, that here the right-most state of the sequence is considered to be the first state,[1] which results in the abbreviated cross diagram in Figure 1c for $\boldsymbol{p} = q_{n,0} \dots q_{1,0}$, $u = a_{0,1} \dots a_{0,m}$, $v = a_{n,1} \dots a_{n,m}$ and $\boldsymbol{q} = q_{n,m} \dots q_{1,m}$.

[1] This makes sense as we will later on define a *left* action of the states on the words over Σ.



(a) Cross diagrams (b) Inverse cross diagrams (c) Abbreviated cross diagram



(d) Multiple crosses combined in one diagram

Figure 1: Single, inverted, combined and abbreviated cross diagrams

For a deterministic and complete automaton $\mathcal{T} = (Q, \Sigma, \delta)$, there exists exactly one cross diagram of the form in Figure 1c for every $\boldsymbol{q} \in Q^*$ and $u \in \Sigma^*$. If $\mathcal{T}$ is additionally invertible (i. e. it is a $\mathscr{G}$-automaton), we define that we have the cross diagram in Figure 1b for $p, q \in Q$ and $a, b \in \Sigma$ whenever we have the cross diagram from Figure 1a. Note that we have flipped the cross diagram along its horizontal axis and inverted the states. In this case, the cross diagram in Figure 1c uniquely exists for all $\boldsymbol{p} \in Q^{\pm *}$ and all $u \in \Sigma^*$ (although we now also allows states from Q^{-1}).

This allows us to define a left action of $Q^{\pm *}$ on Σ^* where the action of $\boldsymbol{q} \in Q^{\pm *}$ on a word $u \in \Sigma^*$ is given by $\boldsymbol{q} \circ u = v$ where v is uniquely obtained from the cross diagram Figure 1c (the empty state sequence acts as the identity on all words by convention). The reader may verify that we indeed have $\boldsymbol{q}^{-1}\boldsymbol{q} \circ u = u = \boldsymbol{q}\boldsymbol{q}^{-1} \circ u$ with our definition of inverting cross diagrams.

For two state sequences $\boldsymbol{p}, \boldsymbol{q} \in Q^{\pm *}$ of a $\mathscr{G}$-automaton $\mathcal{T} = (Q, \Sigma, \delta)$, we define the relation

$$\boldsymbol{p} =_{\mathcal{T}} \boldsymbol{q} \iff \forall u \in \Sigma^* : \boldsymbol{p} \circ u = \boldsymbol{q} \circ u.$$

It turns out that this relation is a congruence, which allows us to consider the monoid $Q^{\pm *}/=_{\mathcal{T}}$ formed by its classes. In fact, this monoid has a group structure (where the class of $\boldsymbol{q}^{-1}$ is the inverse of the class of $\boldsymbol{q}$) and this is the *group generated* by $\mathcal{T}$. Any group generated by some $\mathscr{G}$-automaton is called an *automaton group*.

We use the common graphical depiction of automata, which results in a $\Sigma \times \Sigma$-labeled finite directed graph. If this graph does not have any cycles except for the a/a labeled self-loops at an identity state,[2] we say that the automaton is *finitary*. The *depth* of a finitary $\mathscr{G}$-automaton is the minimal number d such that, after reading at least d many letters,

[2]Note that any complete finite automaton must contain a cycle and that, thus, every finitary $\mathscr{G}$-automaton has an identity state.

we are always in the identity state (regardless where we started). A group generated by a finitary $\mathscr{G}$-automaton is a *finitary* automaton group. Since, with a finitary $\mathscr{G}$-automaton, a state sequence may only act non-trivially on the first d letters (where d is the depth of the generating automaton), a finitary automaton group is necessarily finite. On the other hand, any finite group G is generated by the finitary automaton (G, G, δ) with $\delta = \{g \xrightarrow{h/gh} \mathbb{1} \mid g, h \in G\}$. Thus, studying finitary automaton groups is the same as studying finite groups but we are interested in a certain way of presenting these groups.

Complexity. We need some notions from complexity theory for this paper. However, we will not go into details about complexity theory and refer the reader to standard text books on the topic (such as [20]) instead. Regarding complexity classes, we need the class coNP which contains all problems whose complement can be solved in non-deterministic polynomial time (i. e. is in NP) and the class PSpace of problems solvable in polynomial space (where it does not matter whether we consider deterministic or non-deterministic algorithms by Savitch's theorem [20, Theorem 7.5]). Additionally, we need LogSpace-computable functions (where LogSpace refers to deterministic logarithmic space). When it comes to reductions, we will exclusively work with *many-one* LogSpace-*reductions*. Formally, such a reduction from a problem A to a problem B is a LogSpace-computable function f mapping instances of A to instances of B such that positive instances are mapped to positive instances and negative instances are mapped to negative ones. A problem A is $\mathcal{C}$-*hard* for some complexity class $\mathcal{C}$ if any problem $C \in \mathcal{C}$ can be reduced to A (using a many-one LogSpace-reduction). Typically, this is done by reducing a problem which is already known to be $\mathcal{C}$-hard to A (as many-one LogSpace-reductions are closed under composition, see e. g. [20, Proposition 8.2]). If a $\mathcal{C}$-hard problem is also contained in $\mathcal{C}$, it is $\mathcal{C}$-*complete*.

Balanced Iterated Commutators. In addition to the normal commutator of two elements, we also need iterated commutators which we recursively split in the middle.

Definition 2.1 (compare to [25, Definition 3])**.** For words $\alpha, \beta, \boldsymbol{q}_1, \dots, \boldsymbol{q}_D \in Q^{\pm *}$ where $D = 2^d$ is a power of two, we define $B_{\beta,\alpha}[\boldsymbol{q}_D, \dots, \boldsymbol{q}_1]$ by induction on d and let

$$B_{\beta,\alpha}[\boldsymbol{q}_1] = \boldsymbol{q}_1 \quad \text{and}$$
$$B_{\beta,\alpha}[\boldsymbol{q}_D, \dots, \boldsymbol{q}_1] = \left[B_{\beta,\alpha}[\boldsymbol{q}_D, \dots, \boldsymbol{q}_{\frac{D}{2}+1}]^{\beta},\ B_{\beta,\alpha}[\boldsymbol{q}_{\frac{D}{2}}, \dots, \boldsymbol{q}_1]^{\alpha}\right].$$

This also immediately yields an operation $B_{\beta,\alpha}[g_D, \dots, g_1]$ for group elements $g_1, \dots, g_D$ using the natural evaluation in the group.

The reason for introducing balanced iterated commutators is that we may use them to simulate a D-ary logical conjunction in groups. The idea here is that the neutral element $\mathbb{1}$ belongs to $\bot$ and all other elements are considered to belong to $\top$. One direction of the simulation then works in any group as we state in the following fact.[3]

[3] The fact can be proved using a simple induction on the structure of the balanced iterated commutators, see [25, Fact 4].

