Reversible Two-Party Computations

Martin Kutrib and Andreas Malcher
Institut für Informatik, Universität Giessen
Arndtstr. 2, 35392 Giessen, Germany
{kutrib,andreas.malcher}@informatik.uni-giessen.de

Deterministic synchronous systems consisting of two finite automata running in opposite directions on a shared read-only input are studied with respect to their ability to perform reversible computations, which means that the automata are also backward deterministic and, thus, are able to uniquely step the computation back and forth. We study the computational capacity of such devices and obtain on the one hand that there are regular languages that cannot be accepted by such systems. On the other hand, such systems can accept even non-semilinear languages. Since the systems communicate by sending messages, we consider also systems where the number of messages sent during a computation is restricted. We obtain a finite hierarchy with respect to the allowed amount of communication inside the reversible classes and separations to general, not necessarily reversible, classes. Finally, we study closure properties and decidability questions and obtain that the questions of emptiness, finiteness, inclusion, and equivalence are not semidecidable if a superlogarithmic amount of communication is allowed.

1 Introduction

Watson-Crick automata have been introduced in [7] as a formal model for DNA computing. The motivation for such automata comes from processes observed in nature and laboratories. Basically, the idea is to consider finite automata with two reading heads that run on either strand of a double stranded DNA-molecule. It is noted in [20] that in nature enzymes moving along DNA strands may obey the biochemical direction of the single strands of the DNA sequence. Hence, so-called $5' \to 3'$ Watson-Crick automata have been introduced in [20], which are two-head finite automata where the heads start at opposite ends of a strand and move in opposite physical directions. It is known that no additional information is encoded in the second strand provided that the complementarity relation of the double stranded sequence is one-to-one. In this case, $5' \to 3'$ Watson-Crick automata share a common input sequence.

Watson-Crick automata and $5' \to 3'$ Watson-Crick automata have intensively been investigated in the last years from different points of view. Descriptional complexity aspects of Watson-Crick automata are studied in [6]. $5' \to 3'$ Watson-Crick automata with several runs, which means that both heads are sweeping between both ends of the input, are investigated in [18] and a hierarchy with respect to the number of runs has been obtained. The aspect of the amount of communication between the two heads that is necessary in accepting computations is highlighted in [12] where $5' \to 3'$ Watson-Crick automata with restricted communication are introduced and a finite hierarchy concerning the amount of communication could be obtained. The concept of sensing heads, where one head can sense the presence of the other head, has been applied to $5' \to 3'$ Watson-Crick automata in [21, 24]. The concept of jumping automata, where the input is processed in a discontinuous way, has been introduced and investigated for $5' \to 3'$ Watson-Crick automata in [9]. Finally, the impact of replacing the underlying devices of finite automata by finite transducers or pushdown automata is studied in [23] and in [5, 22], respectively.

Another line of research in recent years is the study of reversible devices. Here, a computation is considered *reversible* if every configuration has at most one unique successor configuration and at most one

Zs. Gazdag, Sz. Iván, G. Kovásznai (Eds.): 16th International
Conference on Automata and Formal Languages (AFL 2023)
EPTCS 386, 2023, pp. 142–154, doi:10.4204/EPTCS.386.12

unique predecessor configuration. The study of such devices that perform logically reversible computations is motivated by Landauer's question of whether logical irreversibility is an unavoidable feature of useful computers. This question is of particular interest, since Landauer has demonstrated that whenever a physical computer throws away information about its previous state it must generate a corresponding amount of entropy that results in heat dissipation. A detailed discussion and suitable references can be found in [2]. Reversible variants of many computational models have been studied in the literature. For Turing machines the first investigations on reversible computations date back to the sixties of the last century. It is shown in the work of Lecerf [17] and Bennett [2] that it is possible for every Turing machine to construct an equivalent reversible Turing machine. Hence, every irreversible computation can be made reversible. This is no longer true if finite automata are considered. On the one hand, it is known that reversible one-way deterministic finite automata are computationally weaker than one-way deterministic finite automata in general [1] (cf. also [8]). On the other hand, two-way deterministic finite automata and reversible two-way deterministic finite automata are equally powerful [10]. Similar results are known for multihead finite automata. In case of one-way motion, the reversible variant is computationally weaker than the general model ([14]), whereas in case of two-way motion the computational power of the reversible variant and the general model coincides [19]. Several more types of devices as, for example, queue automata [16], one-way counter machines with multiple counters [15], and parallel communicating finite automata [3] have been investigated with respect to reversibility. An overview of the topic is given in [11].

The aspect of reversibility has been studied for Watson-Crick automata in [4]. One result is that every regular language can be accepted by a reversible Watson-Crick automaton. Here, it is essential that the complementarity relation of the double stranded sequence is *not* one-to-one. If the complementarity relation is one-to-one, another result of [4] gives that the computational power of reversible Watson-Crick automata and reversible two-head finite automata ([14]) coincides. In this paper, we study $5' \to 3'$ Watson-Crick automata having a one-to-one complementarity relation and to differentiate the notation from other variants we will call the devices in question *two-party Watson-Crick systems*. This paper can be seen as a continuation of [12] where communication restricted two-party Watson-Crick systems are introduced and a strict four-level hierarchy depending on the number of messages sent was established, where the levels are given by $O(1)$, $O(\log(n))$, $O(\sqrt{n})$, and $O(n)$ messages allowed. Moreover, it could be shown that the questions of emptiness, finiteness, inclusion, and equivalence are not semidecidable, that is, not recursively enumerable, even if the communication is reduced to a limit $O(\log(n) \cdot \log(\log(n)))$. Here, we complement these results. After defining the model and giving two illustrating examples in Section 2 we show in Section 3 that there are regular languages which can not be accepted by any reversible two-party Watson-Crick systems with any amount of communication. This is in strong contrast to general two-party Watson-Crick systems where no communication is necessary to accept regular languages. This result can be used in Section 4 in which closure properties are investigated. It turns out that reversible two-party Watson-Crick systems are closed under complementation and reversal, whereas they are not closed under union, intersection, intersection with regular languages, concatenation, iteration, length-preserving homomorphism, and inverse homomorphism. In Section 5, we can extend the strict four-level hierarchy depending on the number of messages sent from [12] to reversible two-party Watson-Crick systems. Moreover, we obtain that for every level the reversible systems are computationally weaker than the general systems. Finally, we discuss in Section 6 decidability questions. In a first step, we show that the questions of emptiness, finiteness, inclusion, and equivalence are not semidecidable for reversible two-party Watson-Crick systems essentially disregarding the number of messages communicated. In a second step, we refine the argumentation and apply and adapt a result from [14] which enables us to show that the questions of emptiness, finiteness, inclusion, and equivalence are not semidecidable for reversible two-party Watson-Crick systems even if the number of messages allowed is bounded by $O(\log(n) \cdot \log(\log(n)))$.

