Sweep Complexity Revisited*

Szilárd Zsolt Fazekas[1][0001−5319−0395] and Robert Mercaş[2][0001−6034−433X]

[1] Akita University, Department of Mathematical Science and Electrical-Electronic-Computer Engineering
szilard.fazekas@ie.akita-u.ac.jp
[2] Loughborough University, Department of Computer Science
R.G.Mercas@lboro.ac.uk

Abstract. We study the sweep complexity of DFA in one-way jumping mode answering several questions posed earlier. This measure is the number of times in the worst case that such machines have to return to the beginning of their input after having skipped some of the symbols. The class of languages accepted by these machines strictly includes the regular class and constant sweep complexity allows exactly the acceptance of regular languages. However, we show that there exist machines with higher than constant complexity still only accepting regular languages and that in general the sweep complexity of an automaton does not distinguish between accepting regular and non-regular languages. We establish separation results for asymptotic classes defined by this complexity measure and give a surprising exponential/logarithmic relation between factors of certain inputs which can be verified by such machines.

Keywords: automata · deterministic · one-way jumping · sweep complexity.

1 Introduction

In roughly the last three decades several non-classical models of automata have been introduced to study the effect of processing inputs with simple machines in a non-sequential way. Such models include restarting automata [10], jumping automata [12], input revolving automata [4] and automata with translucent letters [13]. However, these models are either strictly more powerful or accept a class incomparable with the regular one.

One-way jumping finite automata (OWJFA) were introduced [5] to study the power of deterministic finite automata (DFA) performing non-sequential processing without completely discarding structural information about the inputs á la jumping automata. The resulting model is in a sense a minimal extension of finite automata. Machines are specified in exactly the same way as DFA allowing partial transition functions. The only change is the behaviour of the machine when encountering a letter for which the current state has no outgoing transition defined. In the classical case such inputs are rejected, but in one-way jumping

* The first author was supported by JSPS Kakenhi Grant JPxxxxxxxxxxxxxxxx.

mode the letters are skipped temporarily to be processed later. The relative order of the skipped symbols is maintained, and the automaton moves back to the beginning after each pass (called *sweeps* here), seeing only the symbols previously skipped. Therefore one can also view this model as a DFA with an input tape which works as a restricted queue, or one that reads and erases symbols from a circular tape always jumping clockwise to the nearest letter for which it has a defined transition from the current state. When the transition function is complete, no symbols are skipped, so the machine behaves as ordinary DFA, which means that the class of languages accepted by DFA in one-way jumping mode trivially includes all regular languages.

Various properties of the accepted language class [1] and the status of fundamental decidability questions have been settled [2]. More powerful machines with this new processing mode have also been investigated, such as nondeterministic finite automata [3, 6], two-way finite automata [7], pushdown automata and linear bounded automata [6]. While the language classes defined by the models have no nontrivial closure properties under usual language operations, the accepting power and decidability issues raised some intriguing problems.

Except for linear bounded automata, the machine models mentioned above become more powerful when they are allowed to jump to the nearest symbol readable in the current state, which is not surprising. However, it has proved challenging to get a clear picture of just how powerful the new processing mode is, even in the simplest case when one starts from DFA. Such automata can accept all regular languages and the language class defined by them is incomparable with the context-free class, but included in the context-sensitive class and in DTIME(n^2) [1]. The separation results make use of combinations of a handful of regular languages together with a very simple type of non-regular languages which contain words having letter counts in a certain ratio, e.g., the frequently used $L_{ab} = \{w \in \{a, b\}^* \mid w \text{ contains as many } a\text{'s as } b\text{'s}\}$ accepted by the machine $\mathcal{A}$ in Fig 1. While this was enough to establish virtually all separations of interest, it left a significant gap in our understanding of the model: can such machines accept any ('interesting') non-regular languages apart from the ones which establish linear relationships among letter counts?

To try to answer the question above and to get closer to a decision procedure which would tell whether a given machine accepts a regular language, we continue the investigation of sweep complexity of DFA in one-way jumping mode, started recently [8]. Sweep count can also be viewed as a measure of non-regular resources used by a machine posing the natural question of how much of this resource is needed to be able to accept non-regular languages? It has been shown that constant sweep complexity does not increase the accepting power of the machines [9]. It was conjectured that, in fact, any automaton with higher than constant sweep complexity accepts a non-regular language. In Section 3 we refute that conjecture by exhibiting a small DFA accepting a regular language while processing some inputs of length n in $\Omega(\log n)$ sweeps. We also show that there is no non-trivial upper bound on the sweep complexity of regular languages, that is, there are machines with linear complexity accepting regular languages.

A natural question regarding the new complexity measure is whether there exists a meaningful hierarchy which does not collapse to the extremes of $\mathcal{O}(1)$ and $\mathcal{O}(n)$. The aforementioned example shows that automata with logarithmic complexity exist, which answers another question posed earlier. Furthermore, following the line of computational complexity theory, we set out to explore whether the language classes defined through asymptotic complexity form a true hierarchy, that is whether there are languages which can be accepted by a machine with $\mathcal{O}(f(n))$ complexity but not by any with $o(f(n))$ complexity, for various functions $f(n)$. In Section 4 we demonstrate that such a hierarchy exists by presenting languages with $\Theta(\log n)$ and $\Theta(n)$ sweep complexity, respectively.

