Theoretical Computer Science 410 (2009) 3250–3260

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

On the descriptional complexity of Watson–Crick automata

Elena Czeizler [a,*], Eugen Czeizler [a,1], Lila Kari [a], Kai Salomaa [b]

[a] Department of Computer Science, University of Western Ontario, London, Ontario N6A 5B7, Canada
[b] School of Computing, Queen's University, Kingston, Ontario K7L 3N6, Canada

ARTICLE INFO

Keywords:
Watson–Crick automata
State complexity
Determinism

ABSTRACT

Watson–Crick automata are finite state automata working on double-stranded tapes, introduced to investigate the potential of DNA molecules for computing. In this paper, we continue the investigation of descriptional complexity of Watson–Crick automata initiated by Păun et al. [A. Păun, M. Păun, State and transition complexity of Watson–Crick finite automata, in: G. Ciobanu, G. Paun (Eds.), Fundamentals of Computation Theory, FCT'99, in: LNCS, vol. 1684, 1999, pp. 409–420]. In particular, we show that any finite language, as well as any unary regular language, can be recognized by a Watson–Crick automaton with only two, and respectively three, states. Also, we formally define the notion of determinism for these systems. Contrary to the case of non-deterministic Watson–Crick automata, we show that, for deterministic ones, the complementarity relation plays a major role in the acceptance power of these systems.

1. Introduction

One of the current trends in nanoengineering is to develop nanomachines which can parse molecules of DNA and perform a finite number of tasks, e.g., the development of artificial enzymes [14] or smart drug design [15]. One of the first theoretical abstractions for such nanomachines is *Watson–Crick automata* [3], which is based on the idea of finite automata running on complete DNA-molecules. Formally, these machines are finite automata, with two independent reading heads, working on double-stranded sequences. The two strands of the input are separately scanned from left to right by read-only heads controlled by a common state. One of the main features of these automata is that characters on corresponding positions from the two strands of the input are related by a complementarity relation similar to the Watson–Crick complementarity of the DNA nucleotides. Several variants of these systems were investigated, e.g., in [9] and [12]; for a comprehensive presentation, we refer to both Chapter 5 from [11] as well as to [1] for a recent survey.

In this paper, we continue the study of the descriptional complexity of Watson–Crick automata initiated in [10]. Since, at each step, Watson–Crick automata can read blocks of more than one letter, a special feature of these systems is that, for a given number of states, even two or three, one can already define an infinite number of distinct automata. This is different from most of the usually considered models, such as ordinary finite automata or Turing machines. Thus, Watson–Crick automata allow, in some sense, to encode state information in the finite but unbounded number of transitions, and this makes it essentially more difficult to prove lower bounds for the number of states. In particular, we prove that several intricate families of languages can be accepted by Watson–Crick automata with a small, constant number of states. For instance, we show that any finite language and any unary regular language can be recognized by a Watson–Crick automaton with two, and respectively three, states. Also, we provide a family of languages which generates an infinite hierarchy from the

* Corresponding address: Department of IT, Åbo Akademi University, Turku 20520, Finland. Tel.: +1 519 661 2111.
E-mail addresses: elczeizl@abo.fi (Elena Czeizler), eczeizle@abo.fi (Eugen Czeizler), lila@csd.uwo.ca (L. Kari), ksalomaa@cs.queensu.ca (K. Salomaa).
[1] Current affiliation: Department of IT, Åbo Akademi University, Turku 20520, Finland.

0304-3975/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2009.05.001

point of view of the state complexity of the block automata recognizing them. Then, we show that this hierarchy collapses when we consider the case of Watson–Crick automata, that is, we prove that three states are enough when recognizing any language from this family. Recall that *block automata*, or *block-NFA*, are finite automata which, similarly to the case of Watson–Crick automata, can read an arbitrarily long finite sequence of characters at a time.

Also, we formally define the notion of deterministic Watson–Crick automata and investigate their properties. Although determinism is a well established notion in automata theory, it has never been considered yet in relation to Watson–Crick automata. In this paper, we define the notion of determinism using a syntactic property of the rewriting rules of the automaton. We also consider a weaker operational definition of determinism, namely weak determinism, and show that it is undecidable whether a given non-deterministic Watson–Crick automaton is weakly deterministic. For non-deterministic Watson–Crick automata, it was proved in [8] that we can always suppose the complementarity relation to be the identity. Hence, a natural question is whether the structure of the complementarity relation plays an active role in the deterministic case. Thus, we define the notion of strong determinism, embedding both the deterministic feature and the fact that the complementarity relation is the identity. We prove that these three levels of abstraction (i.e., weak determinism, determinism, and strong determinism) are all distinct from each other. Furthermore, we also show that non-deterministic Watson–Crick automata are strictly stronger than strongly deterministic ones.

The paper is organized as follows. In the next section, we fix our terminology and recall some known results. In Section 3, we investigate some properties of deterministic Watson–Crick automata, on the three levels of abstraction mentioned above. Also, we look at the relation between the acceptance power of these three variants of deterministic Watson–Crick automata. In Section 4, we investigate the state complexity of both deterministic and non-deterministic Watson–Crick automata. A preliminary version of this paper was given in [2].