2 Definitions and Preliminaries

We denote the set of nonnegative integers by $\mathbb{N}$. We write Σ^* for the set of all words over the finite alphabet Σ. The empty word is denoted by λ, and $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$. The reversal of a word w is denoted by w^R and for the length of w we write $|w|$. We use $\subseteq$ for inclusions and $\subset$ for strict inclusions.

A two-party Watson-Crick system is a device of two finite automata working independently and in opposite directions on a common read-only input data. The automata communicate by broadcasting messages. The transition function of a single automaton depends on its current state, the currently scanned input symbol, and the message currently received from the other automaton. Both automata work synchronously and the messages are delivered instantly. Whenever the transition function of (at least) one of the single automata is undefined the whole systems halts. The input is accepted if at least one of the automata is in an accepting state. A formal definition is as follows.

A *deterministic two-party Watson-Crick system* (DPWK) is a construct $\mathscr{A} = \langle \Sigma, M, \rhd, \lhd, A_1, A_2 \rangle$, where Σ is the finite set of *input symbols*, M is the set of possible *messages*, $\rhd \notin \Sigma$ and $\lhd \notin \Sigma$ are the *left and right endmarkers*, and each $A_i = \langle Q_i, \Sigma, \delta_i, \mu_i, q_{0,i}, F_i \rangle$, $i \in \{1,2\}$, is basically a *deterministic finite automaton* with *state set* Q_i, *initial state* $q_{0,i} \in Q_i$, and set of *accepting states* $F_i \subseteq Q_i$. Additionally, each A_i has a *broadcast function* $\mu_i : Q_i \times (\Sigma \cup \{\rhd, \lhd\}) \to M \cup \{\perp\}$ which determines the message *to be sent*, where $\perp \notin M$ means *nothing to send*, and a *(partial) transition function* $\delta_i : Q_i \times (\Sigma \cup \{\rhd, \lhd\}) \times (M \cup \{\perp\}) \to Q_i \times \{0, +\}$, where $+$ means to move the head one square and 0 means to keep the head on the current square.

The automata A_1 and A_2 are called *components* of the system $\mathscr{A}$, where the so-called *upper* component A_1 starts at the left end of the input and moves from left to right, and the *lower* component A_2 starts at the right end of the input and moves from right to left. A *configuration* of $\mathscr{A}$ is represented by a string $\rhd v_1 \overrightarrow{p} x v_2 y \underleftarrow{q} v_3 \lhd$, where $v_1 x v_2 y v_3$ is the input and it is understood that component A_1 is in state p with its head scanning symbol x, and component A_2 is in state q with its head scanning symbol y. System $\mathscr{A}$ starts with component A_1 in its initial state scanning the left endmarker and component A_2 in its initial state scanning the right endmarker. So, for input $w \in \Sigma^*$, the initial configuration is $\overrightarrow{q_{0,1}} \rhd w \lhd \underleftarrow{q_{0,2}}$. A computation of $\mathscr{A}$ is a sequence of configurations beginning with an initial configuration. One step from a configuration to its successor configuration is denoted by $\vdash$. Let $w = a_1 a_2 \cdots a_n$ be the input, $a_0 = \rhd$, and $a_{n+1} = \lhd$, then we set

$$a_0 \cdots a_{i-1} \overrightarrow{p} a_i \cdots a_j \underleftarrow{q} a_{j+1} \cdots a_{n+1} \vdash a_0 \cdots a_{i'-1} \overrightarrow{p_1} a_{i'} \cdots a_{j'} \underleftarrow{q_1} a_{j'+1} \cdots a_{n+1},$$

for $0 \leq i, j \leq n+1$, iff $\delta_1(p, a_i, \mu_2(q, a_j)) = (p_1, d_1)$ and $\delta_2(q, a_j, \mu_1(p, a_i)) = (q_1, d_2)$, $i' = i + d_1$ and $j' = j - d_2$. As usual we define the reflexive, transitive closure of $\vdash$ by $\vdash^*$.

A computation *halts* when the successor configuration is not defined for the current configuration. This may happen when the transition function of one component is not defined. The language $L(\mathscr{A})$ accepted by a DPWK $\mathscr{A}$ is the set of inputs $w \in \Sigma^*$ such that there is some computation beginning with the initial configuration for w and halting with at least one component being in an accepting state.

Now we turn to *reversible* two-party Watson-Crick systems. Basically, reversibility is meant with respect to the possibility of stepping the computation back and forth. So, the system has also to be backward deterministic. That is, any configuration must have at most one predecessor which, in addition, is computable by a two-party Watson-Crick system. In particular for the read-only input tape, the machines reread the input symbol which they have been read in a preceding forward computation step. Therefore, for reverse computation steps the head of the upper component is either moved to the *left* or stays stationary, whereas the head of the lower component is either moved to the *right* or stays stationary. One can imagine that in a forward step, first the input symbol is read and then the input head is moved to its new position, whereas in a backward step, first the input head is moved to its new position and then the input symbol is read.

So, a deterministic two-party Watson-Crick system $\mathscr{A}$ is said to be *reversible* (REV-PWK) if and only if there exist *reverse transition functions* $\delta_i^{\leftarrow} : Q_i \times (\Sigma \cup \{\rhd, \lhd\}) \times (M \cup \{\perp\}) \to Q_i \times \{0, -\}$ and *reverse broadcast functions* $\mu_i^{\leftarrow} : Q_i \times (\Sigma \cup \{\rhd, \lhd\}) \to M \cup \{\perp\}$ inducing a relation $\vdash^{\leftarrow}$ from a configuration to its *predecessor configuration*, such that

$$a_0 \cdots a_{i'-1} \overrightarrow{p_1} a_{i'} \cdots a_{j'} \underleftarrow{q_1} a_{j'+1} \cdots a_{n+1} \vdash^{\leftarrow} a_0 \cdots a_{i-1} \overrightarrow{p} a_i \cdots a_j \underleftarrow{q} a_{j+1} \cdots a_{n+1}$$

if and only if

$$a_0 \cdots a_{i-1} \overrightarrow{p} a_i \cdots a_j \underleftarrow{q} a_{j+1} \cdots a_{n+1} \vdash a_0 \cdots a_{i'-1} \overrightarrow{p_1} a_{i'} \cdots a_{j'} \underleftarrow{q_1} a_{j'+1} \cdots a_{n+1}.$$

In the following, we study the impact of communication in deterministic two-party Watson-Crick systems. The communication is measured by the total number of messages sent during a computation, where it is understood that $\perp$ means no message and, thus, is not counted.

