An Automaton Group with PSPACE-Complete Word Problem

Jan Philipp Wächter
Universität Stuttgart, Institut für Formale Methoden der Informatik (FMI), Universitätsstraße 38, 70569 Stuttgart, Germany
jan-philipp.waechter@fmi.uni-stuttgart.de

Armin Weiß
Universität Stuttgart, Institut für Formale Methoden der Informatik (FMI), Universitätsstraße 38, 70569 Stuttgart, Germany
armin.weiss@fmi.uni-stuttgart.de

Abstract

We construct an automaton group with a PSPACE-complete word problem, proving a conjecture due to Steinberg. Additionally, the constructed group has a provably more difficult, namely EXPSPACE-complete, compressed word problem. Our construction directly simulates the computation of a Turing machine in an automaton group and, therefore, seems to be quite versatile. It combines two ideas: the first one is a construction used by D'Angeli, Rodaro and the first author to obtain an inverse automaton semigroup with a PSPACE-complete word problem and the second one is to utilize a construction used by Barrington to simulate circuits of bounded degree and logarithmic depth in the group of even permutations over five elements.

2012 ACM Subject Classification Theory of computation → Transducers

Keywords and phrases automaton group, word problem, PSPACE, compressed word problem

Digital Object Identifier 10.4230/LIPIcs.STACS.2020.6

Funding *Armin Weiß*: Funded by DFG project DI 435/7-1.

1 Introduction

The word problem is one of Dehn's fundamental algorithmic problems in group theory [10]: given a word over a finite set of generators for a group, decide whether the word represents the identity in the group. While, in general, the word problem is undecidable [18, 7], many classes of groups have a decidable word problem. Among them is the class of automaton groups. In this context, the term *automaton* refers to finite state, letter-to-letter transducers. In such automata, every state q induces a length-preserving, prefix-compatible action on the set of words, where an *input word* u is mapped to the *output word* obtained by reading u starting in q. The group or semigroup generated by the automaton is the closure under composition of the actions of the different states and a (semi)group arising in this way is called an *automaton (semi)group*.

The interest in automaton groups was stirred by the observation that many groups with interesting properties are automaton groups. Most prominently, the class contains the famous Grigorchuk group (which is the first example of a group with sub-exponential but super-polynomial growth and admits other peculiar properties, see [14] for an accessible introduction). There is also a quite extensive study of algorithmic problems in automaton (semi)groups: the conjugacy problem and the isomorphism problem (here the automaton is part of the input) – the other two of Dehn's fundamental problems – are undecidable for automaton groups [23] . For automaton semigroups, the order problem could be proved to be undecidable [12, Corollary 3.14]. Recently, this could be extended to automaton groups

[13] (see also [4]). On the other hand, the undecidability result for the finiteness problem for automaton semigroups [12, Theorem 3.13] could not be lifted to automaton groups so far.

The undecidability results show that the presentation of groups using automata is still quite powerful. Nevertheless, it is not very difficult to see that the word problem for automaton groups is decidable. One possible way is to show an upper bound on the length of an input word on which a state sequence[1] not representing the identity of the group acts non-trivially. In the most general setting, this bound is $|Q|^n$ where Q is the state set of the automaton and n is the length of the state sequence. Another viewpoint is that one can use a non-deterministic guess and check algorithm to solve the word problem. This algorithm uses linear space proving that the word problem for automaton (semi)groups is in PSPACE. This approach seems to be mentioned first by Steinberg [22, Section 3] (see also [9, Proposition 2 and 3]). In some special cases, better algorithms or upper bounds are known: for example, for contracting automaton groups (and this includes the Grigorchuk group), the witness length is bounded logarithmically [17] and the problem, thus, is in LOGSPACE; other examples of classes with better upper bounds or algorithms include automata with polynomial activity [5] or Hanoi Tower groups [6]. On the other hand, Steinberg conjectured that there is an automaton group with a PSPACE-complete word problem [22, Question 5]. As a partial solution to his problem, an inverse automaton semigroup with a PSPACE-complete word problem has been constructed in [9, Proposition 6][2]. In this paper, our aim is to finally prove the conjecture for groups.

In order to do so, we adopt the construction used by D'Angeli, Rodaro and the first author from [9, Proposition 6]. This construction uses a master reduction and directly encodes a Turing machine into an automaton. Already in [9, Proposition 6], it was also used to show that there is an automaton group whose word problem with a rational constraint (which depends on the input) is PSPACE-complete. To get rid of this rational constraint, we apply an idea used by Barrington [3] to transform NC^1-circuits (circuits of bounded fan-in and logarithmic depth) into bounded-width polynomial-size branching programs. Similar ideas predating Barrington have been attributed to Gurevich (see [15]) and given by Mal'cev [16]. Nevertheless, this paper is fully self-contained and no previous knowledge of either [9] or [3] is needed.

In addition, we also investigate the compressed word problem for automaton groups. Here, the (input) state sequence is given as a so-called *straight-line program* (a context-free grammar which generates exactly one word). By uncompressing the input sequence and applying the above mentioned non-deterministic linear-space algorithm, one can see that the compressed word problem can be solved in EXPSPACE. Thus, the more interesting part is to prove that this algorithm cannot be improved significantly: we show that there is an automaton group with an EXPSPACE-hard compressed word problem. This result is interesting because, by taking the direct product, we obtain a group whose (ordinary) word problem is PSPACE-complete and whose compressed word problem is EXPSPACE-complete and, thus, provably more difficult by the space hierarchy theorem [21, Theorem 6] (or e. g. [2, Theorem 4.8]). To the best of our knowledge, this is the first example of a group for which this is possible.

---

[1] In order to avoid ambiguities, we call a word over the states of the automaton a *state sequence*. So, in our case the input for the word problem is a state sequence.

[2] In fact, the semigroup is generated by a partial, invertible automaton. A priori, this seems to be a stronger statement than that the semigroup is inverse and also an automaton semigroup. That is why the cited paper uses the term 'automaton-inverse semigroup'. Only later, it was shown that the two concepts actually coincide [8, Theorem 25].

Explicit previous results on the compressed word problem for automaton groups do not seem to exist. However, it was observed by Gillibert [11] that the proof of [9, Proposition 6] also yields an automaton semigroup with an EXPSPACE-complete compressed word problem in a rather straightforward manner. For the case of groups, it is possible to adapt the construction used by Gillibert to prove the existence of an automaton group with an undecidable order problem [13] slightly to obtain an automaton group with a PSPACE-hard compressed word problem [11].

2 Preliminaries

Words and Alphabets with Involution. We use common notations from formal language theory. In particular, we use Σ^* to denote the set of words over an alphabet Σ including the empty word. If we want to exclude the empty word, we write Σ^+. For any alphabet Q, we define a natural involution between Q and a disjoint copy $Q^{-1} = \{q^{-1} \mid q \in Q\}$ of Q: it maps $q \in Q$ to $q^{-1} \in Q^{-1}$ and vice versa. In particular, we have $(q^{-1})^{-1} = q$. The involution extends naturally to words over $Q \cup Q^{-1}$: for $q_1, \dots, q_n \in Q \cup Q^{-1}$, we set $(q_n \dots q_1)^{-1} = q_1^{-1} \dots q_n^{-1}$. This way, the involution is equivalent to taking the group inverse if Q is a generating set of a group.

Turing Machines and Complexity. We assume the reader to be familiar with basic notions of complexity theory such as configurations for Turing machines, computations and reductions in logarithmic space as well as complete and hard problems for PSPACE and the class EXPSPACE. See [19] or [2] for standard text books on complexity theory. We only consider deterministic, single-tape machines and write their configurations as word $c_1 \dots c_{i-1} p c_i \dots c_n$ where the c_j are symbols from the tape alphabet and p is a state. In this configuration, the machine is in state p and its head is over the symbol c_i.

▶ **Fact 1** (Folklore). *Consider a deterministic Turing machine with state set P and tape alphabet Δ. After a straightforward transformation of the transition function and states, we can assume that the symbol $\gamma_i^{(t+1)}$ at position i of the configuration at time step $t+1$ only depends on the symbols $\gamma_{i-1}^{(t)}, \gamma_i^{(t)}, \gamma_{i+1}^{(t)} \in \Gamma$ at position $i-1$, i and $i+1$ at time step t. Thus, we may always assume that there is a function $\tau : \Gamma^3 \to \Gamma$ with $\Gamma = P \uplus \Delta$ mapping the symbols $\gamma_{i-1}^{(t)}, \gamma_i^{(t)}, \gamma_{i+1}^{(t)} \in \Gamma$ to the uniquely determined symbol $\gamma_i^{(t+1)}$ for all i and t.*

Group Theory and A_5. For elements h and g of a group G, we write g^h for the conjugation $h^{-1}gh$ of g with h and $[h, g]$ for the commutator $h^{-1}g^{-1}hg$. For the neutral element of a group, we write $\mathbb{1}$. We write $\boldsymbol{p} =_G \boldsymbol{q}$ or $\boldsymbol{p} = \boldsymbol{q}$ in G if two words $\boldsymbol{p}$ and $\boldsymbol{q}$ over the generators (and their inverses) of a group evaluate to the same group element.

