Submission to the Journal of Automata, Languages and Combinatorics
Submitted on January 21, 2022

COMPLEXITIES FOR JUMPS AND SWEEPS

Szilárd Zsolt Fazekas[(A)] Robert Mercaş[(B)] Olivia Wu[(B)]

[(A)] *Graduate School of Engineering Science, Akita University, Japan*
szilard.fazekas@ie.akita-u.ac.jp

[(B)] *Department of Computer Science, Loughborough University, United Kingdom*
R.G.Mercas@lboro.ac.uk (R. Mercaş) olivia.wu98@gmail.com (O. Wu)

ABSTRACT

The recently introduced model of one-way jumping finite automata skips over letters for which it does not have defined transitions instead of halting and rejecting as classical machines do. It is known that as an acceptor it is strictly more powerful than classical finite automata. The extra power comes from the ability of temporarily jumping over parts of the input. Here we define classes of machines and their accepted languages when this resource is bounded asymptotically, similar to computational complexity classes. We initiate the study of the gap between the resulting constant and linear jumping complexity classes. We conjecture that there is no intermediate jumping complexity but show that for a restriction of the model where the length of the jumped-over factors is limited, machines with logarithmic jumping complexity exist. We also introduce a measure called sweep complexity in order to get closer to a characterization of the regular language class in terms of one-way jumping machines with limited resources.

1. Introduction

Jumping finite automata (JFA) have been introduced to model non-contiguous computations with finite state control and no storage [13]. The model can accept some non-regular languages by checking linear relationships between letter counts, but at the price of giving up most information on the order of the letters, jumping around nondeterministically in the input. This also means that JFA accept only permutation-closed languages, therefore not all regular languages. JFA has been studied extensively with regards to the class of accepted languages, pumping lemmas for them [12] and parsing complexity [8]. From the beginning (see general JFA [13]) the model has been extended to reduce its nondeterminism. The model was also combined with Watson-Crick automata [10] and extended into two dimensions [9] to allow the processing of complementary double strands and picture languages, respectively.

One-way jumping transitions were proposed [4] as a deterministic and at least partially structure preserving processing mode for JFA and they refer to a machine jumping over the current tape symbol whenever its current state has no transition defined for it. This processing mode can also be seen as a finite state machine having

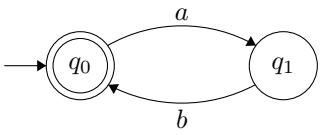

Figure 1: ROWFJA M_{ab} accepting binary words with equal number of a's and b's

its input in a first-in-first-out (FIFO) storage where we can only write the most recently read symbol or nothing, making it a very restricted version of queue automata, a model with a rich history [11]. Looking at it through these glasses, one-way jumping machines can model processes when the input comes in a stream or on a circular assembly line and there is limited time to deal with each piece. If the time is exhausted, then the current piece of information/assembly is left for processing later.

One-way jumping machines behave exactly as their classical counterparts when transitions are defined for the current state and input letter which makes the model capable of accepting any language accepted by its classical counterpart. At first, deterministic finite automata (DFA) equipped with the new jumping mode, the so called right one-way jumping finite automata (ROWJFA), were introduced and studied [4]. Various properties of the accepted language class [1] and the status of fundamental decidability questions have been settled [2]. Since then various classes of machines with this new processing mode have been investigated, such as nondeterministic finite automata [3, 5], two-way finite automata [6], pushdown automata and linear bounded automata [5]. The topics addressed have mainly been accepting power, closure properties of the accepted language class and decidability questions. 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 machines are, even in the simplest case when one starts from DFA.

We know that such machines 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 clever use of combinations of a handful of mostly regular languages together with a very simple type of non-regular languages which contains 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 M_{ab} in Fig 1. While this is enough to establish virtually all separations of interest, it leaves a significant gap in our understanding of the model: can such a machine accept any ('interesting') non-regular language apart from the ones which establish linear relationships among letter counts?

Trying to get a better picture of the kind of languages accepted by DFA with one-way jumping looks necessary also in order to attack the hardest problem to date related to the model: how to decide whether the language accepted is regular or not. Were there a constructive decision procedure for the problem, one may be able

to apply it to represent regular languages more succinctly by incomplete DFA with jumps. This ties in to the more general problem of finding suitably defined 'canonical' minimal ROWJFA for a given language. Simple examples such as in Fig. 2 can demonstrate that machines having fewest states are not unique for a given language, but imposing some order on the machines and searching for the first, or just any with the fewest number of states is a possible avenue.

Starting from an idea which led to a sufficient condition for a ROWJFA to accept a regular language, we start exploring the boundary between machines which accept regular languages and ones which accept more complex languages. This involves quantifying the non-regularity of the language accepted, which we formalize by jump complexity and sweep complexity, respectively. These measures count the number of jumps, consecutive jumps, and so called sweeps. This paper is meant to start the conversation about jump complexity and its relatives for these one-way jumping machines. While we do not reach any definitive results, we identify promising directions and derive some nontrivial fundamental results concerning the complexities.

In Section 2 we give some common definitions and give an alternative interpretation of the one-way jumping machine model. In Section 3 we introduce the complexity measures considered and raise a number of problems with regards to them. In Section 4 we analyze the sweep complexity of machines with limited number of consecutive jumps. In Section 5 we develop necessary conditions for machines to have superconstant ($\boldsymbol{\omega}(1)$) jump and sweep complexity. In Section 6 we draw some conclusions and talk about potential future work.

2. Preliminaries

A *deterministic finite automaton* (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 can transition 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 $a \in \Sigma$. We denote this by $\mathbf{p}w \Rightarrow \mathbf{q}w'$ and the reflexive and transitive closure of the relation $\Rightarrow$ by $\Rightarrow^*$. A word w is *accepted* by a DFA M if there exists $\mathbf{f} \in F$, such that $\mathbf{s}w \Rightarrow^* \mathbf{f}$. The language accepted by M is $\{w \in \Sigma^* \mid \exists \mathbf{f} \in F : \mathbf{s}w \Rightarrow^* \mathbf{f}\}$.

A *nondeterministic finite automaton* (NFA) is a quintuple $M = (Q, \Sigma, R, \mathbf{s}, F)$,

Figure 2: Different minimal ROWJFA accepting the language $\{w \in \{a, b\}^* \mid |w|_b > 0\}$.

where everything is defined the same way as for DFA, except for R, which is relaxed to being any relation from $Q \times \Sigma$ to Q, not necessarily a function.