Let $f : \mathbb{N} \to \mathbb{N}$ be a mapping. If all $w \in L(\mathscr{A})$ are accepted with computations where the total number of messages sent is bounded by $f(|w|)$, then $\mathscr{A}$ is said to be *communication bounded by* f. We denote the class of DPWKs that are communication bounded by f by DPWK(f). In case of reversible DPWKs we have to consider the number of messages sent in reverse computations as well. If all $w \in L(\mathscr{A})$ are accepted with computations where the total number of messages sent in forward computations and in reverse computations is each bounded by $f(|w|)$, then $\mathscr{A}$ is said to be *communication bounded by* f and the corresponding class of REV-PWKs is denoted by REV-PWK(f).

In general, the *family of languages accepted* by devices of type X is denoted by $\mathscr{L}(X)$. To illustrate the definitions we start with two examples.

Example 1. The non-regular language $L = \{ a^n b^n \mid n \geq 1 \}$ is accepted by a REV-PWK. The principal idea of the construction is that the upper component starts with one time step delay and then moves its head with maximum speed to the right, whereas the lower component immediately starts to move its head with maximum speed to the left. Both components communicate in every time step the symbol they read. When the lower component has read the rightmost a of the a-block after having passed the b-block, the transition functions ensure that the upper component has to read the leftmost b of the b-block after having passed the a-block. When the lower component has reached the left endmarker, it waits for one time step. To accept the input, the upper component has to read the right endmarker in the final step. In the backward computation the upper component immediately starts, whereas the lower component starts with with one time step delay. When the upper component has read the rightmost a of the a-block after having passed the b-block, the transition functions ensure that the lower component has to read the leftmost b of the b-block after having passed the a-block. Finally, when the upper component has reached the right endmarker, it waits for one time step. To reach the initial configuration the lower component has to read the left endmarker in the next time step.

For the precise construction of a REV-PWK accepting the language $L = \{ a^n b^n \mid n \geq 1 \}$ we define $\mathscr{A} = \langle \{a,b\}, \{a,b,\rhd,\lhd\}, \rhd, \lhd, A_1, A_2 \rangle$ where

$$A_1 = \langle \{p_0, p_1, \ldots, p_5\}, \{a,b\}, \delta_1, \mu_1, p_0, \{p_5\} \rangle \text{ and } A_2 = \langle \{q_0, q_1, \ldots, q_5\}, \{a,b\}, \delta_2, \mu_2, q_0, \{\} \rangle.$$

The broadcast functions μ_1, μ_2 and the reverse broadcast functions $\mu_1^{\leftarrow}, \mu_2^{\leftarrow}$ are defined as $\mu_1(p,x) = \mu_1^{\leftarrow}(p,x) = x$ and $\mu_2(q,x) = \mu_2^{\leftarrow}(q,x) = x$ for all $p \in \{p_0, p_1, \ldots, p_5\}$, $q \in \{q_0, q_1, \ldots, q_5\}$, and $x \in \{a, b, \triangleright, \triangleleft\}$. The transition functions δ_1, δ_2 and $\delta_1^{\leftarrow}, \delta_2^{\leftarrow}$ are as follows.

A_1 forward			
(1)	$\delta_1(p_0, \triangleright, \triangleleft)$	=	$(p_1, 0)$
(2)	$\delta_1(p_1, \triangleright, b)$	=	$(p_2, +)$
(3)	$\delta_1(p_2, a, b)$	=	$(p_2, +)$
(4)	$\delta_1(p_2, a, a)$	=	$(p_3, +)$
(5)	$\delta_1(p_3, b, a)$	=	$(p_3, +)$
(6)	$\delta_1(p_3, b, \triangleright)$	=	$(p_4, +)$
(7)	$\delta_1(p_4, \triangleleft, \triangleright)$	=	$(p_5, 0)$

A_1 backward			
(1)	$\delta_1^{\leftarrow}(p_1, \triangleright, \triangleleft)$	=	$(p_0, 0)$
(2)	$\delta_1^{\leftarrow}(p_2, \triangleright, b)$	=	$(p_1, -)$
(3)	$\delta_1^{\leftarrow}(p_2, a, b)$	=	$(p_2, -)$
(4)	$\delta_1^{\leftarrow}(p_3, a, a)$	=	$(p_2, -)$
(5)	$\delta_1^{\leftarrow}(p_3, b, a)$	=	$(p_3, -)$
(6)	$\delta_1^{\leftarrow}(p_4, b, \triangleleft)$	=	$(p_3, -)$
(7)	$\delta_1^{\leftarrow}(p_5, \triangleleft, \triangleright)$	=	$(p_4, 0)$

A_2 forward			
(1)	$\delta_2(q_0, \triangleleft, \triangleright)$	=	$(q_1, +)$
(2)	$\delta_2(q_1, b, \triangleright)$	=	$(q_2, +)$
(3)	$\delta_2(q_2, b, a)$	=	$(q_2, +)$
(4)	$\delta_2(q_2, a, a)$	=	$(q_3, +)$
(5)	$\delta_2(q_3, a, b)$	=	$(q_3, +)$
(6)	$\delta_2(q_3, \triangleright, b)$	=	$(q_4, 0)$
(7)	$\delta_2(q_4, \triangleright, \triangleleft)$	=	$(q_5, 0)$

A_2 backward			
(1)	$\delta_2^{\leftarrow}(q_1, \triangleleft, \triangleright)$	=	$(q_0, -)$
(2)	$\delta_2^{\leftarrow}(q_2, b, \triangleright)$	=	$(q_1, -)$
(3)	$\delta_2^{\leftarrow}(q_2, b, a)$	=	$(q_2, -)$
(4)	$\delta_2^{\leftarrow}(q_3, a, a)$	=	$(q_2, -)$
(5)	$\delta_2^{\leftarrow}(q_3, a, b)$	=	$(q_3, -)$
(6)	$\delta_2^{\leftarrow}(q_4, \triangleright, b)$	=	$(q_3, 0)$
(7)	$\delta_2^{\leftarrow}(q_5, \triangleright, \triangleleft)$	=	$(q_4, 0)$