With A_5 we denote the alternating group of degree five, i.e. the group of even permutations of five elements. It was used by Barrington [3] to convert logical circuits of bounded fan-in and logarithmic depth (so-called NC^1*-circuits*) to bounded-width, polynomial-size branching programs. We will not require this actual result (or knowledge of the involved concepts) in the following, but we will make heavy use of the next lemma and the idea to use iterated commutators, which we will outline below.

▶ **Lemma 2** (see Lemma 1 and 3 of [3]). *There are $\sigma, \alpha, \beta \in A_5$ such that $\sigma = [\sigma^\beta, \sigma^\alpha]$.*

From now on, σ, α and β will refer to those mentioned in Lemma 2 (unless stated otherwise).

Word Problem. The *word problem* of a group G generated by a finite set Q is the problem:

Constant: the group G
Input: $\boldsymbol{q} \in (Q \cup Q^{-1})^*$
Question: is $\boldsymbol{q} = \mathbb{1}$ in G?

In addition, if $\mathcal{C}$ is a class of groups, we also consider the *uniform* word problem for $\mathcal{C}$. Here, the group $G \in \mathcal{C}$ is part of the input (in a suitable representation).

Automata. We use the word *automaton* to denote what is more precisely called a letter-to-letter, finite state transducer. Formally, an automaton is a triple $\mathcal{T} = (Q, \Sigma, \delta)$ consisting of a finite set of *states* Q, an input and output alphabet Σ and a set $\delta \subseteq Q \times \Sigma \times \Sigma \times Q$ of *transitions*. For a transition $(p, a, b, q) \in Q \times \Sigma \times \Sigma \times Q$, we usually use the more graphical notation $p \xrightarrow{a/b} q$ where a is the *input* and b is the *output*. Additionally, we use the common way of depicting automata (see e. g. in Example 3). We will usually work with *deterministic* and *complete* automata, i. e. automata where we have $d_{p,a} = \left|\{p \xrightarrow{a/b} q \in \delta \mid b \in \Sigma, q \in Q\}\right| = 1$ for all $p \in Q$ and $a \in \Sigma$. In other words, for every $a \in \Sigma$, every state has exactly one transition with input a.

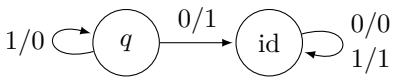
A *run* of an automaton $\mathcal{T} = (Q, \Sigma, \delta)$ is a sequence

$$q_0 \xrightarrow{a_1/b_1} q_1 \xrightarrow{a_2/b_2} \cdots \xrightarrow{a_n/b_n} q_n$$

of transitions from δ. It *starts* in q_0 and *ends* in q_n. Its *input* is $a_1 \dots a_n$ and its *output* is $b_1 \dots b_n$. If $\mathcal{T}$ is complete and deterministic, then, for every state $q \in Q$ and every word $u \in \Sigma^*$, there is exactly one run starting in q with input u. We write $q \circ u$ for its output and $q \cdot u$ for the state in which it ends. This notation can be extended to multiple states. To avoid confusion, we usually use the term *state sequence* instead of 'word' (which we reserve for input or output words) for elements $\boldsymbol{q} \in Q^*$. Now, for states $q_1, q_2, \dots, q_\ell \in Q$, we set $q_\ell \dots q_2 q_1 \circ u = q_\ell \dots q_2 \circ (q_1 \circ u)$ inductively. If the state sequence $\boldsymbol{q} \in Q^*$ is empty, then $\boldsymbol{q} \circ u$ is simply u.

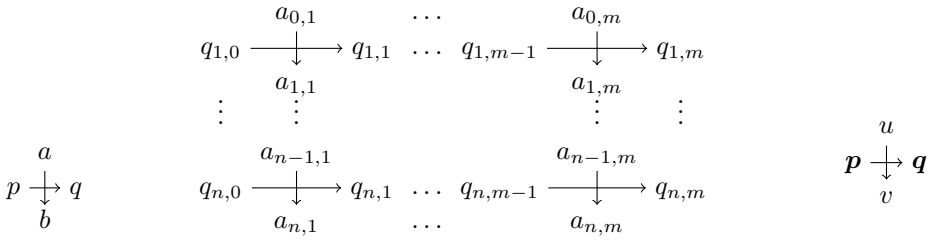
This way, every state $q \in Q$ (and even every state sequence $\boldsymbol{q} \in Q^*$) induces a map $\Sigma^* \to \Sigma^*$ and every word $u \in \Sigma^*$ induces a map $Q \to Q$. If all states of an automaton induce bijective functions, we say it is *invertible* and call it a *$\mathscr{G}$-automaton*. For a $\mathscr{G}$-automaton $\mathcal{T}$, all bijections induced by the states generate a group (with composition as operation), which we denote by $\mathscr{G}(\mathcal{T})$. A group is called an *automaton group* if it arises in this way. Clearly, $\mathscr{G}(\mathcal{T})$ is generated by the maps induced by the states of $\mathcal{T}$ and, thus, finitely generated.

▶ **Example 3.** The typical first example of an automaton generating a group is the *adding machine* $\mathcal{T} = (\{q, \mathrm{id}\}, \{0, 1\}, \delta)$:



It obviously is deterministic and complete and, therefore, we can consider the map induced by state q. We have $q^3 \circ 000 = q^2 \circ 100 = q \circ 010 = 110$. From this example, it is easy to see that the action of q is to increment the input word (which is interpreted as a reverse/least significant bit first binary representation $\overleftarrow{\mathrm{bin}}(n)$ of a number n). The inverse is accordingly to decrement the value. As the other state id acts like the identity, we obtain that the group $\mathscr{G}(\mathcal{T})$ generated by $\mathcal{T}$ is isomorphic to the infinite cyclic group.

Similar to extending the notation $\boldsymbol{q} \circ u$ to state sequences, we can also extend the notation $q \cdot u$. For this, it is useful to introduce *cross diagrams*, another notation for transitions of



(a) Cross diagrams. **(b)** Multiple combined cross diagrams. **(c)** Abbreviated cross diagram.

Figure 1 Combined and abbreviated cross diagrams.

automata. For a transition $p \xrightarrow{a/b} q$ of an automaton, we write the cross diagram given in Figure 1a. Multiple cross diagrams can be combined into a larger one. For example, the cross diagram in Figure 1b indicates that there is a transition $q_{i,j-1} \xrightarrow{a_{i-1,j}/a_{i,j}} q_{i,j}$ for all $1 \le i \le n$ and $1 \le j \le m$. Typically, we omit unneeded names for states and abbreviate cross diagrams. Such an abbreviated cross diagram is depicted in Figure 1c. If we set $q_{n,0} \dots q_{1,0} = \boldsymbol{p}$, $u = a_{0,1} \dots a_{0,m}$, $v = a_{n,1} \dots a_{n,m}$ and $\boldsymbol{q} = q_{n,m} \dots q_{1,m}$, then it indicates the same transitions as the one in Figure 1b. It is important to note here, that the right-most state in $\boldsymbol{p}$ is actually the one to act first.

If we have the cross diagram from Figure 1c, we set $\boldsymbol{p} \cdot u = \boldsymbol{q}$. This is the same, as setting $q_n \dots q_1 \cdot u = [q_n \dots q_2 \cdot (q_1 \circ u)](q_1 \cdot u)$ inductively and, with the definition from above, we already have $\boldsymbol{p} \circ u = v$.

Normally, we cannot simply re-order the rows of a cross diagram as the output interferes and we could get into different states. However, we can clearly re-order rows if they act like the identity:

▶ **Fact 4.** *Let $\mathcal{T} = (Q, \Sigma, \delta)$ be a deterministic automaton, $w \in \Sigma^*$ and $\boldsymbol{q_1}, \dots, \boldsymbol{q_\ell} \in Q^*$ such that $\boldsymbol{q}_1 \circ w = \dots = \boldsymbol{q}_\ell \circ w = w$.*

Then, for any $\boldsymbol{p} = \boldsymbol{p}_k \cdots \boldsymbol{p}_1 \in \{\boldsymbol{q}_1, \dots, \boldsymbol{q}_\ell\}^*$, we have $\boldsymbol{p} \cdot w = (\boldsymbol{p}_k \cdot w) \cdots (\boldsymbol{p}_1 \cdot w)$.