Finally we mention that sweep complexity as an idea has been studied in other contexts, too: an interesting and thorough investigation of a similar flavor established infinite hierarchies in terms of sweep count for iterated uniform finite transducers [11], although that model is significantly more powerful than ours, so the techniques used there do not translate here as far as we can tell.

2 Preliminaries

We consider words over a finite alphabet, e.g., $\Sigma = \{a, b\}$. The set of all words over Σ is Σ^* which includes the empty word ε.

A *DFA* is a quintuple $M = (Q, \Sigma, R, \mathbf{s}, F)$, where Q is the finite set of states, Σ is the finite input alphabet, $\Sigma \cap Q = \emptyset$, $R : Q \times \Sigma \to Q$ is the transition function, $\mathbf{s} \in Q$ is the start state, and $F \subseteq Q$ is the set of final states. Elements of R are referred to as (transition) rules of M and we write $\mathbf{p}y \to \mathbf{q} \in R$ instead of $R(\mathbf{p}, y) = \mathbf{q}$. A configuration of M is a string in $Q \times \Sigma^*$.

A DFA transitions from a configuration $\mathbf{p}w$ to a configuration $\mathbf{q}w'$ if $w = aw'$ and $\mathbf{p}a \to \mathbf{q} \in R$, with $\mathbf{p}, \mathbf{q} \in Q$, $w, w' \in \Sigma^*$ and $a \in \Sigma$. By extending the meaning of $\to$ we denote this by $\mathbf{p}w \to \mathbf{q}w'$ and the reflexive and transitive closure of $\to$ by $\to^*$. A word w is *accepted* by a DFA M if there exists $\mathbf{f} \in F$, such that $\mathbf{s}w \to^* \mathbf{f}$. The language accepted by M is $\{w \in \Sigma^* \mid \exists \mathbf{f} \in F : \mathbf{s}w \to^* \mathbf{f}\}$.

One-way jumping automata

The *one-way jumping relation* (denoted by $\circlearrowright$) between configurations from $Q\Sigma^*$, was originally defined in [5]. Here we follow the slightly different definition of [8] which does not change the accepting power of the model, but is more convenient.

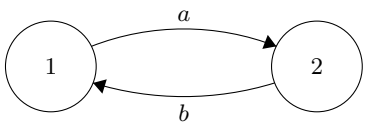

Fig. 1: The only two-state ROWJFA satisfying Lemma 1

position :	0	1	2	3	4	5	6
input	**a**	**d**	**c**	**b**	**c**	**b**	**a**
after sweep 1	ε	**d**	**c**	**b**	**c**	**b**	ε
after sweep 2	ε	**d**	**c**	ε	**c**	ε	ε
after sweep 3	ε	**d**	ε	ε	ε	ε	ε
after sweep 4	ε	ε	ε	ε	ε	ε	ε

Fig. 2: The computation table for *adcbcba* by the machine in Example 1.

A tuple $M = (Q, \Sigma, R, \mathbf{s}, F)$ representing a *deterministic right one-way jumping automaton* (ROWJFA) is defined the same way as a DFA, where the configurations are also elements of the set $Q \times \Sigma^*$. Let $\Sigma_p = \{b \in \Sigma \mid \exists \mathbf{q} \in Q$ such that $\mathbf{p}b \to \mathbf{q} \in R\}$ be the set of all of the letters from Σ for which we have a transition defined from state $\mathbf{p}$. A jumping transition (or jump, for short), denoted $\circlearrowright$, is defined between configurations $\mathbf{p}ax$ and $\mathbf{p}xa$ if state $\mathbf{p}$ cannot read the letter a, formally:

$$\mathbf{p}ax \circlearrowright \mathbf{p}xa, \text{ if } a \in \Sigma \setminus \Sigma_p.$$

A ROWJFA can transition from configuration $\mathbf{p}ax$ to configuration $\mathbf{q}y$, which we denote by $\mathbf{p}ax \vdash \mathbf{q}y$, if

(*i*) $\mathbf{p}ax \to \mathbf{q}y$, where $x = y$ and $\mathbf{p}a \to \mathbf{q} \in R$, as defined earlier, or
(*ii*) $\mathbf{p}ax \circlearrowright \mathbf{p}xa$, when $a \in \Sigma \setminus \Sigma_p, \mathbf{p} = \mathbf{q}$ and $xa = y$.

A word w is accepted by M if $\mathbf{s}w \vdash^* \mathbf{f}$. The language accepted by M is defined by $L(M) = \{x \in \Sigma^* \mid \exists \mathbf{f} \in F : \mathbf{s}x \vdash^* \mathbf{f}\}$.

While some texts define DFA having complete transition functions, our DFAs allow partially defined ones. Indeed, the pairs $(\mathbf{p}, a) \in Q \times \Sigma$ for which no transition is defined enable the ROWJFA to perform a jump as opposed to rejecting the input as a DFA would. Hence, a ROWJFA with a complete transition function is just a DFA.