We note that it is shown in [13] that $L = \{\, a^n b^n \mid n \geq 1 \,\}$ is not accepted by any reversible pushdown automaton. ∎

Example 2. The non-context-free language $L' = \{\, w\$w^R\$a^{|w|} \mid w \in \{a,b\}^* \,\}$ is accepted by a REV-PWK. Here, the principal idea is that the upper component waits at the left endmarker, while the lower component moves across the a-block. Having reached the second \$, both components move with maximum speed and test the structure $w\$w^R$ by communicating in every time step they read. If no error occurred, the upper component moves to the second \$, while the lower component waits at the first \$. Finally, both components move with maximum speed and test the length of w equals the length of the a-block. The moving of the components in the backward computation is straightforward. ∎

3 Reversibility versus Irreversibility

We now turn to the question of whether reversible two-party Watson-Crick systems are weaker than irreversible ones or not; it turns out that they are. In fact, there are languages accepted by irreversible two-party Watson-Crick systems that do not need any communication which cannot be accepted by any reversible two-party Watson-Crick system regardless of the number of communications. To show this claim, we will use regular witness languages. Let $\Sigma \supseteq \{a,b\}$ be an alphabet and $I \subseteq \Sigma^*$ be regular such that $I = I^R$. Then we define $L_I = \{\, a^{m_1} bvba^{m_2} \mid m_1, m_2 \geq 0, v \in b^* \text{ or } v \in I \,\}$. So, the words in L_I have a nonempty prefix of a's, followed by a b, followed by a factor of b's or a factor from I, followed by a b, followed by a nonempty suffix of a's.

Theorem 3. *Let $\Sigma \supseteq \{a,b\}$ and $I \subseteq \Sigma^*$. Then language L_I is not accepted by any REV-PWK.*

Proof. Assume for the purpose of contradiction that L_I is accepted by some REV-PWK $\mathscr{A}$. Since we do not limit the number of possible communications, we simply assume that both components send a message at every time step. In this case, for the sake of easier writing, we can assume that there is *one common* finite-state control for both components. This control receives a pair of input symbols in every step, changes the state, and moves the components if required. Now we can argue that the system is irreversible if there are two reachable states that have a common successor state for the same pair of input symbols.

We denote this system M, its set of states Q, and its transition function δ. We now consider accepting computations on words $w = a^x b^y a^z \in L_I$, where x,y,z are long enough. In a first phase of such a computation, eventually at least one component has to start to move across the a-prefix or a-suffix. Otherwise the overall computation would loop forever. Since $L_I = L_I^R$, we can safely assume that the upper component moves. The lower component may move across the a-suffix or stay stationary on the endmarker or some a. We choose x and z large enough such that M runs into a state cycle in this phase. Moreover, we choose z that large that the upper component arrives at the first b after the a-prefix before the lower component has passed the a-suffix. Let $p_1, p_2, \dots, p_k$ be the state cycle. We can adjust the length of the prefix such that M moves the upper component on the first b while entering state p_k. So, we have a configuration of the form $p_k\colon \rhd aa\cdots a\overrightarrow{b}b\cdots baa\cdots \underleftarrow{\sigma}\cdots$, where the state of M is written in front of $\rhd$, and $\sigma = a$ or $\sigma = \lhd$, and the components are scanning the symbols indicated by the arrows. Next, we can enlarge z such that M runs again in a state loop while the upper component is reading b's and the lower component is reading $\lhd$ or a's. Assume that the sequence of states passed through is extended from p_k by $p'_1, p'_2, \dots, p'_i, p''_1, \dots p''_j, p''_1$. Then we know $\delta(p'_i,(b,\sigma_1)) = (p''_1, d_1, d_2)$ and $\delta(p''_j,(b,\sigma_2)) = (p''_1, d_1, d_2)$, where d_1, d_2 indicate whether the components are moved or not. Since M is reversible, we derive $p'_1, p'_2, \dots, p'_i, p''_1, \dots p''_j, p''_1 = p_1, p_2, \dots, p_k, p_1$ or $(b,\sigma_1) \neq (b,\sigma_2)$ and, thus, $\sigma_1 \neq \sigma_2$ and, hence, $\sigma_1 = \lhd$ and $\sigma_2 = a$. Dependent on whether the loop on the (a,σ)'s is continued on the (b,σ)'s, or the second possibility, we distinguish two cases. A similar distinction will be made in several sub-cases.

Case A The system M continues to loop through the states $p_1, p_2, \dots, p_k$ while reading (b,σ)'s. Recall that the current state determines the last movements of the components. Therefore, the upper component moves across the b's. Moreover, we can choose y and z again large enough such that the upper component runs through several loops and M moves the upper component on the first a of the suffix while entering state p_k. So, we have a configuration of the form $p_k\colon \rhd aa\cdots abb\cdots b\overrightarrow{a}a\cdots a\underleftarrow{\sigma}\cdots$. Now, we can repeat the argument from above and distinguish the two sub-cases that M continues to loop through the states $p_1, p_2, \dots, p_k, p_1$, or $(a,\sigma_1) \neq (a,\sigma_2)$ and, thus, $\sigma_1 \neq \sigma_2$ and, hence, $\sigma_1 = \lhd$ and $\sigma_2 = a$.

Case A.1 The system M continues to loop through the states $p_1, p_2, \dots, p_k$ while reading (a,σ)'s. In this sub-case the upper component may reach the right endmarker before the lower component reaches the b before the a-suffix. Then the remaining computation of M is that of a finite automaton, that is, of the lower component. Since the language $a^*b^*a^*$ is not accepted by any reversible DFA, we obtain a contradiction.

Therefore, the upper component may reach the right endmarker not before the lower component reaches the b before the a-suffix. Now, again we can repeat the argument from above and distinguish the two sub-cases that M continues to loop through the states $p_1, p_2, \dots, p_k$ while moving the lower component or (a,σ_1) must not be equal to (a,σ_2) which can be violated by adjusting the value of z. In this way $\sigma_1 = \sigma_2 = a$, a contradiction. If, however, M continues to loop through the states $p_1, p_2, \dots, p_k$, by almost the same arguments as before we can obtain a contradiction unless M continues to loop through the states $p_1, p_2, \dots, p_k$ until the lower component has reached the left endmarker. In this case, the language $\{a,b\}^+$ is accepted.

