Freezing 1-Tag Systems with States

Szilárd Zsolt Fazekas*
Akita University
Akita, Japan
szilard.fazekas@ie.akita-u.ac.jp

Shinnosuke Seki†
University of Electro-Communications
Tokyo, Japan
s.seki@uec.ac.jp

We study 1-tag systems with states obeying the freezing property that only allows constant bounded number of rewrites of symbols. We look at examples of languages accepted by such systems, the accepting power of the model, as well as certain closure properties and decision problems. Finally we discuss a restriction of the system where the working alphabet must match the input alphabet.

1 Introduction

Tag systems are a class of deterministic string rewriting systems. In each step they read out the first letter, say a, of the current word along with the succeeding $n-1$ letters (where n is a system parameter), refer to the system's transition table (function) δ with the letter read as a key, and append the word $\delta(a)$ at the end of the current word. The system stops if the key is a special halting symbol or if there are less than n letters left in the word. A system in this class is often called an n-tag system including the value of n explicitly. It is well known that n-tag systems are capable of simulating Turing machines for any $n \geq 2$ (see, e.g., [2, 3]). The initial definition is stateless, has no additional storage or intricate rules describing its dynamics; hence such systems are ideal for being simulated in other computational models such as 1D cellular automata or molecular computation models, particularly in their early stage of development wherein knowledge and techniques for programming are yet to be developed.

Recently, the oritatami model of RNA co-transcriptional folding has been introduced [5]. In this model, an abstract RNA sequence folds upon itself greedily while being synthesized in order to achieve various computational tasks *in-silico*. These tasks are usually relatively simple from the viewpoint of computational complexity theory, such as, the detection of some specific molecule for gene expression regulation [15].

The cyclic variant of tag systems (cts) introduced by Cook [3] has been simulated in the oritatami model in order to prove its Turing completeness [5]; periodicity supposed for the sequence to be folded in the oritatami computation also favored this variant. With more tools available for oritatami programming including finite state control [12] and the molecular implementation of the oritatami model within view, a class of tag systems or their functional enhancements with extra features that are not as computationally powerful as the Turing machine gets more significant.

Including states enables even 1-tag systems to simulate Turing machines and they characterize the class of context-sensitive languages if all appended words are restricted to be of length at most 1, that is, the word never gets strictly longer than the initial one. This was shown a long time ago [17], with such 1-tag systems with states being referred to as circular automata. A primary source of computability even under this restriction is that each "cell" of the input tape can be rewritten endlessly in an arbitrary

*Szilárd Zsolt Fazekas was supported by JSPS KAKENHI Grant Number JP23K10976.
†Shinnosuke Seki was supported by JSPS KAKENHI Grant-in-Aids for Scientific Research (B) 20H04141 and (C) 20K11672.

Zs. Gazdag, Sz. Iván, G. Kovásznai (Eds.): 16th International Conference on Automata and Formal Languages (AFL 2023)
EPTCS 386, 2023, pp. 82–95, doi:10.4204/EPTCS.386.8

manner. As it is experimentally not trivial to implement arbitrarily-rewritable media, it makes sense to suppose the *freezing property*, under which a letter can be replaced with only a smaller letter according to some pre-determined order.

Many types of machines processing their input using some first-in-first-out storage have been investigated starting with queue automata and various restrictions thereof [9]. Such models use queues *in addition to* their input tape and are generally quite powerful, whereas freezing 1-tag systems (Fr1TASS) are towards the lower end of computational complexity, as we will see. A model that is rather close in spirit to our subject is the iterated uniform finite transducer [8]. Such transducers can simulate freezing 1-tag systems in a straightforward manner, but are much more powerful in the general case due to the lack of the freezing property. However, limiting the number of so-called 'sweeps' performed by these transducers by a function that is linear in the length of the input might produce systems that are similar in accepting power to our tag systems, although the bound on sweeps does not directly translate into the constant bound on the number of rewritings per position. Another recent computing device with a similarity to freezing tag systems is the one-way jumping automaton, which reads and erases letters on a circular tape [1], a behavior that can be simulated by freezing tag systems, but it is easy to see that those automata are strictly weaker than our current model.

In this paper, we explore basic properties of freezing 1-tag systems. In Section 2 we define the model and two different accepting modes borrowed from the theory of pushdown automata. We show that the accepting modes are equivalent in the general case. In Section 3 we start the study of the accepting power of the model. We can prove that it is between the regular and the context-sensitive languages and that it is not included in the context-free class, but at present we cannot show that the inclusion does not hold in the other direction either. Next, in Section 4 we show that the class of languages accepted is closed under Boolean operations and that with a simple idea one can trade off description size for time complexity when constructing systems for intersection or union. In Section 5 we study some fundamental decision problems such as emptiness, universality, equivalence, and show that they are not decidable by reduction to the Post Correspondence Problem. In Section 6 we discuss a restriction of Fr1TASS where the tape alphabet is the same as the input alphabet. In their restricted form these systems are much weaker and we can use a 'computation flattening' argument to prove negative results about systems with accepting state mode. We conclude the paper with some remarks and suggestions for future research in Section 7.

2 Preliminaries

Let Σ be a finite alphabet and Σ^* be a set of all words over Σ including the empty word λ. The length of a word $w \in \Sigma^*$ is the number of letters that occur in w, and is denoted by $|w|$.

A *1-tag system with states* (1TASS) is a string rewriting system denoted by a tuple $(\Sigma, \Gamma, Q, q_0, F, \delta)$, where Σ is a finite input alphabet, Γ is a finite tape alphabet that includes Σ as its subset, Q is a finite set of states including the initial state q_0, $F \subseteq Q$ is a set of accepting states, and $\delta : Q \times \Gamma \to Q \times \Gamma^*$ is a transition function. The transition function is *freezing* if, for all $q \in Q$, $a \in \Gamma$, and $(p,u) \in \delta(q,a)$, we have $|u| \leq 1$ (length-non-increasing), and furthermore, either $u = \lambda$ or $u \leq a$ according to some predefined order $\leq$ over the elements of Γ. In this case, the 1TASS itself is also said to be freezing. We will refer to these systems as Fr1TASS.

A configuration of a freezing 1TASS $M = (\Sigma, \Gamma, Q, q_0, F, \delta, \tau)$ is a pair (q,w) of the current state $q \in Q$ and a current word $w \in \Gamma^*$. Suppose this system is in the configuration $(p, a_1 a_2 \cdots a_n)$ for some $p \in Q$, $n \geq 0$, and $a_1, a_2, \ldots, a_n \in \Gamma$. Then it can transfer to a configuration (q,v) if $(q,b) \in \delta(p,a_1)$ and $v = a_2 \cdots a_n b$; in this case, we write $(p, a_1 \cdots a_n) \to_M (q,v)$. The reflexive and transitive closure of the

relation $\to_M$ is denoted by $\to_M^*$. In the 1TASS, a word can be considered as being written on a cyclic tape along which a finite-state control moves in the clockwise direction while rewriting.