One-way jumping mode

The *right one-way jumping relation* (denoted by $\circlearrowright$) between configurations from $Q\Sigma^*$, was originally defined in [4]. Here we define it a little differently without changing the accepting power of the ROWJFA model to make it more convenient when talking about complexity measures. We separate the transitions where the machine reads the letter immediately to the right of the head from the transitions where the head needs to jump over said letter, because no transition is defined for it from the current state.

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 $\Sigma_p = \{b \in \Sigma \mid \exists \mathbf{q} \in Q \text{ such that } \mathbf{p}b \to \mathbf{q} \in R\}$ 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:

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

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

(*i*) $\mathbf{p}ax \Rightarrow \mathbf{q}y$, 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 $\mathbf{s}w \models^* \mathbf{f}$. The language accepted by M is defined by

$$L(M) = \{x \in \Sigma^* \mid \exists \mathbf{f} \in F : \mathbf{s}x \models^* \mathbf{f}\}.$$

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

For analyzing regularity conditions of languages accepted by ROWJFA, it seems crucial to consider positions that the automaton jumps over multiple times. So far the only non-trivial sufficient condition for a ROWJFA language to be regular is to put a constant bound on the number of times the automaton reaches the last position of the original input word [7].

We say that states for which there is no transition labeled by the upcoming input letter are *deficient*, i.e., $\mathbf{p} \in Q$ is S-deficient if for all $a \in S \subset \Sigma$ we have $\mathbf{p}a \to \mathbf{q} \notin R$ for every $\mathbf{q} \in Q$. We write $\mathbf{p}$ is a-deficient if $S = \{a\}$ is a singleton.

Sweeps are contiguous parts of the computation on a given input, intuitively, the steps from reading or jumping over the leftmost remaining input letter to reading or jumping over the rightmost one. Formally, for an arbitrary input $w_0 \in L(M)$ let the ROWJFA M have the following computation:

$$\mathbf{p}_0 w_0 \models \mathbf{p}_1 w_1 \models \cdots \models \mathbf{p}_n \in F.$$

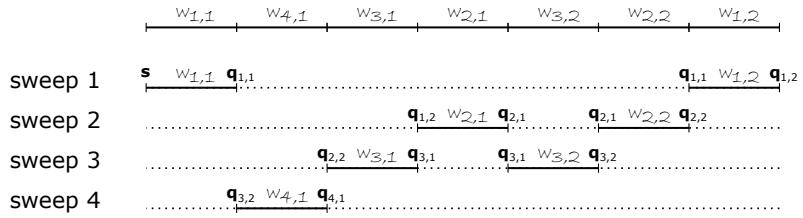

Figure 3: The word $w_{1,1}w_{4,1}w_{3,1}w_{2,1}w_{3,2}w_{2,2}w_{1,2}$ processed by a ROWJFA in 4 sweeps.

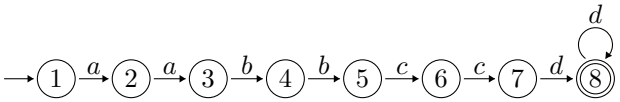

Figure 4: ROWJFA M_1 accepting all w with $|w|_a = |w|_b = |w|_c = 2$ and $|w|_d \geq 1$.

Let $\mathbf{p}_{i_1}, \mathbf{p}_{i_2}, \ldots, \mathbf{p}_{i_k}$ be such that $i_0 = 0$, $i_1 = |w_0|$, $i_k = n$ and for all $j \geq 1$ we have that $i_{j+1} = i_j + \text{jumps(j)}$, where jumps$(j)$ is the number of jumping transitions between configurations i_{j-1} and i_j, that is,

$$\text{jumps}(j) = |\{\ell \mid \mathbf{p}_\ell w_\ell \circlearrowleft \mathbf{p}_{\ell+1} w_{\ell+1} \text{ with } i_{j-1} \leq \ell < i_j\}|$$

transitions $\mathbf{p}_\ell w_\ell \circlearrowleft \mathbf{p}_{\ell+1} w_{\ell+1}$ with $i_{j-1} \leq \ell < i_j$. Then, sweep j is given by the computation $\mathbf{p}_{i_{j-1}} w_{i_{j-1}} \models \cdots \models \mathbf{p}_{i_j} w_{i_j}$ and the number of sweeps needed for M to process w_0 is k.

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.

Example 1. Consider an input $w_{1,1}w_{4,1}w_{3,1}w_{2,1}w_{3,2}w_{2,2}w_{1,2}$, where $w_{i,j} \in \Sigma^+$, which is processed in 4 sweeps as shown in Fig. 3. The first index denotes the sweep in which the factor was read while the second index orders the factors read within the same sweep. The machine first reads the factors for the first sweep, then the ones for the second sweep, etc., so $w_{1,1}w_{1,2}w_{2,1}w_{2,2}w_{3,1}w_{3,2}w_{4,1}$ is the order in which the factors are read.

The computation starts in the initial state $\mathbf{s}$. After reading $w_{i,j}$ the ROWJFA is in state $\mathbf{q}_{i,j}$. State $\mathbf{q}_{1,1}$ is S-deficient with $S = \{a \in \Sigma \mid |w_{4,1}w_{3,1}w_{2,1}w_{3,2}w_{2,2}|_a > 0\}$, while state $\mathbf{q}_{1,2}$ is T-deficient with $T = \{a \in \Sigma \mid |w_{4,1}w_{3,1}|_a > 0\}$, etc. The input is accepted if $\mathbf{q}_{4,1}$ is a final state. The ROWJFA jumps over the letters in $w_{4,1}$ three times before processing them, hence the number of sweeps is four. For a ROWJFA performing such an accepting computation consider the automaton M_1 in Fig. 4 and the input *adcbcba*, processed in the order *aabbccd*.

For a more 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. We add an extra sweep 0 as the first row, representing the initial input before processing. Cell i, j in the table contains either a letter or a symbol representing that the letter has been *read*. By setting the marker to be the empty word ε, we can conveniently talk about the computation in what follows. Once a letter has been marked *read* it stays that way, so each columns is a word of the form $a^\ell \varepsilon^{k-\ell}$ $(= a^\ell)$ for some $a \in \Sigma$ and $1 \leq \ell \leq k$. Consider now the input *adcbcba* from the example before. The computation table is described in Fig. 5.

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

Figure 5: The computation table for *adcbcba* by the machine in Example 1.

Let $A = (a_{i,j}) \in (\Sigma \cup \{\varepsilon\})^{m \times n}$ be such a computation table. We say that some position i is jumped over in sweep j if it is not marked *read* in sweep j so, translated to table terms, $a_{i,j} \in \Sigma$. The next lemma is quite intuitive and we make use of it when analyzing superconstant complexities both in the case of jumps and that of sweeps. It expresses the fact that in an accepting computation every jump must be followed, sooner or later, by a sequential (reading) transition.