2. Preliminaries

Let V be a finite alphabet. We denote by V^* the set of all finite words over V, by λ the empty word, and $V^+ = V^*\backslash\{\lambda\}$. For a word $u \in V^*$, we denote by $|u|$ its *length*, i.e., the number of letters occurring in it; in particular, $|\lambda| = 0$. We say that $u \in V^*$ is a *prefix* of a word v, and denote it by $u \leq v$, if there exists some $t \in V^*$ such that $v = ut$. Two words u and v are *prefix comparable*, denoted by $u \sim_p v$, if one of them is a prefix of the other. For a word $u = u_1 \dots u_n$, with $u_1, \dots, u_n \in V$, we denote by $u^R = u_n \dots u_1$ its reverse. Then, we say that u is *palindrome* if $u = u^R$.

Now let $\rho \subseteq V \times V$ be a symmetric relation, called *the Watson–Crick complementarity relation on* V. As suggested by the name, this relation is biologically inspired by the Watson–Crick complementarity of nucleotides in the double stranded DNA molecule. In accordance with the representation of DNA molecules, viewed as two strings written one on top of the other, we write $\binom{V^*}{V^*}$ instead of $V^* \times V^*$ and an element $(w_1, w_2) \in V^* \times V^*$ as $\binom{w_1}{w_2}$.

We denote $\left[\begin{smallmatrix} V \\ V \end{smallmatrix}\right]_\rho = \left\{\left[\begin{smallmatrix} a \\ b \end{smallmatrix}\right] \mid a, b \in V,\ (a, b) \in \rho\right\}$ and $WK_\rho(V) = \left[\begin{smallmatrix} V \\ V \end{smallmatrix}\right]_\rho^*$. The set $WK_\rho(V)$ is called *the Watson–Crick domain* associated to V and ρ. An element $\left[\begin{smallmatrix} a_1 \\ b_1 \end{smallmatrix}\right]\left[\begin{smallmatrix} a_2 \\ b_2 \end{smallmatrix}\right] \dots \left[\begin{smallmatrix} a_n \\ b_n \end{smallmatrix}\right] \in WK_\rho(V)$ can be also written in a more compact form as $\left[\begin{smallmatrix} w_1 \\ w_2 \end{smallmatrix}\right]$, where $w_1 = a_1 a_2 \dots a_n$ and $w_2 = b_1 b_2 \dots b_n$.

The essential difference between $\binom{w_1}{w_2}$ and $\left[\begin{smallmatrix} w_1 \\ w_2 \end{smallmatrix}\right]$ is that $\binom{w_1}{w_2}$ is just an alternative notation for the pair (w_1, w_2), whereas $\left[\begin{smallmatrix} w_1 \\ w_2 \end{smallmatrix}\right]$ implies that the strings w_1 and w_2 have the same length and the corresponding letters are connected by the complementarity relation.

A *(non-deterministic) Watson–Crick finite automaton* is a 6-tuple $\mathcal{M} = (V, \rho, Q, q_0, F, P)$, where: V is the (input) alphabet, $\rho \subseteq V \times V$ is the complementarity relation, Q is a finite set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and P is a finite set of transition rules of the form $q\binom{w_1}{w_2} \to q'$, denoting the fact that if the automaton is in a state q and parses $w_1 \in V^*$ on the upper strand and $w_2 \in V^*$ on the lower strand, then it enters the state q'.

A *configuration* of a Watson–Crick automaton is a pair $\left(s, \binom{u}{v}\right)$ where s is the current state of the automaton and $\binom{u}{v}$ is the part of the input word which has not been read yet. Now, a *transition* between two configurations is defined as follows. For $\binom{u_1 v_1}{u_2 v_2} \in \binom{V^*}{V^*}$ and $q, q' \in Q$ we write $q\binom{u_1 v_1}{u_2 v_2} \Rightarrow q'\binom{v_1}{v_2}$ if and only if $q\binom{u_1}{u_2} \to q'$. Let $\Rightarrow^*$ denote the reflexive and transitive closure of the relation $\Rightarrow$. Then, for a given $\left[\begin{smallmatrix} w_1 \\ w_2 \end{smallmatrix}\right] \in WK_\rho(V)$, a *computation* is a sequence of transitions $q_0\left[\begin{smallmatrix} w_1 \\ w_2 \end{smallmatrix}\right] \Rightarrow^* s\binom{u_1}{u_2}$ with $u_1, u_2 \in V^*$, starting from the initial state and such that either $s \in F$ and $u_1 = u_2 = \lambda$ or there are no more applicable transitions from configuration $\left(s, \binom{u_1}{u_2}\right)$. The *language accepted by a Watson–Crick* automaton is:

$$L(\mathcal{M}) = \left\{ w_1 \in V^* \mid q_0 \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \Rightarrow^* s \binom{\lambda}{\lambda}, \text{ with } s \in F, w_2 \in V^*, \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \in WK_\rho(V) \right\}.$$

Hence, a word w_1 is accepted by $\mathcal{M}$ if there exists a complementary word w_2 such that starting from the initial state, after parsing the whole input $\begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$ the automaton is in a final state. By convention, as suggested also in [10], whenever we compare the languages accepted by two Watson–Crick automata, we ignore the empty word.