Empty tape vs accepting state. The model above can be introduced with or without deletion, which as we will see shortly, does not make a difference with respect to the accepting power. In the first case we allow transitions in which a letter is replaced by the empty word, effectively erasing the cell from the tape. In this scenario we can define acceptance conditions similar to pushdown automata: the machine accepts when the tape is empty (ET) or when the machine enters an accepting state (AS). Formally, starting from an initial configuration (q_0, w), M can accept an input word $w \in \Sigma^*$ in two different modes: *by an accepting state* if $(q_0, w) \to_M^* (q_f, v)$ for some accepting state $q_f \in F$ and a word $v \in \Gamma^*$, while *by the empty tape* if $(q_0, w) \to_M^* (q, \lambda)$ for some $q \in Q$. Where the distinction is necessary, the languages accepted by the system M by accepting state and by empty tape will be denoted by $L(M)_{AS}$ and $L(M)_{ET}$, respectively. Note that the AS mode as defined here introduces a technical problem: a system in this mode either does not accept the empty word or it accepts every input. This is due to fact that accepting the empty word requires that the initial state is final, as well. However, then such a system accepts any input right away as it is already in a final state. Therefore, to simplify the presentation we allow AS mode systems to have a special transition from the initial state to a final state on reading λ and require that such transition is only used when the input is empty. This allows stating our results in a more general form without emphasizing this caveat whenever talking about AS mode Fr1TASS.

First we show that the two conditions lead to the same computational power, which simplifies our exposition further on as we will not have to specify the accepting mode. The following technical lemma states that AS mode machines do not actually need to erase any symbol from their tape.

Lemma 1. *For any Fr1TASS $A = (\Sigma, \Gamma_1, Q_1, q_1, F_1, \delta_1)$ there exists a Fr1TASS $B = (\Sigma, \Gamma_2, Q_1, q_1, F_1, \delta_2)$ such that $L(A)_{AS} = L(B)_{AS}$ and the transition function of B does not erase symbols, that is, $\delta_2(q,a) = (q', \lambda)$ is not allowed for any $q, q' \in Q$ and $a \in \Gamma_2$.*

Proof. The proof is straightforward. If A does not erase symbols, then $B = A$ concludes the argument. If it does, then B simulates all non-erasing transition of A and for each erasing transition of A of the form $\delta_1(q,a) = (q', \lambda)$ we set $\delta_2(q,a) = (q', \square)$, where $\square \in \Gamma_2 \setminus \Gamma_1$ is a new symbol standing in for the positions erased by A. We set $\square$ to be the smallest letter in Γ_2 and we add a loop $\delta_2(q, \square) = (q, \square)$ to each state q, ensuring that B performs the same computations as A. □

We now show that accepting modes ET and AS are equivalent.

Lemma 2 (ET simulates AS)**.** *For any Fr1TASS $A = (\Sigma, \Gamma_1, Q_1, q_1, F_1, \delta_1)$ there exists a Fr1TASS $B = (\Sigma, \Gamma_2, Q_1, q_1, F_1, \delta_2)$ such that $L(A)_{AS} = L(B)_{ET}$.*

Proof. In this case the ET machine will be almost identical to the AS one. To get the ET machine accepting the same language, we simply add erasing loops $\delta(f,a) = (f, \lambda)$ to all accepting states f for all tape symbols a. By the definition of AS, when the machine reaches an accepting state, it halts, so we may assume w.l.o.g. that in A there are no outgoing transitions from any accepting state, so adding the aforementioned transitions does not introduce non-determinism. If the AS machine reaches a final state by an input, the same input will take the ET machine into the same state, whereby it will erase all the remaining symbols from the tape. Conversely, by Lemma 1 we may assume that A never erases its tape, therefore the only words leading to an empty tape in B will be the ones accepted by A. □

Lemma 3 (AS simulates ET)**.** *For any Fr1TASS $A = (\Sigma, \Gamma_1, Q_1, q_1, F_1, \delta_1)$ there exists a Fr1TASS $B = (\Sigma, \Gamma_2, Q_1, q_1, F_1, \delta_2)$ such that $L(A)_{ET} = L(B)_{AS}$.*

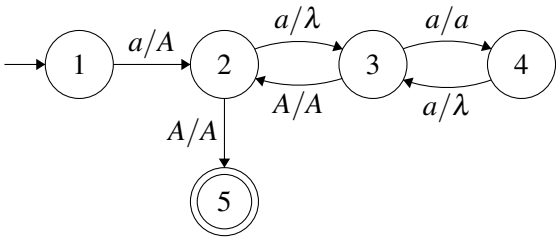

Figure 1: Fr1TASS accepting the language $\{a^{2^n} \mid n \geq 0\}$.

Proof. Similar to the argument in Lemma 1, we can replace erasing transitions $\delta_1(q,a) = (q',\lambda)$ with $\delta_2(q,a) = (q',\square)$ for some newly introduced smallest letter $\square$, and add $\square$-labeled loops to all states. We create two copies of the tape alphabet of A, marked and unmarked. Because of this, when the computation begins, B can mark the first letter to keep track of the start of the input. All operations on that symbol will be done with marked symbols. We duplicate the whole state diagram of A, such that the two copies of the states will 'remember' whether some symbol other than $\square$ was read since last passing the marked start. If the marked start is read twice with no non-$\square$ symbol in between, it means that on the given input A erased the tape, so B will transition to an accepting state. $\square$

2.1 Examples

Example 1. Even over a unary alphabet the ability to repeatedly read the tape allows freezing 1TASS to accept complex languages, such as the well-known non-context-free language $\{a^{2^n} \mid n \geq 0\}$. The tag system in Fig. 1 is intuitive and is essentially the same as for iterated uniform finite state transducers ([8], Lemma 20). The system erases every second occurrence of a which means that in each sweep it halves the length of the remaining tape content. Together with marking the first position with a special symbol at the start this allows Fr1TASS to process correct inputs in logarithmically many sweeps in the length of the input.

Example 2. Our next example is the marked copy language $\{\#w\#w \mid w \in \{a,b\}^*\}$. It is well known that the language is not context-free by a simple application of the Bar-Hillel pumping lemma. A simple Fr1TASS as in Fig. 2 can accept this language by matching and erasing pairs of letters at the same distance from the two special markers # iteratively, accepting the language in linearly many sweeps.