Lemma 2. *In any accepting computation table, for any column* $\mathbf{c}_i$ *with* $|\mathbf{c}_i| > 1$, *there exists a column* $\mathbf{c}_j$ *such that* $|\mathbf{c}_j| = |\mathbf{c}_i| - 1$.

Proof. Let the input be $a_1 \cdots a_n$. The machine reads a_i in sweep $|\mathbf{c}_i|$, so there is a sweep $|\mathbf{c}_i| - 1$. The machine needs to read at least one letter in each sweep (> 0), so there exists some j such that a_j is read in sweep $|\mathbf{c}_i| - 1$, which means $|\mathbf{c}_j| = |\mathbf{c}_i| - 1$. □

Corollary 3. *If an accepting computation table has a column of height* n, *then for each* $i \in \{1, \ldots, n-1\}$ *it has a column of height* i.

Further, we can establish some useful relation between the number of letters jumped over in the first sweep and the number of sweeps needed in the computation. Let us denote the length of each row to be the number of remaining elements in it, namely the number of symbols not read yet.

Lemma 4. *For any accepting computation table with rows* $\mathbf{r}_0, \mathbf{r}_1, \ldots, \mathbf{r}_m$ *we have, for* $0 \leq i \leq m$, *that* $|\mathbf{r}_i| \geq m - i$.

Proof. In each sweep the machine reads at least one letter and trivially the only letters that can be read in a sweep are the ones that have not been read in previous sweeps. This implies the statement. □

Before going on to define the complexity measures we would like to analyze, we recall the aforementioned result on languages accepted with constant-bounded number of sweeps.

Theorem 5. *[7] For any ROWJFA $\mathcal{A} = (Q, \Sigma, R, \boldsymbol{s}, F)$, if there exists a constant k, such that for any word $w \in L(\mathcal{A})$ the number of sweeps needed by $\mathcal{A}$ to process w is at most k, then the language $L(\mathcal{A})$ is regular.*

The theorem is proved by simulating the computation of the ROWJFA with an NFA, which for each sweep guesses the states in which it starts as well as guessing which parts of the input are read in the sweep and accepting if a matching set of guesses is possible. Here we give a somewhat simplified version of the original proof which actually allows a stronger statement.

Theorem 6. *For any ROWJFA $\mathcal{A} = (Q, \Sigma, R, \boldsymbol{s}, F)$ and any constant k, the set of words accepted by $\mathcal{A}$ in at most k sweeps is regular.*

Proof. We prove the statement by constructing an NFA, which simulates accepting runs of $\mathcal{A}$ performed in at most k sweeps. Let this NFA be $\mathcal{B} = (Q', \Sigma, R', \mathbf{s}', F')$. The states of the NFA have $2k$ components, tracking the computation in each of the sweeps

$$Q' = \{\mathbf{q}_{\mathbf{s}_1,\mathbf{c}_1,\ldots,\mathbf{s}_k,\mathbf{c}_k} \mid \mathbf{s}_i, \mathbf{c}_i \in Q, 1 \leq i \leq k\}.$$

The indices $\mathbf{s}_i, \mathbf{c}_i$ of the states are interpreted as follows:

- the integer i is the number of the sweep,
- the state $\mathbf{s}_i$ is the one in which we started sweep i, and
- the state $\mathbf{c}_i$ is the current one in sweep i.

We can keep track of the computation performed in sweep i by tracking the change of state of $\mathcal{A}$ in the components $\mathbf{c}_i$ of the states of the NFA, while remembering in $\mathbf{s}_i$ the state we started the sweep in. The NFA has the transitions

$$\mathbf{q}_{\mathbf{s}_1,\mathbf{c}_1,\ldots,\mathbf{s}_{i-1},\mathbf{c}_{i-1},\mathbf{s}_i,\mathbf{c}_i,\mathbf{s}_{i+1},\mathbf{c}_{i+1},\ldots\mathbf{s}_k,\mathbf{c}_k} a \to \mathbf{q}_{\mathbf{s}_1,\mathbf{c}_1,\ldots,\mathbf{s}_{i-1},\mathbf{c}_{i-1},\mathbf{s}_i,\mathbf{c}'_i,\mathbf{s}_{i+1},\mathbf{c}_{i+1},\ldots,\mathbf{s}_k,\mathbf{c}_k} \tag{1}$$

for each $i \in \{1, \ldots, k\}$ and all combinations of indices satisfying the conditions that

- states $\mathbf{c}_1, \ldots, \mathbf{c}_{i-1}$ are a-deficient, so they do not change;
- we have $\mathbf{c}_i a \to \mathbf{c}'_i \in R$;
- the components $\mathbf{c}_{i+1}, \ldots, \mathbf{c}_k$ do not change because the letter has been read in sweep i so it is not on the tape in consequent sweeps.

The second condition makes sure that each letter is read in some sweep, so if a final state is reached at the end of sweep k, the input is accepted.

The initial state is $\mathbf{s}'$. In addition to the transitions described above, the automaton has the following transitions:

$$\mathbf{s}'\lambda \to \mathbf{q}_{\mathbf{s},\mathbf{s},\mathbf{p}_2,\mathbf{p}_2,\ldots,\mathbf{p}_k,\mathbf{p}_k},$$

for all possible values $\mathbf{p}_2, \ldots, \mathbf{p}_k \in Q$. These are the only nondeterministic transitions of $\mathcal{B}$ and they are responsible for guessing the correct states in which each of the sweeps starts when accepting an input. We have final states of the form

$$\mathbf{q}_{\mathbf{s},\mathbf{c}_1,\mathbf{s}_2,\mathbf{c}_2,\ldots,\mathbf{s}_k,\mathbf{c}_k},$$

with $\mathbf{c}_{i-1} = \mathbf{s}_i$, for all $i \in \{2, \ldots, k\}$, meaning that sweeps start in the state in which the previous sweep finished; $\mathbf{c}_k \in F$ meaning that the machine reached a final state in some sweep $\ell \leq k$ (and did not change afterwards).