Case A.2 The sequence of states passed through to reach the configuration $p_k\colon \triangleright aa\cdots abb\cdots b\overrightarrow{d}a\cdots a\underleftarrow{\sigma}\cdots$ is extended from state p_k by the states $q'_1, q'_2, \ldots, q'_{i'}, q''_1, \ldots q''_{j'}, q''_1$, and we have $\delta(q'_{i'}, (a, \sigma_1)) = (q''_1, d_1, d_2)$ and $\delta(q''_{j'}, (a, \sigma_2)) = (q''_1, d_1, d_2)$, and therefore $(a, \sigma_1) \neq (a, \sigma_2)$ which implies $\sigma_1 = \lhd$ and $\sigma_2 = a$.

Now, the upper component may or may not reach the right endmarker before the lower component reaches the b before the a-suffix. We obtain a contradiction almost literally as in Case A.1.

Case B The sequence of states passed through to reach the configuration $\triangleright aa\cdots a\overrightarrow{b}b\cdots baa\cdots\underleftarrow{\sigma}\cdots$ in state p_k is extended from p_k by $p'_1, \ldots, p'_i, p''_1, \ldots p''_j, p''_1$. Then we have $\delta(p'_i, (b, \sigma_1)) = (p''_1, d_1, d_2)$ and $\delta(p''_j, (b, \sigma_2)) = (p''_1, d_1, d_2)$, and therefore, $(b, \sigma_1) \neq (b, \sigma_2)$ which implies $\sigma_1 = \lhd$ and $\sigma_2 = a$.

Case B.1 If the upper component moves in the state cycle $p''_1, \ldots p''_j$, then we can choose z again large enough such that the upper component reaches the first a after the b-factor before the lower component reaches the b before the a-suffix. So, a configuration $\triangleright aa\cdots abb\cdots b\overrightarrow{d}a\cdots\underleftarrow{a}\cdots$ is reached in some state from the cycle. We obtain a contradiction along the argumentation as in Case A.1.

Case B.2 If the upper component does not move in the state cycle $p''_1, \ldots p''_j$, then a configuration $\cdots\overrightarrow{b}b\cdots\underleftarrow{b}aa\cdots$ is reached in some state from the cycle.

Assume that from here the computation continues in the same state cycle until the lower component has reached the left endmarker. Then the upper component stays on the current input in this phase, and the remaining computation of M is that of a finite automaton, that is, of the upper component on its remaining input of the form b^*a^*, which is not accepted by any reversible DFA. So, we obtain a contradiction.

We conclude that the computation cannot continue in the same state cycle. If it continues in some state cycle $q'_1, q'_2, \ldots, q'_{i'}, q''_1, \ldots q''_{j'}, q''_1$ while both components read b's, then we have $\delta(q'_{i'}, (b, b)) = (q''_1, d_1, d_2)$ and $\delta(p''_{j'}, (b, b)) = (p''_1, d_1, d_2)$ which violates the reversibility.

If the computation continues in some state cycle $q'_1, q'_2, \ldots, q'_{i'}, q''_1, \ldots q''_{j'}, q''_1$ after at least one component has passed across the b-factor, we obtain a similar contradiction with input pairs (a, b), (b, a), or (a, a).

This concludes the case analysis. Since in any possible case a contradiction is derived, the initial assumption that L_I is accepted by some REV-PWK is wrong and the assertion follows. □

The result of Theorem 3 that there is a regular language that is not accepted by any REV-PWK together with Example 1 showing that the non-regular language $\{\, a^n b^n \mid n \geq 1 \,\}$ is accepted by a REV-PWK proves that the class of languages accepted by REV-PWK and the regular languages are incomparable. Since $\{\, a^n b^n \mid n \geq 1 \,\}$ is a linear and real-time deterministic context-free language, we immediately obtain the incomparability to the linear context-free languages as well as to the real-time deterministic context-free languages. It is shown in [13] that every regular language can be accepted by a reversible pushdown automaton. Moreover, it is shown that the language $\{\, a^n b^n \mid n \geq 1 \,\}$ cannot be accepted by any reversible pushdown automaton. Hence, the classes of languages accepted by REV-PWK and reversible pushdown automata are incomparable as well.

4 Closure Properties

The goal of this section is to collect some closure properties of the families $\mathscr{L}$(REV-PWK). For this purpose, the regular languages L_I can be used very well in several cases. In particular, we consider Boolean operations (complementation, union, intersection) and AFL operations (union, intersection with regular languages, homomorphism, inverse homomorphism, concatenation, iteration). The positive closure under reversal is trivial. The results are summarized in Table 1 at the end of the section.

Proposition 4. *The family $\mathscr{L}$(REV-PWK) is closed under complementation.*

Proposition 5. *The family $\mathscr{L}$(REV-PWK) is not closed under union, intersection, and intersection with regular languages.*

Proof. Let $\Sigma = \{a, b\}$. For $I = \emptyset$, we consider the regular language $L_\emptyset = \{\, a^{m_1} bb^{m_3} ba^{m_2} \mid m_1, m_2, m_3 \geq 0 \,\}$. By Theorem 3, the regular language $L_\emptyset$ does not belong to the family $\mathscr{L}$(REV-PWK). On the other hand, the language Σ^* does belong to the family. Since $\Sigma^* \cap L_\emptyset = L_\emptyset$, we obtain the non-closure under intersection with regular languages.

The non-closure under intersection is witnessed by the languages, $L_1 = \{\, a^m bbv \mid m \geq 0, v \in \{a,b\}^* \,\}$ and $L_2 = \{\, vbba^m \mid m \geq 0, v \in \{a,b\}^* \,\}$.

We show that L_1 is accepted by some more or less trivial REV-PWK without any communication as follows.

The lower component does nothing, that is, it loops in its non-accepting initial state on the right endmarker. The behavior of the upper component is depicted as a state graph in Figure 1. If and only if the component has seen a correct prefix of the form a^*bb it halts in an accepting state (the rest of the input cannot affect the computation result any more and, by definition, there is no need to read it).

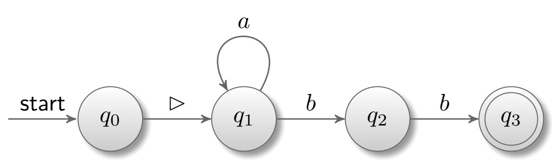

Figure 1: State graph of the upper component of a REV-PWK accepting L_1.