Although the notion of determinism is well established in automata theory, it has never been considered in relation with Watson–Crick automata. Intuitively, this notion suggests that for each configuration we have at most one option to continue. Here, we propose three variants for this concept, each on a different level of abstraction. The first definition that we suggest illustrates the intuitive idea we presented above.

Definition 1. We say that a Watson–Crick automaton is *weakly deterministic* if in every configuration that can occur in some computation of the automaton there is at most one possibility to continue the computation.

Note that the previous definition does not provide a clear description of the structure of the transition rules of a deterministic automaton. Thus, we introduce our second definition.

Definition 2. We call a Watson–Crick automaton *deterministic* if whenever we have two rewriting rules of the form $q\left(\begin{smallmatrix} u \\ v \end{smallmatrix}\right) \to q'$ and $q\left(\begin{smallmatrix} u' \\ v' \end{smallmatrix}\right) \to q''$, then $u \nsim_p u'$ or $v \nsim_p v'$.

Clearly, the deterministic constraint is stronger than the weak one. However, unexpectedly, Example 2 shows that a weakly deterministic automaton need not be deterministic.

With the previous two definitions, for a given pair of words from the domain, $\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \in WK_\rho(V)$, there exists a unique computation. However, depending on the complementary relation ρ, the automaton can choose various words w_2 on the lower strand. Thus, in order to eliminate this selection, we introduce our third definition.

Definition 3. We call a Watson–Crick automaton *strongly deterministic* if it is deterministic and the Watson–Crick complementarity relation is the identity.

Note that the notion of strong determinism does not change if ρ is allowed to be a non-identity one-to-one function. As shown later by Theorem 4, strongly deterministic Watson–Crick automata are less powerful than deterministic ones.

Depending on the type of the states and of the rewriting rules, there are four subclasses of Watson–Crick automata. We say that a Watson–Crick automaton $\mathcal{M}$ is

- *stateless* if it has only one state, i.e., $Q = F = \{q_0\}$;
- *all-final* if all the states are final, i.e., $Q = F$;
- *simple* if for any rewriting rule $s\left(\begin{smallmatrix} w_1 \\ w_2 \end{smallmatrix}\right) \to s'$, either $w_1 = \lambda$ or $w_2 = \lambda$;
- *1-limited* if for any rewriting rule $s\left(\begin{smallmatrix} w_1 \\ w_2 \end{smallmatrix}\right) \to s'$, we have $|w_1 w_2| = 1$.

Recently, in [8], it was proved that for non-deterministic Watson–Crick automata we can always suppose the complementarity relation ρ to be the identity, denoted, from now on, by $\iota \subseteq V \times V$. However, this is not true anymore for the deterministic case, as shown later by Theorem 4.

3. Properties of deterministic Watson–Crick automata

In this section we investigate various aspects of the three types of deterministic Watson–Crick automata.

3.1. Deterministic Watson–Crick automata: Subclasses equivalence

One of the basic properties of non-deterministic Watson–Crick automata is that they are equivalent with simple and 1-limited ones, respectively, see [11]. The following two results show that this property still holds when we look at their deterministic variants.

Theorem 1. *Deterministic Watson–Crick automata are equivalent with deterministic simple Watson–Crick automata.*

Proof. Let $\mathcal{M} = (V, \rho, Q, q_0, F, P)$ be a deterministic Watson–Crick automaton. We want to construct an equivalent deterministic Watson–Crick automaton $\mathcal{M}' = (V, \rho, Q', q_0, F, P')$, where for any state $q \in Q'$ all rewriting rules from $q, q\left(\begin{smallmatrix} u_i \\ v_i \end{smallmatrix}\right) \to q_i$ with $1 \le i \le n$, satisfy exactly one of the following conditions:

$$\text{either } u_i = \lambda \text{ for all } 1 \le i \le n \text{ and } v_j \nsim_p v_k \text{ for any } 1 \le j \ne k \le n, \tag{1}$$

$$\text{or } v_i = \lambda \text{ for all } 1 \le i \le n \text{ and } u_j \nsim_p u_k \text{ for any } 1 \le j \ne k \le n. \tag{2}$$

Thus, we introduce some new intermediate states and transform the rewriting rules to achieve the above constraint.

If for a state q there exists only one rewriting rule $q\left(\begin{smallmatrix} u \\ v \end{smallmatrix}\right) \to q'$, with $u, v \ne \lambda$, then we introduce a new intermediate state $s \notin Q$ and the rewriting rules $q\left(\begin{smallmatrix} u \\ \lambda \end{smallmatrix}\right) \to s$ and $s\left(\begin{smallmatrix} \lambda \\ v \end{smallmatrix}\right) \to q'$. This newly introduced state s will not be used in any other rewriting rule. Note that if either $u = \lambda$ or $v = \lambda$, then this rule is already of the desired form, so we do not do anything.