Sweeps are contiguous sequences of transitions on a given input, consisting of the steps from reading or jumping over the leftmost remaining input letter to reading or jumping over the rightmost one. If a position is jumped over, then the input symbol in that position is processed in a later sweep. The number of sweeps needed to process the whole input is the number of times the automaton reaches the last position of the original input word or, equivalently, one more than the maximum number of times any position is jumped over.

For an intuitive picture of sweeps, consider the computation of a ROWJFA M on input w as a table with rows representing the k sweeps needed to process w and columns representing positions in the input word. Cell i, j in the table contains either a letter or a symbol representing that the letter has been *read*, e.g., ε. Once a letter has been marked *read* and erased it stays that way, so each column is a word of the form $a^\ell \varepsilon^{k-\ell}$ $(= a^\ell)$ for some $a \in \Sigma$ and $1 \leq \ell \leq k$.

Example 1. Consider the automaton M_1 in Fig. 3 and the input *adcbcba*, processed in the order *aabbccd*. The ROWJFA jumps over the letter d three times before processing it, hence the number of sweeps is four. Moreover, its computation table is described in Fig. 2.

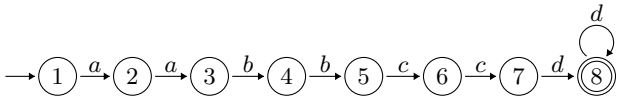

Fig. 3: ROWJFA M_1 accepting all w with $|w|_a = |w|_b = |w|_c = 2$ and $|w|_d \geq 1$.

In order to be able to analyze quantitatively the boundary between regular and non-regular languages accepted by the one-way jumping model, as well as how non-regular the accepted language is, when it is the case, the following complexity measure was proposed [8], which gives us the number of sweeps performed by a machine in the 'worst case' for an input of length n.

Let $w \in L(M)$, and let

$$\mathbf{p}_0 w \models \mathbf{p}_1 w_1 \models \mathbf{p}_2 w_2 \models \ldots \models \mathbf{p}_m, \text{ where } \mathbf{p}_0 = s \text{ and } \mathbf{p}_m \in F,$$

be the computation of M on the input w. Sweep 1 consists of $\mathbf{p}_0 w \models^* \mathbf{p}_{|w|} w_{|w|}$, and we say that sweep 1 ends in configuration $\mathbf{p}_{|w|} w_{|w|}$. Then, for all $i \geq 1$, if sweep i ends in configuration $\mathbf{p}_{si} w_{si}$, then sweep $i+1$ is the sequence of configurations $\mathbf{p}_{si} w_{si} \models^* \mathbf{p}_{si+|w_{si}|} w_{si+|w_{si}|}$. The last sweep ends in configuration $\mathbf{p}_m$, that is, when all input symbols have been read. We define

$$E(M, w) = \{\text{the number of sweeps performed by } M \text{ on } w\}.$$

When $w \notin L(M)$, then we set[3] $E(M, w) = 0$. The *sweep complexity* of a machine M is a function $sc_M : \mathbb{N} \to \mathbb{N}$, with $sc_M(n)$ being the maximum number of sweeps M makes on processing inputs $w \in L(M)$ of length n, formally:

$$sc_M(n) = \max\{E(M, w) \mid w \in \Sigma^n\}.$$

In a certain sense the most "non-regular" word of each length is considered. With this in mind, we can define complexity classes in the usual manner: the class SWEEP($f(n)$) consists of languages accepted by some one-way jumping machine with sweep complexity $\mathcal{O}(f(n))$.

Each machine considered up to the point when the above measures where introduced [8] had either constant or linear sweep complexity, so it seemed that there is a gap between. Moreover, the examples with linear complexity accepted non-regular languages, while as the theorem below states, the constant complexity languages are exactly the regular languages.

Theorem 1. *[9] ROWJFA with $\mathcal{O}(1)$ sweep complexity accept regular languages.*

The sufficient condition above was conjectured to be also necessary for regularity in general, evidenced by the known examples at that point.

Next, we investigate the apparent gap between constant and linear complexities and show that the presumed condition above is not necessary for regularity. Our search for machines with non-constant sweep complexity is directed by the following structural lemma, which says that such machines need to have two 'complementary deficient states' in a cycle.

Lemma 1. *[8] If a ROWJFA has sweep complexity $\omega(1)$ then its state diagram has a closed walk with states $\mathbf{p}$ and $\mathbf{q}$, such that $\mathbf{p}au \to^* \mathbf{q}bv \to^* \mathbf{p}$ for $a, b \in \Sigma$, $u, v \in \Sigma^*$ and $\mathbf{p}$ has no transition defined for b, while $\mathbf{q}$ has no transition for a.*

[3] Another option is to count the sweeps of non-accepting computations, too. We follow here the definition of [8], as this allows straightforward generalization to NFAs later on. Moreover, if non-accepting computations count, it is trivial to construct machines accepting regular languages with any implementable sweep complexity.