Since $L_2 = L_1^R$ and the closure of $\mathscr{L}$(REV-PWK) under reversal, we conclude that L_2 belongs to $\mathscr{L}$(REV-PWK) as well. However, $L_1 \cap L_2 = L_I$ for $I = b\{a,b\}^*b$ and, thus, the non-closure under intersection follows.

The non-closure under union follows from the closure under complementation and the non-closure under intersection by De Morgan's law. □

Proposition 6. *The family $\mathscr{L}$(REV-PWK) is not closed under concatenation and iteration.*

Proof. The witness language for both operations is $L = \{\, a^n b^n \mid n \geq 1 \,\}$ which belongs to $\mathscr{L}$(REV-PWK) by Example 1.

For the concatenation we consider $L \cdot L$ and for the iteration we consider L^*.

Essentially, using a different but similar language, in [18] it is shown that for n long enough both components have to scan some symbol from each two factors whose lengths have to be compared simultaneously. This argument applies also here. However, the two components can simultaneously stay in two corresponding factors at most for one such pair. This implies that neither the language $L \cdot L$ nor the language L^* is accepted even by any not necessarily reversible DPWK. □

Proposition 7. *The family $\mathscr{L}$(REV-PWK) is not closed under length-preserving homomorphisms.*

Proposition 8. *The family $\mathscr{L}$(REV-PWK) is not closed under inverse homomorphisms.*

Family	$\overline{}$	$\cup$	$\cap$	$\cap_{\text{reg}}$	$\cdot$	$*$	$h_{\text{len.pres.}}$	h^{-1}	R
REV-PWK	✓	✗	✗	✗	✗	✗	✗	✗	✓

Table 1: Closure properties of the language families discussed.

5 Restricted Communication

The REV-PWK considered in the previous sections may communicate arbitrarily often. In this section, we want to consider DPWK and REV-PWK with a restricted amount of communications. According to the definition in Section 2 we have a function $f : \mathbb{N} \to \mathbb{N}$ and define that a DPWK is communication bounded by f if all words w in the language are accepted with computations where the total number of messages sent is bounded by $f(|w|)$. A REV-PWK is communication bounded by f if, in addition, the total number of messages sent in reverse computations is bounded by $f(|w|)$ as well. Here, we will study the language class with constant communication, where $f \in O(1)$, the class with logarithmic communication, where $f \in O(\log(n))$, the class with square root communication, where $f \in O(\sqrt{n})$, and the class with arbitrary, i.e., linear communication, where $f \in O(n)$. The relations of these classes have been investigated for DPWK in [12]. Here, we will complement the results for REV-PWK and clarify the relations between reversible and general, possibly irreversible, devices. We start with an example presenting a non-semilinear language that is accepted by a REV-PWK($O(\log(n))$).

Example 9. The language $L_{expo} = \{\, a^{2^0}ba^{2^2}b\cdots ba^{2^{2m}}ca^{2^{2m+1}}b\cdots ba^{2^3}ba^{2^1} \mid m \geq 1 \,\}$ is accepted by a REV-PWK. The rough idea of the construction is that in a first phase the components compare the lengths 2^0 with 2^1, 2^2 with 2^3, ..., and 2^{2m} with 2^{2m+1}. The first phase ends when both components reach the center symbol c. In a second phase, the components compare the length 2^{2m} with 2^{2m-1}, 2^{2m-2} with 2^{2m-3}, ..., and 2^2 with 2^1. To achieve this the lower component has to wait on the c until the upper component has moved across the block $a^{2^{2m+1}}$. To realize the comparisons, the upper component moves across its a-blocks with half speed, whereas the lower component moves across its a-blocks with full speed, that is, one square per step. The length comparisons in the first and second phase are checked by communicating when a b, c, or the right endmarker is reached which must happen synchronously.

The length of an accepted input is $n = 2^{2m+2} + 2m$. There are communications only on symbols b, c, and $\lhd$ both in forward computations and reverse computations. Hence, there are at most $2m+3$ communications in forward computations as well as in reverse computations. Thus, the REV-PWK constructed is a REV-PWK($O(\log(n))$) and L_{expo} belongs to $\mathscr{L}$(REV-PWK($O(\log(n))$)). ■

Lemma 10. *The language $L_{lin} = \{\, wcw^R \mid w \in \{0,1\}^* \,\}$ belongs to $\mathscr{L}$(REV-PWK($O(n)$)).*

Proof. A REV-PWK accepting L_{lin} will move its both components synchronously towards the middle marker c as long as the input symbol read and communicated in every step is equal. In case of in-equivalence the computation halts non-accepting. If both components reach the middle marker c at the same time, the first task is nearly accomplished. It remains for the lower component to read the input completely and to halt non-accepting in case of another symbol c occurring. Since both components move synchronously and communicate in every step, it is clear that L_{lin} can be accepted by a REV-PWK($O(n)$). □

As a combination of Example 9 and Lemma 10 we obtain the following lemma.

Lemma 11. *$\hat{L}_{expo} = \{\, a^{2^0}x_1a^{2^2}x_2\cdots x_ma^{2^{2m}}ca^{2^{2m+1}}x_m\cdots x_2a^{2^3}x_1a^{2^1} \mid m \geq 1$ and $x_i \in \{0,1\}, 1 \leq i \leq m \,\}$ belongs to $\mathscr{L}$(REV-PWK($O(\log(n))$)).*

Proof. It can be observed from the construction in Example 9 that in the first phase both components communicate on every symbol b and c. So, on the corresponding input from $\hat{L}_{expo}$ both components can communicate on every symbol 0, 1, and c in order to simulate the REV-PWK accepting L_{lin} as a subtask. $\square$

With similar ideas it is possible to show the following lemma.