The NFA defined above has an accepting path for input w if and only if $w \in L(\mathcal{A})$, because:

- in any transition of type (1) exactly one of the c_i changes and that happens according to the transition table of $\mathcal{A}$.
- a letter $a \in \Sigma$ can only be processed in sweep i if in the sweeps before the machine jumped over it, ensured by the condition that $c_1, \ldots, c_{i-1}$ are a-deficient.
- the condition $c_{i-1} = s_i$ in the final states ensures that the computation continues between sweeps correctly.
- if a word was not accepted in k sweeps by $\mathcal{A}$, that happens
  - ∗ either because the whole input was processed in k sweeps but $\mathcal{A}$ stopped in a non-final state r, in which case the component c_k of $\mathcal{B}$ will be equal to r, so $\mathcal{B}$ also rejects;
  - ∗ or because some letter a of the input has not been read in the first k sweeps by $\mathcal{A}$, which means that the state of $\mathcal{A}$ in each of those sweeps is a-deficient, in which case $\mathcal{B}$ will get stuck when reaching that letter and hence the input will not be accepted.

This concludes our proof. □

If all words are accepted by a machine with a constant number of sweeps then the above turns into the previous weaker version. The sufficient condition in Theorem 5 of constant-bounded number of sweeps was conjectured to be necessary, as well, evidenced by the known examples so far.

3. Complexity measures

We are interested in 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. To be able to analyze this quantitatively, we define a number of complexity measures, which give us the number of jumps/sweeps performed by a machine in the 'worst case' for an input of length n.

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

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

be a computation in M on the input w. We define

$$E(C_M(w)) = \{i \geq 1 \mid \mathbf{p}_{i-1} w_{i-1} \circlearrowleft \mathbf{p}_i w_i\}.$$

In words, $E(C_M(w))$ contains all the jumping steps in the computation $C_M(w)$.

We now define the *jumping complexity* of the computation of M on the word w by

$$jc_M(w) = \begin{cases} \min\{card(E(C_M(w))) \mid C_M(w) \text{ is a computation of } M \text{ on } w\} \\ 0, \text{ if } w \notin L(M). \end{cases}$$

In other words, the jumping complexity of a word with respect to M is computed by taking into consideration the "least non-regular computation" if there is one. Equivalently, the jumping complexity of a word with respect to M is the number of jumping steps of a computation with the minimal number of jumping steps. Note that these definitions[1] work for the more general model of nondeterministic machines, NROWJFA, but this is outside the scope of our paper, as this work only considers deterministic ROWJFA and the complexity classes defined by them, and therefore in all cases there is a single computation to be considered.

The jumping complexity of an automaton M as above is a mapping from $\mathbb{N}$ to $\mathbb{N}$ defined by

$$jc_M(n) = \max\{jc_M(w) \mid |w| = n\}.$$

As one can see, the most "non-regular" word of each length is considered.

Similarly to the jump complexity above we can now define the *sweep complexity* of a machine M as 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. With these in mind, we can define complexity classes in the usual manner, JUMP($f(n)$) and SWEEP($f(n)$) being the classes of languages accepted by some one-way jumping machine with jump complexity $\mathcal{O}(f(n))$ and sweep complexity $\mathcal{O}(f(n))$, respectively.

Using the notation above we can summarize what we know so far and initial observations as follows:

Proposition 7. *(1) For any ROWJFA M we have $sc_M(n) \leq n$ and*

$$sc_M(n) - 1 \leq jc_M(n) \leq \sum_{k=n-sc_M(n)+1}^{n-1} k \leq \frac{n(n-1)}{2}.$$

(2) For any function f we have JUMP($f(n)$) $\subseteq$ SWEEP($f(n)$) $\subseteq$ JUMP(n^2).
(3) JUMP(1) = SWEEP(1) = REG.

[1] this form of the definitions was suggested by Victor Mitrana

(4) SWEEP(1) $\subsetneq$ JUMP(n^2).

Proof. (1) and (2): observe that in any computation the number of sweeps is at most one more than the number of jumps, because to have any letters to read in a given sweep, said letter had to be jumped over previously. On the other hand, in each sweep the machine needs to consume one of the input letters, otherwise the word is not accepted, so the number of sweeps is at most the length of the input. This means that the length of the remaining input decreases between sweeps, so we get that a straightforward upper bound on $jc_M(n)$ is the sum of the series $(n-1)+(n-2)+\cdots$ with the number of terms being $sc_M(n)-1$, which derives the quadratic upper bound of $\frac{n(n-1)}{2}$. Statement (3) follows from Theorem 5. The separation in (4) is illustrated by the omnipresent example of the nonregular language L_{ab}: this language is not regular, so it is not in SWEEP(1), but it is accepted by the simple ROWJFA in Fig. 1, so it is in JUMP(n^2). □

The maximum number of jumps, with respect to the input word's length, in all (current) examples in the literature seem to all fall in one of the three following categories:

- constant-bounded, e.g., machines with complete transition functions (0 jumps);
- linear, e.g., the machine M_1 in Fig. 4 which accepts everything in at most 4 sweeps, so has sweep complexity 4 which gives jump complexity less than $4n$; at the same time, for inputs of the form $d^kccbbaa$ it makes over $4k$ jumps;
- quadratic, e.g., the machine M_{ab} mentioned earlier: for inputs of the form a^kb^k in each sweep two letters are read and all the remaining ones are jumped over, so we get $jc_M(n) = \frac{n(n-2)}{8}$.

We seem to have a similar gap for sweep complexity since the sweep complexity of each machine considered so far is either constant or linear. Our first inquiry is into these apparent gaps: do there exist machines whose jump complexity is, e.g., $\Theta(\log n)$ or $\Theta(n \log n)$? Are there machines with, say, $\Theta(\log n)$ sweep complexity?

4. Jump Restrictions

In this section of our work we consider variations of the ROWJFA in which we limit the number of jumps that are allowed, *per state*. In other words for any accepted input of length n and for each state $\mathbf{p}$ the machine can perform $f(n)$ jumps while in state $\mathbf{p}$. By Theorem 6, bounding the number of sweeps by a constant leads to regularity of the accepted language, so we can immediately conclude that a constant bound on the total number of jumps allowed per state and per computation also leads to regularity, since this trivially implies a constant number of sweeps no larger than the product between the number of states and the bound on the jumps on each of them.

Not wanting to relax the bound on the number of jumps too much, we investigate what happens when we limit the number of jumps per state, but we do this relative to each visit. Hence, if we are not allowed to do more than k jumps per state visit, for some fixed positive integer k, it means that in between every $k+1$ jumps, that

the ROWJFA does, the machine has to process one symbol or otherwise reject the input. Note that this is a *restriction of the model*, not a complexity class.