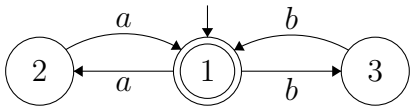

Fig. 4: A ROWJFA accepting $\{w \in \{a,b\}^* \mid |w|_a \equiv 0 \bmod 2, |w|_b \equiv 0 \bmod 2\}$ with sweep complexity $\Theta(\log n)$.

3 Regular languages with non-constant sweep complexity

In this section we show that there is no sweep complexity separation between regular and non-regular languages by exhibiting automata which accept regular languages while requiring superconstant number of sweeps.

Consider first the automaton $\mathcal{B}$ with states $\{\mathbf{1}, \mathbf{2}, \mathbf{3}\}$ where $\mathbf{1}$ is initial and final, and transitions are $\{\mathbf{1}a \to \mathbf{2}, \mathbf{2}a \to \mathbf{2}, \mathbf{1}b \to \mathbf{3}, \mathbf{3}b \to \mathbf{1}\}$, described in Fig. 4.

Proposition 1. *$L(\mathcal{B})$ is regular.*

Proof. We claim that $L(\mathcal{B}) = \{w \in \{a,b\}^* \mid |w|_a \equiv 0 \bmod 2, |w|_b \equiv 0 \bmod 2\}$. This is obviously a regular language (i.e., Fig. 7 where **00** is the final state).

The computation for a word w is rejected if it finishes in either $\mathbf{2}$ or $\mathbf{3}$. However, the only time that the machine ends up in state $\mathbf{2}$ is when it read an odd number of a's, and, similarly, it ends in $\mathbf{3}$ when it read an odd number of b's. Since both of these types of words are rejected, we conclude. □

Theorem 2. *The sweep complexity of $\mathcal{B}$ is $\Theta(\log n)$.*

Proof. Firstly, observe that in any sweep, while in $\mathbf{1}$ or $\mathbf{2}$, the automata reduces any block of a's to at most a single a, and, similarly, while in $\mathbf{1}$ or $\mathbf{3}$, the automata reduces any block of b's to at most a single b. Thus, the number of sweeps necessary to process a word w consisting of n unary blocks is never higher than that of processing the word $(ab)^n$. Now consider the inputs $(ab)^n$ (and $(ba)^n$). Starting with the first b (respectively, a) every third symbol is jumped while the rest is read. This means that from an arbitrary word with k unary blocks, after one sweep at most $\frac{k}{3} + 2$ remain. This immediately gives us that the machine makes at most logarithmically many sweeps. As for the other side, consider an input $w = (ab)^{6^k}$. Per the previous argument, after $i \leq$ sweeps the remaining input will be $(ab)^{\frac{6^k}{3^i}}$ or $(ba)^{\frac{6^k}{3^i}}$ depending on the parity of i, so the number of sweeps is at least $\log_6 |\frac{w}{2}| = k$. Eventually, the input will be accepted according to Proposition 1, so the sweep complexity of $\mathcal{B}$ is also $\Omega(\log n)$. □

The above results showcase the existence of ROWJFAs that accept regular languages while performing a logarithmic number of sweeps. Next we construct of a ROWJFA that accepts a regular language while requiring a linear number of sweeps in the worst case. Consider the automaton $\mathcal{C}$ in Fig. 5 defined as

$$\mathcal{C} = \{\{\mathbf{A0}, \mathbf{A1}, \mathbf{A2}, \mathbf{A3}, \mathbf{B1}, \mathbf{B2}, \mathbf{B3}\}, \{a,b\}, R, \mathbf{A0}, \{\mathbf{B1}\}\},$$

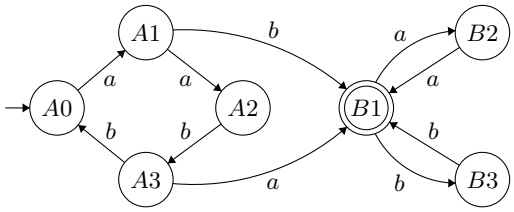

Fig. 5: A ROWJFA accepting $\{w \in \{a,b\}^* \mid |w|_a \equiv 1 \mod 2, |w|_b \equiv 1 \mod 2\}$ with sweep complexity $\Theta(n)$.

where the transitions from R are given by the edges in the figure.

Proposition 2. *The sweep complexity of $\mathcal{C}$ is $\Theta(n)$.*

Proof. To see that the complexity is $\Omega(n)$, consider the word $a^{2n+1}b^{2n+1}$, for $n > 1$. In this case, from **A0** we go first to **A2** where we jump over all the remaining a's, then we move back to **A0** where we jump over all the remaining b's, and we are left with $a^{2n-1}b^{2n-1}$ to process. After the nth sweep, we are only left with ab to process, which takes us from **A0** to **B1**, and we accept.

For the $\mathcal{O}(n)$ complexity, observe that the above computation is indeed the longest possible. Once we reach **B1** we either accept or reject a word in at most $\mathcal{O}(\log n)$ sweeps, same as in Theorem 2. Of course, this part also directly follows from the fact that all ROWJFA process their inputs in $\mathcal{O}(n)$ sweeps. □

Proposition 3. *$L(\mathcal{C})$ is regular.*

Proof. We show that $L(\mathcal{C}) = \{w \in \{a,b\}^* \mid |w|_a \equiv 1 \bmod 2, |w|_b \equiv 1 \bmod 2\}$. This is obviously a regular language (i.e., Fig. 7 where **11** is the final state).