Balanced Iterated Commutators. We lift the notation g^h for conjugation and $[h, g]$ for the commutator from groups to words over the generators: for an alphabet Q and words $\boldsymbol{p}, \boldsymbol{q} \in Q^*$, we write $\boldsymbol{p}^{\boldsymbol{q}} = \boldsymbol{q}^{-1}\boldsymbol{p}\boldsymbol{q}$ and $[\boldsymbol{q}, \boldsymbol{p}] = \boldsymbol{q}^{-1}\boldsymbol{p}^{-1}\boldsymbol{q}\boldsymbol{p}$ using the natural involution for $Q \cup Q^{-1}$. We also need a balanced version of an iterated commutator; in fact, it will be crucial to our constructions.[3]

▶ **Definition 5.** *Let Q be an alphabet and $\alpha, \beta \in (Q \cup Q^{-1})^*$. For $g_t, \dots, g_1 \in (Q \cup Q^{-1})^*$, we inductively define the word $B_{\beta,\alpha}[g_t, \dots, g_1]$ by*

$$B_{\beta,\alpha}[g_1] = g_1$$
$$B_{\beta,\alpha}[g_t, \dots, g_1] = \left[B_{\beta,\alpha}[g_t, \dots, g_{\lfloor \frac{t}{2} \rfloor + 1}]^\beta,\ B_{\beta,\alpha}[g_{\lfloor \frac{t}{2} \rfloor}, \dots, g_1]^\alpha\right].$$

▶ **Lemma 6.** *On input of $g_t, \dots, g_1$, one can compute $B_{\beta,\alpha}[g_t, \dots, g_1]$ in logarithmic space.*

[3] Here, we make an exception as α and β do not refer to the corresponding permutations from Lemma 2 but are arbitrary words. Later on, however, we will apply the definition usually in such a way that α and β indeed are related to the ones from Lemma 2.

▶ Remark 7. For readers familiar with the notions: using a more careful but tedious analysis (and appropriate padding symbols), one can see that the reduction from $g_1, \dots, g_t$ to $B_{\beta,\alpha}[g_t, \dots, g_1]$ can not only be done in logarithmic space but actually it is a DLOGTIME-uniform projection reduction (compare to the proof of Barrington's result in [24, Theorem 4.52]).

If we substitute σ, α and β by the actual elements from A_5, we can see (using a simple induction) that $B_{\beta,\alpha}[g_t, \dots, g_1]$ works as a t-ary logical conjunction:

▶ **Lemma 8.** *For all $g_t, \dots, g_1 \in \{\mathrm{id}, \sigma\} \subseteq A_5$, we have*

$$B_{\beta,\alpha}[g_t, \dots, g_1] =_{A_5} \begin{cases} \sigma & \text{if } g_1 = \cdots = g_t = \sigma \\ \mathbb{1} & \text{otherwise.} \end{cases}$$

▶ Remark 9. Something similar can be done with the free group of rank two (instead of A_5) (see [20]). This is interesting because A_5 cannot be realized as an automaton group over an alphabet with less than five elements but the free group of rank three can be generated by an automaton with binary alphabet [1, 25].

3 Word Problem

In this section, we will show our main result:

▶ **Theorem 10.** *There is an automaton group with a* PSPACE*-complete word problem:*

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

In order to prove this theorem, we are going to adapt the construction used in [9, Proposition 6] to show that there is an inverse automaton semigroup with a PSPACE-complete word problem and that there is an automaton group whose word problem with a single rational constraint is PSPACE-complete. The main idea is to use a master reduction. Our automaton operates in two modes. In the first mode, which we will call 'TM-mode', it will interpret its input word as a sequence of configurations of a (suitable) PSPACE-machine and verifies that the configuration sequence constitutes a valid computation of the Turing machine. This verification is done by multiple states (where each state is responsible for a different verification part) and the information whether the verification was successful is stored in the state, **not** by manipulating the input word. So we have *successful states* and *fail states*. Upon reading a special input symbol, the automaton will switch into a second mode, the 'A_5-mode'. More precisely, successful states go into a state which acts like σ from Lemma 2 and the fail states go into an identity state id. Finally, to extract the information from the states, we use the iterated commutator from Definition 5.

The idea for the TM-mode is similar to the approach taken by Kozen to show PSPACE-completeness of the DFA Intersection Problem where the input word is interpreted as a sequence of configurations of a PSPACE Turing machine where each configuration is of length $s(n)$:

$$\gamma_1^{(0)}\gamma_2^{(0)}\gamma_3^{(0)} \cdots \gamma_{s(n)}^{(0)} \# \gamma_1^{(1)}\gamma_2^{(1)}\gamma_3^{(1)} \cdots \gamma_{s(n)}^{(1)} \# \cdots$$

In Kozen's proof, there is an acceptor for each position i of the configurations with $1 \leq i \leq s(n)$ which checks for all t whether the transition from $\gamma_i^{(t)}$ to $\gamma_i^{(t+1)}$ is valid. In our case, however,

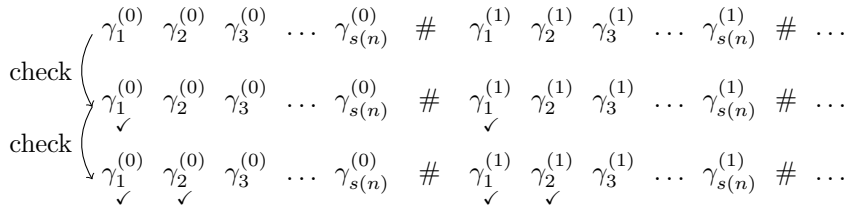


Figure 2 Illustration of the checkmark approach.

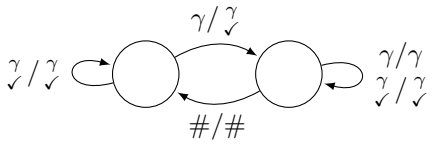


Figure 3 Adding a check-mark yields a non-invertible automaton.

the automaton must not depend on the input (or its length n) and we have to handle this a bit differently. The first idea is to use a 'check-mark approach'. First, we check all first positions for valid transitions. Then, we put a check-mark on all first positions, which tells us that we now have to check all second positions (i. e. the first ones without a check-mark). Again, we put a check-mark on all these, continue with checking all third positions and so on (see Figure 2).

The problem with this approach is that the check-marking leads to an intrinsically non-invertible automaton (see Figure 3). To circumvent this, we generalize the check-mark approach: before each symbol $\gamma_i^{(t)}$ of a configuration, we add a 0^k block (of sufficient length k). In the spirit of Example 3, we interpret this block as representing a binary number. We consider the symbol following the block as 'unchecked' if the number is zero; for all other numbers, it is considered as 'checked'. Now, checking the next symbol boils down to incrementing each block until we have encountered a block whose value was previously zero (and this can be detected while doing the addition). This idea is depicted in Figure 4. It would also be possible to have the check-mark block after each symbol instead of before (which might be more intuitive) but it turns out that our ordering has some technical advantages.

Proof of Theorem 10. Since the uniform word problem for automaton groups is in PSPACE [22] (see also [9, Proposition 2 and 3]), so is the word problem of any (fixed) automaton group. Therefore, we only have to show the hardness part of the result.

Consider an arbitrary PSPACE-complete problem and let M be a deterministic, polyno-

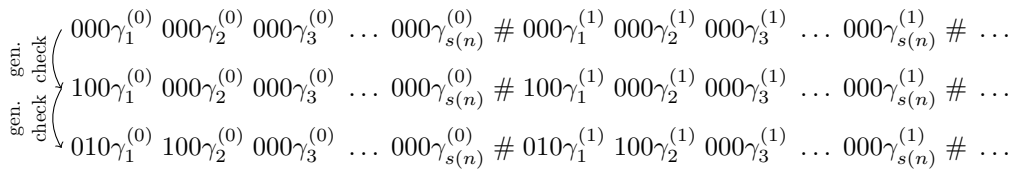


Figure 4 The idea of our generalized check-marking approach.