Let us suppose now that for a state q we have several rewriting rules from q and let us denote $P_q = \left\{ q \begin{pmatrix} u_i \\ v_i \end{pmatrix} \to q_i \mid 1 \leq i \leq n \right\}$ the set of all such rules. We now describe an inductive procedure which stops after we transform all these rules into ones of the form (1) or (2). First, we define an equivalence relation for a set of words $W \subseteq V^+$. We say that for two words $u, v \in W$, $u \equiv v$ if and only if there exists $w \in W$ such that $w \leq u$ and $w \leq v$, i.e., w is a prefix of both u and v.
Case 1: If for some $1 \leq i \leq n$, $u_i = \lambda$, then for all $1 \leq j \leq n$, $v_j \neq \lambda$ and, moreover, $v_i \not\sim_p v_k$ for any $k \neq i$, since the initial automaton is deterministic. We partition the set P_q into equivalence classes using the relation $\equiv$ for the set of words occurring on the lower strand. For all classes containing only one rule $q \begin{pmatrix} u_i \\ v_i \end{pmatrix} \to q_i$, we have one of the following two possibilities. If $u_i = \lambda$, then we leave the rule unchanged, as it is already of the form (2). Otherwise, we introduce a new state s and replace the rule with the following two: $q \begin{pmatrix} \lambda \\ v_i \end{pmatrix} \to s$ and $s \begin{pmatrix} u_i \\ \lambda \end{pmatrix} \to q_i$. Note that at least one such class exists since for a rule with $u_i = \lambda$, $v_i \not\sim_p v_k$ for any $k \neq i$. Suppose now that there exists a class C containing at least two rules. Then, there exists a rule $q \begin{pmatrix} u_k \\ v_k \end{pmatrix} \to q_k$ in C such that v_k is a prefix of all the words occurring on the lower strand of the rules from C. Next, we introduce a new state s and we transform all the rules from C as follows. First, we introduce the rule $q \begin{pmatrix} \lambda \\ v_k \end{pmatrix} \to s$. Then, each rule $q \begin{pmatrix} u_j \\ v_j \end{pmatrix} \to q_j$ from C is replaced by $s \begin{pmatrix} u_j \\ v'_j \end{pmatrix} \to q_j$, where $v_j = v_k v'_j$ with $v'_j \in V^*$ and $|v'_j| < |v_j|$. Furthermore, for each newly introduced state s the set of rules P_s contains strictly less elements than P_q. So, we can repeat inductively the same procedure for each new set P_s. Moreover, note that after we make these transformations, all the rules initiating from state q are of the form (2).
Case 2: Suppose now that in P_q, $u_i \neq \lambda$ for all $1 \leq i \leq n$. Then we partition the set P_q into equivalence classes using the relation $\equiv$ for the set of words occurring on the upper strand. For all classes containing only one rule $q \begin{pmatrix} u_i \\ v_i \end{pmatrix} \to q_i$, we have one of the following two possibilities. If $v_i = \lambda$, then we leave the rule unchanged as it is already of the form (1). Otherwise, we introduce a new state s and replace the rule with the following two: $q \begin{pmatrix} u_i \\ \lambda \end{pmatrix} \to s$ and $s \begin{pmatrix} \lambda \\ v_i \end{pmatrix} \to q_i$. On the other hand, for all classes C containing at least two rules, we proceed as follows. By the definition of the relation $\equiv$, there exists a rule $q \begin{pmatrix} u_k \\ v_k \end{pmatrix} \to q_k$ in C such that u_k is a prefix of all the words occurring on the upper strand of the rules from C. Next, we introduce a new state s and we transform all the rules from C as follows. First, we introduce the rule $q \begin{pmatrix} u_k \\ \lambda \end{pmatrix} \to s$. Then, each rule $q \begin{pmatrix} u_j \\ v_j \end{pmatrix} \to q_j$ from C is replaced by $s \begin{pmatrix} u'_j \\ v_j \end{pmatrix} \to q_j$, where $u_j = u_k u'_j$ with $u'_j \in V^*$ and $|u'_j| < |u_j|$. Furthermore, although for each newly introduced state s the set of rules P_s contains at most an equal number of elements as P_q, the words on the upper strands are strictly shorter. So, we can repeat inductively the same procedure for each new set P_s. Moreover, note that after we make these transformations, all the rules initiating from state q are of the form (1).

Since, at each step, we strictly decrease the length of the words on the upper or on the lower strand of each rule, this procedure ends after finitely many steps. In addition, all the rules starting from a given state $q \in Q'$ are either of the form (1) or (2).

Since each newly introduced state acts just as an intermediate, the language accepted by $\mathcal{M}'$ is exactly $L(\mathcal{M})$. By the way we constructed the rules in P', $\mathcal{M}'$ is a deterministic simple Watson–Crick automaton. □

Using a similar technique, we can transform a deterministic simple Watson–Crick automaton into a deterministic 1-limited one. Thus, we can state the following result.