To show that indeed $L(\mathcal{C})$ is the language accepting every binary word that has odd number of a's and b's, first note that the right hand side automaton consisting only of the **B**-labelled states, accepts every language that has an even number of a's and b's, as shown by Proposition 2.

To reach **B1** we have to read exactly one a and one b starting from either **A0** or **A2**. Since from the start state **A0** we can reach **A0** or **A2** by processing an even number of a's and b's, possibly with jumps, our conclusion follows. □

As a consequence of Propositions 2 and 3, we know that the class of regular languages has no upper bound in terms of sweep complexity. The left hand cycle in the automata $\mathcal{C}$ described in Fig. 5 also showcases that while the conditions from Lemma 1 are necessary for non-regularity (as it requires superconstant complexity), they are not sufficient.

4 Separation results for the language classes SWEEP($\log n$) and SWEEP(n)

Consider the prolongable morphism $\varphi(a) = abab$, $\varphi(b) = b$ starting from the word ab. We get $\varphi(ab) = ababb$, $\varphi^2(ab) = \varphi(ababb) = ababbababbb$, etc. The

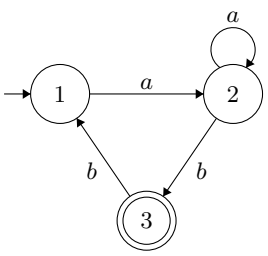

Fig. 6: The ROWJFA $\mathcal{D}$ accepts a non-regular language with $\Theta(\log n)$ sweeps.

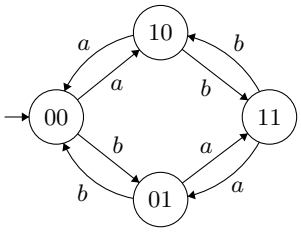

Fig. 7: DFA accepting words with even (for **00** final state) or odd (for **11** final state) number of a's and b's.

infinite word $\phi = \lim_{n\to\infty} \varphi^n(ab) = ababbababbb\ldots$ is a fixed point of ϕ. It is easy to see that in ϕ all a's stand alone, that is, we never have blocks of a's longer than 1, and the lengths of the blocks of b's is $1, 2, 1, 3$, and so on[4]. Each block of b's gets longer by one on each application of φ, because of the a preceding it. Moreover, when applying φ, each a introduces a new block of b's of length 1, and the number of a's doubles. A simple inductive argument shows that the last block of b's in $\varphi^n(ab)$ has length n, and is preceded by 2^{n-1} occurrences of a's separated by blocks of b's.

Lemma 2. *Consider the morphism $\varphi : \{a, b\}^* \to \{a, b\}^*$ given by $\varphi(a) = abab$, $\varphi(b) = b$. The following statements hold for any $n \geq 1$:*

(i) $\varphi^n(ab) \in (ababb^+)^+$;
(ii) *if* $\varphi^n(ab) = ab^{k_1} \cdots ab^{k_m}$, *then* $\varphi^{n+1}(ab) = abab^{k_1+1}abab^{k_2+1} \cdots abab^{k_m+1}$;
(iii) $\varphi^n(ab) = ab^{k_1} \cdots ab^{k_m}$, *where* $m = 2^n$, $k_m = n + 1$ *and* $k_{2i-1} = 1$ *for all* $i \in \{1, \ldots, 2^{n-1}\}$.

Proof. When $n = 1$, then $\varphi(ab) = ababb$, so for $n = 1$ all three claims hold. Suppose they hold for n, so $\varphi^n(ab) = abab^{k_1}abab^{k_2} \cdots abab^{k_m}$ with the conditions of (iii) satisfied by m and $k_1, \ldots, k_m$. Then,

$$\begin{aligned}\varphi^{n+1}(ab) &= \varphi(abab^{k_1} \cdots abab^{k_m}) = \varphi(ab)\varphi(ab^{k_1}) \cdots \varphi(ab)\varphi(ab^{k_m}) \\ &= (ababb)(abab^{k_1+1}) \cdots (ababb)(abab^{k_m+1})\end{aligned}$$

From this we can immediately conclude that (i) and (ii) hold for any $n \geq 1$. Further, by the equation above we have $\varphi^{n+1}(ab) = ab^{\ell_1} \cdots ab^{\ell_{m'}}$ with $m' = 2m = 2 \cdot 2^n = 2^{n+1}$. Finally, because of (ii) we also get that $\ell_{m'} = k_m + 1 = n + 2$ and $\ell_{2i-1} = 1$ for all $i \in \{1, \ldots, 2^n\}$. □

In what follows we analyze the language accepted by the automaton $\mathcal{D} = (\{\mathbf{1}, \mathbf{2}, \mathbf{3}\}, \{a, b\}, \{\mathbf{1}a \to \mathbf{2}, \mathbf{2}a \to \mathbf{2}, \mathbf{2}b \to \mathbf{3}, \mathbf{3}b \to \mathbf{1}\}, \mathbf{1}, \{\mathbf{3}\})$, described in Fig. 6.

