String Assembling Systems and Watson-Crick Finite Automata*

András Murvai, György Vaszil

Faculty of Informatics, University of Debrecen, Kassai út 26, 4028 Debrecen, Hungary
halimifoliumlycium@gmail.com
vaszil.gyorgy@inf.unideb.hu

Abstract: We explore the relationship of Watson-Crick automata and string assembling systems. In the general case, Watson-Crick automata are more powerful, so restricting the study to the stateless variant is of interest. We show that the class of languages of stateless Watson-Crick automata are strictly included in the class of languages of automata with at least two states, then compare string assembling systems with the stateless variant. It turns out, that there are languages that can be generated by string assembling systems, but not accepted by stateless Watson-Crick automata, but the question concerning the exact relationship of the language classes remains open.

1 Introduction

The double stranded structure of DNA motivated the introduction and study of double stranded strings with the tools and techniques of formal language theory, that is, the definition of string operations which model the biochemical processes that the corresponding strings can undergo, see the monograph [7] for more details. Two such models, related to the topic of this paper are sticker systems and Watson-Crick automata.

Sticker systems were introduced in [3], they use double stranded string "pieces" as building blocks to assemble longer double stranded strings when their single stranded ends stick together and form double strands according to the Watson-Crick complementarity relation. Such systems describe formal languages, sets of double stranded strings which can be "stick" together starting from a set of initial pieces. Depending on how we specify the details of the functioning of the system, simple languages (like regular languages which can be described by finite automata), or more complicated languages (like recursively enumerable languages which can be described by Turing machines) can be generated by sticker systems. As we will see later, string assembling systems (one of the computing models studied in this paper) are similar to this, from a certain point of view they can be thought of as special cases of sticker systems.

*Research supported by the construction EFOP-3.6.3-VEKOP-16-2017-00002 supported by the European Union, co-financed by the European Social Fund, and by grant K 120558 of the National Research, Development and Innovation Office of Hungary (NKFIH), financed under the K 16 funding scheme.

The other model interesting for our investigations is called Watson-Crick finite automaton, and it was introduced in [2]. Such automata are similar to ordinary finite automata in the sense that they read strings written on their tape, and either accept or reject them, thus defining a formal language as the set of strings that are accepted. Similarly to sticker systems, Watson-Crick automata work on double stranded strings, thus, they have a double stranded tape that contains two complementary strings which are read by two separate reading heads being able to move independently of each other, one on the upper and one on the lower strand of the tape. Concerning their computational power, these types of automata can describe more complicated language classes than ordinary finite automata. As we will see examples of this later, even some non-context-free context-sensitive languages can be accepted.

2 Preliminaries

For a finite alphabet of symbols V, let V^* denote the set of all strings over V, and let $V^+ = V^* \setminus \{\lambda\}$ where λ denotes the empty string. We consider two languages $L_1, L_2 \subseteq V^*$ equal if they differ in at most the empty string, $L_1 \setminus \{\lambda\} = L_2 \setminus \{\lambda\}$. The length of a word $w \in V^*$ is denoted by $|w|$, the number of occurrences of a symbol $x \in V$ in w is denoted by $|w|_x$.

A double stranded string over V is a pair of strings $(w_1, w_2) \in V^* \times V^*$, it can also be written as $\binom{w_1}{w_2}$, while the set of pairs $V^* \times V^*$ can be written as $\binom{V^*}{V^*}$.

We denote the last letters and the first letters of a string pair $\binom{u}{v}$ as $end(\binom{u}{v})$ and $bgn(\binom{u}{v})$:

1. If $u = u'x$, $v = v'y$ for some $x, y \in V$, $u', v' \in V^*$, then $end(\binom{u'x}{v'y}) = \binom{x}{y}$. Similarly, the first letters of the pair $\binom{u}{v}$ for $u = xu'$ and $v = yv'$ are denoted as $bgn(\binom{xu'}{yv'}) = \binom{x}{y}$ where $x, y \in V$, $u', v' \in V^*$.
2. Otherwise, if one of the strings is empty, we have $end(\binom{\lambda}{v'y}) = \binom{\lambda}{y}$, $end(\binom{u'x}{\lambda}) = \binom{x}{\lambda}$, $bgn(\binom{\lambda}{yv'}) = \binom{\lambda}{y}$, and $bgn(\binom{xu'}{\lambda}) = \binom{x}{\lambda}$ for

$x, y \in V,\ u', v' \in V^*$.

The *complementarity relation* is a symmetric relation on the letters of the alphabet $\rho \subseteq V \times V$. We call the set of sequences of pairs of complementary symbols a *Watson-Crick domain* and denote it with $WK_\rho(V)$, more formally,

$$WK_\rho(V) = \{ \begin{bmatrix} x \\ y \end{bmatrix} \mid x, y \in V,\ (x,y) \in \rho \}^*.$$

The string pair $\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} \dots \begin{bmatrix} x_n \\ y_n \end{bmatrix} \in WK_\rho(V)$ can also be written as $\begin{bmatrix} w \\ w' \end{bmatrix}$ where $w = x_1 x_2 \dots x_n$, $w' = y_1 y_2 \dots y_n$. We also denote the complement of $x \in V$ by $\overline{x}$, that is, $(x, \overline{x}) \in \rho$, and the complement of $\overline{x}$ is $\overline{\overline{x}}$, so $\overline{\overline{x}} = x$, and $(\overline{x}, x) \in \rho$.