Corollary 2. *Deterministic Watson–Crick automata are equivalent with deterministic 1-limited Watson–Crick automata.*

3.2. Relations among non-deterministic and deterministic Watson–Crick automata

As stated in [11], a 1-limited Watson–Crick automaton can be interpreted as a one-way two-headed automaton where the two strands are interrelated through the complementarity relation. Furthermore, as arbitrary Watson–Crick automata are equivalent with 1-limited ones (see [11]) we can also state that they are equivalent with one-way two-headed automata. Moreover, by Corollary 2, deterministic Watson–Crick automata using the identity complementarity relation are equivalent with deterministic one-way two-headed automata. Thus, we can prove the following result.

Theorem 3. *Non-deterministic Watson–Crick automata are more powerful than strongly deterministic ones.*

Proof. Let $L = \{w \in V^* \mid w = w^R\}$ be the set of palindrome words and $L' = V^* \setminus L$ be its complement. It is known (see [6] and [16]) that L' can be recognized by a non-deterministic one-way two-headed automaton, but not by a deterministic one. Thus, L' can be recognized by a nondeterministic Watson–Crick automaton, but not by a strongly deterministic one. □

The next example shows that if we use a non-injective complementarity relation ρ, then we can construct a deterministic Watson–Crick automaton accepting the language L' from the previous result. Thus, L' cannot be used to differentiate between the accepting power of deterministic and non-deterministic Watson–Crick automata.

Example 1. Let $\mathcal{M} = (V, \rho, Q, q_0, F, P)$ be a Watson–Crick automaton, where $V = \{a, b, v_a, v_b, c\}$, $\rho = \{(a, a), (a, v_a), (v_a, a), (b, b), (b, v_b), (v_b, b), (c, a), (a, c), (c, b), (b, c)\}$, $Q = \{q_0, q_1, q_a, q_b\}$, $F = \{q_1\}$, and we have the following transitions:

- $q_0 \binom{\lambda}{x} \to q_0$, $q_0 \binom{\lambda}{v_x} \to q_x$, with $x \in \{a, b\}$,
- $q_x \binom{y}{z} \to q_x$, with $x, y, z \in \{a, b\}$,
- $q_x \binom{zy}{c} \to q_1$, with $x, y, z \in \{a, b\}$, $x \neq y$,
- $q_1 \binom{x}{\lambda} \to q_1$, with $x \in \{a, b\}$.

It is easy to see that $\mathcal{M}$ is deterministic. Let $w \in \{a, b\}^*$, $w = w_1 w_2 \dots w_n$ with $w_i \in \{a, b\}$. If $w \neq w^R$, then there exists a position k on the first half of w such that $w_k \neq w_{n-k}$. The automaton $\mathcal{M}$ accepts the word w only when we choose, as its complement, the word $w_1 \dots w_{k-1}\, v_{w_k}\, w_{k+1} \dots w_{n-1} c$. On the other hand, if w is a palindrome word, then it will not be accepted, regardless of what complement we use; thus, $L(\mathcal{M}) = \{w \in \{a, b\}^+ \mid w \neq w^R\}$.

Thus, we can formulate the following result showing that the complementarity relation plays an active role for deterministic Watson–Crick automata, contrary to the non-deterministic case, see [8].

Theorem 4. *Strongly deterministic Watson–Crick automata are strictly weaker than deterministic ones.*

Now, a natural and interesting question, which still remains open, is whether non-deterministic Watson–Crick automata are equivalent to deterministic ones, clearly, using a non-injective complementarity relation.

As we already stated in the previous section, the deterministic constraint is stronger than the weakly deterministic one. The following example presents a weakly deterministic Watson–Crick automaton which is not deterministic.

Example 2. Let us consider the non-regular language $L = \{a^n b^n \mid n \geq 1\} \cup \{b^n a^n \mid n \geq 1\}$. Then, we take $\mathcal{M} = (V, \iota, Q, q_0, F, P)$, where $V = \{a, b\}$, $Q = \{q_0, q_1, q_2, q_3, q_4\}$, $F = \{q_3, q_4\}$, and P contains the following productions:

- $q_0 \binom{a}{\lambda} \to q_1$ and $q_0 \binom{\lambda}{b} \to q_2$,
- $q_1 \binom{a}{\lambda} \to q_1$, $q_1 \binom{b}{a} \to q_3$, $q_3 \binom{b}{a} \to q_3$, $q_3 \binom{\lambda}{b} \to q_3$,
- $q_2 \binom{\lambda}{b} \to q_2$, $q_2 \binom{b}{a} \to q_4$, $q_4 \binom{b}{a} \to q_4$, $q_4 \binom{a}{\lambda} \to q_4$.

Clearly, the language recognized by this automaton is L. The only non-deterministic choice can be made in q_0 at the beginning of a computation. However, given an input, this choice becomes uniquely determined. Thus, $\mathcal{M}$ is a weakly deterministic Watson–Crick automaton which is not deterministic, due to the first two rules in q_0.

It is not yet known whether weakly deterministic Watson–Crick automata recognize more languages than deterministic ones.