[4] The sequence given by the lengths of b blocks is A001511 in OEIS; its most relevant characterization for us is that $a(n) - 1$ is the number of trailing zeros in the binary expansion of n, since this means that $a(n) - 1$ is $\log n$ for powers of 2

Lemma 3. *For any $n \geq 0$, the ROWJFA $\mathcal{D}$ accepts $\varphi^n(ab)$ in $n+1$ sweeps.*

Proof. We show that the machine accepts $\varphi^n(ab)$, for any $n \geq 0$. From state **1** after reading/jumping through a factor of the form $ababb^+$ the automaton gets back to state **1**. In fact, $\mathbf{1}abab^k w \vdash \mathbf{1}wab^{k-1}$, for any $k \geq 1$, so in one sweep the factor $abab^k$ is reduced to ab^{k-1}. From Lemma 2 we can see that we can write $\varphi^{n+1}(ab) = abab^{k_1+1}abab^{k_2+1} \cdots abab^{k_m+1}$, which means that one sweep of $\mathcal{D}$ acts as the inverse of φ on those words when starting from state **1**, that is,

$$\mathbf{1}\varphi^{n+1}(ab) = \mathbf{1}abab^{k_1+1}abab^{k_2+1} \cdots abab^{k_m+1} \vdash^* \mathbf{1}ab^{k_1}ab^{k_2} \cdots ab^{k_m} = \mathbf{1}\varphi^n.$$

This means that in n sweeps the machine reduces $\varphi^n(ab)$ to $\varphi^0(ab)$. Finally, for $n = 0$, we have $\varphi^0(ab) = ab$, which is accepted by $\mathcal{D}$ in a single sweep. □

Lemma 4. *The ROWJFA $\mathcal{D}$ accepts a non-regular language.*

Proof. By Lemma 3 we know that for any n the machine accepts $\varphi^n(ab)$, which means that for arbitrarily long unary factors consisting of b's, there is some word in $L(\mathcal{D})$ having such a factor as a suffix. Our strategy is to first establish a non-linear relation between the length of those unary factors and the length of the preceding factors in all words accepted by $\mathcal{D}$. Then, by a pumping argument we show that a classical finite automaton cannot verify such a non-linear relation, therefore $L(\mathcal{D})$ cannot be regular.

Claim 1. Words of the form wb^n are only accepted if $|w| \in \Omega(2^{\frac{n}{2}})$.

Proof of Claim 1: In any sweep, any block of a's which the automaton starts to read is read and erased completely through a sequence of transitions $\mathbf{1}a^k bu \to^* \mathbf{2}bu$. For the automaton to jump over a block of a's, it needs to arrive to its start in state **3**. Then it jumps over it to the next b, after which it starts and reads completely the following block of a's, as described earlier. This means that the machine can never jump over two consecutive blocks of a's. From here we get that if at the beginning of the sweep the number of a blocks was ℓ, then after the sweep it is at most $\lfloor \frac{\ell}{2} \rfloor + 1$.

Furthermore, in each sweep, each block of b's is reduced by at most 2. This means that the automaton needs at least $\frac{n}{2}$ sweeps to read a block b^n, in each of which it reduces the number of a blocks by half (or more). Thus we can conclude that in order to accept a word with a suffix b^n, we have to start out with at least $2^{\frac{n}{2}} + 1$ blocks of a's preceding it. ∇

Claim 2. No finite automaton can accept $L(\mathcal{D})$.

Proof of Claim 2: Suppose the opposite, i.e., that there exists some complete DFA $\mathcal{F}$ having N states such that $L(\mathcal{F}) = L(\mathcal{D})$. We know that there are words in the language with arbitrarily long suffixes of b's, so there is a $wb^m \in L(\mathcal{F})$ for some word w and exponent $m > N$. By a usual pumping argument, this means that there exists some ℓ with $0 < \ell < N$ such that $wb^{m+i\cdot\ell} \in L(\mathcal{F})$ for any $i \geq 0$. However, for a large enough i this contradicts Claim 1, as the block of b's can outgrow any upper bound in terms of the length of $|w|$. ∇

Our result follows as a result of Claims 1 and 2. □

Lemma 5. *The sweep complexity of $\mathcal{D}$ is $\Theta(\log n)$.*

Proof. By Lemma 3 we have that the sweep complexity of $\mathcal{D}$ is $\Omega(\log n)$, so what remains to show is that it is also $\mathcal{O}(\log n)$.

We first note that within a sweep all blocks of a's separated by bb are fully processed (including any prefix of a's), while for any symbols a that were jumped, the entire block that they were part of it was jumped. Following the argument in the proof of Claim 1 of Lemma 4, in each sweep the number of blocks of a's is reduced by at least half, which means that after $\mathcal{O}(\log n)$ sweeps there are no more blocks of a on the tape. Then, the machine either accepts in one sweep or it rejects the input. This leads to our conclusion. □

The results of Lemmas 4 and 5 mean that we have separation between SWEEP(1) and SWEEP($\log n$).