Definition 8. A *deterministic right one-way $f(n)$-jumping automaton* ($\text{ROWJFA}_{f(n)}$) is defined as a ROWJFA $M = (Q, \Sigma, R, \mathbf{s}, F)$ with an additional condition that from no state can we do more than $f(n)$ consecutive jumps ($\circlearrowright$ transitions) while accepting an input word of length n. Formally, we have ta computation $\mathbf{p}_1 w_1 \models \mathbf{p}_2 w_2 \models \cdots \models \mathbf{p}_m w_m$, on an input w_1 of length n, is accepting if $\mathbf{p}_1 = \mathbf{s}$, $\mathbf{p}_m \in F$, $w_m = \varepsilon$ and $|w_i| > |w_{i+f(n)+1}|$, for every $i \in \{1, \ldots, m - f(n) - 1\}$.

Looking at the above definition another way, this restriction can also be seen as one on the length of the jumps made, i.e., the ROWJFA cannot jump over any factor of length greater than k.

The complexity definitions from Section 3 extend naturally to the restricted model, hence the jumping (sweep) complexity of a ROWJFA_k is the function which gives the maximum number of jumps (sweeps) needed to accept words of length n.

Proposition 9. *For any positive integer k, the sweep complexity of any ROWJFA_k is $O(\log n)$.*

Proof. Consider an input of length n which is accepted by such a ROWJFA_k. Since the ROWJFA_k does no more than k consecutive jumps it must be the case that in each of the sweeps the it reads at least $\lfloor \frac{m}{k+1} \rfloor$ symbols, if m is the length of the sequence still to be processed by the automaton. Hence, we easily deduce that the automaton does $O(\log_{\frac{k+1}{k}} n)$ sweeps in order to process the whole word, and we conclude. □

Instead of a fixed k, considering sublinear functions $f(n)$ (where n is the length of the word) leads to a new complexity measure, but for the rest of this section our focus is only on constant functions $f(n) = k$.

For any arbitrary integer $k > 1$, consider now the morphism ϕ_k defined over the alphabet $\Sigma_k = \{1, 2, \ldots, k\}$ with $\phi_k(i) = i+1$, for $i < k$, and $\phi_k(k) = k\,1\,2$ (a sequence of length 3). As usual, let $\phi_k^0(i) = i$ and $\phi_k^{\ell+1}(i) = \phi_k(\phi_k^\ell(i))$. Since this morphism is prolongable on k, that is $\phi_k^n(k)$ is a prefix of $\phi_k^{n+1}(k)$, we immediately get that $\phi_k^\omega(k)$ is a fixed point. For $k = 2$ this is exactly the *OEIS* recursive sequence **A125058** which is defined as $a(0) = 1$, $a(1) = 2$, and then $a(n+2) = a(n+1)a(n)a(n+1)$.

Remark 10. For all positive integers k and ℓ with $\ell > 2$, in the sequence $\phi_k^\ell(k)$, reading from left to right, the elements are non-decreasing up to k, with k being either followed by another k, or by a 1. Moreover, every other element is either followed by a consecutive number, or by one equal to it.

We can also deduce the following result with respect to the images of ϕ.

Lemma 11. *For every integer $k > 1$, the sum of the elements of $\phi_k^\ell(k)$ is less than three times the sum of the elements of $\phi_k^{\ell-1}(k)$.*

Proof. Let $\phi_k^{\ell-1} = a_1 \cdots a_m$ with $a_i \in \{1, \ldots, k\}$ for all $i \in \{1, \ldots, m\}$. By the definition of ϕ the sum of elements of the image ϕ of each letter a, denoted $S_\phi(a)$, is

$$S_\phi(a) = \begin{cases} a+1, & \text{if } 1 \leq a \leq k-1, \\ k+1+2 = k+3, & \text{if } a = k, \end{cases}$$

Since $a \geq 1$ and $k \geq 2$, we get that $S_\phi(a) \leq \frac{5}{2} \cdot a$, for any $a \in \{1, \ldots, k\}$. The sum of the elements of $\phi_k^\ell(k)$ is then $S_\phi(a_1) + \cdots + S_\phi(a_m) \leq \frac{5}{2}(a_1 + \cdots + a_m)$, and the statement follows. □

The following conclusion is immediate.

Corollary 12. *For any $\ell \geq 0$ and $k \geq 2$, we have $\ell + k > \log_3 S$, where S is the sum of the elements of $\phi_k^\ell(k)$.*

Proof. Let S_n denote the sum of the elements of $\phi_k^n(k)$. We will argue by induction on ℓ. If $\ell = 0$ we have $S_0 = k$, so the statement holds. For the induction step, assume that the hypothesis holds for $\ell = n$, that is, $n + k > \log_3 S_n$. By Lemma 11 we have that $S_{n+1} < 3 \cdot S_n$, so $n + 1 + k > \log_3 3S_n > \log_3 S_{n+1}$, which means that the statement holds. □

Consider the machine $M_{ab} = (\{\mathbf{q}_0, \mathbf{q}_1\}, \{a, b\}, \{\mathbf{q}_0 a \to \mathbf{q}_1, \mathbf{q}_1 b \to \mathbf{q}_0\}, \mathbf{q}_0, \{\mathbf{q}_1\})$ shown in Fig. 1. Let us denote the automaton M_{ab}, when jumps no longer than some positive k are allowed, by $M_{ab}^{(k)}$. Note that a word accepted by $M_{ab}^{(k)}$ cannot have a unary factor of length $k+2$ or higher, since such a factor would require a jump of at least $k+1$ symbols in order to be processed by the automaton.

While for ROWJFAs with no restrictions the maximum number of sweeps needed to process a language is linear, as we have seen earlier, and such languages exist, whenever we introduce restrictions on the length of the jumps, then the bound lowers to logarithmic, as shown in Proposition 9, and the next result shows that such languages exist.

Theorem 13. *For every fixed $k > 1$, there exists an ROWJFA$_{k-1}$ with sweep complexity $\Omega(\log n)$.*

Proof. The choice of $k-1$ in our statement is due to legibility when working with the morphism ϕ from above. We will show that $M_{ab}^{(k-1)}$ has sweep complexity $\Omega(\log n)$.