The difference between $\begin{pmatrix} w_1 \\ w_2 \end{pmatrix}$ and $\begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$ is that $\begin{pmatrix} w_1 \\ w_2 \end{pmatrix}$ only gives a different notation for (w_1, w_2), while in the case of $\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \in WK_\rho(V)$, $|w_1| = |w_2|$ and w_2 is the complement of w_1, $w_2 = \overline{w}_1$ (also $\overline{w}_2 = w_1$).

2.1 Watson-Crick automata

Now we recall the definition of Watson-Crick automata from [2], see also the monograph [7] for more details.

The construct $M = \langle V, \rho, Q, q_0, F, \delta \rangle$ is a *Watson-Crick automaton* (WK automaton in short), where

- V is the input alphabet,
- $\rho \subseteq V \times V$ is the complementarity relation,
- Q is the nonempty finite set of states,
- $q_0 \in Q$ is the initial state,
- $F \subseteq Q$ is the set of accepting states, and
- $\delta : Q \times \begin{pmatrix} V^* \\ V^* \end{pmatrix} \to 2^Q$ is a finite relation, the state transition relation.

A *configuration* of such a WK automaton is denoted as $\begin{pmatrix} w_1 \\ w_2 \end{pmatrix} q \begin{pmatrix} w_3 \\ w_4 \end{pmatrix}$ where $\begin{pmatrix} w_1 \\ w_2 \end{pmatrix} \in \begin{pmatrix} V^* \\ V^* \end{pmatrix}$ is the part of the input which is already read, $q \in Q$ is the state of the WK automaton, and $\begin{pmatrix} w_3 \\ w_4 \end{pmatrix} \in \begin{pmatrix} V^* \\ V^* \end{pmatrix}$ is the part of the input which is not read yet.

The *transition* between two configurations is denoted by $\Rightarrow$, and defined as follows.

Let $\begin{pmatrix} u_1 \\ v_1 \end{pmatrix}, \begin{pmatrix} u_2 \\ v_2 \end{pmatrix}, \begin{pmatrix} u_3 \\ v_3 \end{pmatrix} \in \begin{pmatrix} V^* \\ V^* \end{pmatrix}$, $\begin{bmatrix} u_1 u_2 u_3 \\ v_1 v_2 v_3 \end{bmatrix} \in WK_\rho(V)$, $q, q' \in Q$. Then

$$\begin{pmatrix} u_1 \\ v_1 \end{pmatrix} q \begin{pmatrix} u_2 u_3 \\ v_2 v_3 \end{pmatrix} \Rightarrow \begin{pmatrix} u_1 u_2 \\ v_1 v_2 \end{pmatrix} q' \begin{pmatrix} u_3 \\ v_3 \end{pmatrix}$$

if and only if, $q' \in \delta(q, \begin{pmatrix} u_2 \\ v_2 \end{pmatrix})$. We may also write

$$\begin{pmatrix} u_1 \\ v_1 \end{pmatrix} \Rightarrow \begin{pmatrix} u_1 u_2 \\ v_1 v_2 \end{pmatrix}$$

if we are not interested in the states of the automaton and in the rest of the input to be read.

If the reflexive and transitive closure of $\Rightarrow$ is denoted by $\Rightarrow^*$, then the *language accepted by M* is

$$L(M) = \{ w \in V^* \mid q_0 \begin{bmatrix} w \\ \overline{w} \end{bmatrix} \Rightarrow^* \begin{bmatrix} w \\ \overline{w} \end{bmatrix} q_f, \text{ where } \begin{bmatrix} w \\ \overline{w} \end{bmatrix} \in WK_\rho(V),\ q_f \in F \}.$$

2.2 String assembling systems

Now we recall the definition of string assembling systems from [5]. These also work with double stranded strings, but in their case, the complementarity relation is the identity relation. Similarly to sticker systems mentioned in the introduction, they use a set of double stranded string pieces as building blocks to generate a language. They are able to "glue" two double stranded strings together if the last letters of the first string and the first letters of the second string coincide, and these letters overlap in the resulting string.

The 4-tuple $S = \langle \Sigma, A, T, E \rangle$ is a *string assembling system* (SAS in short), where

- Σ is a finite alphabet,
- $A \subset \Sigma^+ \times \Sigma^+$ is the finite set of axioms of the form $\begin{pmatrix} uv \\ u \end{pmatrix}$ or $\begin{pmatrix} u \\ uv \end{pmatrix}$ with $u \in \Sigma^+,\ v \in \Sigma^*$,
- $T \subset \Sigma^+ \times \Sigma^+$ is the finite set of assembly units,
- $E \subset \Sigma^+ \times \Sigma^+$ is the finite set of ending assembly units of the form $\begin{pmatrix} uv \\ v \end{pmatrix}$ or $\begin{pmatrix} v \\ uv \end{pmatrix}$ with $u \in \Sigma^*,\ v \in \Sigma^+$.