Example 3: accepting nondeterministic context-free languages. As will be detailed later, we were not able to prove that Fr1TASS cannot accept the language of palindromes, even though we conjecture that is the case. Our next idea was that perhaps such systems cannot even detect positions of the input at a certain ratio of the length from the beginning. Somewhat surprisingly, though, this proved to be false. Let us explain how a freezing 1TASS can detect the center of a given input. The idea can be adapted to detect positions at other linearly defined distances from the start. The problem is formalized as follows: modify a given a freezing 1TASS so that, given an input $w = a_0a_1\cdots a_{n-1}$ of length $n \geq 0$, it can mark $a_{\lfloor n/2 \rfloor}$ as a preprocess.

Solving this problem is equivalent to marking the letters in the first half somehow. Let $k = \lfloor n/2 \rfloor$. The following algorithm first marks k letters of w (Step 2), and then move each mark "rightward" across the first letter a_0, which is distinguished from the other letters (Step 1).

1. Mark the first letter as $s_0a_1\cdots a_{n-1}$.
2. Mark every other letter as $s_0\overline{a_1}a_2\overline{a_3}\cdots$. This results in $s_0\overline{a_1}a_2\overline{a_3}\cdots a_{n-2}\overline{a_{n-1}}$ if n is even, or $s_0\overline{a_1}a_2\overline{a_3}\cdots\overline{a_{n-2}}a_{n-1}$ if n is odd.

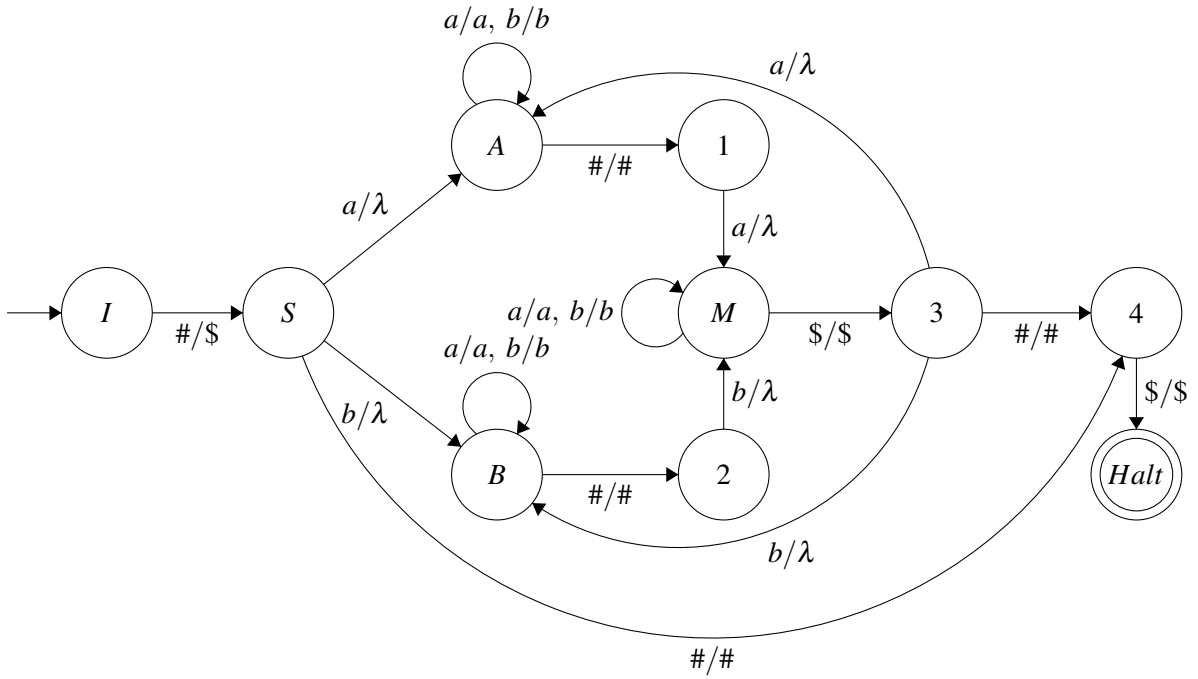

Figure 2: Fr1TASS accepting the language $\{\#w\#w \mid w \in \{a,b\}^*\}$.

The following steps will be repeated until step 3 is unsuccessful, i.e., no overlined position is preceded by an unoverlined one. The read-write head starts at s_0, after finishing steps 1 and 2.

3. Find the first overlined letter, say $\overline{a_j}$, after at least one unoverlined one.
4. Remove the overline as $\overline{a_j} \to a'_j$ (prime is must due to the freezing property).
5. Scan the tape until s_0, which might be overlined.
6. Overline the first unoverlined letter, which might have been primed.

One overline per iteration is shifted to the first half of the word. After repeating Steps 3-6 k times, Step 3 fails to find an unoverlined letter prior to an overlined one; thus the system escapes from this loop. At this point, the k overlines have been all moved to the left half of the word as $\overline{s_0}\overline{a'_1}\overline{a_2}\overline{a'_3}\cdots\overline{a'_{k-1}}a_k\cdots a_{n-1}$. The first unoverlined letter is $a_{\lfloor n/2 \rfloor}$.

3 Power of Fr1TASS

Let $\Sigma = \{1,\ldots,k\}$ be the ordered alphabet of a 1TASS A and consider an input $w = a_1\cdots a_n$, where $a_i \in \Sigma$. We can perceive the computation performed by A as happening in 'sweeps' on a circular tape. A sweep starts at the first position and ends when we reach that position again. Formally, the length of a *sweep* of computation can be defined inductively. Let $r_1 = n$ and then r_i (for $i > 1$) is defined as the length of the remaining input w_i after $i-1$ sweeps, i.e., if $m = \sum_{\ell=1}^{i-1} r_\ell$, then for some $q \in Q$ we have $(q_0, w) \to^m (q, w_i)$ and we set $r_i = |w_i|$. Assuming that the computation stops, the freezing property of A imposes that one of the following is true for each $i > 1$:

1. $r_i < r_{i-1}$.

2. $w_{i-1} = b_1 \cdots b_{r_i}$ and $w_i = c_1 \cdots c_{r_i}$ with $c_j \leq b_j$ for each $j \in \{1,\ldots,r_i\}$, and there exists some $j \in \{1,\ldots,r_i\}$ such that $c_j < b_j$.
3. $w_{i-1} = w_i$, but the sweeps $i-1$ and i start in different states.

From here, we can put an upper bound on the number of sweeps in a computation, and therefore an upper bound on the number of steps. Case 1 can happen at most n times. From Case 3 we get that there can be at most $|Q|$ consecutive sweeps that do not change the tape. The number of times Case 2 occurs is bounded by the total number of possible rewritings. If the tape content is $w = b_1 \cdots b_n$ with $b_i \in \{1,\ldots,k\}$ for all $1 \leq i \leq n$, then each letter can be rewritten to a smaller letter at most $k-1$ times, which means that Case 2 cannot occur more than $\sum_{\ell=1}^{n} b_\ell$ times, and $\sum_{\ell=1}^{n}(b_\ell - 1) \leq n(k-1)$. This means that the number of sweeps performed on an input of length n is $O(n)$. In each sweep we make at most n steps to the right and a Turing machine simulating Fr1TASS would need to make at most n steps to the left at the end of each sweep to return to the beginning. Thus we can conclude that the class of languages accepted by freezing 1TASS is included in DTIME(n^2).