Fact 2.2 (see [25, Fact 4]). *Let a group G be generated by the alphabet Q and let $\alpha, \beta, \boldsymbol{q}_1, \ldots, \boldsymbol{q}_D \in Q^{\pm *}$ for some $D = 2^d$. If there is some $1 \leq i \leq D$ with $\boldsymbol{q}_i = \mathbb{1}$ in G, we have $B_{\beta,\alpha}[\boldsymbol{q}_D, \ldots, \boldsymbol{q}_1] = \mathbb{1}$ in G.*

The reason that we use balanced iterated commutators (instead of the usual ones of the form $\left[g_D, [g_{D-1}, \ldots, g_1]\right]$) is that, this way, the depth remains logarithmic in the number of entries. This allows us to compute the balanced iterated commutator from its entries in logarithmic space.

Fact 2.3 (see [25, Lemma 7]). *The balanced commutator $B_{\beta,\alpha}[\boldsymbol{q}_D, \ldots, \boldsymbol{q}_1]$ can be computed from $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_D \in Q^{\pm *}$ and $\alpha, \beta \in Q^{\pm *}$ in logarithmic space.*

Normally, we cannot simply add balanced iterated commutators to cross diagrams and expect the resulting diagram to still hold. However, this is possible if all the entries (and the conjugating elements α and β) act trivially on the input word (which can be seen by a simple induction on the structure of the balanced iterated commutators).

Fact 2.4 (see [25, Fact 8]). *Let $\mathcal{T} = (Q, \Sigma, \delta)$ be a $\mathscr{G}$-automaton, $u \in \Sigma^*$, $\alpha, \beta, \boldsymbol{q}_1, \ldots, \boldsymbol{q}_D \in Q^{\pm *}$ with $D = 2^d$ then the cross diagrams*

$$\begin{array}{ccc} u & & \\ \boldsymbol{q}_1 & \to & \boldsymbol{q}'_1 \\ u & & \\ \vdots & \vdots & \vdots \\ u & & \\ \boldsymbol{q}_D & \to & \boldsymbol{q}'_D \\ u & & \end{array} \qquad \begin{array}{ccc} u & & \\ \alpha & \to & \alpha' \\ u & & \\ & \textit{and} & \\ u & & \\ \beta & \to & \beta' \\ u & & \end{array}$$

imply the cross diagram

$$\begin{array}{ccc} u & & \\ B_{\beta,\alpha}[\boldsymbol{q}_D, \ldots, \boldsymbol{q}_1] & \to & B_{\beta',\alpha'}[\boldsymbol{q}'_D, \ldots, \boldsymbol{q}'_1] \\ u & & \end{array} .$$

The Group A_5. We have already seen in Fact 2.2 that the balanced iterated commutator collapses to $\mathbb{1}$ (which corresponds to $\bot$) if one of its entries is equal to $\mathbb{1}$ (i. e. corresponds to $\bot$). This is one of the two directions to use the commutators as logical conjunctions. The other direction, however, does not hold for all elements of all groups. That is why we next look at the group A_5 of even permutations on the five-element set $\{a_1, \ldots, a_5\}$.

In A_5 there is a non-identity element which is its own commutator (up to suitable conjugation).[4]

[4] Such an element exists since there are two five-cycles in A_5 whose commutator is again a five-cycle and since five-cycles are always conjugate (see [1, Lemma 1 and 3]).

Fact 2.5 (compare to [25, Example 5]). *There are elements $\sigma, \alpha, \beta \in A_5$ with $\sigma \neq \mathbb{1}$ and $\sigma = [\sigma^\beta, \sigma^\alpha]$.*

The idea is now that we use σ from Fact 2.5 as an element representing $\top$. Again, using a simple induction on the structure of balanced iterated commutators (where Fact 2.5 is used for the inductive step), we obtain that balanced iterated commutators whose entries are all σ (which corresponds to $\top$) also evaluate to σ (i. e. to $\top$).

Fact 2.6 (compare to [25, Example 5]). *For the elements $\sigma, \alpha, \beta \in A_5$ from Fact 2.5, we have $B_{\beta,\alpha}[\sigma, \dots, \sigma] = \sigma$ for any number of entries in the commutator (which is a power of two).*

3 The Word Problem

The Uniform Word Problem. The uniform word problem for finitary automaton groups is the decision problem

Input: a finitary $\mathscr{G}$-automaton $\mathcal{T} = (Q, \Sigma, \delta)$ and a state sequence $\boldsymbol{q} \in Q^{\pm *}$
Question: is $\boldsymbol{q} = \mathbb{1}$ in $\mathscr{G}(\mathcal{T})$?

In this section, we will show that it is coNP-complete.

Proposition 3.1. *The uniform word problem for finitary automaton groups is in* coNP.

Proof. We solve the complement of the problem by a guess and check approach in NP. First, we guess a witness u on which $\boldsymbol{q}$ acts non-trivially. The length of a shortest such witness is at most the depth of the automaton $\mathcal{T}$, which, in turn, is bounded by the size of $\mathcal{T}$. Thus, the witness can be guessed in linear time.

Then, we compute $u_i = q_i \dots q_1 \circ u$ for $\boldsymbol{q} = q_\ell \dots q_1$ (with $q_i \in Q$, $1 \leq i \leq \ell$) state by state. This requires time $|\boldsymbol{q}| \cdot |u|$ and is, thus, certainly possible in polynomial time. □

Proposition 3.2. *The uniform word problem for finitary automaton groups is* coNP*-hard (under many-one* LogSpace*-reductions). This remains true if we fix a set with five elements as the alphabet of the input automaton.*

Proof. We reduce the NP-hard[5] satisfiability problem for boolean formulae

Input: a boolean formula φ in 3-conjunctive normal form
Question: is φ satisfiable?

to the complement of the stated problem by using a many-one LogSpace-reduction. In other words, we need to map (in logarithmic space) a boolean formula φ in 3-conjunctive normal form over a set of variables $\mathbb{X} = \{x_1, \dots, x_N\}$ to a finitary $\mathscr{G}$-automaton $\mathcal{T}$ and a state sequence $\boldsymbol{q}$ such that $\boldsymbol{q}$ does **not** act as the identity if and only if φ is satisfiable.

As φ is in 3-conjunctive normal form, we may write $\varphi = \bigwedge_{k=1}^{K} C_k$ where every clause C_k contains exactly three distinct literals over $\mathbb{X}$. Without loss of generality, we may

[5] This is a well-known classical NP-complete problem, see e. g. [20, Problem 9.5.5]

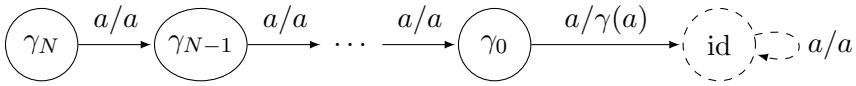


Figure 2: Schematic depiction of the automaton part for the states $\{\alpha_n, \beta_n \mid 0 \leq n \leq N\}$. This part exists for $\gamma \in \{\alpha, \beta\}$ (where α and β refer to the elements defined in Fact 2.5) and the dashed state refers to the already defined identity state. All transitions exist for all $a \in \Sigma$.

assume that no clause contains the same variable as a positive and a negative literal (as such clauses are satisfied by all assignments and can, thus, be dropped). In other words, we have $C_k = (\neg)x_{n_1} \vee (\neg)x_{n_2} \vee (\neg)x_{n_3}$ for three pairwise distinct n_1, n_2 and n_3 with $1 \leq n_1, n_2, n_3 \leq N$.