Theorem 3. *SWEEP*$(1) \subsetneq$ *SWEEP*$(\log n)$

Proof. Lemma 5 says $L(\mathcal{D}) \in$ SWEEP($\log n$). By Theorem 1 we know that SWEEP(1) is included in the class of regular languages. Finally, by Lemma 4 we have that $L(\mathcal{D})$ is not regular which means that $L(D) \notin$ SWEEP(1). □

Lemma 6. *Any automaton which accepts* $L_{ab} = \{w \in \{a,b\}^* \mid |w|_a = |w|_b\}$ *has sweep complexity* $\Theta(n)$.

Proof. We know that every machine has sweep complexity $\mathcal{O}(n)$, so it is enough to show that it is not possible to accept L_{ab} with sublinear sweep complexity. For that we assume that such an automaton, say $\mathcal{F} = (Q, \Sigma, R, s, F)$ exists, and derive a contradiction.

If $\mathcal{F}$ had linear sweep complexity, then it could have computations on infinitely many inputs in which all sweeps process a constant number of symbols. However, with sublinear complexity we get that for any constant C and for all long enough inputs $w \in L_{ab}$, during the processing of w at least one sweep reads more than C symbols. We also know that $a^n b^n \in L_{ab}$ for any $n \geq 0$. Let $C = 2|Q|$ where $|Q|$ is the number of states of $\mathcal{F}$ and consider an input $w = a^m b^m$ with m large enough that the machine reads more than C symbols in some sweep while processing w. The remaining input at the beginning of said sweep is $a^k b^\ell$ for some k, ℓ such that $k + \ell > C$. During the sweep the machine reads $a^{k'} b^{\ell'}$ where $k' + \ell' > C$. This means that either $k' > |Q|$ or $\ell' > |Q|$. Without loss of generality we can assume $k' > |Q|$. This gives us that while reading $a^{k'}$ the automaton must visit some state $\mathbf{p}$ at least twice while reading only a's, so we get that $\mathbf{p}a^r \to^* \mathbf{p}$ for some $r > 0$. But then, by a usual pumping argument the machine also needs to accept $a^{n+r} b^n \notin L_{ab}$ contradicting our assumption that $L(\mathcal{F}) = L_{ab}$ and concluding the proof. □

Theorem 4. *For any* $f : \mathbb{N} \to \mathbb{N}$ *with* $f(n) \in o(n)$ *we have SWEEP*$(f(n)) \subsetneq$ *SWEEP*(n).

Proof. By Lemma 6 we know that $L_{ab} \notin$ SWEEP($f(n)$) for any sublinear function $f(n)$. The two-state automaton $\mathcal{A}$ accepts the language with sweep complexity $\Theta(n)$. This is easy to see when considering the worst-case inputs of the form $a^n b^n$ for $n \geq 0$. □

5 Concluding remarks

Apart from the complexity considerations listed below we think the proof of Lemma 4 contains a detail worth emphasizing: the automaton can verify a logarithmic/exponential relation between two factors of suitably chosen inputs! We found this very surprising since we still basically deal with DFA which cannot store information and cannot 'choose' which symbols to read or jump over[5].

We presented automata for all pairings of regular and non-regular languages with logarithmic and linear worst case sweep complexity. This way we disproved the conjecture on the constant sweep requirement for regularity [9] and answered several questions regarding sweep complexity posed in [8]:

1. Is the language of each machine with $\omega(1)$ complexity non-regular? NO, by Section 3.
2. Is there a machine with sweep complexity between constant and linear, that is, $\omega(1)$ and $o(n)$? YES, by Theorem 2 (and Lemma 5).
3. Is there a *language* with sweep complexity between constant and linear, that is, all machines accepting it have superconstant complexity and at least one has sublinear? YES, by Theorem 3.
4. Is there an upper bound in terms of sweep complexity on machines accepting regular languages? NO, by Propositions 2 and 3.
5. Are machines less complex in the case of binary alphabets, given that the complementary deficient pairs of Lemma 1 are predetermined? NO, illustrated by the fact that all results have been obtained over a binary alphabet.

These coarser forms of Questions 2 and 3 have been answered here, but for a complete picture one would want to know whether there exist machines with arbitrary (constructible) sublinear complexity and its equivalent for languages. The most obvious choices for such a study would probably be complexities $\Theta(\log^k n)$ and $\Theta(n^\epsilon)$, for constants $k > 1$ and $\epsilon < 1$. Another angle related to Question 4 is to study the lower bound of non-regularity: logarithmic complexity can produce non-regular languages, but can we do it with less of this 'non-regular' resource? In the case of Question 5, our answer may be refined, as there may by some sublinear $f(n)$ such that the machines of $\Theta(f(n))$ complexity all accept regular or all accept non-regular languages, although we have not seen anything that indicates such perplexing behaviour.

Another interesting direction relates to our original motivation in looking at the complexity of these automata, deciding regularity. The question more generally becomes, is it decidable given a machine or language and a function $f(n)$, whether the machine/language has $\Theta(f(n))$ complexity (or its one-sided variants with $\mathcal{O}$ and Ω)? We suspect that the answer is yes at least in the case of constant and linear functions but have no idea about the logarithmic and more complicated cases.