3.3. Strongly deterministic stateless Watson–Crick automata

Even though strongly deterministic Watson–Crick automata are weaker than non-deterministic ones, they still prove to be more powerful than finite automata. The following example shows that even if we take strongly deterministic stateless Watson–Crick automata, we can still recognize some non-regular languages.

Example 3. Consider the non-regular language $L = \{a^{2n} b^{2n} \mid n \geq 1\} \cup \{b^{2n} a^{2n} \mid n \geq 1\}$. Let $\mathcal{M} = (V, \iota, Q, q_0, F, P)$, where $V = \{a, b\}$, $Q = F = \{q_0\}$, i.e., the automaton is stateless, and P contains the following three productions:

$$q_0 \binom{aa}{a} \to q_0, \qquad q_0 \binom{b}{a} \to q_0, \qquad q_0 \binom{b}{bb} \to q_0.$$

Let us consider the case when both reading heads of the Watson–Crick automaton are on the same position (e.g., at the beginning of the input word). Depending on whether the first letter of the rest of the input word is either a or b, the next time when the reading heads of the automaton are again on the same position will be after the automaton parses a block of the form $a^{2n} b^{2n}$ or $b^{2n} a^{2n}$, respectively, for some $n \geq 1$. Since a word is accepted only when both reading heads have finished parsing the input, and thus they are on the same position, we conclude that an accepting word must contain one or more blocks and, moreover, each of them is of the form $a^{2n} b^{2n}$ or $b^{2n} a^{2n}$ for some $n \geq 1$. Thus, the language accepted by the automaton is L^*, which is a non-regular language. Furthermore, note that the Watson–Crick automaton $\mathcal{M}$ is deterministic.

Thus, we can naturally ask now what kind of languages can be accepted by strongly deterministic stateless Watson–Crick automata. In [11] it was proved that if L is the language accepted by a non-deterministic stateless Watson–Crick automaton, then $L = L^+$. For the case of strongly deterministic automata, we refine this result using prefix codes. We say that a language L is a *prefix code* if no two (distinct) words of the language are prefix comparable. That is, for any two words $w_1, w_2 \in L$ such that $w_1 \neq w_2$, we have $w_1 \not\sim_p w_2$. For more detailed information on prefix codes we refer to [7].

Proposition 5. *For any strongly deterministic stateless Watson–Crick automaton $\mathcal{M}$, there exists a prefix code L such that $L(\mathcal{M}) = L^*$.*

Proof. Let $L \subseteq L(\mathcal{M})$ be the language obtained from $L(\mathcal{M})$ by taking all those non-empty words whose proper prefixes are not accepted by the automaton $\mathcal{M}$, i.e.,

$$L = \{w \in L(\mathcal{M}) \mid w \neq \lambda \text{ and for all } v \in L(\mathcal{M}) \setminus \{\lambda\} \text{ if } v \leq w \text{ then } v = w\}.$$

Clearly, L is a prefix code and $L^* \subseteq L(\mathcal{M})$. We show that if $v \in L(\mathcal{M})$, then $v \in L^*$.

Let q_0 be the state of the automaton $\mathcal{M}$. Then, all the rewriting rules from $\mathcal{M}$ are of the form $q_0 \binom{u_i}{v_i} \to q_0$ for $1 \leq i \leq n$, where for all $1 \leq i \neq j \leq n$ either $u_i \not\sim_p u_j$ or $v_i \not\sim_p v_j$ (or both). Let us consider now a non-empty word v recognized by the automaton $\mathcal{M}$, i.e., $v \in L(\mathcal{M})$, such that $v \notin L$. Then, there must exist a word $w \in L$, such that $w \leq v$; let $v = ww'$ for some w'. Let $q_0 \binom{u_{i_1}}{v_{i_1}} \to q_0$ and $q_0 \binom{u_{j_1}}{v_{j_1}} \to q_0$ be the first rewriting rules applied when recognizing w and v, respectively. Since $w \leq v$ we conclude that $u_{i_1} \sim_p u_{j_1}$ and also $v_{i_1} \sim_p v_{j_1}$. Thus, since $\mathcal{M}$ is deterministic, it implies that the two rules must coincide, i.e., $u_{i_1} = u_{j_1}$ and $v_{i_1} = v_{j_1}$. Similarly, we can conclude that all the rewriting rules applied when recognizing w and $v = ww'$ are exactly the same, until we start parsing w'. So, at some moment in the recognition process of v, after parsing w, both heads of the Watson–Crick automaton are at the beginning of w'. Since q_0 is the only state of $\mathcal{M}$ we conclude that w' is also accepted by the automaton, i.e., $w' \in L(\mathcal{M})$.

To conclude, for any non-empty word $v \in L(\mathcal{M}) \setminus L$, there exists $w \in L$ such that $v = ww'$ and $w' \in L(\mathcal{M})$. By applying this process inductively, we obtain $v \in L^+$. □

Now, it seems natural to ask whether strongly deterministic stateless Watson–Crick automata can recognize the Kleene star of any prefix code. We start this analysis by looking first at finite prefix codes.