A *derivation step* of the SAS S as above is denoted by $\Rightarrow$, and defined as follows. We say that we added the unit $\begin{pmatrix} xu_2 \\ yv_2 \end{pmatrix}$ to the unit $\begin{pmatrix} u_1 \\ v_1 \end{pmatrix}$, denoted as

$$\begin{pmatrix} u_1 \\ v_1 \end{pmatrix} \Rightarrow \begin{pmatrix} u_1 u_2 \\ v_1 v_2 \end{pmatrix},$$

if and only if, $\begin{pmatrix} x \\ y \end{pmatrix} = end(\begin{pmatrix} u_1 \\ v_1 \end{pmatrix})$ for some $x, y \in \Sigma$, and $\begin{pmatrix} xu_2 \\ yv_2 \end{pmatrix} \in T \cup E$. Moreover, if $|u_1 u_2| \geq |v_1 v_2|$, then $u_1 u_2$ can be written as $u_1 u_2 = v_1 v_2 u'$ for some $u' \in \Sigma^*$, or the other way around, if $|u_1 u_2| \leq |v_1 v_2|$, then $u_1 u_2 v' = v_1 v_2$ for some $v' \in \Sigma^*$.

A sequence of derivation steps is a *derivation*. A derivation is *successful*, if after choosing an axiom from A, we

add assembling units from T, then close the derivation by adding a unit from E, in such a way that the upper and lower string of the string pair which is produced is identical.

If the reflexive and transitive closure of $\Rightarrow$ is denoted by $\Rightarrow^*$, then the language generated by S is

$$L(S) = \{w \in \Sigma^+ \mid \begin{pmatrix} u \\ v \end{pmatrix} \Rightarrow^* \begin{bmatrix} w \\ w \end{bmatrix} \text{ is a successful derivation}\}.$$

Now we introduce some additional notation that will be useful later. The string pair generated in k derivation steps, or in the case of WK automata, the string pair read in k transition steps, is denoted by $\begin{pmatrix} u_{(k)} \\ v_{(k)} \end{pmatrix}$, that is, $\begin{pmatrix} u_0 \\ v_0 \end{pmatrix} \Rightarrow^* \begin{pmatrix} u_0u_1 \dots u_{k-1}u_k \\ v_0v_1 \dots v_{k-1}v_k \end{pmatrix} = \begin{pmatrix} u_{(k)} \\ v_{(k)} \end{pmatrix}$. The pair $\begin{pmatrix} u_{(0)} \\ v_{(0)} \end{pmatrix}$ denotes an axiom, or in the case of WK automata, the pair $\begin{pmatrix} \lambda \\ \lambda \end{pmatrix}$, representing the fact that nothing has been read from the input yet.

Let Δ_k denote the difference of the length of upper and lower element of the string pair $\begin{pmatrix} u \\ v \end{pmatrix}$ added in the kth derivation step, or read in the kth state transition, that is,$\Delta_k = |u| - |v|$, and let $\Delta_{(k)}$ denote the difference of the generated strings or the strings read on the upper and lower pert of the input tape in k derivation or state transition steps, that is, for $\begin{pmatrix} u_{(k)} \\ v_{(k)} \end{pmatrix}$, let $\Delta_{(k)} = |u_{(k)}| - |v_{(k)}|$.

Let us also denote the class of languages accepted or generated by a computational model X by $\mathscr{L}(X)$, so let $\mathscr{L}(SAS)$ and $\mathscr{L}(WK)$ denote the classes of languages generated by SAS and accepted by WK automata, respectively.

3 The computational power of Watson-Crick automata and string assembling systems

In the following, we would like to point out some of the similarities of WK automata and SAS, and then investigate whether these similarities help us to establish a relationship between their computational power.

Consider the WK automata $M = \langle V, \rho, Q, q_0, F, \delta \rangle$ and the SAS $S = \langle \Sigma, A, T, E \rangle$. They both work with double stranded strings, and there is a relation between the upper and lower strings in both cases. In the case of SAS, this relation is the identity relation, in WK automata, on the other hand, the relation can be an arbitrary symmetric relation. This might seem to be an important difference, but it is not. It is known from [4], that the computational power of WK automata is not influenced by the complementarity relation. For any WK automata M, we can construct an M', such that it uses the relation $\rho = \{(x,x) \mid x \in V\}$, and $L(M) = L(M')$. Based on this result, from now on we assume that $\rho = \rho_{id}$, the identity relation.

We can also find similarities in the functioning of the two models. Adding building units to the generated string by a SAS from A, T, or E, corresponds to reading substrings of the input by a WK automaton with a transition from the initial state, a transition between arbitrary states, and a transition leading to and accepting state, respectively. While the states of WK automata are explicitly given, "states" of a SAS are implicit, they are "hidden" in the overlapping pairs of letters of the building units.