Recall that M_{ab} accepts all words having the same number of a's and b's when there is no restriction on its jumps. In our case, the automaton $M_{ab}^{(k-1)}$ cannot do more than $k-1$ consecutive jumps before reading a letter. Consider the binary words given by the function $\Psi^\ell : \{a, b\} \to \{a, b\}^*$ defined as $\Psi^\ell(a_1) = a_1^{e_1} a_2^{e_2} a_3^{e_3} a_4^{e_4} \cdots a_m^{e_m}$, where $a_i \neq a_{i+1}$ for any $i \in \{1, \ldots, m-1\}$ and $e_1 e_2 \cdots e_m = \phi_k^\ell(k)$. That is, every number in $\phi_k^\ell(k)$ represents the power of a symbol from $\{a, b\}$, different from the previous one. As an example, since $\phi_2^3(2)$ is

21222122122122212,

the word $\Psi^3(a)$, starting with an a, is the word

$$a^2b^1a^2b^2a^2b^1a^2b^2a^1b^2a^2b^1a^2b^2a^2b^1a^2.$$

Note that the complementary word is $\Psi^3(b)$, in which we just substitute every a for a b, and vice-versa. We call the description given by ϕ, where we replace blocks of letters by their exponent, the *block representation* of a particular word.

We first show that the words $\Psi^\ell(a)\Psi^\ell(b)$ are accepted by our machine for every integer $\ell \geq 0$. Indeed, since the latter half of the word is just the complement of the former one, the number of a's and b's in the word is the same. Moreover, by definition, $\Psi^\ell(a)$ and $\Psi^\ell(b)$ have blocks of length at most k. Since the former half of the words, namely $\Psi^\ell(a)$, ends in a, while the latter, namely $\Psi^\ell(b)$, starts with b we also ensure the fact that no more than k consecutive occurrences of the same letter exist in the word after concatenating the two.

Finally, we need to make sure that no unary factor of length greater than k is created, after a concatenation of neighbouring blocks following a sweep. We note that instead of working with actual words we can restrict ourselves to the block representations, keeping in mind that whenever a block has exponent 0, then the blocks to the left and right should be concatenated. That is, a representation of the word *aabbabbaab* would be 221221, and after a sweep through the word we are left with 110110, that is, with the word *abba* represented by 121.

Following Remark 10, since the only time that a 1 occurs is when it is preceded by a k and is followed by a 2, this will be the only instance when two or more separate blocks are combined. Since we read exactly one symbol from each block during a sweep, the consecutive blocks k 1 2 become $(k-1)$ 0 1 and are merged into a single block k. Moreover, in this case, the values of all of the other blocks (that do not lead to concatenations) decrease by 1. It is not difficult to see that in fact this is $\phi_k^{\ell-1}(k)$ and therefore each such sweep renders the previous iteration of our morphism.

Following the conclusion above, since $\phi^\ell(k)$ is reduced to $\phi^0(k) = k$ after ℓ such interations, the word $\Psi^\ell(a)\Psi^\ell(b)$ needs at least ℓ sweeps in order for it to be accepted by $M_{ab}^{(k-1)}$. From here we get our asymptotic lower bound via Corollary 12. □

5. Beyond constant-bounded complexity for jumps and sweeps

We start this section by looking at what it entails when a machine has constant-bounded jump complexity.

First, we will fix some terminology. Consider a ROWJFA $M = (Q, \Sigma, R, \mathbf{s}, F)$ and let $\mathbf{q} \in Q$ be a state of M. We say that $\mathbf{q}$ is *cyclic* if it is in a cycle, i.e., there is some word $w \in \Sigma^+$ such that $\mathbf{q}w \Rightarrow^* \mathbf{q}$. State $\mathbf{q}$ is *reachable* if there is a word $w_1 \in \Sigma^*$ such that $\mathbf{s}w_1 \Rightarrow^* \mathbf{q}$ and it is co-reachable if there is a word $w_2 \in \Sigma^*$ and $\mathbf{f} \in F$ such that $\mathbf{q}w_2 \Rightarrow^* \mathbf{f}$. Recall that a state is deficient if there is at least one letter for which it has no defined transition.

Proposition 14. *If the jump complexity of a machine M is not bounded by any constant, then M has a state $\mathbf{p}$ which is deficient, cyclic, reachable and co-reachable, but the converse does not hold.*

Proof. If the jump complexity of M is not bounded by any constant, then for any non-negative integer k there exists some word w_k such that the least non-regular accepting computation (the one with the fewest jumps) makes at least $k+1$ jumps. If we set $k = |Q|$ this means that the machine is in the same state $\mathbf{p}$ at least two times when making a jump. As $\mathbf{p}$ occurs on an accepting path, the reachability and co-reachability conditions are met. Also, such a state $\mathbf{p}$ must be deficient, otherwise the machine could not jump while in this state (which is part of our assumption). Since we reach this state multiple times during a computation on w_k, we also get that state $\mathbf{p}$ is in some cycle.

As for the converse not holding, we have to show that having such a deficient state in a cycle is not sufficient to guarantee that the machine has superconstant jump complexity.

Consider the machine with input alphabet $\{a, b\}$ and transitions $\mathbf{1}a \to \mathbf{2}$, $\mathbf{2}a \to \mathbf{1}$ and $\mathbf{1}b \to \mathbf{3}$, where $\mathbf{1}$ is the initial state and $\mathbf{3}$ is final (see Fig. 6). We can see that it satisfies the conditions of Proposition 14. Namely, state $\mathbf{2}$ is b-deficient (cannot read the symbol b), is in the cycle $\mathbf{2} \to \mathbf{1} \to \mathbf{2}$, and there is a walk from the initial state $\mathbf{1}$ to the final state $\mathbf{3}$ which traverses it. The only other deficient state is $\mathbf{3}$, but it has no outgoing transitions, so once we reach it, the computation ends. Since the only symbol to be jumped is b and the only words accepted contain exactly one occurrence of the symbol b, as per the automaton description, we immediately conclude that the maximum number of jumps for this machine is actually 1. If we reach b for the first time with a configuration $\mathbf{2}ba^{2k+1}$, then we accept in the next sweep without another jump, while if we reach b with a configuration $\mathbf{2}ba^{2k}$, then we reject. □

Now that we know that having a deficient state in a cycle is not sufficient to have superconstant jump complexity, we can try to tighten the condition. The example in the proof above suggests that if the machine makes multiple jumps there should be another state in a cycle which can read the letters jumped over. In fact, as we will see later we can prove not only this, but that an analogous (but stronger) result holds also when we consider sweep complexity instead of jump complexity.

We need a technical lemma first, which can be easily derived from the basic properties of computation tables.