The class of languages accepted by Fr1TASS strictly includes the class of regular languages. We can see that the inclusion is strict (even for unary languages) from the examples of the previous section. In order to show that the inclusion holds, we simulate a deterministic finite automaton by an AS mode Fr1TASS. The construction is simple but we need to bear in mind the fact that if the Fr1TASS reaches a final state reading a word w consisting only of symbols of the input alphabet, that results in the system accepting any word from the language $w\Sigma^*$ according to the definition.

Lemma 4. *For any regular language R there exists a Fr1TASS A such that $L(A)_{AS} = R$.*

Proof. Let $M = (\Sigma, Q, q_0, F, \delta)$ be a deterministic finite automaton accepting R. We construct the Fr1TASS $A = (\Sigma, \Sigma \cup \{\Box\}, Q \cup \{f_A\}, q_0, \{f_A\}, \delta')$, where $\Box \notin \Sigma$, $f_A \notin Q$, as follows. For each transition $\delta(q,a) = q'$ of the DFA, the Fr1TASS has a transition $\delta'(q,a) = (q', \Box)$. For each $f \in F$ we add the transitions $\delta'(f, \Box) = (f_a, \lambda)$. The system A walks the same path in the transition diagram as M does, but it replaces each letter by the $\Box$ symbol to mark it read. If reading the input takes the original system to a final state then the simulating Fr1TASS will have a $\Box$-labeled transition to a final state of its own. $\Box$

Due to the AS and ET modes being equivalent, we can also construct an ET mode Fr1TASS for any regular language. At the other extreme, it is easy to see that linear bounded automata can simulate Fr1TASS, which means that the class of languages accepted by these systems is included in the class of context-sensitive languages. With respect to the other classes of the Chomsky hierarchy, we conjecture that the power of Fr1TASS is incomparable, but we lack the tools to show both sides of such statements. The examples from the previous section demonstrate that not every Fr1TASS language is context-free. We found the ability of Fr1TASS to mark the middle letter of a word counterintuitive, due to the fact that although (even one turn) pushdown automata can accept the related language of words with a in the middle, it needs non-determinism to do so, as we will prove below. As Fr1TASS are deterministic, they cannot guess the middle and match the length of prefixes and suffixes. Nevertheless, nondeterminism is not required for Fr1TASS to mark the middle as shown in Example 3.

Consider $L_a = \{uav \mid |u| = |v|\}$. As we will show now, this language is not a deterministic context-free language. We will use the so-called DCFL pumping lemma below, due to Yu [16].

Lemma 5. *Let L be a deterministic context-free language. Then there exists a constant n for L such that for any pair of words $w, w' \in L$ if*

1. *$w = xy$ and $w' = xz$, $|x| > n$, and*
2. *first letter of y = first letter of z,*

then either 3. or 4. holds:

3. *there is a factorization $x = x_1x_2x_3x_4x_5$, with $|x_2x_4| \geq 1$ and $|x_2x_3x_4| \leq n$, such that for all $i \geq 0$ we have that $x_1{x_2}^i x_3{x_4}^i x_5 y$ and $x_1{x_2}^i x_3{x_4}^i x_5 z$ are in L;*
4. *there exist factorizations $x = x_1x_2x_3, y = y_1y_2y_3$ and $z = z_1z_2z_3$, with $|x_2| \geq 1$ and $|x_2x_3| \leq n$, such that for all $i \geq 0$ we have that $x_1{x_2}^i x_3 y_1{y_2}^i y_3$ and $x_1{x_2}^i x_3 z_1{z_2}^i z_3$ are in L.*

Theorem 6. *$L_a = \{uav \mid |u| = |v|\}$ is not a deterministic context-free language.*

Proof. Suppose that L_a were a DCFL and hence, that Lemma 5 applied. Let n be the constant from the lemma and consider the words $x = b^{n+2}ab^{n+1}$, $y = a$ and $z = ab^{2n+4}$. It is easy to see that both xy and xz are in L_a and long enough to meet the two conditions of the lemma, and the first letter of both y and z is a. Now we will show that assuming either conclusion of the lemma leads to a contradiction. First, suppose conclusion 3. is true. Depending on the factorization of x, we have the following cases:

1. $x_1, x_2, x_4, x_5 \in b^*$, $x_3 \in b^*ab^*$: for $x_1x_2^0x_3x_4^0x_5y$ to be in L_a, we need $|x_2| = |x_4|$ and from the lemma we know they are not empty, so let $x_2 = x_4 = b^k$ for some positive $k \leq \frac{n}{2}$. However, then $x_1x_2^0x_3x_4^0x_5z = x_1x_2^0x_3x_4^0x_5ab^{2n+4}$, where the length of $x_1x_2^0x_3x_4^0x_5 = 2n+4-2k$, so the letter at the middle of the word is b, contradicting $x_1x_2^0x_3x_4^0x_5z \in L_a$.
2. x_1 or x_5 contains a. In both cases $x_1x_2^0x_3x_4^0x_5y$ will result in a word with fewer b's on one side of the first a than the other, a contradiction.
3. x_2 or x_4 has an a. In both cases $x_1x_2^0x_3x_4^0x_5y$ will have only one a, at the end of the word, a contradiction.

Now suppose conclusion 4. is true. Since the factorization $x = x_1x_2x_3$ has the property $|x_2x_3| < n$ and $|x_2| \geq 1$, we know that $x_2 = b^k$, for some positive $k \leq n$. However, this means that for any factorization $y = y_1y_2y_3$, the word $x_1x_2^0x_3y_1y_2^0y_3$ is of the form $b^{n+2}ab^{n+1-k}$ or $b^{n+2}ab^{n+1-k}a$. As neither of those is in L_a, because $k \geq 1$, we arrived at a contradiction again. Consequently, L_a is not a deterministic context-free language. □

4 Closure properties

We can show that the class of languages accepted by Fr1TASS forms a Boolean algebra, i.e., it is closed under union, intersection and complement. For the first two, we can adapt the classical construction used in the case of finite automata: the machine simulating the union/intersection of two others will have pairs of states representing the states of the starting machines and its alphabet will also consist of pairs of letters, to keep track of the tape of both simulated machines.