3.1 Known results on WK automata and SAS

As we have seen, SAS can basically add three types of units to the generated strings, while WK automata can have an arbitrary number of states, so it seems to be reasonable to review results related to the number of states of WK automata. In [6] the authors show that from the point of view of state complexity, WK automata are more efficient than ordinary finite automata, there is a sequence L_k, $k \geq 1$ of regular languages, such that the number of states of the automata sequence accepting L_k is unbounded, while all languages of the sequence can be accepted with WK automata having a fixed number of states. Building on these results, in [1] it is shown that arbitrary finite languages can be accepted with WK automata having two states, and that arbitrary unary regular languages can be accepted by WK automata having three states.

Concerning the power of SAS, already in the introductory paper [5], the authors show that there are unary regular languages that cannot be generated by SAS, but all finite languages can be. From our point of view, especially interesting is the result concerning the relationship of languages accepted by stateless two head finite automata (that is, two head automata with one state) and languages generated by SAS. While the languages accepted by stateless WK automata strictly include the languages accepted by stateless two-head automata, Theorem 1 which we are going to prove in the next section, can be considered as the extension of this result.

3.2 Comparing the power of WK automata and SAS

Let us start by establishing the relationship between languages of (unrestricted) WK automata and SAS.

Proposition 1.

$$\mathscr{L}(SAS) \subset \mathscr{L}(WK).$$

Proof. According to Theorem 3.1 of [5], languages generated by SAS can be accepted by nondeterministic one-way two-head finite automata, and since one-way two-head finite automata can be simulated by WK automata, it is clear that $\mathscr{L}(SAS) \subseteq \mathscr{L}(WK)$.

On the other hand, $\mathscr{L}(WK) \neq \mathscr{L}(SAS)$, since according Theorem 3.6 in [5] the language $L = \{a\} \cup \{a^{2n} \mid n \geq 2\}$

cannot be generated by any SAS, but can be accepted by the following WK automaton.

Let $M = \langle \{a\}, \rho_{id}, \{q_0, q_f, q_{f'}\}, q_0, \{q_f, q_{f'}\}, \delta \rangle$ where

$$\delta(q_0, \begin{pmatrix} a \\ a \end{pmatrix}) = q_f, \qquad \delta(q_0, \begin{pmatrix} aaaa \\ aaaa \end{pmatrix}) = q_{f'},$$
$$\delta(q_{f'}, \begin{pmatrix} aa \\ aa \end{pmatrix}) = q_{f'}.$$

□

As we have seen, in the general case, the computational power of WK automata is greater than the power of SAS. Now we continue by restricting the power of WK automata by considering its variants with reduced number of states. First, we look at the relationship between stateless WK automata and SAS.

In the following, we denote the classes of WK automata with m states by $WK_{|Q|=m}$ and by $\mathscr{L}(WK_{|Q|=m})$ the class of languages they accept.

Theorem 1. *Let M be a nondeterministic stateless WK automaton. Then there exists an SAS S generating the language accepted by M with an initial marker, that is, $\$L(M) = L(S)$ where $\$ \notin V$.*

Proof. Consider $M = \langle V, \rho_{id}, \{q\}, q, \{q\}, \delta \rangle$, we construct the SAS $S = \langle \Sigma, A, T, E \rangle$ as follows.

- $\Sigma = V \cup \{\$\}$ ($\$ \notin V$),
- $A = \{ \begin{pmatrix} \$ \\ \$ \end{pmatrix} \}$,
- $T = \{ \begin{pmatrix} xu \\ yv \end{pmatrix} \mid$ for all $\begin{pmatrix} x \\ y \end{pmatrix} \in \Sigma \times \Sigma$ pairs of letters and transitions $\delta(q, \begin{pmatrix} u \\ v \end{pmatrix}) = q \}$,
- $E = T \cup \{ \begin{pmatrix} \$ \\ \$ \end{pmatrix} \}$.

First we show that any $w \in L(M)$ can be generated by S. M accepts the empty word, so S must generate \$, thus $\begin{pmatrix} \$ \\ \$ \end{pmatrix} \in A$, $\begin{pmatrix} \$ \\ \$ \end{pmatrix} \in E$.

Let s_u and s_l denote the already read part of the upper and lower strands of the input of M. For all possible pairs of letters $end(\begin{pmatrix} \$s_u \\ \$s_l \end{pmatrix}) = \begin{pmatrix} x \\ y \end{pmatrix}$, if $\begin{pmatrix} u \\ v \end{pmatrix}$ can be read by M, there has to be a unit $\begin{pmatrix} xu \\ yv \end{pmatrix}$ which can be added by S. We have added all possible building units for the reading of $\begin{pmatrix} u \\ v \end{pmatrix}$ to S, so if a string pair can be read by M, then the corresponding building unit can be added by S.