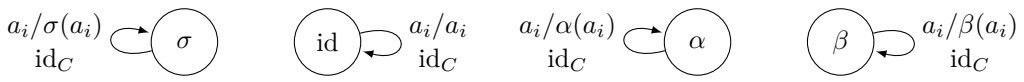
mially space-bounded Turing machine deciding it[4] with input alphabet Λ, tape alphabet Δ, blank symbol $\square$, state set P, initial state p_0 and accepting states $F \subseteq P$. Thus, for any input word of length n, all configurations of M are of the form $\square\Delta^\ell P\Delta^m\square$ with $\ell + 1 + m = s(n)$ for some polynomial s. This makes the problem

Constant:	the PSPACE machine M
Input:	$w \in \Lambda^*$
Question:	does M reach a configuration with a state from F from the initial configuration $\square p_0 w \square^{s(n)-n-1}\square$?

PSPACE-complete. From the machine M, we construct the $\mathscr{G}$-automaton $\mathcal{T}$ and, from the input w, we construct the state sequence $\boldsymbol{q}$. The input words for the automaton will be interpreted to have two parts separated by a special symbol \$. The first part will represent a computation of the machine M and the automaton will be in the TM-mode (mentioned above) while reading it. At the separation symbol, the automaton will switch into the A_5-mode and will operate letter-wise as σ, α, β (from Lemma 2) or the identity on the second part of the word. For the state sequence, we define $\boldsymbol{q} = B_0[f, \boldsymbol{q}_{s(n)}, \dots, \boldsymbol{q}_1, \boldsymbol{c}', \boldsymbol{c}_{s(n)}, \dots, \boldsymbol{c}_1, r]$ where we use the short-hand notation B_0 for the balanced commutator B_{β_0,α_0} from Definition 5. We will define the individual entries $f, \boldsymbol{q}_{s(n)}, \dots, \boldsymbol{q}_1, \boldsymbol{c}', \boldsymbol{c}_{s(n)}, \dots, \boldsymbol{c}_1, r$ of the balanced commutator in detail below. The general idea is that they are used to check different parts of the computation. For example, r is responsible for checking that the first part of the input word is in the correct form for a computation and $\boldsymbol{q}_i$ is responsible for checking that all transitions at position i are valid transitions of M. The individual entries do not change the first part of the word (or – more precisely – any changes are reverted) and operate as the identity if the check failed or as σ if the check succeeded on the second word part. This way, if a single check fails, $\boldsymbol{q}$ as a whole will operate as the identity and, if all checks succeed, it will operate as σ on the second word part by Lemma 8. The commutator acts as a logical conjunction where σ is interpreted as `true` and the identity is interpreted as `false`.

Let $\Gamma = \Delta \uplus P$. From now on, we will not work with M anymore but rather only with its corresponding $\tau : \Gamma^3 \to \Gamma$ from Fact 1.

Construction of the Automaton. The automaton $\mathcal{T}$ works in the way described above and is the union of several simpler automata. For the alphabet, we use $\Sigma = \Gamma \uplus \{0, 1\} \uplus \{\#, \$\}$ with new letters 0, 1, # and \$. The letters 0 and 1 will be used for the generalized check-mark approach described above, the letter # is used to separate individual configurations and \$ acts as an 'end-of-computation' symbol switching the automaton from the TM-mode to the A_5-mode. For the A_5-mode, we choose $a_1, \dots, a_5 \in \Sigma$ arbitrarily (but distinct) and assume that σ, α and β (from Lemma 2) operate on $\{a_1, \dots, a_5\}$ such that $\sigma(a_1) \neq a_1$. Furthermore, we set $C = \Sigma \setminus \{a_1, \dots, a_5\}$. With this, the first part of the automaton $\mathcal{T}$ used for the A_5-mode is



where the a_i-transitions exist for all $i \in \{1, \dots, 5\}$ and we use the convention that id_X indicates x/x-transitions for all $x \in X \subseteq \Sigma$. Obviously, the state id acts as the identity and the action of the state σ on a word is to apply the permutation σ letter-wise (and to ignore letters from C), which justifies the dual use in notation as we can identify σ in the

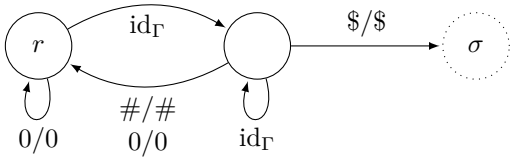
[4] Alternatively, we could also use a PSPACE-universal Turing machine for our construction.

automaton group with σ in A_5. For the intuition, it helps to see id as a 'fail' state and σ as an 'okay' state in the following. In the end, we will implement this intuition basically using the iterated commutator from Definition 5. For this commutator, we also need the conjugating elements α and β, which work in the same way as σ. However, they do not have an intuitive semantic and are mostly there for technical reasons.

Now, let us describe the part of the automaton used for the TM-mode. First, we need two states which ignore everything in the TM-mode and then go to α or β:

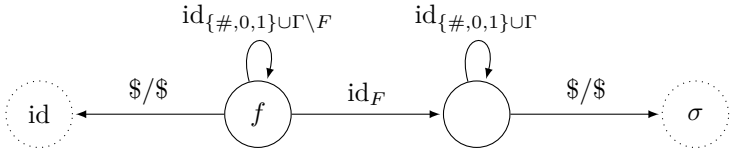
where dotted states refer to the states defined above.

The next part of our automaton is used to check that the input word (for the TM-mode) is of the form $(0^*\Gamma)^+ (\#(0^*\Gamma)^+)^*$:



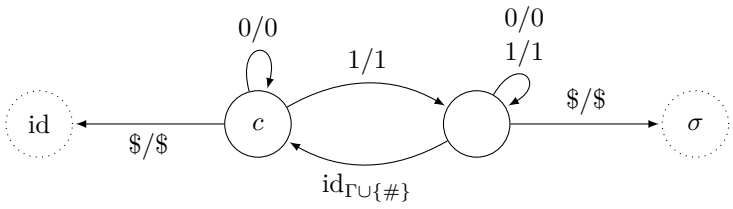
Here, we use the convention that, whenever a transition is missing for some $x \in \Sigma$, there is an implicit x/x-transition to the state id (as defined above). Note that we do not check that the factors in $(0^*\Gamma)^+$ correspond to well-formed configurations for the Turing machine. This will be done implicitly by checking that the input word belongs to a valid computation of the Turing machine, which we describe below.

Next, we need a part which checks whether the input word contains a final state (if this is not the case, we want to 'reject' the word):



Finally, we come to the more complicated parts of $\mathcal{T}$. The first one is for the generalized check-marking as described above and is depicted in Figure 5. In fact, we need this part *twice*: once for $g = \sigma$ and once for $g = \mathrm{id}$. Notice that, during the TM-mode phase (i. e. before the first \$), the two versions behave exactly the same way; the only difference is after switching to the A_5-mode: while $\checkmark_{\mathrm{id}}$ still always acts like the identity, $\checkmark_\sigma$ acts non-trivially on suitable input words.

Additionally, we also need an automaton part verifying that every configuration symbol has been check-marked (in the generalized sense):



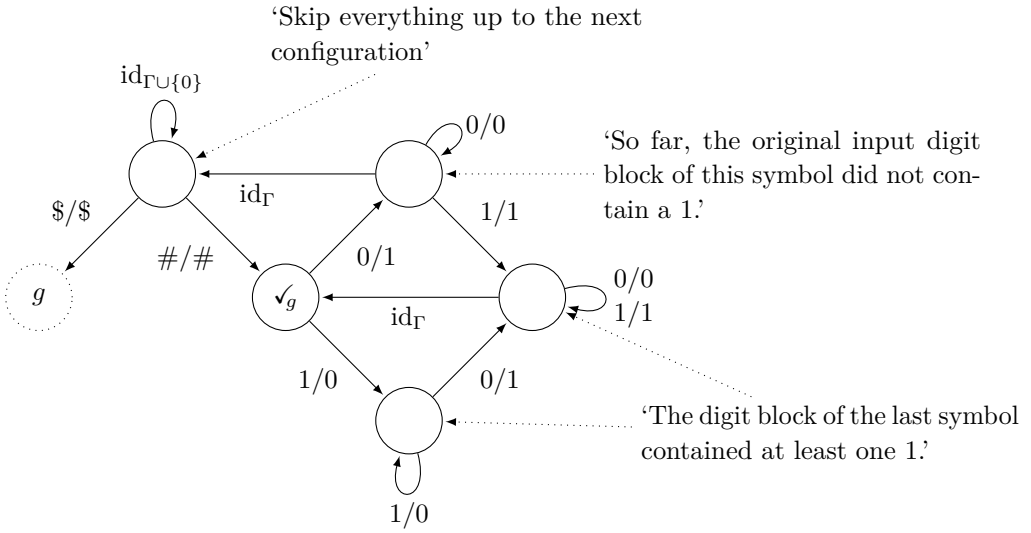
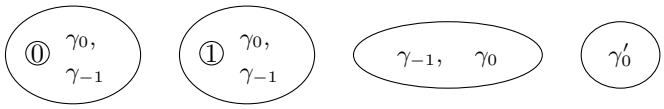


Figure 5 The automaton part used for generalized check-marking.