As the alphabet of the automaton $\mathcal{T}$, we use the set $\Sigma = \{a_1, \dots, a_5\}$ with five elements. From this set, we take two arbitrary letters and identify them with $\bot$ and $\top$, respectively. This allows us to encode an assignment $\mathcal{A} : \mathbb{X} \to \mathbb{B}$ as the word $\langle \mathcal{A} \rangle = \mathcal{A}(x_N) \dots \mathcal{A}(x_1)$ of length N.[6] Note that a word $w \in \Sigma^*$ of length N encodes an assignment (i. e. $w = \langle \mathcal{A} \rangle$ for some assignment $\mathcal{A}$) if and only if $w \in \{\bot, \top\}^*$.

The general idea is now that we check for every clause C_k whether the first N letters of the input form an encoding of an assignment satisfying C_k. If this is not the case (i. e. if a letter different to $\bot$ and $\top$ appears or if the encoded assignment does not satisfy C_k), we will go into an identity state, which can be thought of as a "fail" state. Otherwise, we end in a state corresponding to σ from Fact 2.5. Finally, we will use the balanced commutator from Definition 2.1 to make a conjunction of all these checks.

We will give a precise definition of the automaton $\mathcal{T} = (Q, \Sigma, \delta)$ by describing various parts. First, we need the mentioned identity state $\mathrm{id} \in Q$ (with the transitions $\{\mathrm{id} \xrightarrow{a/a} \mathrm{id} \mid a \in \Sigma\} \subseteq \delta$).

In order to eventually realize the balanced iterated commutator as a logical conjunction, we also need some technical states for the elements α and β from Fact 2.5. These states ignore the first N letters and then act as α or β on the $(N+1)$-th letter. For this, we use the states $\{\alpha_n, \beta_n \mid 0 \leq n \leq N\} \subseteq Q$ with the transitions

$$\begin{aligned} &\{\alpha_n \xrightarrow{a/a} \alpha_{n-1}, \beta_n \xrightarrow{a/a} \beta_{n-1} \mid 0 < n \leq N, a \in \Sigma\} \\ \cup\, &\{\alpha_0 \xrightarrow{a/\alpha(a)} \mathrm{id}, \beta_0 \xrightarrow{a/\beta(a)} \mathrm{id} \mid a \in \Sigma\} \subseteq \delta. \end{aligned}$$

This results in the automaton part graphically depicted in Figure 2 (for $\gamma \in \{\alpha, \beta\}$). Note that, for $\gamma \in \{\alpha, \beta\}$ and all $1 \leq n \leq N$, we have the cross diagram

$$\begin{array}{ccccc} & w & & a & \\ \gamma_n & \downarrow & \gamma_0 & \downarrow & \mathrm{id} \\ & w & & \gamma(a) & \end{array} \tag{1}$$

[6] Note that the right-most letter here corresponds to the first variable x_1. We could have done this the other way round as well but it turns out that this numbering has some technical advantages.

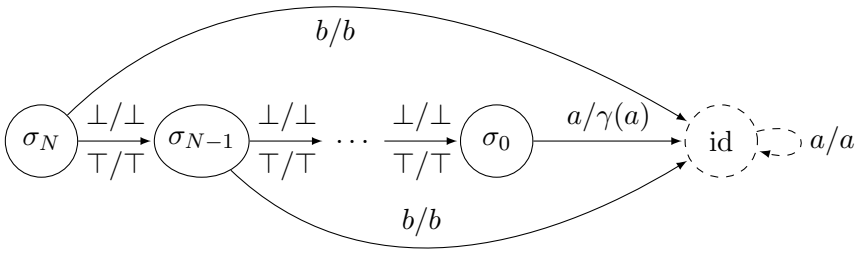


Figure 3: The automaton part for the states $\{\sigma_n \mid 0 \leq n \leq N\}$ where σ refers to the element defined in Fact 2.5), the dashed state refers to the already defined identity state and the transitions exist for all $a \in \Sigma$ and $b \in \Sigma \setminus \{\bot, \top\}$.

for all $w \in \Sigma^*$ of length n and $a \in \Sigma$. In particular, we have the invariant that γ_N acts trivially on the first N letters of all words. Also note that this part of the automaton does not introduce any cycles (in fact, after at most $N+1$ many letters we always end up in the identity state) and that it can be computed in logarithmic space as we only need to count up to the value of N.

Then, we need states that check whether the first N letters are either $\bot$ or $\top$ and, if this is the case, act like σ (from Fact 2.5) on the $(N+1)$-th letter. Otherwise, they will go to the identity state as a "fail" state. For this, we use the states $\{\sigma_n \mid 0 \leq n \leq n\} \subseteq Q$ together with the transitions

$$\left\{\sigma_n \xrightarrow{\bot/\bot} \sigma_{n-1}, \sigma_n \xrightarrow{\top/\top} \sigma_{n-1}, \sigma_n \xrightarrow{b/b} \mathrm{id} \mid 0 < n \leq N, b \in \Sigma \setminus \{\bot, \top\}\right\}$$
$$\cup \left\{\sigma_0 \xrightarrow{a/\sigma(a)} \mathrm{id} \mid a \in \Sigma\right\} \subseteq \delta.$$

See Figure 3 for a graphical representation. By construction, we obtain for all $0 \leq n \leq N$ the cross diagram

$$\sigma_n \begin{array}{c} w \\ \vdots \\ w \end{array} \begin{cases} \sigma_0 & \text{if } w \in \{\bot, \top\}^* \\ \mathrm{id} & \text{otherwise} \end{cases} \tag{2}$$

for all $w \in \Sigma^*$ of length n. Recall that, for a word $w \in \Sigma^*$ of length N, we have $w = \langle \mathcal{A} \rangle$ for some assignment $\mathcal{A}$ if and only if $w \in \{\bot, \top\}^*$ (i. e. if we are in the upper case in the above diagram). We have, in particular, that σ_N does not change the first N letters. Note that this part does not introduce any cycles either and may be computed in logarithmic space (as we again only need a counter up to N).

Most interesting are those parts of the automaton which are used to verify whether a clause is satisfied. In order to describe these parts, consider the clause $C_k = L_3 \vee L_2 \vee L_1$ for all $1 \leq k \leq K$ where L_i for $i \in \{1, 2, 3\}$ is either a positive or a negative literal of a variable x_{n_i}. Without loss of generality, we may assume $1 \leq n_3 < n_2 < n_1 \leq N$ and we say that x_{n_i} appears *positively* in C_k if $L_i = x_{n_i}$ and it appears *negatively* in C_k if $L_i = \neg x_{n_i}$ (for some $i \in \{1, 2, 3\}$). If a variable appears neither positively nor negatively, we say that it does *not* appear in C_k.