M is always in an accepting state, so the only requirement for a string to be accepted is the property that it can be read completely. In the last reading step, reading some $\begin{pmatrix} u \\ v \end{pmatrix}$, the two heads reach the end of the upper and lower strings simultaneously. All string pairs corresponding to some unit in T have an appropriate transition in δ, so S can finish the string generation by adding some unit from T, which must contain all units that can end the generation, so

$$E = T \cup \{ \begin{pmatrix} \$ \\ \$ \end{pmatrix} \}.$$

Now we show that any $\$w \in L(S)$ can be accepted by M. For all $\$w$, we have a derivation

$$\begin{pmatrix} \$ \\ \$ \end{pmatrix} \Rightarrow \begin{pmatrix} \$u_1 \\ \$v_1 \end{pmatrix} \Rightarrow \begin{pmatrix} \$u_1u_2 \\ \$v_1v_2 \end{pmatrix} \Rightarrow \ldots \Rightarrow \begin{pmatrix} \$u_1u_2 \ldots u_t \\ \$v_1v_2 \ldots v_t \end{pmatrix},$$

where $\$w = \$u_1u_2 \ldots u_t = \$v_1v_2 \ldots v_t$, $\begin{pmatrix} xu_i \\ yv_i \end{pmatrix} \in T \cup E$, $1 \leq i \leq t$. Since we created the building units in such a way that their initial pairs of letters contain all possible combinations, even if $\begin{pmatrix} x \\ y \end{pmatrix} = end(\begin{pmatrix} u_{(i-1)} \\ v_{(i-1)} \end{pmatrix})$ holds, the ability to add unit $\begin{pmatrix} xu_i \\ yv_i \end{pmatrix}$ depends only on the string pair $\begin{pmatrix} u_i \\ v_i \end{pmatrix}$. As $\delta(q, \begin{pmatrix} u_i \\ v_i \end{pmatrix}) = q$, and M is stateless, only the read string pair $\begin{pmatrix} u_i \\ v_i \end{pmatrix}$ determines the success of the reading a word. The derivation above (besides the leading \$ sign) is a successful reading of the word w

$$\begin{pmatrix} u_1 \\ v_1 \end{pmatrix} \Rightarrow \begin{pmatrix} u_1u_2 \\ v_1v_2 \end{pmatrix} \Rightarrow \ldots \Rightarrow \begin{pmatrix} u_1u_2 \ldots u_t \\ v_1v_2 \ldots v_t \end{pmatrix},$$

thus $w \in L(M)$. □

Although formally the above theorem does not say anything about the containment relationship betweem $\mathscr{L}(SAS)$ and $\mathscr{L}(WK_{|Q|=1})$, it still establishes some kind of connection between the two languages classes. Moreover, our statement is "stronger" then the similar statement in [5] which states that for all languages L accepted by stateless two-head finite automata, a corresponding SAS generating L' can be constructed in such a way that all $w \in L$ corresponds to $w' \in L'$ with $h(w') = w$ for a homomorphism deleting four symbols from w' (and not just one, as in our case).

Let us now relax the requirement of statelessness, and consider WK automata with two states

Lemma 1. *There is a WK automaton with two states which accepts the language $L = \{w \mid w \in \{a,b\}^*, |w|_a = 2(1+2i),\ i \geq 0\}$.*

Proof. The language can be accepted by the following WK automaton.

Let $M = \langle \{a,b\}, \rho_{id}, \{q_0, q_f\}, q_0, \{q_f\}, \delta \rangle$ with δ:

$$\delta(q_0,\binom{a}{\lambda})=q_0,\quad \delta(q_f,\binom{a}{\lambda})=q_f,$$
$$\delta(q_0,\binom{b}{x})=q_0,\quad \delta(q_f,\binom{b}{x})=q_f,$$
$$\delta(q_0,\binom{\lambda}{xy})=q_f,\quad \delta(q_f,\binom{\lambda}{xy})=q_0,$$

where $x,y\in\{a,b\}$.

We first show that $L(M)\subseteq L$. For any $w\in L(M)$ we have a transition sequence

$$q_0\binom{w}{w}\Rightarrow\binom{u_0}{v_0}q\binom{w_1}{w_1'}\Rightarrow\binom{u_0u_1}{v_0v_1}q'\binom{w_2}{w_2'}\Rightarrow\dots$$
$$\dots\Rightarrow\binom{u_0u_1\dots u_t}{v_0v_1\dots v_t}q_f$$

where in the first and last configuration M is in states q_0 and q_f, respectively, and in one of the two states in the intermediate configurations, $\binom{u_i}{v_i}\in\{\binom{a}{\lambda},\binom{b}{x},\binom{\lambda}{xy}\mid x,y\in\{a,b\}\}$, $0\le i\le t$, correspond to the reading steps.

In any case, by reading $\binom{u_i}{v_i}=\binom{b}{x}$, $\Delta_i=0$, and M remains in the same state, by reading $\binom{u_i}{v_i}=\binom{a}{\lambda}$, $\Delta_i=1$ and M remains in the same state, by reading $\binom{u_i}{v_i}=\binom{\lambda}{xy}$, $\Delta_i=-2$, and M changes to the other state.

It is clear that $\Delta_{(i)}$ can only increase after a reading step if $\binom{a}{\lambda}$ was read, and can only decrease if $\binom{\lambda}{xy}$ was read. Therefore, in order for $\Delta_{(i)}=0$, some $\binom{\lambda}{xy}$ need to be read, together with reading twice as many $\binom{a}{\lambda}$. Thus, if $\Delta_{(i)}=0$, the already read part of the upper string contains an even number of as.