The last part is for checking the validity of the transition at all first so-far unchecked positions. While it is not really difficult, this part is a bit technical. Intuitively, for checking the transition from time step $t-1$ to time step t at position i, we need to compute $\gamma_i^{(t)} = \tau(\gamma_{i-1}^{(t-1)}, \gamma_i^{(t-1)}, \gamma_{i+1}^{(t-1)})$ from the configuration symbol at positions $i-1$, i and $i+1$ for time step $t-1$. We store $\gamma_i^{(t)}$ in the state (to compare it to the actual value). Additionally, we need to store the last two symbols of configuration t we have encountered so far (for computing what we expect in the next time step later on) and whether we have seen a 1 or only 0s in the check-mark digit block. For all this, we use the states



with $\gamma_{-1}, \gamma_0, \gamma_0' \in \Gamma$. The idea is the following. In the ⓪ and ① states, we store the value we expect for the first unchecked symbol (γ_0) and the last symbol we have seen in the current configuration (γ_{-1}). We are in the ⓪-state if we have not seen any 1 in the digit block yet and in the ① if we did. The latter two are used to skip the rest of the current configuration and to compute the symbol we expect for the first unchecked position in the next configuration (γ_0').

We use these states in the transitions schematically depicted in Figure 6. Here, the dashed transitions exist for all γ_{-1}' and γ_1 in Γ but go to different states, respectively, and the dotted states correspond to the respective non-dotted states with different values for γ_0 and γ_{-1} (with the exception of σ, which corresponds to the state defined above). We also define $q_{\gamma'}$ as the state on the bottom right (for every $\gamma' \in \Gamma$, respectively).

The automaton parts depicted in Figure 5 and Figure 6 are best understood with an example. Consider the input word

$$100\gamma_1^{(0)}\ 000\gamma_2^{(0)}\ 000\gamma_3^{(0)}\ \#\ 100\gamma_1^{(1)}\ 000\gamma_2^{(1)}\ 000\gamma_3^{(1)}\ \$$$

where we consider the $\gamma_i^{(t)}$ to form a valid computation. If we start in state $q_{\gamma_2^{(0)}}$ and read the above word, we immediately take the 1/1-transition and go into the corresponding ① state where we skip the rest of the digit block. Using the dashed transition, the next symbol

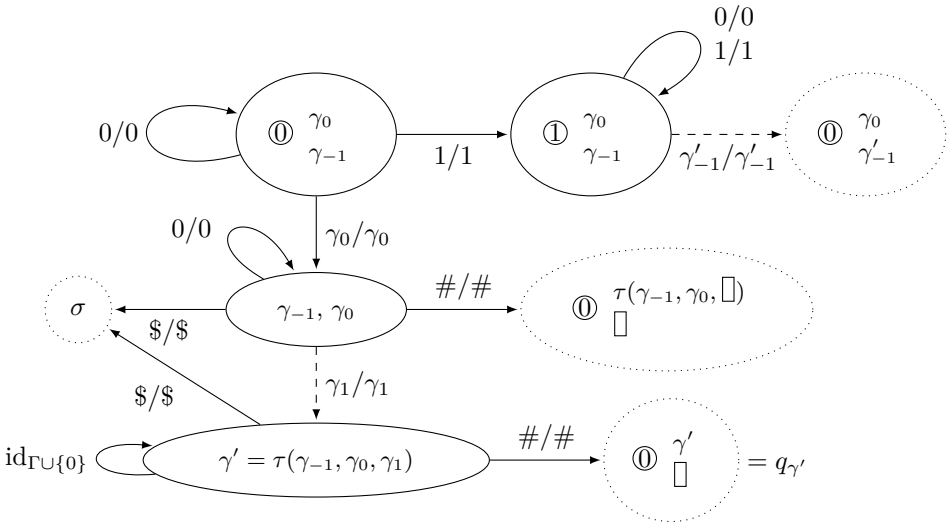


Figure 6 Schematic representation of the transitions used for checking Turing machine transitions and definition of q'_γ; the dashed transitions exist for all γ'_{-1} and γ_1 in Γ but go to different states, respectively

$\gamma_1^{(0)}$ takes us back into a ⓪-state where the upper entry is still $\gamma_2^{(0)}$ but the lower entry is now $\gamma_1^{(0)}$ (i. e. the last configuration symbol we just read). We loop at this state while reading the next three 0s and, since the next symbol $\gamma_2^{(0)}$ matches with the one stored in the state, we get into the states with entries $\gamma_1^{(0)}, \gamma_2^{(0)}$ where we skip the next three 0s again. Reading $\gamma_3^{(0)}$ now gets us into the state with entry $\gamma_2^{(1)}$ since we have $\tau(\gamma_1^{(0)}, \gamma_2^{(0)}, \gamma_3^{(0)}) = \gamma_2^{(1)}$ by assumption that the $\gamma_i^{(t)}$ form a valid computation. Here, we read #/# and the process repeats for the second configuration, this time starting in $q_{\gamma_2^{(2)}}$. When reading the final \$, we are in the state with entry $\tau(\gamma_1^{(1)}, \gamma_2^{(1)}, \gamma_3^{(1)})$ and finally go to σ. Notice that during the whole process, we have not changed the input word at all!

If we now start reading the input word again in state $\checkmark_\sigma$ (see Figure 5 and also refer to Figure 4), we turn the first 1 into a 0, go to the state at the bottom, turn the next 0 into a 1 and go to the state on the right, where we ignore the next 0. When reading $\gamma_1^{(0)}$, we go back to $\checkmark_\sigma$. Next, we take the upper exit and turn the next 0 into a 1. The remaining 0s are ignored and we remain in the state at the top right until we read $\gamma_2^{(0)}$ and go to the state at the top left. Here, we ignore everything up to #, which gets us back into $\checkmark_\sigma$. The second part works in the same way with the difference that we go to σ at the end since we encounter the \$ instead of #. The output word, thus, is

$$010\gamma_1^{(0)}\ 100\gamma_2^{(0)}\ 000\gamma_3^{(0)}\ \#\ 010\gamma_1^{(1)}\ 100\gamma_2^{(1)}\ 000\gamma_3^{(1)}\ \$$$

and we have check-marked the next position in both configurations.

This concludes the definition of the automaton and the reader may verify that $\mathcal{T}$ is indeed a $\mathscr{G}$-automaton since all individual parts are $\mathscr{G}$-automata. Furthermore, apart from the check-marking, all states except σ, α and β (which belong to the A_5-mode and are only entered when reading \$) act like the identity.

Definition of the State Sequence. To describe the actual reduction, we have to define the state sequence $\boldsymbol{q}$ such that $\boldsymbol{q}$ depends only on the input word w for the Turing machine. Similar to the automaton $\mathcal{T}$, this sequence consists of multiple parts. Each part will verify a certain aspect of the input word and the general idea is that, after reading a word $u\$$, we

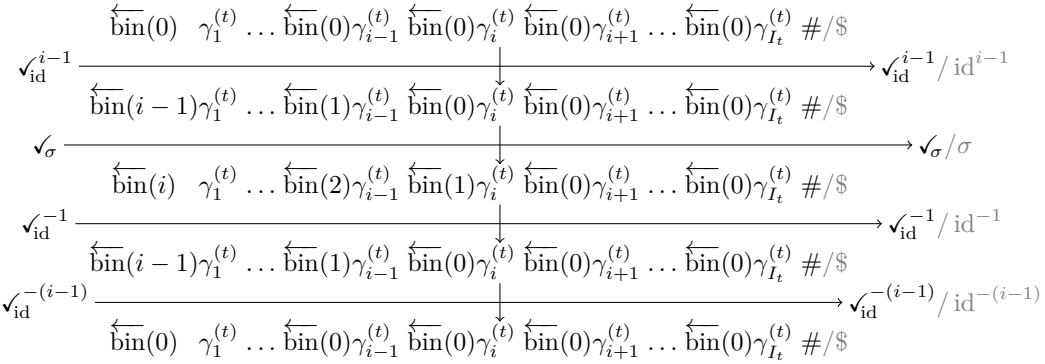
are either in σ (if u satisfies the criterion we are currently checking) or in id (if it does not). Finally, using the balanced commutator from Definition 5, we can find whether any of the criteria was not satisfied. For this to work easily, we define the individual parts in such a way that the output will be $u\$$ again, which will allow us to apply Fact 4.