Proposition 6. *Let $L \subseteq V^*$ be a finite prefix code. Then, there exists a strongly deterministic stateless Watson–Crick automaton recognizing the language L^*.*

Proof. Let $L = \{w_1, \ldots, w_n\} \subseteq V^*$, with $n \geq 1$, be a finite prefix code. We construct a stateless Watson–Crick automaton $\mathcal{M} = (V, \iota, \{q_0\}, q_0, \{q_0\}, P)$ where P contains all the rewriting rules of the form $q_0 \binom{w_i}{w_i} \to q_0$ with $1 \leq i \leq n$. It is easy to see that the automaton $\mathcal{M}$ accepts the language L^*. Since L is a prefix code, the constructed Watson–Crick automaton is deterministic. □

As illustrated by Example 3, there exist also some infinite prefix codes L such that the language L^* is recognized by a deterministic stateless Watson–Crick automaton. However, this cannot be generalized for the Kleene closure of any infinite prefix code, since there are countably many Watson–Crick automata, while the set of all prefix codes is uncountable. Thus, we have the following result.

Proposition 7. *There exists an infinite prefix code L such that the language L^* cannot be accepted by any strongly deterministic stateless Watson–Crick automaton.*

The following example presents such an infinite prefix code.

Example 4. Let us take the infinite prefix code $L = \{a^n b \mid n \geq 1\}$ and assume there exists a strongly deterministic stateless Watson–Crick automaton $\mathcal{M} = (V, \iota, \{q_0\}, q_0, \{q_0\}, P)$ accepting L^*. Since $\mathcal{M}$ has a finite number of transition rules and it accepts all words of the form $a^n b$ with $n \geq 1$, it must contain a rule of the form $q_0 \binom{a^i}{a^j} \to q_0$ with $i, j \geq 0$. If $i = j$, then we would also have $a^i \in L(\mathcal{M})$, which is a contradiction. Thus, from now on we can suppose without loss of generality that $i > j \geq 0$; the case when $j > i \geq 0$ is symmetric. Moreover, in P we cannot have any other rule of the form $q_0 \binom{a^{i_1}}{a^{j_1}} \to q_0$ since $\mathcal{M}$ is deterministic. Thus, all other rules from P must read the letter b at least on one string. Let now $P = \{p_1, \ldots, p_k\}$ be the set of all productions from $\mathcal{M}$, where $p_1 : q_0 \binom{a^i}{a^j} \to q_0$ and all the others involve also the character b. When the automaton $\mathcal{M}$ accepts words from $L \subseteq L^*$ any of the productions $p_2, \ldots, p_k$ may be used at most on the last two steps. Since L is infinite, this means that there exist two indices l_1 and l_2 such that the rules p_{l_1} and p_{l_2} are used for the last two steps when accepting an infinite number of words from L; let this set be $L' = \{a^{n_1} b,\ a^{n_2} b, \ldots\}$. Let now $p_{l_1} : q_0 \binom{u_{l_1}}{v_{l_1}} \to q_0$ and $p_{l_2} : q_0 \binom{u_{l_2}}{v_{l_2}} \to q_0$. Then, $|u_{l_1}|_a + |u_{l_2}|_a - |v_{l_1}|_a - |v_{l_2}|_a = C$, where C is a constant. However, when accepting words from L', by applying rule p_1 we can obtain an arbitrarily large difference between the two reading heads, e.g., larger than C. So, we cannot use only the rules p_1, p_{l_1}, and p_{l_2} to accept all the words from L'.

3.4. Undecidability results

The next result gives an undecidability property for Watson–Crick automata.

Theorem 8. *It is undecidable whether a given non-deterministic Watson–Crick automaton is weakly deterministic.*

Proof. In order to prove this, we use the fact that it is undecidable whether a Turing machine accepts the empty word, see [4]. Given a deterministic Turing machine T, we construct a Watson–Crick automaton $\mathcal{M}$ which verifies whether the input is a valid sequence of consecutive configurations of the Turing machine starting from the empty tape. Since the Turing machine is deterministic, we can simulate each of its transitions with deterministic rewriting rules of the Watson–Crick automaton. The only moment when we can continue a computation in $\mathcal{M}$ in a non-deterministic manner, i.e., by choosing to apply one rule from several possible ones, is when the Turing machine halts. The detailed description of the Watson–Crick automaton is very technical and we include it in the Appendix.