If w is accepted, then $\Delta_{(i)}=0$ and M must be in state q_f. In order for M to be in state q_f, an odd number $(1+2k,k\ge 0)$ of reading $\binom{\lambda}{xy}$ had to happen, which means that $2(1+2k)$ number of reading $\binom{a}{\lambda}$ had to happen. Thus, M can only accept words containing $2(1+2k)$ number of as.

Now we show that M can accept any $w\in L$. First it needs to read $\binom{a}{\lambda}$ and $\binom{b}{x}$ (choosing the appropriate x) until it reaches the end of the upper string. We are in state q_0, since we have started in q_0 and remained in q_0 all the way. The reading head of the lower string is behind the upper head by the number of as, by $2(1+2k)$. In order to catch up with the upper head, we need to read $\binom{\lambda}{xy}$ a number of times. At each such reading step, M changes its state. Since the number if as in w is $2(1+2k)$ and we read two letters from the lower string, the lower reading head reaches the end of the word after $1+2k$, that is, after an odd number of reading steps. Therefore, M enters q_f when started in q_0, so w is accepted. □

Corollary 1. *The language $L=\{w\mid w\in\{a,b\}^*,|w|_a=(1+2i)k,\ i\ge 0\}$ can be accepted for any $k\ge 2$ by a WK automaton similar to the above, with*

$$\delta(q_0,\binom{a}{\lambda})=q_0,\qquad \delta(q_f,\binom{a}{\lambda})=q_f,$$
$$\delta(q_0,\binom{b}{x})=q_0,\qquad \delta(q_f,\binom{b}{x})=q_f,$$
$$\delta(q_0,\binom{\lambda}{x_1x_2\dots x_k})=q_f,\quad \delta(q_f,\binom{\lambda}{x_1x_2\dots x_k})=q_0,$$

where $x,x_1,x_2,\dots,x_k\in\{a,b\}$.

Thus, $L=\{w\mid w\in\{a,b\}^*,|w|_a=2(1+2i),\ i\ge 0\}$ can be accepted by a WK automaton with two states, but it cannot be accepted by stateless WK automata, as we will show in the following.

Lemma 2. *If M is a stateless WK automaton, then $L(M)=L(M)^*$.*

Proof. Let $M=\langle V,\rho_{id},\{q\},q,\{q\},\delta\rangle$ be a stateless WK automaton. It is clear that $L(M)\subseteq L(M)^*$. To show the converse inclusion, consider $L(M)^*=\bigcup_{i=0}^{\infty}L(M)^i$, and a word $w\in L(M)^*$. There are three possible cases.

1. If $w\in L(M)^0$, then $w=\lambda\in L(M)$,
2. if $w\in L(M)^1$, then $w\in L(M)$,
3. if $w\in L(M)^k,\ k\ge 2$, then w can be written as $w=w_1w_2\dots w_k$, where $w_i\in L(M),\ 1\le i\le k$.

The first two cases are clear. In the third case, $\begin{bmatrix}w_1\\w_1\end{bmatrix}q\begin{bmatrix}w_2\dots w_k\\w_2\dots w_k\end{bmatrix}$ is a possible configuration, since $w_1\in L$, so the two reading heads can be positioned after w_1.

If a configuration $\begin{bmatrix}w_1\dots w_{i-1}\\w_1\dots w_{i-1}\end{bmatrix}q\begin{bmatrix}w_i\dots w_k\\w_i\dots w_k\end{bmatrix}$ can occur, then also $\begin{bmatrix}w_1\dots w_i\\w_1\dots w_i\end{bmatrix}q\begin{bmatrix}w_{i+1}\dots w_k\\w_{i+1}\dots w_k\end{bmatrix}$ is a possible configuration, since $\begin{bmatrix}w_i\\w_i\end{bmatrix}$ can be read (as $w_i\in L$). Therefore, w can be read in such a way that both reading heads reach the end of the tape, and since there is only one state, M accepts w. □

Using the above lemma, we can show the following.

Corollary 2. *$L=\{w\mid w\in\{a,b\}^*,|w|_a=2+4i,i\ge 0\}$ cannot be accepted by any stateless WK automaton.*

Proof. If $L=L(M)$ for a stateless WK automaton M, then $L=L^*$, according to the lemma above. Let $w\in L$ be such, that $|w|_a=2$. Then $w^2\in L^*$, but $w^2\notin L$, so $L\ne L^*$, therefore $L\ne L(M)$, which is a contradiction. □

Now we show that the language that we found not to be in $\mathscr{L}(WK_{|Q|=1})$ can be generated by SAS.

Lemma 3. *$L = \{w \mid w \in \{a,b\}^*, |w|_a = 2+4i,\ i \geq 0\}$ can be generated by SAS.*

Proof. Let $S = \langle \Sigma, A, T, E \rangle$ with

- $\Sigma = \{a,b\}$,
- $A = \{\binom{aa}{aa}, \binom{ab}{a}, \binom{ba}{b}, \binom{bba}{bb}, \binom{bbb}{b}\}$,
- $T = \{\binom{x_1 b}{x_2 x_3}, \binom{xabb}{y}, \binom{xbab}{y}, \binom{xbba}{y}, \binom{xaa}{y}, \binom{x_1 a}{x_2 x_3 x_4}\}$ where $x, x_1, x_2, x_3, x_4,\ y \in \Sigma$,
- $E = \{\binom{a}{a}, \binom{b}{b}\}$.