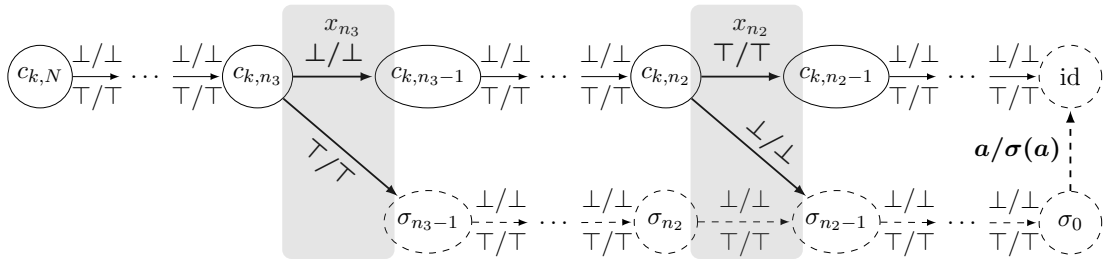


Figure 4: Part of the automaton for the states $\{c_{k,n} \mid 0 < n \leq N\}$. We assume x_{n_3} to appear positively in C_k while x_{n_2} is assumed to appear negatively. The part for x_{n_1} is not drawn for space reasons. Dashed states and transitions are already defined above. The transition on the right exists for all $a \in \Sigma$ and missing transitions are of the form b/b and go to id (for $b \in \Sigma \setminus \{\bot, \top\}$).

Now, in order to verify whether the clause C_k is satisfied, we use the states $\{c_{k,n} \mid 0 < n \leq N\} \subseteq Q$ with the transitions

$$\begin{aligned}
&\{c_{k,n} \xrightarrow{\bot/\bot} c_{k,n-1}, c_{k,n} \xrightarrow{\top/\top} c_{k,n-1} \mid 0 < n \leq N, x_n \text{ does not appear in } C_k\} \\
\cup\, &\{c_{k,n} \xrightarrow{\bot/\bot} c_{k,n-1}, c_{k,n} \xrightarrow{\top/\top} \sigma_{n-1} \mid 0 < n \leq N, x_n \text{ appears positively in } C_k\} \\
\cup\, &\{c_{k,n} \xrightarrow{\bot/\bot} \sigma_{n-1}, c_{k,n} \xrightarrow{\top/\top} c_{k,n-1} \mid 0 < n \leq N, x_n \text{ appears negatively in } C_k\} \\
\cup\, &\{c_{k,n} \xrightarrow{b/b} \mathrm{id} \mid 0 < n \leq N, b \in \Sigma \setminus \{\bot, \top\}\} \subseteq \delta
\end{aligned}$$

where we identify $c_{k,0}$ with the identity state id. This results in the automaton part schematically depicted in Figure 4.

The reader may verify that we obtain the cross diagram

$$c_{k,N} \overset{w}{\underset{w}{\dashrightarrow}} \begin{cases} \sigma_0 & \text{if } w = \langle \mathcal{A} \rangle \text{ such that } \mathcal{A} \text{ satisfies } C_k \\ \mathrm{id} & \text{otherwise} \end{cases} \tag{3}$$

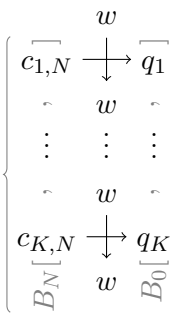
for all $w \in \Sigma^*$ of length N and all $1 \leq k \leq K$ by construction of the automaton. Note here that the "otherwise" case occurs if w contains a letter different to $\bot$ and $\top$ (i. e. it does not encode an assignment) and if w encodes an assignment which does not satisfy C_k. Also note that this part does not introduce any cycles either (again, we are in the identity state after at most $N+1$ many letters) and that we may compute it in logarithmic space since we only need to count up to N.

This concludes the definition of $\mathcal{T}$ and it remains to define $\boldsymbol{q}$. For this, we assume without loss of generality that K is a power of two. We may do this since we can easily just repeat one of the clauses and only have to count up to the next power of two (which can be done in logarithmic space). To actually define $\boldsymbol{q}$, we may now use the balanced commutator from Definition 2.1. In order to simplify our notation a bit, we will simply write $B_n[\boldsymbol{q}_\ell, \dots, \boldsymbol{q}_1]$ for any $0 \leq n \leq N$ instead of $B_{\beta_n,\alpha_n}[\boldsymbol{q}_\ell, \dots, \boldsymbol{q}_1]$ (for any $\boldsymbol{q}_1, \dots, \boldsymbol{q}_\ell \in Q^{\pm *}$). Using this convention, we let

$$\boldsymbol{q} = B_N\left[c_{K,N}, \dots, c_{1,N}\right].$$

Please note that $\boldsymbol{q}$ may be computed in logarithmic space by Fact 2.3.

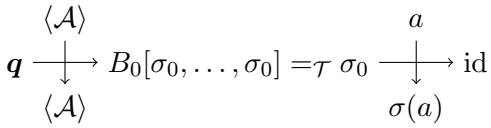
This concludes the definition of the reduction function and it remains to show $\boldsymbol{q} \neq_{\mathcal{T}} \mathrm{id}$ if and only if φ is satisfiable. We start by looking at how $\boldsymbol{q}$ acts on a word $w \in \Sigma^*$ of length N. From the cross diagrams (3), we obtain the black part of the cross diagram



(4)

where we have $q_k = \sigma_0$ if $w = \langle \mathcal{A} \rangle$ for some assignment $\mathcal{A}$ that satisfies C_k and $q_k = \mathrm{id}$ otherwise (i. e. if w does not encode a suitable assignment or if the assignment does not satisfy C_k). By combining Fact 2.4 with the left part of the cross diagrams (1) (for $\gamma = \alpha$ and $\gamma = \beta$ where $n = N$), we may add the balanced commutators to the cross diagram (gray additions above).

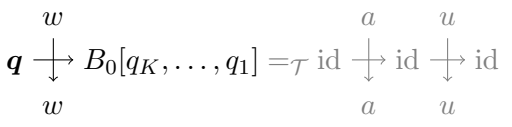
Now, assume that there is some assignment $\mathcal{A} : \mathbb{X} \to \mathbb{B}$ such that $\mathcal{A}$ satisfies φ. Note that σ_0 acts on the first letter like σ (from Fact 2.5) and then like the identity on all following letters. Thus, Fact 2.6 yields $B_0[\sigma_0, \ldots, \sigma_0] =_{\mathcal{T}} \sigma_0$ (where the number of σ_0 inside the commutator is any power of two) and we get the cross diagram



(5)

for this assignment from diagram (4). This shows that $\boldsymbol{q}$ acts non-trivially on $\langle \mathcal{A} \rangle a$ for some $a \in \Sigma$ and, thus, $\boldsymbol{q} \neq_{\mathcal{T}} \mathrm{id}$.

For the other direction, assume that φ is not satisfiable. We will show that $\boldsymbol{q}$ acts as the identity on all words of length at least $N+1$ (and, thus, in particular, also on shorter ones). Consider a word $wau \in \Sigma^*$ where w is of length N and $a \in \Sigma$. From the cross diagram (4), we obtain the black part of the cross diagram



(6)

Now, we have to consider two cases. If w does not encode an assignment, all $q_1, \ldots, q_K$ will be equal to id (by cross diagram (4)). If w does encode an assignment $\mathcal{A}$, this assignment cannot satisfy all clauses in φ (as φ is not satisfiable) and there must be some $1 \leq k \leq K$ such that $\mathcal{A}$ does not satisfy the clause C_k. For the above cross

diagram, this implies $q_k = \mathrm{id}$. Thus, in both cases, there is some $1 \leq k \leq K$ with $q_k = \mathrm{id}$ and we have $B_0[q_K, \ldots, q_1] =_{\mathcal{T}} \mathrm{id}$ by Fact 2.2, which yields the gray additions to the above cross diagram. In particular, we have $\boldsymbol{q} \circ wau = wau$ and, in general, $\boldsymbol{q} =_{\mathcal{T}} \mathrm{id}$ as desired. □

Remark 3.3. The automaton constructed by the reduction for Proposition 3.2 has $(3 + K) \cdot (N + 1) + 1$ many states where N is the numbers of variables in the input formula and K is the number of clauses. The depth of the automaton is $N + 1$.