Theorem 7. *The class of languages accepted by Fr1TASS is closed under union and intersection.*

Proof. Consider two languages, accepted by $A = (\Sigma, \Gamma_1, Q_1, q_1, F_1, \delta_1)$ and $B = (\Sigma, \Gamma_2, Q_2, q_2, F_2, \delta_2)$, respectively. We construct the system

$$C = (\Sigma, \Sigma \cup (\Gamma_1 \times \Gamma_2), Q_1 \times Q_2, (q_1, q_2), F_1 \times F_2, \delta)$$

accepting the language $L(A) \cap L(B)$ as follows. The computation of C will simulate the computations of A and B in parallel, similar to the classical finite automaton construction. By Lemma 1 we may assume that the systems A and B do not erase any symbols, so the length of the word on the tape is the same throughout the computation, making the parallel simulation possible. The difference is that here we have

to observe the freezing property, so the ordering of the tape alphabet Γ and the transition function δ need to be carefully defined. Let the total orderings of Γ_1 and Γ_2 be $\leq_1$ and $\leq_2$, respectively. Those two total orderings naturally define the partial order $\leq_{12}$ on $\Gamma_1 \times \Gamma_2$ as $(a,b) \leq_{12} (c,d)$ if $a \leq_1 c$ and $b \leq_2 d$. By Szpilrajn's theorem [14], every partial order has a linear extension, and we can efficiently construct such a linear order compatible with $\leq_{12}$ by any well-known topological sorting algorithm (e.g. Kahn's [7]), since $\leq_1$ and $\leq_2$ are finite. We define the transition function of C as $\delta((q_1,q_2),a) = ((q_1',q_2'),(b_1,b_2))$ where

$$((q_1',b_1),(q_2',b_2)) = \begin{cases} (\delta_1(q_1,a),\delta_2(q_2,a)) & \text{if } a \in \Sigma \\ (\delta_1(q_1,a_1),\delta_2(q_2,a_2)) & \text{if } a = (a_1,a_2) \notin \Sigma \end{cases}.$$

By the definition of δ we can be certain that the freezing property is preserved, that is, symbols of Γ_2 are rewritten respecting the linear extension of $\leq_{12}$. The proof for the union follows the same line with some small changes. Since the computations are done in parallel, it may happen that one of the machines gets stuck, i.e., it has no outgoing transition from its current state for the current input letter. However, if the other machine accepts, the input should be accepted. To handle such situations we introduce pairs of states $(q_1,\perp)$ and $(\perp,q_2)$ for all $q_1 \in Q_1, q_2 \in Q_2$, where having $\perp$ as one of the state components means the respective machine could not continue its computation. We can reach such states by transitions $\delta((q_1,q_2),a) = ((q_1',\perp),b)$ when $\delta_1(q_1,a) = (q_1',b)$ and $\delta_2(q_2,a)$ is undefined, and then add transitions of the form $\delta((q_1',\perp),a) = ((q_1'',\perp),(b,b))$ if $\delta_1(q_1',a) = (q_1'',(b,b))$ and their counterpart for the $(\perp,q_2)$ states. This way the state component tracking the stuck machine's state will be frozen while the other can continue the computation. □

Theorem 8. *The class of languages accepted by Fr1TASS is closed under complement.*

Proof. For any Fr1TASS that halts on all inputs, it is enough to switch final and non-final states to accept the complement of its language. However, these systems may go into infinite loops, so a system accepting the complement needs to be able to detect that behavior. Each Fr1TASS can be completed with a 'sink state', that is a state from which no other is reachable, and we can direct the transitions for any previously missing state-letter pair into that state. Additionally, we make $n+1$ copies of each state p, say, $p_1,\ldots,p_{n+1}$, where n is the number of states originally. For each i, the states indexed with i are connected among them according to the original transition function, that is $\delta'(p_i,a) = (q_i,b)$ if $\delta(p,a) = (q,b)$, where δ and δ' are the transition functions of the original machine and of the machine for the complement, respectively. In the beginning, we mark the start of the input word with a special symbol to be able to keep track of it. Since the first symbol may need to be changed during the computation of the original machine, we add a marked copy of the original tape alphabet which will only be used to rewrite the first position. Whenever we read the start symbol in some state p_i, we continue on the p_{i+1} states until one of two things happens:

- We change one of the cells on the tape. In this case we continue the computation on the p_1 copies of the states until we reach the start mark.
- We reach the start mark from a p_{n+1} state. This means that the machine made n sweeps without changing any cell on the tape, so the original machine would go into an infinite loop. Instead, here we can simply transition to the sink state defined earlier.

The machine for the complement will have the newly introduced sink state as its only final state. □

Interestingly, the state complexity of intersection and union can be reduced at the expense of time complexity. This is because we can process the input twice instead of simulating both machines in

parallel. First we process it according to the rules of the first machine, and then do so according to the rules of the second one. In order to do this, the number of states in the simulating system only needs to be the sum of the size of state sets of the two starting machines, instead of their product. Moreover, if we 'recycle' the states, the size of the machine for the intersection/union need not increase beyond a constant plus the size of the larger machine participating in the intersection/union. When constructing the machine C to accept $L(A) \cap L(B)$ (or $L(A) \cup L(B)$, respectively), we can reuse the states of the machine by having a tape alphabet with two tracks, say blue and red. Then, we can draw the red transitions completely independent of the blue transitions using the same states as vertices, therefore realizing C on $\max\{|A|,|B|\}+k$ states. The additive constant term k is needed, because after we finish simulating the first machine, we need to freeze the first track of the tape which requires some extra states to cycle through the input and mark each position frozen in the first track. We need to keep track of whether the first machine accepted or rejected the input. We can achieve this without extra states, though, by performing short-circuit evaluation: if the operation is intersection and the first machine rejects then we can reject right away; hence, if the simulation continues to the second machine, we know the first was accepted. The case for union can be treated analogously.

Regarding the regular operations concatenation and Kleene-star, the class of languages accepted by Fr1TASS is probably not closed, but we do not have the tools at present to prove that. In particular, we do not have any necessary conditions for a language to be accepted by Fr1TASS beyond the time complexity bound $O(n^2)$ mentioned earlier, and that bound is not enough to prove negative results regarding closure. The reason we think that the class is not closed under concatenation and Kleene-star is that in general such closure results require either non-deterministically guessing a decomposition of the input into factors of the constituent languages or the possibility of trying all possible decompositions. Neither option seems possible with Fr1TASS.

5 Decision problems and minimal Fr1TASS

Using a construction similar to the freezing 1TASS accepting $\{\#w\#w \mid w \in \Sigma^*\}$, we will show how to reduce the Post Correspondence Problem (PCP) to the emptiness of freezing 1TASS languages. From that we can deduce that emptiness, universality ($= \Sigma^*$) and equivalence are undecidable for this model. We will argue that the undecidability of equivalence also strongly suggests that finding minimal freezing 1TASS for a given language cannot be algorithmically accomplished.