We have chosen the assembly units of the SAS S in such a way that the difference between the length of the upper and lower string indicates how many as must be added to the upper string in order to obtain a correct number. The following four cases are possible.

1. At least 3 as are missing: $\Delta_{(i)} = 3+4n$,
2. at least 2 as are missing: $\Delta_{(i)} = 2+4n$,
3. at least 1 a is missing: $\Delta_{(i)} = 1+4n$,
4. at least 0 a is missing: $\Delta_{(i)} = 0+4n$,

where $n \in \mathbb{Z}$ and we are after the ith derivation steps.

More formally we can express the above as follows. Consider the string pair $\binom{u_{(k)}}{v_{(k)}}$ obtained in k derivation steps. Then the congruence

$$2 - |u_{(k)}|_a \equiv \Delta_{(k)} \pmod 4$$

must hold during the whole generating process: $|u_{(k)}|_a$ is the number of as in the upper string. In case of $|u_{(k)}|_a + 2 = 4i+j,\ (i \geq 0,\ 1 \leq j \leq 4)$, $(4i+4)-(4i+j) = (4i+4) - |u_{(k)}|_a - 2$ indicates the least number of as that we need to add in order for the number to be a multiple of 4. $\Delta_{(k)} = (4-j)+4n,\ n \in \mathbb{Z}$ indicates this too, so

$$(4i+4) - |u_{(k)}|_a - 2 \equiv 2 - |u_{(k)}|_a \equiv \Delta_{(k)} = (4-j)+4n \pmod 4.$$

The axioms fulfill this condition.

Let us assume, that the congruence holds after k derivation steps. After adding the unit $\binom{u_{k+1}}{v_{k+1}}$ in the $(k+1)$th step, the congruence still holds, since

- if $\binom{x_1 b}{x_2 x_3}, \binom{a}{a}, \binom{b}{b}$, then $|u_{(k+1)}|_a = |u_{(k)}|_a$ so $\Delta_{(k)} = \Delta_{(k+1)}$, thus $2 - |u_{(k+1)}|_a = 2 - |u_{(k)}|_a \equiv \Delta_{(k)} = \Delta_{(k+1)}$ also hold.

Using that $a \equiv b \pmod m$ if and only if $m|(a-b)$, that is, if $a - b \equiv 0 \pmod m$, in the remaining cases we show the congruence $(2 - |u_{(k)}|_a) - \Delta_{(k)} \equiv 0 \pmod 4$ holds.

- If $\binom{xabb}{y}, \binom{xbab}{y}, \binom{xbba}{y}$, then $|u_{(k+1)}|_a = |u_{(k)}|_a + 1$ and $\Delta_{(k+1)} = \Delta_{(k)} + 3$ holds, so $2 - |u_{(k+1)}|_a - \Delta_{(k+1)} = 2 - (|u_{(k)}|_a + 1) - (\Delta_{(k)} + 3) = 2 - |u_{(k)}|_a - \Delta_{(k)} - 4 \equiv 0 \pmod 4$ also holds;
- if $\binom{xaa}{y}$, then $|u_{(k+1)}|_a = |u_{(k)}|_a + 2$ and $\Delta_{(k+1)} = \Delta_{(k)} + 2$ hold, so $2 - |u_{(k+1)}|_a - \Delta_{(k+1)} = 2 - (|u_{(k)}|_a + 2) - (\Delta_{(k)} + 2) = 2 - |u_{(k)}|_a - \Delta_{(k)} - 4 \equiv 0 \pmod 4$ also holds;
- if $\binom{x_1 a}{x_2 x_3 x_4}$, then $|u_{(k+1)}|_a = |u_{(k)}|_a + 1$ and $\Delta_{(k+1)} = \Delta_{(k)} - 1$ hold, so $2 - |u_{(k+1)}|_a - \Delta_{(k+1)} = 2 - (|u_{(k)}|_a + 1) - (\Delta_{(k)} - 1) = 2 - |u_{(k)}|_a - \Delta_{(k)} \equiv 0 \pmod 4$ also holds.

Thus, all words generated by S belong to the language L.

Now we show that arbitrary words of L can be generated by S. The assembling units of S are chosen in such a way that as and bs can follow each other in arbitrary order in the upper string, and the lower string of each unit can be arbitrary (only the length of the strings are fixed), so it is possible to add to the lower strings the appropriate letters.

Axioms are chosen as follows. We create the least number of axioms which satisfy the above mentioned congruence and provide the arbitrary letter order. The idea is to create axioms of the form $\binom{u}{v}$ with upper strings of a fixed length at most $n \in \mathbb{N}$, and the lower strings being at most as long as the upper, thus $|u| = i,\ i = 1 \ldots n,\ 1 \leq |v| \leq |u|$. Then we discard those which do not satisfy the congruence, or which are not necessary for the initiation of the generation.