Proposition 3.1 and Proposition 3.2 constitute the two parts of showing that the word problem is coNP-complete.

Theorem 3.4. *The word problem for finitary automaton groups is* coNP*-complete.*

4 The Compressed Word Problem

Straight-line Programs. A *straight-line program* (or *SPL*) is a context-free grammar which generates exactly one word. A *context-free grammar* mainly consists of a set of *rules* where the left-hand side consists of a single *non-terminal symbol* and the right-hand side is a finite word whose letters may be non-terminal or *terminal symbols*. A word is generated by starting at a dedicated non-terminal *starting symbol* and then iteratively replacing non-terminal symbols by matching right-hand sides of rules until only terminal symbols are left. By convention, non-terminal symbols are usually capitalized while terminal symbols are lowercase. More details may be found in any introductory textbook on formal language theory (see e. g. [13]).

Remark 4.1. The word generated by an SLP may be exponential in the size of the SLP. An example for this is given by the rules $A_1 \to a$ and $A_{n+1} \to A_n A_n$ (for $1 \leq n \leq N$).

The Compressed Word Problem. The uniform compressed word problem for finitary automaton groups is the problem

Input: a finitary $\mathcal{G}$-automaton $\mathcal{T} = (Q, \Sigma, \delta)$ and
a straight-line program generating a state sequence $\boldsymbol{q} \in Q^{\pm *}$
Question: is $\boldsymbol{q} = \mathbb{1}$ in $\mathcal{G}(\mathcal{T})$?

The difference to its ordinary version is that the state sequence is not given directly but only by a generating straight-line program. Due to the potential exponential blow-up when decompressing the SLP, the complexity of the compressed version differs in many cases from the one of the ordinary word problem. More information on the compressed word problem may be found in [5, 15].

In this section, we will show that the uniform compressed word problem for finitary automaton groups is PSpace-complete.

Proposition 4.2. *The uniform compressed word problem for finitary automaton groups is in* PSpace.

Proof. We follow the same guess and check approach as in the proof for Proposition 3.1. Since the length of the witness (on which $\boldsymbol{q}$ acts non-trivially) is bounded by the size of $\mathcal{T}$, it can clearly be guessed in linear space. The more interesting part is the "check" part. Here, we cannot simply decompress $\boldsymbol{q}$ and then apply it state by state (since $\boldsymbol{q}$ can be exponentially long). However, we can still compute (and store) the intermediate u_i directly from the SLP. We start with the rule $S \to \alpha_\ell \dots \alpha_1$ where the α_i are either terminal symbols (i. e. states) or non-terminals. We apply the symbols α_i from right to left to u. If α_i is a state, we can directly apply it to the current word. If it is a non-terminal symbol $\alpha_i = B$, we descend recursively into the rule $B \to \beta_k \dots \beta_1$. For this, we have to store where we were in the previous rule (this can, for example, be done using a pointer, which is clearly possible even in linear space). Note that we may assume that the same non-terminal symbol does not appear twice in the same recursive branch as this would correspond to a syntax tree with multiple instances of the same non-terminal symbol on one branch, which cannot occur if the grammar only generates a single word. Thus, in the worst case, we need to store one position for every rule in the input, which is still possible in linear space. □

For the other direction – namely to prove that our problem is PSPACE-hard – we will use the following problem to make the reduction.

Theorem 4.3. *The problem* 3-QBF

Input: *a quantified boolean formula* $\varphi = \neg\forall x_N \neg\forall X_{N-1} \dots \neg\forall x_1 : \varphi_0$ *where* φ_0 *is in 3-conjunctive normal form and contains no other variables than* $\{x_1, \dots, x_N\}$.

Question: *is* φ *true?*

is PSPACE*-complete (under many-one* LOGSPACE*-reductions).*

Proof. We reduce the problem

Input: $\varphi = \exists x_1 \forall x_2 \dots Q_N x_N : \varphi_0$ where φ_0 is in conjunctive normal form

Question: does φ hold?

where $Q_N = \exists$ if N is odd and $Q_N = \forall$ is N is even, which is PSPACE-complete (under many-one LOGSPACE-reductions) by [20, Theorem 19.1], to the special version stated in the theorem.

First, we split up all clauses with more than three literals in the common way[7] by using the fact that $L_1 \vee \dots \vee L_\ell$ and $\exists z : (L_1 \vee L_2 \vee z) \wedge (\neg z \vee L_3 \vee \dots L_\ell)$ (where z is a new, so-far unused variable) are equivalent (i. e. they are satisfied by exactly the same assignments). This introduces additional existential quantifiers at the innermost position.

Clauses with less than three literals can be padded with new variables by using the fact that any literal L is equivalent to $\forall z : (L \vee z)$ (where z is again a new variable). This introduces additional universal quantifiers at the innermost position.

[7] This is usually used to prove that 3-SAT is NP-complete (see e. g. [20, Problem 9.5.2]).

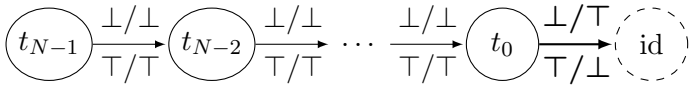


Figure 5: The additional automaton part with the states $\{t_n \mid 0 \leq n < N\}$. Missing transitions are b/b transitions to the identity state (for $b \in \Sigma \setminus \{\bot, \top\}$).

We may ensure that the quantifiers are alternating between $\exists$ and $\forall$ by adding dummy variables not appearing in the matrix (i. e. the inner part of the formula without quantifiers) of the formula. This results in a formula of the form $\exists x_{N'} \forall x_{N'-1} \ldots \exists x_2 \forall x_1 : \varphi_0'$ where φ_0' is in 3-conjunctive normal form which is equivalent to the original formula.

Finally, we use the equivalence of $\exists z : \psi$ and $\neg \forall z : \neg \psi$ to eliminate all existential quantifiers.

Note that each of these steps can be computed in LogSpace and that, thus, the whole reduction can be done in LogSpace. □

Proposition 4.4. *The uniform compressed word problem for finitary automaton groups is* PSpace*-hard (under many-one* LogSpace*-reductions). This remains true if we fix a set with five elements as the alphabet of the input automaton.*

Proof. We reduce 3-QBF from Theorem 4.3 to the (complement of the) compressed word problem for fintary automaton groups (in LogSpace). For this, assume that we get a quantified boolean formula $\varphi = \neg \forall x_N \neg \forall x_{N-1} \ldots \neg \forall x_1 : \varphi_0$ where φ_0 is in 3-conjunctive normal form and contains no other variables than $\{x_1, \ldots, x_N\}$. First, we perform the LogSpace-computable reduction described in the proof of Proposition 3.2 using φ_0 as the input. This yields a finitary $\mathscr{G}$-automaton $\mathcal{T} = (Q_0, \Sigma, \delta_0)$ with $\Sigma = \{a_1, \ldots, a_5\}$ and a state sequence $\boldsymbol{q}_0 \in Q_0^{\pm *}$. Then we perform a second reduction on this output (which is possible since LogSpace-computable functions are closed under composition, see, for example, [20, Proposition 8.2]). Here, we need to compute a finitary $\mathscr{G}$-automaton $\mathcal{T}$ and a state sequence $\boldsymbol{q}$ encoded as an SLP such that $\boldsymbol{q} \neq_{\mathcal{T}} \mathrm{id}$ if and only if φ holds.