An instance of PCP consists of two sets of words $U = \{u_1,\ldots,u_n\}$ and $V = \{v_1,\ldots,v_n\}$ and the instance is positive if there exists some finite sequence $k_1,\ldots,k_\ell$ (a solution), with $k_i \in \{1,\ldots,n\}$, such that $u_{k_1}\cdots u_{k_\ell} = v_{k_1}\cdots v_{k_\ell}$. It is a well-known fact that it is undecidable whether an instance of PCP is positive ([13]).

Let us fix the alphabet of the PCP instance as Γ, that is, $U,V \subseteq \Gamma^*$, and let $\Gamma' = \{1,\ldots,n\} \cup \Gamma$. The alphabet of the machine will be $\Sigma = \{\#\} \cup \bigcup_{a\in\Gamma'}\{a,\overline{a}\}$. Choose any ordering of the alphabet such that $a < \overline{a}$ for each $a \in \Gamma'$. We construct a freezing 1TASS that accepts the language

$$\{\#k_1\cdots k_\ell\#u_{k_1}\cdots u_{k_\ell}\#v_{k_1}\cdots v_{k_\ell} \mid u_{k_1}\cdots u_{k_\ell} = v_{k_1}\cdots v_{k_\ell}\},$$

where $k_i,u_i,v_i \in (\bigcup_{a\in\Gamma'}\{\overline{a}\})^*$. The machine needs to check whether the input satisfies the following three conditions: (1) the middle part is indeed $u_{k_1}\cdots u_{k_\ell}$, (2) the last part is indeed $v_{k_1}\cdots v_{k_\ell}$ and (3) check whether $u_{k_1}\cdots u_{k_\ell} = v_{k_1}\cdots v_{k_\ell}$. As (2) can be done the same way as (1) and in parallel to it, and (3) has been illustrated before as the machine for $\{\#w\#w\}$, we will only detail (1). Figure 3 illustrates the part of the system for checking (1). Since the factor between the second and third # and the one after the

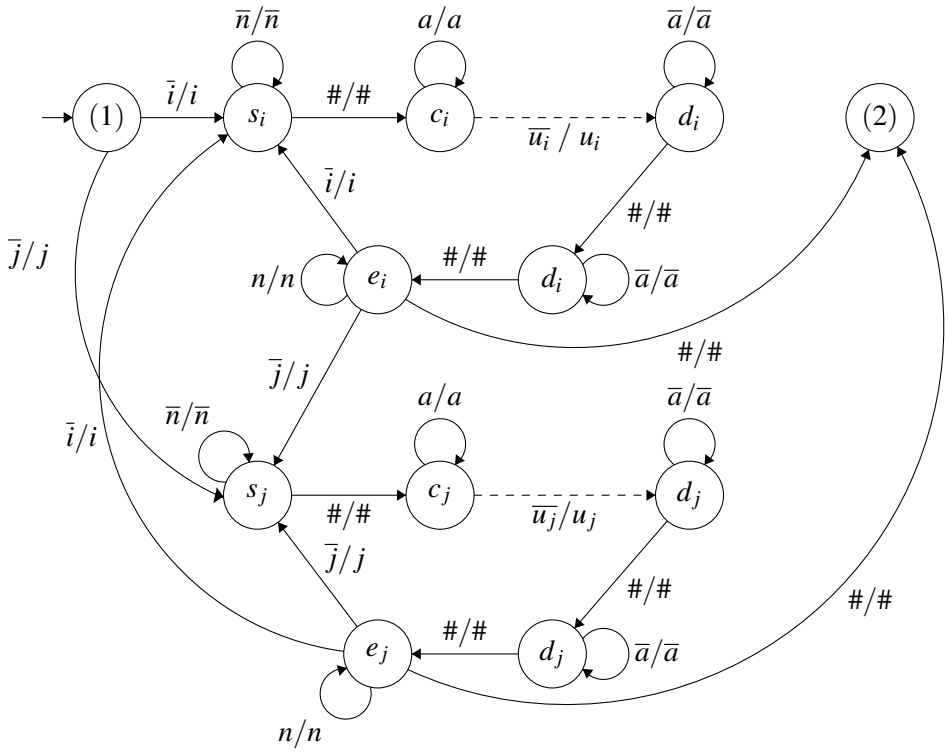

Figure 3: The parts of the 1TASS for matching the first portion of the input containing the numbers, $k_1 \cdots k_\ell$, to the second portion, $u_{k_1} \cdots u_{k_\ell}$. If the number read is i, that is, the symbol $\overline{i}$, then we continue from s_i, if it is $\overline{j}$ then continue from s_j, and so on. Then, the machine looks for the # symbol after which it ignores the already matched parts of $u_{k_1} \cdots u_{k_\ell}$. Finding the first unmatched symbols, it matches them against u_i, after which it returns to the beginning and reads the next number.

third # needs to be checked twice, first for (1) and (2), respectively, then for (3), all the input except the separators # needs to be marked by overlines at the beginning.

1. Checking whether a word equals $u_i = x_1 \cdots x_m$ is easy: we set up $m+1$ states $q_0, \ldots, q_m$ such that $\delta(q_i, \overline{x_i}) = (q_{i+1}, x_i)$. For all $a \neq x_i$, the state q_i has no outgoing transitions, therefore immediately rejecting the input on reading those letters.
2. We read the first unmatched number after the first #, say i. We move without changing over all following symbols until we reach the next #. Then move over all matched symbols, i.e., symbols without overline. From the first symbol with overline, we match u_i to the input, as above. If successful, move over all following symbols until we meet the second #. Move over all symbols without overline and start the process again.

This 1TASS will accept the solutions to the PCP instance, if any. Since PCP is undecidable, deciding whether the language accepted by a freezing 1TASS is empty, is also undecidable. This means that language equivalence is undecidable: if we let freezing 1TASS A and B be such that A does not accept any input, while B accepts the solutions of a PCP instance, then deciding equivalence amounts to deciding whether the PCP instance is positive. Similarly, if we let $L(C) = \Sigma^* \setminus L(B)$, where B accepts the solutions to a PCP instance, then a decision algorithm that could tell whether $L(C) = \Sigma^*$, would decide whether

the PCP instance has solutions, so universality is also undecidable.

Minimization of 1TASS. From the undecidability of the language equivalence, we can draw certain conclusions regarding the minimization of such systems. Say we define minimal 1TASS as ones having the fewest number of states and/or transitions. We instantly get that the following statements cannot both be true, otherwise equivalence would be decidable by the same isomorphism checking method as for DFA:

1. For each freezing 1TASS A there is a unique (up to renaming the states) minimal 1TASS B with $L(A) = L(B)$.
2. There is an algorithm to find for each freezing 1TASS A a minimal freezing 1TASS B with $L(A) = L(B)$.

If 1. holds then we cannot find the unique minimal system. Therefore we could assume that 1. does not hold and try to devise and algorithm for finding *a* minimal system.