This way, for $i = 1$ we have $\binom{a}{a}$ and $\binom{b}{b}$ as axioms, but we discard these, because they do not satisfy the congruence. For $i = 2$ we have the units $\binom{aa}{a}, \binom{aa}{aa}, \binom{ab}{a}, \binom{ab}{ab}, \binom{ba}{b}, \binom{ba}{ba}, \binom{bb}{b}$, and $\binom{bb}{bb}$. We keep $\binom{aa}{aa}, \binom{ab}{a}, \binom{ba}{b}$, so we can begin the generation of words starting with aa, ab, ba. We cannot generate words starting with bb, so for $i = 3$ we only consider the units starting with bb $\binom{bba}{b}, \binom{bba}{bb}, \binom{bba}{bba}, \binom{bbb}{b}, \binom{bbb}{bb}, \binom{bbb}{bbb}$. We keep $\binom{bba}{bb}, \binom{bbb}{b}$, so we are able to start the generation of words beginning with bb. We can start the generation with all possible combinations of letters, so no more axioms are needed.

We proceed with units in T in a similar manner: we create the least possible number of elements that satisfy

the congruence, but we also allow that lower strings are longer than the upper strings. Let us suppose that the current string pair satisfies the congruence. If we would like to continue the upper string with b, then

- we need to add an appropriate letter also to the lower string, so we need a unit of the form $\begin{pmatrix} x_1 b \\ x_2 x_3 \end{pmatrix}$.

If we would like to continue the upper string with a, then

- we need to add two letters to the lower string, so we have the unit $\begin{pmatrix} x_1 a \\ x_2 x_3 x_4 \end{pmatrix}$, or
- we need to add further letters to the upper string. If to the upper string
  - we would like to add an a, then we need to add three letters, so we have the units $\begin{pmatrix} xabb \\ y \end{pmatrix}$, $\begin{pmatrix} xbab \\ y \end{pmatrix}$ and $\begin{pmatrix} xbba \\ y \end{pmatrix}$;
  - we would like to add two as, then we have the unit $\begin{pmatrix} xaa \\ y \end{pmatrix}$.

The number of letters needed to be added to the upper or lower strings are determined by the congruence.

With these units, we can generate arbitrary strings of as and bs on the upper string, while the lower can only be the same length as the upper if the congruence is satisfied. In this case the generation can be finished with the ending units $\begin{pmatrix} a \\ a \end{pmatrix}$ or $\begin{pmatrix} b \\ b \end{pmatrix}$. □

Based on the above, we have the next statement.

Theorem 2.

$$\mathscr{L}(SAS) \not\subseteq \mathscr{L}(WK_{|Q|=1}).$$

Proof. The statement is clear by combining Lemma 3 and Corollary 2. □

4 Conclusion

We have started to investigate the relationship between the computational power of WK automata and string assembling systems. We know that in the general case, WK automata are stronger than SAS, so the comparison with WK automata having a restricted number of states is of interest.

First we have shown that WK automata with at least two states are strictly more powerful than stateless WK automata, then constructed a language that can be generated by SAS, but cannot be accepted by stateless WK automata. The exact relationship of the two language classes, that is the question whether there are languages accepted by stateless WK automata which cannot be generated by SAS, remains open.

References

[1] Elena Czeizler, Eugen Czeizler, Lila Kari, and Kai Salomaa. On the descriptional complexity of watson-crick automata. *Theor. Comput. Sci.*, 410(35):3250–3260, 2009.

[2] Rudolf Freund, Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa. Watson-crick finite automata. In Harvey Rubin and David Harlan Wood, editors, *DNA Based Computers, Proceedings of a DIMACS Workshop, Philadelphia, Pennsylvania, USA, June 23-25, 1997*, volume 48 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 297–327. DIMACS/AMS, 1997.

[3] Lila Kari, Gheorghe Păun, Grzegorz Rozenberg, Arto Salomaa, and Sheng Yu. DNA computing, sticker systems, and universality. *Acta Informatica*, 35(5):401–420, May 1998.

[4] Dietrich Kuske and Peter Weigel. The role of the complementarity relation in watson-crick automata and sticker systems. In Cristian Calude, Elena Calude, and Michael J. Dinneen, editors, *Developments in Language Theory, 8th International Conference, DLT 2004, Auckland, New Zealand, December 13-17, 2004, Proceedings*, volume 3340 of *Lecture Notes in Computer Science*, pages 272–283. Springer, 2004.

[5] Martin Kutrib and Matthias Wendlandt. String assembling systems. *RAIRO Theor. Informatics Appl.*, 46(4):593–613, 2012.

[6] Andrei Păun and Mihaela Păun. State and transition complexity of watson-crick finite automata. In Gabriel Ciobanu and Gheorghe Paun, editors, *Fundamentals of Computation Theory, 12th International Symposium, FCT '99, Iasi, Romania, August 30 - September 3, 1999, Proceedings*, volume 1684 of *Lecture Notes in Computer Science*, pages 409–420. Springer, 1999.

[7] Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa. *DNA Computing - New Computing Paradigms*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 1998.