Lemma 12. $\hat{L}_{poly} = \{\, ax_1a^5x_2\cdots x_ma^{4m+1}ca^{4m+3}x_m\cdots x_2a^7x_1a^3 \mid m \geq 0$ *and* $x_i \in \{0,1\}, 1 \leq i \leq m\}$ *belongs to* $\mathscr{L}$*(REV-PWK*$(O(\sqrt{n}))$*).*

It is shown in [12] that L_{lin} does not belong to $\mathscr{L}(\text{DPWK}(O(f)))$ if $f \in \frac{n}{\omega(\log(n))}$. Hence, L_{lin} does not belong to $\mathscr{L}(\text{REV-PWK}(O(\sqrt{n})))$. It is also shown in [12] that $\hat{L}_{poly}$ does not belong to $\mathscr{L}(\text{DPWK}(O(f)))$ if $f \in O(\log(n))$. Thus, $\hat{L}_{poly}$ does not belong to $\mathscr{L}(\text{REV-PWK}(O(\log(n))))$. Finally, it is known due to [12] that every language in $\mathscr{L}(\text{DPWK}(O(1)))$ is semilinear. Since $\hat{L}_{expo}$ is not semilinear, it does not belong to $\mathscr{L}(\text{REV-PWK}(O(1)))$. Together with Lemma 10, Lemma 11, and Lemma 12 we obtain the following proper hierarchy:

$$\mathscr{L}(\text{REV-PWK}(O(1))) \subset \mathscr{L}(\text{REV-PWK}(O(\log(n)))) \subset \mathscr{L}(\text{REV-PWK}(O(\sqrt{n}))) \subset \mathscr{L}(\text{REV-PWK}(O(n)))$$

Theorem 3 presents a regular language that is not accepted by any REV-PWK$(O(n))$. Since the regular languages belong to $\mathscr{L}(\text{DPWK}(O(1)))$ we immediately obtain proper inclusions between reversible and general language classes with the same amount of communication. These results and the other results of this section are summarized in Figure 2.

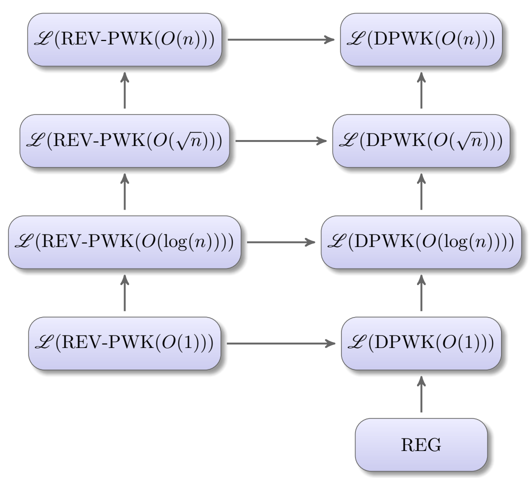

Figure 2: Relationships between language families induced by two-party Watson-Crick systems. An arrow between families indicates a strict inclusion.

6 Decidability Questions

In this section, we will discuss several decidability questions for REV-PWK. It has been shown in [12] that the questions of emptiness, finiteness, inclusion, and equivalence are decidable for general, possibly irreversible, DPWK in case of a finite number of communications. This result leads immediately to the following decidability results for REV-PWK in case of a finite number of communications.

Theorem 13. *Let $k \geq 0$ be a constant. Then emptiness, finiteness, inclusion, and equivalence are decidable for REV-PWK(k).*

Next, we want to obtain that the decidability questions become undecidable if a non-constant number of communications is used. In a first step, we show that the questions of emptiness, finiteness, inclusion, and equivalence are undecidable and, moreover, not even semidecidable for REV-PWK in case of a linear number of communications used. In a second step, we will obtain the same non-semidecidability results with a superlogarithmic number of communications used.

It has been shown in [14] that the questions of testing emptiness, finiteness, inclusion, and equivalence are not semidecidable for *reversible two-head finite automata*. The difference between such automata and DPWK is that the former move their two heads in the same direction from left to right, whereas the latter move both heads in opposite directions. Now, the idea is to simulate a reversible two-head finite automaton by a REV-PWK.

The non-semidecidability results for reversible two-head finite automata are obtained by showing that the set VALC_M of suitably encoded valid computations of a deterministic linearly space bounded one-tape, one-head Turing machine M, so-called linear bounded automaton (LBA) can be accepted by a reversible two-head finite automaton. It should be noted that the due to technical reasons the definition of the set VALC_M in [14] considers valid computations on inputs of length at least 2.

Now, we will construct a REV-PWK$(O(n))$ that accepts the set $\text{VALC}'_M = \{\, w^R c w \mid w \in \text{VALC}_M \,\}$, where the set VALC_M is defined over some alphabet A and $c \notin A$ is a new symbol.

Lemma 14. *Let M be an LBA. Then, a REV-PWK$(O(n))$ accepting VALC'_M can effectively be constructed.*

Proof. Let M be an LBA. A REV-PWK M' accepting VALC'_M has to accomplish two tasks. First, M' will test the structure $w^R c w$ disregarding whether w belongs to VALC_M or not. To achieve this task we use a similar approach as described in the proof of Lemma 10. Both components will move synchronously towards the middle marker c as long as the input symbol read and communicated in every step is equal. The structure $w^R c w$ is correctly tested, if both components reach the middle marker c at the same time. Then, the first task is nearly accomplished, but it remains for the lower component, while accomplishing the second task, to read the input completely and to halt non-accepting in case of another symbol c occurring. Since both components move synchronously and communicate in every step, it is clear that the first task can be realized by a REV-PWK$(O(n))$.

For the second task, we first observe that the remaining input for both components is the same word w and it remains to be checked whether or not w belongs to VALC_M. This can now be realized by implementing the construction given in [14] for two-head finite automata. The head 1 is simulated by the upper component and head 2 is simulated by the lower component, whereby the middle marker c is interpreted as the left endmarker for the two-head finite automaton. In this construction the lower component reads the input completely and can halt non-accepting if another symbol c is read. Since the two-head finite automaton is reversible, the second task and, therefore, the complete construction can be realized by a REV-PWK$(O(n))$. $\square$

This leads immediately to the following non-semidecidability results.

Theorem 15. *The problems of testing emptiness, finiteness, inclusion, and equivalence are not semidecidable for a given REV-PWK$(O(n))$.*

Proof. Let M be an LBA accepting inputs over the alphabet Σ. According to Lemma 14 we can effectively construct a REV-PWK$(O(n))$ M' accepting VALC'_M. Clearly, $L(M') = \text{VALC}'_M$ is empty if and only if VALC_M is empty if and only if $L(M)$ is either empty or contains some words from the finite set $\{\lambda\} \cup \Sigma$. The latter words have to be considered, since M may accept words of length less than two. Since the word problem is decidable for LBAs and emptiness is not semidecidable for LBAs, the non-semidecidability of emptiness follows.

We also obtain that $L(M') = \text{VALC}'_M$ is finite if and only if VALC_M is finite if and only if $L(M)$ is finite. Since finiteness is not semidecidable for LBAs, the non-semidecidability of finiteness follows.