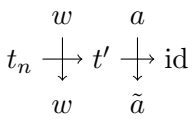
To obtain the automaton $\mathcal{T} = (Q, \Sigma, \delta)$, we extend $\mathcal{T}_0$ by some additional states (but keep the alphabet the same: $\Sigma = \{a_1, \ldots, a_5\}$). The new states are $\{t_n \mid 0 \leq n < N\}$ with the additional transitions

$$\left\{ t_n \xrightarrow{\bot/\bot} t_{n-1}, t_n \xrightarrow{\top/\top} t_{n-1}, t_n \xrightarrow{b/b} \mathrm{id} \mid 0 < n < N, b \in \Sigma \setminus \{\bot, \top\} \right\}$$
$$\cup \left\{ t_0 \xrightarrow{\bot/\top} \mathrm{id}, t_0 \xrightarrow{\top/\bot} \mathrm{id}, t_0 \xrightarrow{b/b} \mathrm{id} \mid b \in \Sigma \setminus \{\bot, \top\} \right\} \subseteq \delta$$

(where $\bot$ and $\top$ refer to the special elements from Σ chosen in the proof of Proposition 3.2 and id is the identity state of $\mathcal{T}_0$). This new automaton part is depicted in Figure 5. Note that we have not introduced any cycles and that this new part may be computed in logarithmic space (as we only need a counter up to the value of N). By construction, we obtain the cross diagram

$$\begin{array}{ccccc} & w & & a & \\ t_n & \longrightarrow & t_0 & \longrightarrow & \mathrm{id} \\ & w & & \neg a & \end{array} \tag{7}$$

for all $0 < n < N$ where $w \in \{\bot, \top\}^*$ is of length n, $a \in \{\bot, \top\}$ and $\neg a$ denotes the negation of a (i. e. $\neg a = \top$ if $a = \bot$ and $\neg a = \bot$ if $a = \top$). For general words $w \in \Sigma^*$ of length $0 < n < N$ and letters $a \in \Sigma$, we get the cross diagram



(8)

where we have $t' = t_0$ and $\tilde{a} = \neg a$ if $wa \in \{\bot, \top\}^*$ and $t' = \mathrm{id}$ and $\tilde{a} = a$ otherwise.

We will define the state sequence $\boldsymbol{q}$ inductively and will use this inductive structure in the end to compute an SLP generating $\boldsymbol{q}$. We already have $\boldsymbol{q}_0$ and, for $0 < n \leq N$, we let

$$\boldsymbol{q}'_n = B_N[\boldsymbol{q}_{n-1}^{t_{N-n}}, \boldsymbol{q}_{n-1}] \text{ and } \boldsymbol{q}_n = \left(\boldsymbol{q}'_n\right)^{-1} \sigma_N$$

(using the notation B_n for B_{β_n,α_n} introduced in the proof of Proposition 3.2).

The reason for choosing the $\boldsymbol{q}_n$ in this way is to satisfy a certain invariant. To state it, recall that φ_0 is already given, let $\varphi'_0 = \varphi_0$ and

$$\varphi'_n = \forall x_n : \varphi_{n-1} \text{ and } \varphi_n = \neg\varphi'_n$$

for $0 < n \leq N$. Note that this means

$$\varphi_n = \neg\forall x_n \ldots \neg\forall x_1 : \varphi_0$$

and φ'_n is the same except that it misses the out-most negation. In particular, we have $\varphi_N = \varphi$. Before we finally state the invariant, we extend the notation $\langle\mathcal{A}\rangle$ from the proof of Proposition 3.2 to assignments $\mathcal{A} : \{x_{n+1}, \ldots, x_N\} \to \mathbb{B}$ (for $0 \leq n \leq N$) by letting $\langle\mathcal{A}\rangle = \mathcal{A}(x_N) \ldots \mathcal{A}(x_{n+1}) \in \{\bot, \top\}^* \subseteq \Sigma^*$. Note that $|\langle\mathcal{A}\rangle|$ has length $N - n$ and that the empty word is the encoding of an empty assignment. Now, the invariant we want to satisfy with the $\boldsymbol{q}_n$ is that, for all $0 \leq n \leq N$, all words $u \in \Sigma^*$ of length $N - n$ and all words $v \in \Sigma^*$ of length n, we have the black part of the cross diagram

$$\boldsymbol{q}_n = \begin{cases} \sigma_N \;\; u \;\; v = \begin{cases} \sigma_0 & \text{if } uv \in \{\bot, \top\}^* \\ \mathrm{id} & \text{otherwise} \end{cases} \\ (\boldsymbol{q}'_n)^{-1} \;\; u \;\; v = \tau \begin{cases} (\sigma_0)^{-1} & \text{if } u = \langle\mathcal{A}\rangle \text{ s. t. } \mathcal{A} \text{ satisfies } \varphi'_n \text{ and } v \in \{\bot, \top\}^* \\ \mathrm{id} & \text{otherwise} \end{cases} \end{cases}$$

(9)

where we let $\boldsymbol{q}'_0 = \boldsymbol{q}_0$ and use the convention that the empty assignment satisfies a (closed)[8] formula if and only if the formula holds. Note that the (black) "otherwise"

[8] A formula is *closed* if it does not have any free variables, i. e. if all appearing variables are bound by a quantifier.

case includes the case that u or v is not from $\{\bot, \top\}^*$ and the case that u encodes an assignment not satisfying φ'_n.

As soon as we have established this invariant for some n, we immediately also get a version where we take the inverses of the states (this is possible since the action is trivial; normally, we would have to additionally flip the diagram along the horizontal axis). Using the cross diagram (2) (from the proof of Proposition 3.2), we may add an additional line for σ_N and obtain the gray additions to the above diagram for $0 < n \leq N$. Note that the product of the state sequences on the right hand side acts trivially if $u = \langle \mathcal{A} \rangle$ for some $\mathcal{A}$ which satisfies φ'_n (this is the case if and only if $\mathcal{A}$ does not satisfy $\varphi_n = \neg\varphi'_n$) and $v \in \{\bot, \top\}^*$. It also acts trivially if $uv \notin \{\bot, \top\}^*$. On the other hand, it acts like σ_0 if $u = \langle \mathcal{A} \rangle$ for some $\mathcal{A}$ which does satisfy $\varphi_n = \neg\varphi'_n$ and $v \in \{\bot, \top\}^*$. This yields the cross diagram

$$\boldsymbol{q}_n \begin{array}{c} u \\ \downarrow \\ u \end{array} \begin{array}{c} v \\ \downarrow \\ v \end{array} =_{\mathcal{T}} \begin{cases} \sigma_0 & \text{if } u = \langle \mathcal{A} \rangle \text{ s.t. } \mathcal{A} \text{ satisfies } \varphi_n \text{ and } v \in \{\bot, \top\}^* \\ \mathrm{id} & \text{otherwise} \end{cases} \tag{10}$$

for all $0 \leq n \leq N$, all words $u \in \Sigma^*$ for length $N - n$ and all words $v \in \Sigma^*$ of length n.