[5] Iterated uniform finite transducers can also verify such relationships, albeit their computing power is much stronger. [11]

References

1. Beier, S., Holzer, M.: Properties of right one-way jumping finite automata. Theoretical Computer Science **798**, 78 – 94 (2019)
2. Beier, S., Holzer, M.: Decidability of right one-way jumping finite automata. International Journal of Foundations of Computer Science **31**(6), 805–825 (2020)
3. Beier, S., Holzer, M.: Nondeterministic right one-way jumping finite automata. Information and Computation **284**, 104687 (2022), selected papers from DCFS 2019
4. Bensch, S., Bordihn, H., Holzer, M., Kutrib, M.: On input-revolving deterministic and nondeterministic finite automata. Information and Computation **207**(11), 1140–1155 (2009)
5. Chigahara, H., Fazekas, S.Z., Yamamura, A.: One-way jumping finite automata. International Journal of Foundations of Computer Science **27**(3), 391–405 (2016)
6. Fazekas, S.Z., Hoshi, K., Yamamura, A.: The effect of jumping modes on various automata models. Natural Computing (2021)
7. Fazekas, S.Z., Hoshi, K., Yamamura, A.: Two-way deterministic automata with jumping mode. Theoretical Computer Science **864**, 92–102 (2021)
8. Fazekas, S.Z., Mercaş, R., Wu, O.: Complexities for jumps and sweeps. J. Autom. Lang. Comb. **27**(1-3), 131–149 (2022)
9. Fazekas, S.Z., Yamamura, A.: On regular languages accepted by one-way jumping finite automata. In: 8th Workshop on Non-Classical Models of Automata and Applications, Short Papers. pp. 7–14 (2016)
10. Jančar, P., Mráz, F., Plátek, M., Vogel, J.: Restarting automata. In: Reichel, H. (ed.) Fundamentals of Computation Theory. pp. 283–292. Springer Berlin Heidelberg, Berlin, Heidelberg (1995)
11. Kutrib, M., Malcher, A., Mereghetti, C., Palano, B.: Descriptional complexity of iterated uniform finite-state transducers. Information and Computation **284**, 104691 (2022)
12. Meduna, A., Zemek, P.: Jumping finite automata. International Journal of Foundations of Computer Science **23**(7), 1555–1578 (2012)
13. Nagy, B., Otto, F.: Finite-state acceptors with translucent letters. In: BILC 2011 - 1st International Workshop on AI Methods for Interdisciplinary Research in Language and Biology, ICAART 2011. pp. 3–13 (2011)

6 Appendix

6.1 Counting the sweep complexity of rejected words

We defined the sweep complexity of a machine as the maximum number of sweeps over all *accepted* words of length n. One could also take into account the sweep count of rejected words. However, this would allow a trick in a sense to 'artificially' increase the sweep complexity of machines with complexity $o(n)$ without affecting regularity. Let A be a machine accepting a regular language and B a non-regular language with sweep complexities $f(n)$ and $g(n)$, respectively, such that $f(n) \in o(g(n))$. The we can easily construct a ROWJFA accepting $aL(A)$ with sweep complexity $g(n)$ by adding a new initial state from which reading a takes us to the initial state of A while reading b takes us to the initial state of B. We set all states of B non-final and this way we get that on inputs starting with b the machine performs B's computations but never accepts anything. Moreover, $aL(A)$ is regular if and only if $L(A)$ was.

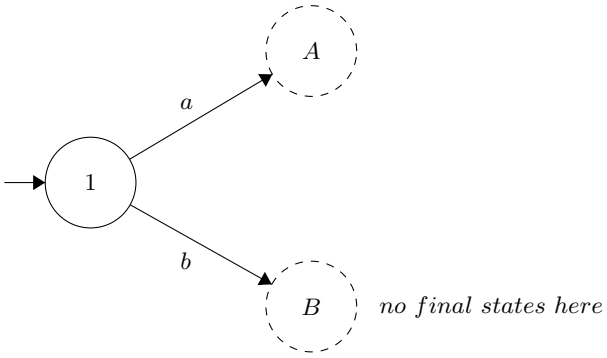

Fig. 8: Artificially increasing the complexity by adding non-functional states to the automaton.

6.2 Classical DFA accepting the languages referenced in Propositions 1 and 3

The language accepted by the (minimal) DFA in Fig. 7, where **00** is a final state, is $L(\mathcal{C}) = \{w \in \{a,b\}^* \mid |w|_a \equiv 0 \mod 2, |w|_b \equiv 0 \mod 2\}$.

The language accepted by the (minimal) DFA in Fig. 7, where **11** is a final state, is $L(\mathcal{C}) = \{w \in \{a,b\}^* \mid |w|_a \equiv 1 \mod 2, |w|_b \equiv 1 \mod 2\}$.

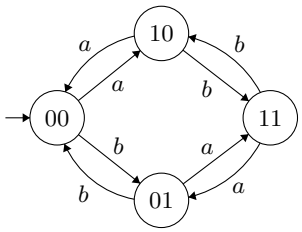

Fig. 9: DFA accepting words with even (when **00** is the final state) or odd (when **11** is the final state) number of a's and b's.