Finally, it is easy to effectively construct a REV-PWK(1) that accepts nothing. Hence, the non-semidecidability of equivalence and inclusion follows immediately. □

Our next step is to obtain these non-semidecidability results also for REV-PWK with less communication. Our approach is to define another variant of VALC'_M in which the length of each configuration is enlarged while the same amount of communication is being kept. A similar approach has been used in [12] for general, possibly irreversible, DPWK. However, here the details are quite different and more complicated since the construction has to be reversible. The detailed and lengthy construction is omitted here. With all these prerequisites it is possible to show the following theorem.

Theorem 16. *The problems of testing emptiness, finiteness, inclusion, and equivalence are not semidecidable for a given REV-PWK*$(O(\log(n) \cdot \log(\log(n))))$.

References

[1] Dana Angluin (1982): *Inference of reversible languages*. J. ACM 29(3), pp. 741–765, doi:10.1145/322326.322334.

[2] Charles H. Bennett (1973): *Logical Reversibility of Computation*. IBM J. Res. Dev. 17, pp. 525–532, doi:10.1147/rd.176.0525.

[3] Henning Bordihn & György Vaszil (2021): *Reversible parallel communicating finite automata systems*. Acta Inf. 58(4), pp. 263–279, doi:10.1007/s00236-021-00396-9.

[4] Kingshuk Chatterjee & Kumar Sankar Ray (2017): *Reversible Watson-Crick automata*. Acta Inf. 54(5), pp. 487–499, doi:10.1007/s00236-016-0267-0.

[5] Kingshuk Chatterjee & Kumar Sankar Ray (2017): *Watson-Crick pushdown automata*. Kybernetika 53(5), pp. 868–876, doi:10.14736/kyb-2017-5-0868.

[6] Elena Czeizler, Eugen Czeizler, Lila Kari & Kai Salomaa (2009): *On the descriptional complexity of Watson-Crick automata*. Theor. Comput. Sci. 410, pp. 3250–3260, doi:10.1016/j.tcs.2009.05.001.

[7] Rudolf Freund, Gheorghe Păun, Grzegorz Rozenberg & Arto Salomaa (1997): *Watson-Crick Finite Automata*. In: DIMACS Workshop on DNA Based Computers, University of Pennsylvania, Philadelphia, pp. 305–317, doi:10.1090/dimacs/048/22.

[8] Markus Holzer, Sebastian Jakobi & Martin Kutrib (2018): *Minimal Reversible Deterministic Finite Automata*. Int. J. Found. Comput. Sci. 29, pp. 251–270, doi:10.1142/S0129054118400063.

[9] Radim Kocman, Zbynek Krivka, Alexander Meduna & Benedek Nagy (2022): *A jumping $5' \to 3'$ Watson-Crick finite automata model*. Acta Inf. 59(5), pp. 557–584, doi:10.1007/s00236-021-00413-x.

[10] Attila Kondacs & John Watrous (1997): *On the Power of Quantum Finite State Automata*. In: Foundations of Computer Science (FOCS 1997), IEEE Computer Society, pp. 66–75, doi:10.1109/SFCS.1997.646094.

[11] Martin Kutrib (2014): *Aspects of Reversibility for Classical Automata*. In C. S. Calude, G. R. Freivalds & K. Iwama, editors: Computing with New Resources, LNCS 8808, Springer, pp. 83–98, doi:10.1007/978-3-319-13350-8_7.

[12] Martin Kutrib & Andreas Malcher (2011): *Two-Party Watson-Crick Computations*. In: *Implementation and Application of Automata (CIAA 2010)*, *LNCS* 6482, Springer, pp. 191–200, doi:10.1007/978-3-642-18098-9_21.

[13] Martin Kutrib & Andreas Malcher (2012): *Reversible Pushdown Automata*. *J. Comput. Syst. Sci.* 78, pp. 1814–1827, doi:10.1016/j.jcss.2011.12.004.

[14] Martin Kutrib & Andreas Malcher (2017): *One-way reversible multi-head finite automata*. *Theor. Comput. Sci.* 682, pp. 149–164, doi:10.1016/j.tcs.2016.11.006.

[15] Martin Kutrib & Andreas Malcher (2022): *Reversible Computations of One-Way Counter Automata*. In Henning Bordihn, Géza Horváth & György Vaszil, editors: *NCMA 2022*, *EPTCS* 367, pp. 126–142, doi:10.4204/EPTCS.367.9.

[16] Martin Kutrib, Andreas Malcher & Matthias Wendlandt (2016): *Reversible Queue Automata*. *Fund. Inform.* 148, pp. 341–368, doi:10.3233/FI-2016-1438.

[17] Yves Lecerf (1963): *Logique Mathématique: Machines de Turing réversible*. *C. R. Séances Acad. Sci.* 257, pp. 2597–2600.

[18] Peter Leupold & Benedek Nagy (2010): $5' \to 3'$ *Watson-Crick Automata with Several Runs*. *Fund. Inform.* 104, pp. 71–91, doi:10.3233/FI-2010-336.

[19] Kenichi Morita (2011): *Two-Way Reversible Multi-Head Finite Automata*. *Fund. Inform.* 110, pp. 241–254, doi:10.3233/FI-2011-541.

[20] Benedek Nagy (2007): *On* $5' \to 3'$ *Sensing Watson-Crick Finite Automata*. In: *DNA Computing*, *LNCS* 4848, Springer, pp. 256–262, doi:10.1007/978-3-540-77962-9_27.

[21] Benedek Nagy (2013): *On a hierarchy of* $5' \to 3'$ *sensing Watson-Crick finite automata languages*. *J. Log. Comput.* 23, pp. 855–872, doi:10.1093/logcom/exr049.

[22] Benedek Nagy (2020): $5' \to 3'$ *Watson-Crick pushdown automata*. *Inf. Sci.* 537, pp. 452–466, doi:10.1016/j.ins.2020.06.031.

[23] Benedek Nagy & Zita Kovács (2021): *On deterministic 1-limited sensing* $5' \to 3'$ *Watson-Crick finite-state transducers*. *RAIRO Theor. Informatics Appl.* 55, p. 5, doi:10.1051/ita/2021007.

[24] Benedek Nagy, Shaghayegh Parchami & Hamid Mir Mohammad Sadeghi (2017): *A New Sensing* $5' \to 3'$ *Watson-Crick Automata Concept*. In Erzsébet Csuhaj-Varjú, Pál Dömösi & György Vaszil, editors: *AFL 2017*, *EPTCS* 252, pp. 195–204, doi:10.4204/EPTCS.252.19.