First, we simply use the state r to verify that u is from $(0^*\Gamma)^+ (\#(0^*\Gamma)^+)^*$. Thus, we only need to consider the case that u is of the form

$$0^{\ell_1^{(0)}}\gamma_1^{(0)}\ 0^{\ell_2^{(0)}}\gamma_2^{(0)}\ \dots\ 0^{\ell_{I_0}^{(0)}}\gamma_{I_0}^{(0)}\ \#\dots\#\ 0^{\ell_1^{(T)}}\gamma_1^{(T)}\ 0^{\ell_2^{(T)}}\gamma_2^{(T)}\ \dots\ 0^{\ell_{I_T}^{(T)}}\gamma_{I_T}^{(T)} \tag{$\dagger$}$$

with $\gamma_i^{(t)} \in \Gamma$ any further.

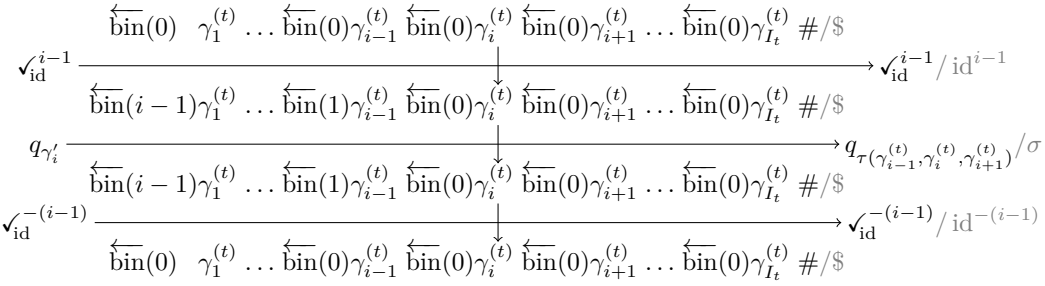
Next, we need to verify that, for every $1 \le i \le s(n)$, we can check-mark the first i positions. For this, we use $\boldsymbol{c}_i = \checkmark_{\mathrm{id}}^{-i}\checkmark_\sigma\checkmark_{\mathrm{id}}^{i-1}$ as we have the cross diagram



where $\overleftarrow{\mathrm{bin}}(z)$ denotes the reverse/least significant bit first binary representation of z (of sufficient length). Here, it is useful to observe that, if the j^{th} 0 block with $j \le i$ is not long enough to count to its required value, then we will always end up in id after reading a \$. The same happens if $I_t < i$ (i. e. if one of the configurations is 'too short'). So this guarantees, $I_t \ge s(n)$ for all t.

On the other hand, we use $\boldsymbol{c}' = \checkmark_{\mathrm{id}}^{-s(n)} c \checkmark_{\mathrm{id}}^{s(n)}$ to ensure that, after check-marking the first $s(n)$ positions in every configurations, all symbols have been check-marked (i. e. that no configuration is 'too long'), which guarantees $I_t = s(n)$ for all t.

Now that we have ensured that the word is of the correct form and we can count high enough for our check-marking, we need to actually verify that the $\gamma_i^{(t)}$ constitute a valid computation of the Turing machine with the initial configuration $\gamma'_1 \dots \gamma'_{s(n)} = p_0 w \square^{s(n)-n-1}$ for the input word w. To do this, we define $\boldsymbol{q}_i = \checkmark_{\mathrm{id}}^{-(i-1)} q_{\gamma'_i} \checkmark_{\mathrm{id}}^{i-1}$ for every $1 \le i \le s(n)$ as we have the cross diagram



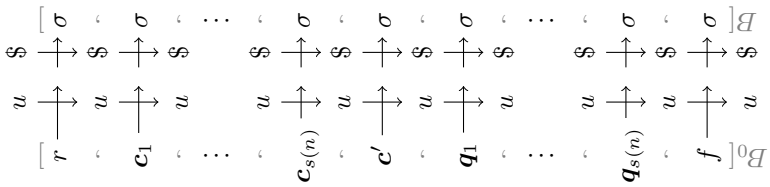


Figure 7 Cross diagram for $\boldsymbol{q}$.

if $\gamma_i^{(t)}$ is the expected γ_i'. Otherwise (if $\gamma_i^{(t)} \neq \gamma_i'$), we always end in state id after reading the \$. Finally, to ensure that the computation is not only valid but also accepting, we use the state f.

Summing this up, we define $\boldsymbol{q} = B_0[f, \boldsymbol{q}_{s(n)}, \ldots, \boldsymbol{q}_1, \boldsymbol{c}', \boldsymbol{c}_{s(n)}, \ldots, \boldsymbol{c}_1, r]$ as mentioned at the begnning of the proof. Remember that we use the short-hand notation B_0 for the balanced commutator B_{β_0,α_0} from Definition 5 and observe that the individual parts of $\boldsymbol{q}$ can indeed be computed in logarithmic space and that, thus, this is also true for $\boldsymbol{q}$ itself by Lemma 6.

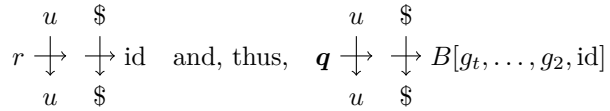
Correctness. We need to prove that the action of $\boldsymbol{q}$ is equal to the identity if and only if the Turing machine does **not** accept the input word w. The easier direction is to assume that the Turing machine accepts on the initial configuration $\square p_0 w \square^{s(n)-n-1}\square$. Let $\gamma_1^{(0)} \ldots \gamma_{s(n)}^{(0)} \vdash \gamma_1^{(1)} \ldots \gamma_{s(n)}^{(1)} \vdash \cdots \vdash \gamma_1^{(T)} \ldots \gamma_{s(n)}^{(T)}$ be the corresponding computation with $\gamma_1^{(0)} = p_0$, $\gamma_2^{(0)} \ldots \gamma_{n+1}^{(0)} = w$ and $\gamma_i^{(T)} \in F$ for some $1 \leq i \leq s(n)$. We choose $\ell = \lceil \log(s(n)) \rceil + 1$ and define

$$u = 0^\ell \gamma_1^{(0)} \ldots 0^\ell \gamma_{s(n)}^{(0)} \# 0^\ell \gamma_1^{(1)} \ldots 0^\ell \gamma_{s(n)}^{(1)} \# \ldots \# 0^\ell \gamma_1^{(T)} \ldots 0^\ell \gamma_{s(n)}^{(T)}.$$

We now let $\boldsymbol{q}$ act on the word $u\$a_1$. Recall that we assume σ to operate non-trivially on a_1. The reader may verify that we have the black part of the cross diagram depicted in Figure 7. From Fact 4, we immediately also obtain the gray additions to the cross diagram where we use B instead of $B_{\beta,\alpha}$ for the balanced commutator from Definition 5. By Lemma 8, we obtain $B[\sigma, \ldots, \sigma] = \sigma$ in A_5 and, thus, in $\mathscr{G}(\mathcal{T})$. Therefore, $\boldsymbol{q}$ acts non-trivially on $u\$a_1$.

For the other direction, assume that no valid computation of M on the initial configuration $\square p_0 w \square^{s(n)-n-1}\square$ contains an accepting state from F. We have to show that $\boldsymbol{q}$ acts like the identity on all words from Σ^*. If the word does not contain a \$, then all individual parts of $\boldsymbol{q}$ act on it like the identity by construction. This is clearly the case for r, $\boldsymbol{c}'$, the $\boldsymbol{q}_i$ and f. For the $\boldsymbol{c}_i$, the only point to note is that $\checkmark_\sigma$ acts in the same way as $\checkmark_{\mathrm{id}}$ on such words.

Thus, we may assume that the word is of the form $u\$v$. If u is not of the form $(0^*\Gamma)^+ (\#(0^*\Gamma)^+)^*$, we have the cross diagram



with $g_2, \ldots, g_t \in \{\sigma, \mathrm{id}\}$. As we have, $B[g_t, \ldots, g_2, \mathrm{id}] =_{A_5} \mathbb{1}$ by Lemma 8, we obtain that $\boldsymbol{q}$ acts like the identity on $u\$v$.