Another possibility is to define minimal systems more tightly, in which case minimization algorithms might exist. We suggest the following possible alternative definitions for a freezing 1TASS A to be minimal:
1. A does not contain strongly equivalent states, i.e., states p,q such that $\delta(p,a) = \delta(q,a)$ for all $a \in \Sigma$. This case is straightforward to deal with along with any unreachable states, but yields little information about the similarity of Fr1TASS.
2. No proper subset of the system (removed transitions or states) accepts $L(A)$. Even the question whether minimality is decidable under this definition is nontrivial, let alone finding such a minimal system for a given Fr1TASS.

6 Fr1TASS with no auxiliary symbols

In this section we look at Fr1TASS that cannot have 'auxiliary' symbols (which cannot appear in the input, but can occur on the tape during the computation), that is, $\Sigma = \Gamma$. This type of restriction leads to dramatic changes in computing power even in the case of machines that can rewrite cells arbitrarily many times [6]. For these systems we can show that AS mode is incomparable to ET mode. Simulating AS with ET mode as done in the general case in Lemma 2 does not work. This is because we can no longer assume that the AS mode machines do not erase their tapes, as the technique used in Lemma 1 is not applicable anymore due to the lack of symbols that can stand in for erased ones. In fact, unary languages provide the proof that under the no-auxiliary-symbols restriction, AS and ET are incomparable.

Lemma 9. *For each Fr1TASS $A = (\{a\}, \{a\}, Q, q_0, F, \delta)$, the language $L(A)_{AS}$ accepted with accepting state is of the form $\{a^n \mid n \geq k\}$ for some fixed k, and the language $L(A)_{ET}$ accepted with empty tape is either finite or equal to a^*.*

Proof. Just like in the case of deterministic finite automata, such systems A have a transition diagram of a loop with a 'handle', due to determinism. There are two types of transitions possible: erasing, that is, $\delta(q,a) = (q', \lambda)$ and non-erasing, that is, $\delta(q,a) = (q', a)$. In AS mode the system accepts and halts as soon as it reaches a final state which means that any input longer than the distance from the initial state to the first final state will be accepted. In fact, any input with more letters than the number of erasing transitions on the path from initial to final state will also be accepted. In ET mode for the system to

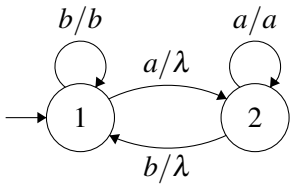

Figure 4: Fr1TASS accepting $\{w \mid |w|_b \leq |w|_a \leq |w|_b + 1\}$ without auxiliary symbols in ET mode.

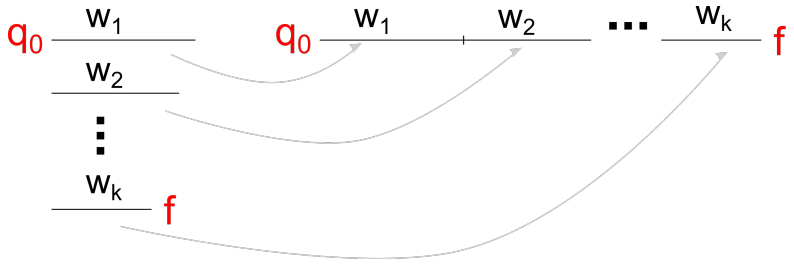

Figure 5: Flattening the computation of an AS mode system with no auxiliary symbols.

accept anything other than the empty word, it needs to have at least one erasing transition. If the only such transitions are on the 'handle', then the accepted language is finite, since the system can only erase finitely many symbols from the tape. If there is an erasing transition in the loop, then all inputs on which the machine reaches the loop will be accepted, since the machine will keep looping until all letters are erased. □

We can also show that AS mode cannot be strictly stronger than ET mode when the tape alphabet is at least binary. Consider the language $L_{ab} = \{w \mid |w|_b \leq |w|_a \leq |w|_b + 1\}$. A machine in ET mode can easily accept this language by reading an a, erasing it, moving to the right until it finds a corresponding b, erasing it and iterating this process (Fig. 4). However, using a 'computation flattening' argument we can prove that a machine in AS mode cannot accept this language.

Lemma 10. *There is no Fr1TASS* $A = (\Sigma, \Sigma, Q, q_0, F, \delta)$ *such that* $L(A)_{AS} = L_{ab}$.

Proof. Assume there is a Fr1TASS A as above that accepts L_{ab}. Take any $w \in L_{ab}$ and let the word on the tape in sweep i of the accepting computation on w be w_i, as defined in the preliminaries. As the word is accepted, there are finitely many, say k, sweeps. If we concatenate the words in the sweeps, we get $w' = w_1 \cdots w_k$. The obtained word w' is a valid input word, because the system can only use the symbols of the input alphabet. On the input w', the system A reaches the same accepting state as on the input w; thus w' is accepted, and the computation requires a single sweep. Moreover, the same final state is reached on input $w'aa$, too. However, this is a contradiction, since $w'aa$ cannot have the required numbers of letter occurrences if w' did. □

Language of palindromes. A very challenging problem is whether the Fr1TASS model can accept the language of palindromes over a non-unary alphabet. Intuitively the model should not be able to accept such a language for the reason described below, but we do not have a proof for this due to the lack of applicable necessary conditions. To verify whether the input is a palindrome, a machine would need to match pairs of letters at the same distance from the middle or from the start and end, respectively. Moreover, the matched pairs would need to be marked to keep track of which parts still need matching. However, in this model, we can only mark symbols *after* a pattern has been identified. This means that if

we start matching letters at the same distance from the middle, then the machine could not guess which is the next unmatched letter in the left half. Conversely, if the machine matches pairs based on their distance from the left and right end, respectively, then it could not guess the next unmatched letter in the right half of the input.

For the restricted model with no auxiliary symbols in AS mode, we can prove that the language of palindromes cannot be accepted, by using the computation flattening argument seen earlier. Let L_{pal} denote the language of palindromes over the binary alphabet $\{a,b\}$, that is, $L_{pal} = \{w \in \{a,b\}^* \mid w = w^R\}$ where w^R is the reverse of w, that is, if $w = a_1 \cdots a_n$ then $w^R = a_n \cdots a_1$.