To prove the invariant (i. e. the black part of cross diagram (9)), we use induction on n. For $n = 0$, we have to show the cross diagram

$$\boldsymbol{q}'_0 = \boldsymbol{q}_0 \begin{array}{c} u \\ \downarrow \\ u \end{array} \begin{array}{c} \varepsilon \\ \downarrow \\ \varepsilon \end{array} =_{\mathcal{T}} \begin{cases} \sigma_0 & \text{if } u = \langle \mathcal{A} \rangle \text{ s.t. } \mathcal{A} \text{ satisfies } \varphi'_0 \text{ (which is equal to } \varphi_0) \\ \mathrm{id} & \text{otherwise} \end{cases}$$

for $u \in \Sigma^*$ of length N. This, however, has already been established by the cross diagrams (5) and (6) in the proof of Proposition 3.2.

For the inductive step from $n-1$ to n, consider a word $u \in \Sigma^*$ of length $N - n$, $a \in \Sigma$ and $v \in \Sigma^*$ of length $n - 1$. We have the black part of the cross diagram

$$\boldsymbol{q}'_n = \begin{array}{l} \overbrace{\boldsymbol{q}_{n-1}}^{} \begin{array}{c} u \\ \downarrow \\ u \end{array} \begin{array}{c} a \\ \downarrow \\ a \end{array} \begin{array}{c} v \\ \downarrow \\ v \end{array} \overbrace{\boldsymbol{p}_{n,0}}^{} \\ t_{N-n} \begin{array}{c} \downarrow \\ \downarrow \end{array} \begin{array}{c} \downarrow \\ \downarrow \end{array} \mathrm{id} \begin{array}{c} \downarrow \\ \downarrow \end{array} \mathrm{id} \\ \boldsymbol{q}_{n-1} \begin{array}{c} u \\ \downarrow \\ u \end{array} \begin{array}{c} \tilde{a} \\ \downarrow \\ \tilde{a} \end{array} \begin{array}{c} v \\ \downarrow \\ v \end{array} \boldsymbol{p}_{n,1} \\ t^{-1}_{N-n} \begin{array}{c} \downarrow \\ \downarrow \end{array} \begin{array}{c} \downarrow \\ \downarrow \end{array} \mathrm{id} \begin{array}{c} \downarrow \\ \downarrow \end{array} \mathrm{id} \\ \underbrace{}_{B_N} \begin{array}{c} u \\ \downarrow \\ u \end{array} \begin{array}{c} a \\ \downarrow \\ a \end{array} \begin{array}{c} v \\ \downarrow \\ v \end{array} \underbrace{}_{B_0} \end{array} \tag{11}$$

where we have

$$\boldsymbol{p}_{n,0} =_{\mathcal{T}} \begin{cases} \sigma_0 & \text{if } ua = \langle \mathcal{A}' \rangle \text{ s.t. } \mathcal{A}' \text{ satisfies } \varphi_{n-1} \text{ and } v \in \{\bot, \top\}^* \\ \mathrm{id} & \text{otherwise} \end{cases}$$

$$\boldsymbol{p}_{n,1} =_{\mathcal{T}} \begin{cases} \sigma_0 & \text{if } u\tilde{a} = \langle \tilde{\mathcal{A}}' \rangle \text{ s.t. } \tilde{\mathcal{A}}' \text{ satisfies } \varphi_{n-1} \text{ and } v \in \{\bot, \top\}^* \\ \mathrm{id} & \text{otherwise.} \end{cases}$$

The shaded parts are due to induction (compare to cross diagram (10)) and lines involving t_{N-n} follow from cross diagram (8).

By Fact 2.4 (and the cross diagram (1) for $\gamma \in \{\alpha, \beta\}$ from the proof of Proposition 3.2), we may add balanced iterated commutators to the above cross diagram and obtain the gray additions.

For the above cross diagram, we do a case distinction. If we have $uav \notin \{\bot, \top\}^*$, we get $\boldsymbol{p}_{n,0} =_{\mathcal{T}} \boldsymbol{p}_{n,1} =_{\mathcal{T}} \mathrm{id}$ and, thus, for the state sequence on the right $B_0[\boldsymbol{p}_{n,1}, \boldsymbol{p}_{n,0}] =_{\mathcal{T}} \mathrm{id}$.

Now, assume $uav \in \{\bot, \top\}^*$ and, in particular, $a \in \{\bot, \top\}$. In this case, we have $u = \langle \mathcal{A} \rangle$ for some $\mathcal{A} : \{x_{n+1}, \dots, x_N\} \to \mathbb{B}$ and $\tilde{a} = \neg a$ (see cross diagram (8)). Let $ua = \langle \mathcal{A}' \rangle$ and $u\tilde{a} = \langle \tilde{\mathcal{A}} \rangle$. Note that we have $\mathcal{A}'(x_n) = a = \neg\tilde{\mathcal{A}}'(x_n)$ (and $\mathcal{A}'(x_m) = \tilde{\mathcal{A}}'(x_m) = \mathcal{A}(x_m)$ for all $n < m \leq N$). If $\mathcal{A}$ satisfies $\varphi'_n = \forall x_n : \varphi_{n-1}$, we, therefore, have that $\mathcal{A}'$ and $\tilde{\mathcal{A}}'$ both satisfy φ_{n-1}. This yields $\boldsymbol{p}_{n,0} =_{\mathcal{T}} \boldsymbol{p}_{n,1} =_{\mathcal{T}} \sigma_0$ (by the above definition of $\boldsymbol{p}_{n,0}$ and $\boldsymbol{p}_{n,1}$) and, thus, for the state sequence on the right $B_0[\boldsymbol{p}_{n,1}, \boldsymbol{p}_{n,0}] =_{\mathcal{T}} \sigma_0$ by Fact 2.6. On the other hand, if $\mathcal{A}$ does not satisfy $\varphi'_n = \forall x_n : \varphi_{n-1}$, we must have that $\mathcal{A}'$ or $\tilde{\mathcal{A}}'$ does not satisfy φ_{n-1}. In this case, we have $\boldsymbol{p}_{n,0} =_{\mathcal{T}} \mathrm{id}$ or $\boldsymbol{p}_{n,1} =_{\mathcal{T}} \mathrm{id}$ and, thus, for the state sequence on the right, $B_0[\boldsymbol{p}_{n,1}, \boldsymbol{p}_{n,0}] =_{\mathcal{T}} \mathrm{id}$ by Fact 2.2. This shows that the cases for the gray additions to cross diagram (11) reflect exactly the black part of cross diagram (9), which shows the invariant.