Therefore, we assume u to be of the form mentioned in Equation † and use a similar argumentation for the remaining cases. If u does not contain a state from F, then we end up in state id after reading \$ for f. As w is not accepted by the machine, this includes in particular all valid computations on the initial configuration $\square p_0 w \square^{s(n)-n-1}\square$. If one of the

0 blocks in u is too short to count to a value required for the check-marking (i. e. one $\ell_i^{(t)}$ is too small), then the corresponding $\boldsymbol{c}_i$ will go to (a state sequence equivalent to) id. This is also true if one configuration is too short (i. e. $I_t < s(n)$ for some t). If one configuration is too long (i. e. $I_t > s(n)$), then this will be detected by $\boldsymbol{c}'$ as not all positions will be check-marked after check-marking all first $s(n)$ positions in every configuration. Finally, $\boldsymbol{q}_i$ yields an id if $\gamma_i^{(0)}$ is not the correct symbol from the initial configuration or if we have $\gamma_i^{(t+1)} \neq \tau(\gamma_{i-1}^{(t)}, \gamma_i^{(t)}, \gamma_{i+1}^{(t)})$ for some t (where we let $\gamma_{-1}^{(t)} = \square = \gamma_{s(n)+1}^{(t)}$). ◀

▶ Remark 11. The constructed automaton has $3\Gamma^2 + \Gamma + 22$ states where Γ is the sum of the number of states and the number of tape symbols for a Turing machine for a PSPACE-hard problem.

▶ Remark 12. Using Remark 9, we can reduce the alphabet of the automaton to a binary one. This, however, involves some technical difficulties mainly for two reasons. First, the original alphabet needs to be compressed into blocks over the binary alphabet and, second, the element to be used in the balanced iterated commutator depends on the position.

4 Compressed Word Problem

In this section, we re-apply our previous construction to show that there is an automaton group with an EXPSPACE-complete compressed word problem. The *compressed word problem* of a group is similar to the normal word problem. However, the input element (to be compared to the neutral element) is not given directly but as a straight-line program. A *straight-line program* is a context-free grammar which generates exactly one word.

▶ **Theorem 13.** *There is an automaton group with an* EXPSPACE*-complete compressed word problem:*

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

Proof. Before starting, we observe that, whenever we have a variable X in a straight-line program generating a word representing a group element g, we can easily obtain a variable X^{-1} generating a word representing g^{-1} by mirroring all rules for X, replacing all letters a by a^{-1} and all variables A by A^{-1} (where we have to continue recursively). Hence, we will always assume that we also have A^{-1} if we have described A in the following.

For the actual proof, we use the same construction as in the proof of Theorem 10, but we start with a Turing machine M for an EXPSPACE-complete problem. Now, all configurations on input of a word w of length n are of the form $\square\Delta^\ell P \Delta^m\square$ with $\ell + 1 + m = s(n)$ where $s(n)$ is of the form 2^{n^e} for some constant $e \in \mathbb{N}$.

Recall that, for the (normal) word problem, we used

$$\boldsymbol{q} = B_0[f, \boldsymbol{q}_{s(n)}, \ldots, \boldsymbol{q}_1, \boldsymbol{c}', \boldsymbol{c}_{s(n)}, \ldots, \boldsymbol{c}_1, r] \tag{‡}$$

for the reduction where

$$\boldsymbol{c}_i = \checkmark_{\mathrm{id}}^{-i} \checkmark_\sigma \checkmark_{\mathrm{id}}^{i-1}, \quad \boldsymbol{c}' = \checkmark_{\mathrm{id}}^{-s(n)} c \checkmark_{\mathrm{id}}^{s(n)} \quad \text{and} \quad \boldsymbol{q}_i = \checkmark_{\mathrm{id}}^{-(i-1)} q_{\gamma_i'} \checkmark_{\mathrm{id}}^{i-1}$$

for $1 \leq i \leq s(n)$. The problem now is that we have exponentially many $\boldsymbol{c}_i$ and $\boldsymbol{q}_i$. Thus, we cannot output them in logarithmic space (or polynomial time), not even if we use a straight-line program. So, we will have to accommodate for this. The good news is that, for

$\boldsymbol{c}'$, this is not a problem: we can output a straight-line program with starting variable C' generating $\boldsymbol{c}'$:

$$C' \to M_{2^{n^e}} c M_{2^{n^e}}^{-1}, \quad M_{2^{n^e}} \to M_{2^{n^e-1}} M_{2^{n^e-1}}, \quad \ldots, \quad M_{2^1} \to M_{2^0} M_{2^0}, \quad M_{2^0} \to \checkmark_{\mathrm{id}}$$

Notice that this program is of polynomial length in n and, clearly, C' evaluates to $\boldsymbol{c}'$.

To circumvent the problem with the $\boldsymbol{c}_i$, we use the fact that the behavior of $\boldsymbol{c}_i$ is structurally the same as the behavior of $\boldsymbol{c}_j$. This can be exploited by defining a slightly modified version of the balanced commutator B_0 (where B_0 is as in the proof of Theorem 10): let $B_c[1] = \checkmark_{\mathrm{id}}^{-1} \checkmark_\sigma$ and

$$\begin{aligned} B_c[k] &= \left[B_c[\lceil \tfrac{k}{2} \rceil]^{\beta_0 \checkmark_{\mathrm{id}}^{\lfloor \frac{k}{2} \rfloor}},\ B_c[\lfloor \tfrac{k}{2} \rfloor]^{\alpha_0} \right] \\ &= \checkmark_{\mathrm{id}}^{-\lfloor \frac{k}{2} \rfloor}\ \beta_0^{-1} B_c[\lceil \tfrac{k}{2} \rceil]^{-1} \beta_0\ \checkmark_{\mathrm{id}}^{\lfloor \frac{k}{2} \rfloor}\ \alpha_0^{-1} B_c[\lfloor \tfrac{k}{2} \rfloor]^{-1} \alpha_0 \\ &\quad\ \checkmark_{\mathrm{id}}^{-\lfloor \frac{k}{2} \rfloor}\ \beta_0^{-1} B_c[\lceil \tfrac{k}{2} \rceil] \beta_0\ \checkmark_{\mathrm{id}}^{\lfloor \frac{k}{2} \rfloor}\ \alpha_0^{-1} B_c[\lfloor \tfrac{k}{2} \rfloor] \alpha_0. \end{aligned}$$

The intuitive idea behind this definition is that we start with all check-mark counters at value zero ('nothing is check-marked'). Then the bottom right part checks that we can check the first $\lfloor \frac{k}{2} \rfloor$ positions. However, the counters are reset to zero during this checking (by induction), so we check-mark the positions again (without modifying the value in the A_5-mode) using a suitable power of $\checkmark_{\mathrm{id}}$. Then the part on the bottom left checks that we can check-mark the next $\lceil \frac{k}{2} \rceil$ positions. After this, the counters are reset to zero. Then the same happens again only using the inverse group elements this time to realize the commutator. Here, it is important to note that the top left part sees the same situation as the bottom left part, so it behaves in the same way except for going to σ^{-1} instead of σ in the A_5-mode.

Formally, we can show $B_c[k] =_{\mathscr{G}(\mathcal{T})} B_0[\boldsymbol{c}_k, \ldots, \boldsymbol{c}_1]$ by induction. The base case holds by definition and, for the induction step, we observe that $\checkmark_{\mathrm{id}}$ commutes with α_0 and β_0 and that we have $\boldsymbol{c}_{i+d} = \checkmark_{\mathrm{id}}^{-d} \boldsymbol{c}_i \checkmark_{\mathrm{id}}^{d}$ in $\mathscr{G}(\mathcal{T})$. Since we have $[y,x]^z = [y^z, x^z]$ in any group, we obtain in $\mathscr{G}(\mathcal{T})$

$$\begin{aligned} B_c[k] &= \left[B_0[\boldsymbol{c}_{\lceil \frac{k}{2} \rceil}, \ldots, \boldsymbol{c}_1]^{\beta_0 \checkmark_{\mathrm{id}}^{\lfloor \frac{k}{2} \rfloor}},\ B_0[\boldsymbol{c}_{\lfloor \frac{k}{2} \rfloor}, \ldots, \boldsymbol{c}_1]^{\alpha_0} \right] && \text{(by induction)} \\ &= \left[B_0\left[\boldsymbol{c}_{\lceil \frac{k}{2} \rceil}^{\checkmark_{\mathrm{id}}^{\lfloor \frac{k}{2} \rfloor}}, \ldots, \boldsymbol{c}_1^{\checkmark_{\mathrm{id}}^{\lfloor \frac{k}{2} \rfloor}}\right]^{\beta_0},\ B_0[\boldsymbol{c}_{\lfloor \frac{k}{2} \rfloor}, \ldots, \boldsymbol{c}_1]^{\alpha_0} \right] \\ &= \left[B_0[\boldsymbol{c}_k, \ldots, \boldsymbol{c}_{\lfloor \frac{k}{2} \rfloor + 1}]^{\beta_0},\ B_0[\boldsymbol{c}_{\lfloor \frac{k}{2} \rfloor}, \ldots, \boldsymbol{c}_1]^{\alpha_0} \right] \\ &= B_0[\boldsymbol{c}_k, \ldots, \boldsymbol{c}_1]. \end{aligned}$$

In particular, this shows that no $B_c[k]$ modifies a word not containing \$ (i. e. all $B_c[k]$ act like the identity in the TM-mode). This is important as we will be using $B_c[s(n)] = B_c[2^{n^e}]$ for checking that we can check-mark all positions in the end.