Theorem 11. *For any Fr1TASS $A = (\Sigma, \Gamma, Q, q, F, \delta)$ we have $L_{pal} \neq L(A)_{AS}$.*

Proof. Suppose that there were a Fr1TASS $A = (\Sigma, \Gamma, Q, q, F, \delta)$ that accepts L_{pal} in accepting state mode. Consider a long palindrome of the form $a^n bwba^n$, where $n > |Q|$, which is accepted by the system in k sweeps. Again, let w_i denote the word on the tape at the beginning of sweep i. Concatenating those words yields the valid input $w' = w_1 \cdots w_k$, which will be accepted with the same transitions as w, but all in one sweep. Since w' is accepted, it must be a palindrome by our assumption, which means that its suffix must be ba^n. Due to the fact that n is larger than the number of states in A, while reading the suffix a^n, the system must enter some state p more than once, reading a^ℓ for some $\ell \geq 1$, between the first two traversals of p. However, this means that the system accepts also words of the form $w'a^{i\ell}$, for all $i \geq 0$. This results in a contradiction for $i = 1$, because the suffix of $w'a^\ell$ is $ba^{n+\ell}$ while its prefix is $a^n b$. Thus, non-palindromes would be also accepted by A. □

7 Concluding remarks

Apart from the decision problems in Section 5, our results have mostly been positive. To establish the limits of the accepting power of Fr1TASS we need negative results separating Fr1TASS languages from other language classes. Although these systems can process symbols in the same position repeatedly, we think that the freezing property allows some form of a pumping lemma, perhaps in combination with a computation flattening argument seen in Section 6. Obtaining such a tool seems quite challenging and will be our main focus in future studies on the topic.

Perhaps with a tool as described above or adapting the Kolmogorov complexity argument of Li et al. [10], one could prove that the language of palindromes is not a Fr1TASS language. This is intuitively a fundamental limitation of such systems with first-in-first-out (FIFO) nature of processing and such questions have proved interesting in their own right with respect to other FIFO style models [11]. If indeed palindromes cannot be accepted with Fr1TASS, then the language class is in some sense a natural counterpart of the class of context-free languages: membership is decidable efficiently and it contains the FIFO-like copy language instead of the LIFO-like palindromes. Interestingly, if one allows Fr1TASS to have non-determinism, then this FIFO limitation seems to vanish: such Fr1TASS could now guess which letters form pairs at equal distance from a reference point (middle or the ends) and verify the guess by marking symbols. The power of nondeterministic Fr1TASS is another topic worth further exploration in our opinion.

Finally, we would like to mention a computational complexity aspect that could be investigated with respect to Fr1TASS. The computation happens in sweeps and those sweeps are a natural resource to measure as the complexity of a given computation. Based on the amount of this resource used, one can define and study asymptotic complexity classes similarly to the case of iterated uniform finite transducers [8] and one-way jumping finite automata [4].

References

[1] Hiroyuki Chigahara, Szilárd Zsolt Fazekas & Akihiro Yamamura (2016): *One-Way Jumping Finite Automata*. *Int. J. Found. Comput. Sci.* 27(3), p. 391, doi:10.1142/S0129054116400165.

[2] John Cocke & Marvin Minsky (1964): *Universality of Tag Systems with $P = 2$*. *Journal of the ACM* 11(1), pp. 15–20, doi:10.1145/321203.321206.

[3] Matthew Cook (2004): *Universality in Elementary Cellular Automata*. *Complex Systems* 15, pp. 1–40, doi:10.25088/ComplexSystems.15.1.1.

[4] Szilárd Zsolt Fazekas, Robert Mercas & Olívia Wu (2022): *Complexities for Jumps and Sweeps*. *J. Autom. Lang. Comb.* 27(1-3), pp. 131–149, doi:10.25596/jalc-2022-131.

[5] Cody Geary, Pierre-Étienne Meunier, Nicolas Robertabanel & Shinnosuke Seki (2018): *Proving the Turing Universality of Oritatami Co-Transcriptional Folding*. In: *Proceedings of the 29th International Symposium on Algorithms and Computation (ISAAC 2018)*, *LIPIcs* 123, pp. 23:1–23:13, doi:10.4230/LIPIcs.ISAAC.2018.23.

[6] Lane A. Hemaspaandra, Proshanto Mukherji & Till Tantau (2005): *Context-Free Languages Can Be Accepted with Absolutely No Space Overhead*. *Information and Computation* 203(2), pp. 163–180, doi:10.1016/j.ic.2005.05.005.

[7] A. B. Kahn (1962): *Topological Sorting of Large Networks*. *Commun. ACM* 5(11), p. 558–562, doi:10.1145/368996.369025.

[8] Martin Kutrib, Andreas Malcher, Carlo Mereghetti & Beatrice Palano (2022): *Descriptional Complexity of Iterated Uniform Finite-State Transducers*. *Information and Computation* 284, p. 104691, doi:10.1016/j.ic.2021.104691. Selected Papers from DCFS 2019, the 21st International Conference on Descriptional Complexity of Formal Systems.

[9] Martin Kutrib, Andreas Malcher & Matthias Wendlandt (2018): *Queue Automata: Foundations and Developments*, pp. 385–431. Springer International Publishing, Cham, doi:10.1007/978-3-319-73216-9_19.

[10] Ming Li, Luc Longpré & Paul Vitányi (1992): *The Power of the Queue*. *SIAM Journal on Computing* 21(4), pp. 697–712, doi:10.1137/0221042.

[11] J. Andres Montoya (2015): *Open Problems Related to Palindrome Recognition: Are There Open Problems Related to Palindrome Recognition?* *J. Autom. Lang. Comb.* 20(1), p. 5–25, doi:10.25596/jalc-2015-005.

[12] Daria Pchelina, Nicolas Schabanel, Shinnosuke Seki & Guillaume Theyssier (2022): *Oritatami Systems Assemble Shapes No Less Complex Than Tile Assembly Model (aTAM)*. In: *Proceedings of the 39th International Symposium on Theoretical Aspects of Computer Science (STACS 2022)*, *LIPIcs* 219, pp. 51:1–51:23, doi:10.4230/LIPIcs.STACS.2022.51.

[13] Emil L. Post (1946): *A Variant of a Recursively Unsolvable Problem*. *Bull. Amer. Math. Soc.* 52, pp. 264–268, doi:10.1090/S0002-9904-1946-08555-9.

[14] Edward Szpilrajn (1930): *Sur l'extension de l'ordre partiel*. *Fundamenta Mathematicae* 16, pp. 386–389, doi:10.4064/fm-16-1-386-389.

[15] Kyle E. Watters, Eric J. Strobel, Angela M. Yu, John T. Lis & Julius B. Lucks (2016): *Cotranscriptional Folding of a Riboswitch at Nucleotide Resolution*. *Nature Structural and Molecular Biology* 23(12), pp. 1124–1131, doi:10.1038/nsmb.3316.

[16] Sheng Yu (1989): *A Pumping Lemma for Deterministic Context-Free Languages*. *Information Processing Letters* 31(1), pp. 47–51, doi:10.1016/0020-0190(89)90108-7.

[17] Charles Zaiontz (1976): *Circular Automata*. In: *Proceedings of the 14th Annual Southeast Regional Conference (ACM-SE 14)*, pp. 350–354, doi:10.1145/503561.503635.