Considering the special case $n = N$ for cross diagram (10), we have obtain

$$\boldsymbol{q}_N \begin{array}{cc} \varepsilon & v \\ \downarrow & \downarrow \\ \varepsilon & v \end{array} =_{\mathcal{T}} \begin{cases} \sigma_0 & \text{if } \varphi_N \text{ holds and } v \in \{\bot, \top\}^* \\ \mathrm{id} & \text{otherwise} \end{cases}$$

for all $v \in \Sigma^*$ of length N. This shows that we have $\boldsymbol{q}_N =_{\mathcal{T}} \mathrm{id}$ if φ_N (which is equal to φ) does **not** hold. If it does hold, on the other hand, we have $\boldsymbol{q}_N \circ \bot^N a = \bot^N \sigma(a) \neq \bot^N a$ for some $a \in \Sigma$ (since σ is not the identity permutation). Thus, we may choose $\boldsymbol{q} = \boldsymbol{q}_N$ as the sought state sequence and it remains to show how an SLP generating $\boldsymbol{q}_N$ can be computed in logarithmic space.

Note that $\boldsymbol{q}_0$ may be computed in logarithmic space (we obtain this from the proof of Proposition 3.2). Thus, we may begin with the rule $A_0 \to \boldsymbol{q}_0$ and add the rules

$$\begin{aligned} A'_n &\to \beta_N^{-1} t_{N-n}^{-1} A_{n-1}^{-1} t_{N-n} \beta_N \; \alpha_N^{-1} A_{n-1}^{-1} \alpha_N \; \beta_N^{-1} t_{N-n}^{-1} A_{n-1} t_{N-n} \beta_N \; \alpha_N^{-1} A_{n-1} \alpha_N \\ &\quad = B_N[A_{n-1}^{t_{N-n}}, A_{n-1}] \quad \text{and} \\ A_n &\to (A'_n)^{-1} \sigma_N \end{aligned}$$

for $1 \leq n \leq N$, where we also implicitly add the rules for $(A'_n)^{-1}$ and A_n^{-1} by mirroring the right-hand sides and inverting every symbol. Note that these rules may be computed in logarithmic space. We choose A_N as our starting symbol and the reader may verify that A'_n generates $\boldsymbol{q}'_n$ and A_n generates $\boldsymbol{q}_n$ (this follows directly from the inductive definitions of the A_n, A'_n and $\boldsymbol{q}_n$, $\boldsymbol{q}'_n$). □

Proposition 4.2 and Proposition 4.4 form the two directions for the following theorem.

Theorem 4.5. *The uniform compressed word problem for finitary automaton groups is* PSPACE*-complete.*

Acknowledgments

The authors would like to thank Armin Weiß for many discussions around the presented topic. The presented results are part of the first author's Bachelor thesis, which was advised by the second author (while he was at FMI).

References

[1] David A. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *Journal of Computer and System Sciences.*, 38(1):150–164, 1989.

[2] Laurent Bartholdi, Michael Figelius, Markus Lohrey, and Armin Weiß. Groups with ALOGTIME-hard word problems and PSPACE-complete circuit value problems. In Shubhangi Saraf, editor, *35th Computational Complexity Conference (CCC 2020)*, volume 169 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 29:1–29:29, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[3] Laurent Bartholdi and Ivan Mitrofanov. The word and order problems for self-similar and automata groups. *Groups, Geometry, and Dynamics*, 14:705–728, 2020.

[4] Laurent Bartholdi and Pedro V. Silva. *Groups Defined by Automata*, chapter 24 of Automata: from Mathematics to Applications (Handbook AutoMathA). 2010. To appear.

[5] Frédérique Bassino, Ilya Kapovich, Markus Lohrey, Alexei Miasnikov, Cyril Nicaud, Andrey Nikolaev, Igor Rivin, Vladimir Shpilrain, Alexander Ushakov, and Pascal Weil. *Complexity and Randomness in Group Theory.* De Gruyter, 2020.

[6] Alex Bishop and Murray Elder. Bounded automata groups are co-ET0L. In Carlos Martín-Vide, Alexander Okhotin, and Dana Shapira, editors, *Language and Automata Theory and Applications*, pages 82–94. Springer International Publishing, 2019.

[7] Ievgen Bondarenko and Jan Philipp Wächter. On orbits and the finiteness of bounded automaton groups. *International Journal of Algebra and Computation*, 31(06):1177–1190, 2021.

[8] Ievgen V. Bondarenko, Natalia V. Bondarenko, Said N. Sidki, and Flavia R. Zapata. On the conjugacy problem for finite-state automorphisms of regular rooted trees. *Groups, Geometry, and Dynamics*, 7:232–355, 2013.

[9] Daniele D'Angeli, Emanuele Rodaro, and Jan Philipp Wächter. On the complexity of the word problem for automaton semigroups and automaton groups. *Advances in Applied Mathematics*, 90:160 – 187, 2017.

[10] Pierre Gillibert. The finiteness problem for automaton semigroups is undecidable. *International Journal of Algebra and Computation*, 24(01):1–9, 2014.

[11] Pierre Gillibert. An automaton group with undecidable order and Engel problems. *Journal of Algebra*, 497:363 – 392, 2018.

[12] Rostislav I. Grigorchuk and Igor Pak. Groups of intermediate growth: an introduction. *L'Enseignement Mathématique*, 54(3-4):251–272, 2008.

[13] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[14] Kenneth Krohn, Ward Douglas Maurer, and John L. Rhodes. Realizing complex boolean functions with simple groups. *Information and Control*, 9(2):190–195, 1966.

[15] Markus Lohrey. *The Compressed Word Problem for Groups*. SpringerBriefs in Mathematics. Springer, 2014.

[16] Gennadiĭ S. Makanin. Decidability of the universal and positive theories of a free group. *Izv. Akad. Nauk SSSR*, Ser. Mat. 48:735–749, 1984. In Russian; English translation in: *Math. USSR Izvestija, 25*, 75–88, 1985.

[17] Anatolij I. Mal'cev. On the equation $zxyx^{-1}y^{-1}z^{-1} = aba^{-1}b^{-1}$ in a free group. *Akademiya Nauk SSSR. Sibirskoe Otdelenie. Institut Matematiki. Algebra i Logika*, 1(5):45–50, 1962.

[18] Ward Douglas Maurer and John L. Rhodes. A property of finite simple non-abelian groups. *Proceedings of the American Mathematical Society*, 16(3):552–554, 1965.

[19] Volodymyr V. Nekrashevych. *Self-similar groups*, volume 117 of *Mathematical Surveys and Monographs*. American Mathematical Society, Providence, RI, 2005.

[20] Christos M. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[21] Said N. Sidki. Automorphisms of one-rooted trees: growth, circuit structure, and acyclicity. *Journal of Mathematical Sciences*, 100(1):1925–1943, 2000.

[22] Pedro V. Silva. Groups and automata: A perfect match. In Martin Kutrib, Nelma Moreira, and Rogério Reis, editors, *Descriptional Complexity of Formal Systems*, pages 50–63, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[23] Benjamin Steinberg. *On some algorithmic properties of finite state automorphisms of rooted trees*, volume 633 of *Contemporary Mathematics*, pages 115–123. American Mathematical Society, 2015.

[24] Zoran Šunić and Enric Ventura. The conjugacy problem in automaton groups is not solvable. *Journal of Algebra*, 364:148–154, 2012.

[25] Jan Philipp Wächter and Armin Weiß. An automaton group with PSpace-complete word problem. *Theory of Computing Systems*, 2022.