The straight-line program for $B_c[s(n)]$ is similar to the one of $\boldsymbol{c}'$ (and uses the variables M_{2^i} defined above). We let

$$\begin{aligned} B_{c,2^\ell} \to\ & M_{2^{\ell-1}}^{-1}\ \beta_0^{-1} B_{c,2^{\ell-1}}^{-1} \beta_0\ M_{2^{\ell-1}}\ \alpha_0^{-1} B_{c,2^{\ell-1}}^{-1} \alpha_0 \\ & M_{2^{\ell-1}}^{-1}\ \beta_0^{-1} B_{c,2^{\ell-1}} \beta_0\ M_{2^{\ell-1}}\ \alpha_0^{-1} B_{c,2^{\ell-1}} \alpha_0 \end{aligned}$$

for $\ell > 0$ and $B_{c,2^0} \to \checkmark_{\mathrm{id}}^{-1} \checkmark_\sigma$. This way, we can compute $B_{c,2^{n^e}}$ as a variable for $B_c[s(n)]$ in logarithmic space.

At first, it seems that we cannot use a similar idea for the $\boldsymbol{q}_i$ as they are responsible for different symbols of the configuration. However, as the initial configuration is $\square p_0 w \square^{2^{n^e}-n}\square$, all $\boldsymbol{q}_i$ except of the first $n+1$ ones are responsible for the same tape symbol (the blank symbol $\square$). We define $B_q[1] = q_\square$; the inductive step is the same as in the definition of $B_c[k]$ and the same ideas apply. This time, however, we want to start with the first $n+1$ positions check-marked (this is the $p_0 w$ part) and, thus, we will be using $\checkmark_{\mathrm{id}}^{-n-1} B_q[s(n)-n-1]\checkmark_{\mathrm{id}}^{n+1}$ and the original $\boldsymbol{q}_i$ for the first $n+1$ positions. With the same arguments as for $B_c[s(n)]$, one can show

$$\checkmark_{\mathrm{id}}^{-n-1} B_q[s(n)-n-1]\checkmark_{\mathrm{id}}^{n+1} =_{\mathscr{G}(\mathcal{T})} B_0[\boldsymbol{q}_{s(n)}, \ldots, \boldsymbol{q}_{n+2}].$$

A straight-line program for $B_q[s(n)-n-1]$ can be obtained in a similar way to the one for $B_c[s(n)]$[5] and the $\checkmark_{\mathrm{id}}$ blocks are short enough to output them directly.

Summing this up, we set

$$Q \to B_0[f,\ \checkmark_{\mathrm{id}}^{-n-1} B_q \checkmark_{\mathrm{id}}^{n+1},\ \boldsymbol{q}_{n+1}, \ldots, \boldsymbol{q}_1,\ C',\ B_{c,s(n)},\ r]$$

where B_q is the starting variable of the straight-line program for $B_q[s(n)-n-1]$. By Lemma 6, we can compute the right-hand side of this rule in logarithmic space. By the construction of the straight-line programs for B_q, $B_{c,s(n)}$ and C' (and the proved equalities), it follows that Q evaluates to

$$B_0[f,\ B_0[\boldsymbol{q}_{s(n)}, \ldots, \boldsymbol{q}_{n+2}],\ \boldsymbol{q}_{n+1}, \ldots, \boldsymbol{q}_1,\ \boldsymbol{c}',\ B_0[\boldsymbol{c}_{s(n)}, \ldots, \boldsymbol{c}_1],\ r].$$

Now, using Lemma 8, it is easy to see that this is equal to $\boldsymbol{q}$ from (‡) in $\mathscr{G}(\mathcal{T})$: before reading the first letter \$, it operates as the identity and, after the first letter \$, it operates either as the identity or as σ depending on whether all components of the commutator act like σ. ◀

▶ **Corollary 14.** *There is an automaton group with a* PSPACE*-complete word problem and an* EXPSPACE*-complete compressed word problem.*

Proof. The result follows from the closure of the class of automaton groups under direct product (which is well-known and easy to see) in combination with Theorem 10 and Theorem 13. ◀

References

1 Stanislav V. Aleshin. A free group of finite automata. *Vestnik Moskovskogo Universiteta. Seriya I. Matematika, Mekhanika*, 4:12–14, 1983.

2 Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

3 David A. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. *J. Comput. Syst. Sci.*, 38(1):150–164, 1989. doi:10.1016/0022-0000(89)90037-8.

4 Laurent Bartholdi and Ivan Mitrofanov. The word and order problems for self-similar and automata groups. *arXiv preprint*, 2017. arXiv:1710.10109.

5 Ievgen V. Bondarenko. Growth of Schreier graphs of automaton groups. *Mathematische Annalen*, 354(2):765–785, 2012. doi:10.1007/s00208-011-0757-x.

[5] Be aware that this is a bit more technical, since we are not only working with powers of two anymore and need to take the rounding into consideration.

6 Ievgen V. Bondarenko. The word problem in Hanoi Towers groups. *Algebra and Discrete Mathematics*, 17(2):248–255, 2014.

7 William W. Boone. The Word Problem. *Annals of Mathematics*, 70(2):207–265, 1959.

8 Daniele D'Angeli, Emanuele Rodaro, and Jan Philipp Wächter. The structure theory of partial automaton semigroups. *arXiv preprint*, 2018. `arXiv:1811.09420`.

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. `doi:10.1016/j.aam.2017.05.008`.

10 Max Dehn. Über unendliche diskontinuierliche Gruppen. *Math. Ann.*, 71:116–144, 1911.

11 Pierre Gillibert. Personal Communication.

12 Pierre Gillibert. The finiteness problem for automaton semigroups is undecidable. *International Journal of Algebra and Computation*, 24(01):1–9, 2014. `doi:10.1142/S0218196714500015`.

13 Pierre Gillibert. An automaton group with undecidable order and Engel problems. *Journal of Algebra*, 497:363–392, 2018. `doi:10.1016/j.jalgebra.2017.11.049`.

14 Rostislav Grigorchuk and Igor Pak. Groups of intermediate growth: an introduction. *L'Enseignement Mathématique*, 54:251–272, 2008.

15 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.

16 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.

17 Volodymyr V. Nekrashevych. *Self-similar groups*, volume 117 of *Mathematical Surveys and Monographs*. American Mathematical Society, Providence, RI, 2005. `doi:10.1090/surv/117`.

18 Petr S. Novikov. On the algorithmic unsolvability of the word problem in group theory. *Trudy Mat. Inst. Steklov*, pages 1–143, 1955. In Russian.

19 Christos M. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

20 David Robinson. *Parallel Algorithms for Group Word Problems*. PhD thesis, University of California, San Diego, 1993.

21 Richard Edwin Stearns, Juris Hartmanis, and Philip M. Lewis. Hierarchies of memory limited computations. In *6th Annual Symposium on Switching Circuit Theory and Logical Design (SWCT 1965)*, pages 179–190. IEEE, 1965.

22 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.

23 Zoran Šunić and Enric Ventura. The conjugacy problem in automaton groups is not solvable. *Journal of Algebra*, 364:148–154, 2012. `doi:10.1016/j.jalgebra.2012.04.014`.

24 Heribert Vollmer. *Introduction to Circuit Complexity*. Springer, Berlin, 1999.

25 Mariya Vorobets and Yaroslav Vorobets. On a free group of transformations defined by an automaton. *Geometriae Dedicata*, 124:237–249, 2007. `doi:10.1007/s10711-006-9060-5`.