Lemma 15. *For any ROWJFA M and for any input $w \in L(M)$ with computation table having rows $\mathbf{r}_0, \mathbf{r}_1, \ldots, \mathbf{r}_m$ we get $|\mathbf{r}_1| > \sqrt{jc_M(w)}$.*

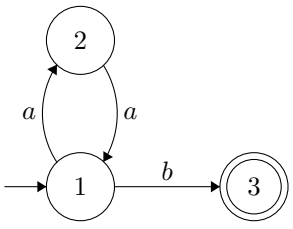

Figure 6: ROWFJA with a cyclic deficient state that only accepts words after at most one jump

Proof. By applying Lemma 4 to the first sweep, that is, with $i = 1$, we know that there cannot be more sweeps than the number of letters jumped over in the first sweep, that is, $|\mathbf{r}_1| \geq m - 1$. By the definition of computation tables we have $|\mathbf{r}_i| > |\mathbf{r}_{i+1}|$, for each $i < m$. Since the lengths of the rows combined (except for, of course, row 0) give the number of jumps the machine makes, we get that the jumping complexity is here $jc_M(w) = \sum_{i=1}^{m} |\mathbf{r}_i| \leq m \cdot |\mathbf{r}_1| < |\mathbf{r}_1|^2$, and the statement follows. □

Proposition 16. *If a ROWJFA has superconstant jump complexity, then it has two cyclic, reachable and co-reachable states* $\boldsymbol{p}$ *and* $\boldsymbol{q}$ *such that state* $\boldsymbol{p}$ *is* a*-deficient, for some* $a \in \Sigma$*, state* $\boldsymbol{q}$ *is reachable from* $\boldsymbol{p}$ *and* $\boldsymbol{q}aw \Rightarrow^* \boldsymbol{q}$*, for some* $w \in \Sigma^*$*.*

Proof. Consider a ROWJFA M with superconstant jump complexity. Then there must exist a word w which is accepted by M with more than $|Q|^4$ jumps, where Q is the state set of M.

Since $jc_M(w) > |Q|^4$, by Lemma 15 we get that in the computation table of w the row $\mathbf{r}_1$ has length at least $\sqrt{jc_M(w)}$, so larger than $|Q|^2$. In other words, the machine jumps over more than $|Q|^2$ letters in the first sweep.

Consider now for each of the letters jumped over in this first sweep, the state in which they were jumped over when first encountered and the state in which they are finally read during the computation. This way we associate a pair of states to each of said letters.

Since the number of different (ordered) pairs of states is $|Q|^2$, we have that at least two of the letters have the same pair $(\mathbf{p}, \mathbf{q})$ of states associated to them. This means that they were first jumped over in state $\mathbf{p}$ and they were both finally read in state $\mathbf{q}$. Let one of the two letters be a. From here we deduce that $\mathbf{p}$ is a-deficient and cyclic and there exists a cycle at $\mathbf{q}$ starting with a transition labeled by a. As this is an accepting computation, we get that both states are reachable and co-reachable. Moreover, as $\mathbf{q}$ occurs in the same computation in later sweeps than the first occurrence of $\mathbf{p}$, the former is reachable from the latter and the proof is concluded. □

The example in Fig. 7 shows that the converse of Proposition 16 does not hold: the conditions on the machine are satisfied with $\mathbf{p} = \mathbf{2}$ and $\mathbf{q} = \mathbf{4}$, but the machine does not accept any words with more than 3 jumps. To see this, note that the only state in which jumps are possible is $\mathbf{2}$, in which the machine can jump over b's. This means that after the first sweep, the remaining input will be a word of the form b^k, where k is the number of jumps performed by the machine up to that point. For $k > 2$, however, it is easy to see that the machine cannot process b^k from any state, so a word with more than two jumps will not be accepted.

We end this section with an observation on what happens when a machine needs more than constant-bounded number of sweeps to accept its language. In particular, we present a consequence of superconstant sweep complexity which might be sufficient to imply it, as well, but we were not able to prove the reverse. This proposition establishes that superconstant sweep complexity implies the existence of two 'complementary' deficient states in a cycle: each of the two states can read some letter that the other cannot.

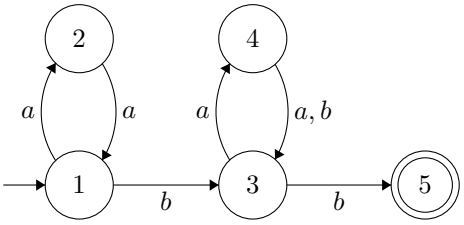

Figure 7: ROWJFA satisfying the conditions of Proposition 16, with b-deficient state **2** and cyclic state **4** which can read b in a cycle, but can not do more than two jumps during any accepting computation.

Proposition 17. *If a ROWJFA has superconstant sweep complexity, then it has two reachable and co-reachable states* $\boldsymbol{p}$ *and* $\boldsymbol{q}$ *such that* $\boldsymbol{p}$ *is* a*-deficient,* $\boldsymbol{q}$ *is* b*-deficient, for some* $a, b \in \Sigma$ *with* $a \neq b$*, and* $\boldsymbol{p}buav \Rightarrow^* \boldsymbol{q}av \Rightarrow^* \boldsymbol{p}$*, for some* $u, v \in \Sigma^*$*.*

Proof. If the number of sweeps by a machine M is not bounded by any constant, then for any k there exists some input $w \in L(M)$ such that some letter of w is read in a sweep $\ell_1 + 1 > k$. Let the leftmost such letter be $w[i_1]$, the i_1th letter of w (so all of $w[1 \dots i_1 - 1]$ is read in the first ℓ_1 sweeps).

First, in each sweep the machine reads at least a letter; similarly, after each series of consecutive jumps the machine reads at least a letter. Also straightforward is that if a letter is read in sweep $\ell_1 + 1$, then in all previous sweeps the machine must have jumped over the letter on this specific position.

What this means for us is that M jumped over position i_1 in all previous sweeps, so ℓ_1 times. As the number of sweeps is not constant-bounded, we can choose ℓ_1 large enough such that the machine jumps over this position i_1 multiple times while in the same state, say $\mathbf{p}_1$, and each such time reads the same letter, say a, before reading anything else. Let the number of times this happens be ℓ_2.

Considering only those ℓ_2 jumps, let us denote the positions of the a letters read after $\mathbf{p}_1$ jumps over i_1 by $x_{1,1}, x_{1,2}, \dots, x_{1,\ell_2}$, meaning that after the jth jump over i_1 the machine in state $\mathbf{p}_1$ reads the letter at position $x_{1,j}$. Observe that in this case we have $x_{1,1} < x_{1,2} < \cdots < x_{1,\ell_2}$. This latter fact is straightforward since, in order to read letter a on position $x_{1,j}$ in some sweep $j \leq \ell_2$, the machine should jump over the whole remaining unread factor in between positions i_1 and $x_{1,j}$, which would be impossible, if a letter a occurring at position $x_{1,i}$ with $i < j$ is to be read later on in the computation while in the same state $\mathbf{p}_1$.

Now we consider the position x_{1,ℓ_1} which is read in a sweep $\geq \ell_2$. Again, by choosing ℓ_2 large enough and applying the argument from before, we have that the position is jumped over multiple times while in the same state $\mathbf{p}_2$. We also know that $\mathbf{p}_2 \neq \mathbf{p}_1$, because $w[x_{1,\ell_1}]$ is read by state $\mathbf{p}_1$.

Repeatedly applying the argument above, we get that we must have $\mathbf{p}_i = \mathbf{p}_1$ for some $i \in [3, |Q| + 1]$. This means that M has a loop with at least two deficient states, namely $\mathbf{p}_1$ and $\mathbf{p}_2$, such that there is a letter which can be read by $\mathbf{p}_1$ but not by $\mathbf{p}_2$, and vice versa. □

6. Conclusions and future work

In this last part we make some conclusions explicit and identify promising research directions/ideas related to the complexity measures defined for one-way jumping finite automata. Based on the examples of ROWJFA and the consequences of having more than constant-bounded many jumps or sweeps while processing inputs, we conjecture that there are no machines with superconstant but sublinear jump or sweep complexity. We think it likely that there is also a gap between linear and quadratic jump complexity. The results we presented are the first structural results on ROWJFA and as such, fundamental in proving or disproving the existence of such a gap.

Logarithmic sweep complexity with other restrictions.
In Section 4 we saw that we can color outside the bounds of this conjecture when the number of *consecutive* jumps is bounded by a constant. In this direction we think it is worth exploring whether sublinear but superconstant bounds on consecutive jumps still allow machines with $\Theta(\log n)$ sweep complexity.

Gap between constant and linear jump complexities.
In the general case - with unrestricted number of consecutive jumps - the necessary conditions presented in Section 5 have been stated with an eye towards a proof strategy for the gap between constant and linear complexities. The conditions derived all take the form of superconstant complexity guaranteeing the existence of a state or pairs of states which are deficient and are in cycles. This opens the possibility of applying pumping arguments in order to prove the conjecture for jump complexity: if the machine visits the deficient cyclic state of Proposition 14 more than constant-bounded times, then an input which forces linearly many such visits may be possible to construct. However, this is far from trivial because of the other transitions involved in the two cycles mentioned in the proposition.

Sweep complexity gap and deciding regularity of ROWJFA languages.
Using Proposition 17, a pumping proof strategy seems promising for proving the complexity gap in the case of sweeps, too. Furthermore, based on some suitably strengthened version of the proposition one might be able to argue that a machine with a certain kind of loop having two complementary deficient states is in some sense similar to the machine for L_{ab}, as in it establishes a linear relationship between letter counts, and deduce that the accepted language is not regular, giving a decidable characterization of regular ROWJFA in terms of sweep complexity. As we mentioned before, the latter problem is a fundamental one in the theory of one-way jumping machines, but there has not been much progress on it over the last five years. Hopefully the ideas presented here can get the ball rolling on the topic.

The effects of alphabet size.
In previous papers on such machines, in some cases when an example language was constructed for separation results or for disproving a closure property, etc., the authors used alphabets with more than two and even three letters. While sometimes those examples can be replaced with binary ones at the expense of making the argument more complex, in others it is unclear whether that is possible. The above is an interesting question also in the case of sweep complexity considerations with respect to Proposition 17: if the alphabet is binary, then there is only one choice of comple-

mentary deficiency, that one is a-deficient and the other is b-deficient, illustrating the 'similarity' to M_{ab} mentioned earlier.

7. Acknowledgements

We would like to thank Victor Mitrana for proposing the idea of jumping complexity and its definitions used in this paper. We originally considered the idea of restrictions on the number of jumps and the length of jumps as approaches to get closer to necessary conditions of regularity and switched to this more systematic approach of defining the complexity classes following his suggestions.

References

[1] S. Beier, M. Holzer, Properties of right one-way jumping finite automata. *Theoretical Computer Science* **798** (2019), 78 – 94.

[2] S. Beier, M. Holzer, Decidability of right one-way jumping finite automata. *International Journal of Foundations of Computer Science* **31** (2020) 6, 805–825.

[3] S. Beier, M. Holzer, Nondeterministic right one-way jumping finite automata. *Information and Computation* (2021), 104687. In Press.

[4] H. Chigahara, S. Z. Fazekas, A. Yamamura, One-way jumping finite automata. *International Journal of Foundations of Computer Science* **27** (2016) 3, 391–405.

[5] S. Z. Fazekas, K. Hoshi, A. Yamamura, The effect of jumping modes on various automata models. *Natural Computing* (2021).

[6] S. Z. Fazekas, K. Hoshi, A. Yamamura, Two-way deterministic automata with jumping mode. *Theoretical Computer Science* **864** (2021), 92–102.

[7] S. Z. Fazekas, A. Yamamura, On regular languages accepted by one-way jumping finite automata. In: *8th Workshop Non-Classical Models of Automata and Applications, Short Papers*. 2016, 7–14.

[8] H. Fernau, M. Paramasivan, M. L. Schmid, Jumping finite automata: Characterizations and complexity. In: F. Drewes (ed.), *Implementation and Application of Automata - 20th International Conference, CIAA 2015, Umeå, Sweden, August 18-21, 2015, Proceedings*. Lecture Notes in Computer Science 9223, Springer, 2015, 89–101.

[9] S. J. Immanuel, D. G. Thomas, Two-dimensional jumping finite automata. *Mathematics for Applications* **5** (2016), 105–122.

[10] R. Kocman, B. Nagy, Z. Krivka, A. Meduna, A jumping 5' $\rightarrow$ 3' watson-crick finite automata model. In: R. Freund, M. Hospodár, G. Jirásková, G. Pighizzini (eds.), *Tenth Workshop on Non-Classical Models of Automata and Applications, NCMA 2018, Košice, Slovakia, August 21-22, 2018*. Österreichische Computer Gesellschaft, 2018, 117–132.

[11] M. Kutrib, A. Malcher, M. Wendlandt, Queue automata: Foundations and developments. In: A. Adamatzky (ed.), *Reversibility and Universality, Essays Presented to Kenichi Morita on the Occasion of his 70th Birthday*. Emergence, Complexity and Computation 30, Springer, 2018, 385–431.

[12] G. Madejski, Jumping and pumping lemmas and their applications. In: *8th Workshop on Non-Classical Models of Automata and Applications (NCMA 2016) Short papers*. 2016, 25–33.

[13] A. Meduna, P. Zemek, Jumping finite automata. *International Journal of Foundations of Computer Science* **23** (2012) 7, 1555–1578.