Intuitively, the Watson–Crick automaton $\mathcal{M}$ receives an input of the form $\begin{bmatrix} \sharp\, q_0\, \sharp\, u_1 q_{i_1} v_1\, \sharp \ldots \sharp\, u_n q_{i_n} v_n\, \sharp \\ \sharp\, q_0\, \sharp\, u_1 q_{i_1} v_1\, \sharp \ldots \sharp\, u_n q_{i_n} v_n\, \sharp \end{bmatrix}$, where q_0 is the initial state of the Turing machine, $q_{i_1}, \ldots, q_{i_n}$ are states of T and $u_i v_i$ is the tape content at step i, where the reading head is on the first character of v_i. Then, the Watson–Crick automaton verifies in a deterministic way that, for any $1 \leq j \leq n-1$, $u_{j+1} q_{i_{j+1}} v_{j+1}$ is a valid configuration obtained from $u_j q_{i_j} v_j$, by applying one of the deterministic rules of T. As soon as the Turing machine enters a final state, we let the Watson–Crick automaton finish reading the rest of the input in a non-deterministic way. Thus, we can make a non-deterministic choice in a computation of the Watson–Crick automaton if and only if the Turing machine enters a final state and halts when started with the empty word as input. Since it is undecidable whether a given Turing machine accepts the empty word, it also becomes undecidable whether the non-deterministic Watson–Crick automaton is weakly deterministic. □

As a consequence of the proof of Theorem 8, we get also an alternative proof for the known undecidability result of the emptiness problem for (non-) deterministic multihead automata [13]. Indeed, in the proof of Theorem 8 we construct a Watson–Crick automaton which accepts an input if and only if it represents a valid sequence of consecutive configurations of a Turing machine starting from the empty tape. However, since it is undecidable whether a Turing machine accepts the empty word, we obtain, in turn, that it is undecidable whether the language recognized by the Watson–Crick automaton is empty or not. Furthermore, the construction from the previous theorem can be easily made fully deterministic.

Corollary 9. *Given a (deterministic) Watson–Crick automaton $\mathcal{M}$, it is undecidable whether the recognized language $L(\mathcal{M})$ is empty.*

4. State complexity of Watson–Crick automata

It is well-known that Watson–Crick automata are more powerful than classical finite automata, see e.g., [11]. In addition, it was shown in [10] that Watson–Crick automata recognize some regular languages in a more efficient manner. We devote this section to the study of state complexity of languages accepted by deterministic and non-deterministic Watson–Crick automata. For more details on state complexity, we refer the reader to [5] and [17]. Note that in [8] it was proved that any non-deterministic Watson–Crick automaton can be transformed into one where the complementarity relation is the identity. Moreover, since this transformation preserves the number of states, when working with non-deterministic Watson–Crick automata we can suppose, without loss of generality, that the complementarity relation ρ is actually the identity $\iota \subseteq V \times V$.

It is well-known that the state complexity of some families of finite languages is unbounded when we consider the finite automata recognizing them. However, as illustrated by our next result, this is not the case anymore when we consider the non-deterministic Watson–Crick automata recognizing them.

Theorem 10. *Any finite language can be recognized by a non-deterministic Watson–Crick automaton with two states.*

Proof. Let $L = \{w_1, \ldots, w_n\} \subset V^*$ be a finite language. We construct the Watson–Crick automaton $\mathcal{M} = (V, \iota, \{q_0, q_1\}, q_0, F, P)$, where $F = \{q_1\}$ and P contains rewriting rules of the form $q_0 \binom{w_i}{w_i} \to q_1$, for all $1 \leq i \leq n$. Clearly, the language recognized by $\mathcal{M}$ is L. □

On the other hand, if we restrict to strongly deterministic Watson–Crick automata, then there exists a family of finite languages with unbounded state complexity.

Theorem 11. *For any $k \geq 2$ there exists a finite language $L_k \subseteq V^*$ such that any strongly deterministic Watson–Crick automaton recognizing L_k needs at least k states.*

Proof. Let $L_k = \{a^i \mid 1 \leq i \leq k-1\}$, with $k \geq 2$, and $\mathcal{M}$ be a strongly deterministic Watson–Crick automaton recognizing it. Since $L_k \subset \{a\}^*$, any rule of $\mathcal{M}$ is of the form $q_1 \binom{a^i}{a^j} \to q_2$ for some $i, j \geq 0$. Furthermore, since $\mathcal{M}$ is deterministic, for every state q_1 we can have at most one transition rule of the form mentioned above.

We claim now that in each final state q_f we accept only one word. Otherwise, let us suppose that in q_f we accept two words $a^i, a^j \in L_k$ with $i < j$. Since $\mathcal{M}$ is deterministic and from every state we have at most one transition rule, any word of the form $a^{i+l(j-i)}$ with $l \geq 0$ is in the language accepted by $\mathcal{M}$. Hence, the accepted language would not be finite.

Thus, a strongly deterministic Watson–Crick automaton accepting L_k needs at least k states: one initial and $k-1$ final ones. Clearly, such a deterministic Watson–Crick automaton can be easily constructed. □

On the other hand, if we look at Watson–Crick automata with non-injective complementarity relations, then the infinite hierarchy from the previous theorem collapses.

Theorem 12. *For any $k \geq 2$, the language $L_k = \{a^i \mid 1 \leq i \leq k-1\}$ can be recognized by a deterministic Watson–Crick automaton with two states and having a non-injective complementarity relation.*

Proof. Let $L_k = \{a^i \mid 1 \leq i \leq k-1\}$, for a given $k \geq 2$. We construct a deterministic Watson–Crick automaton $\mathcal{M} = (V, \rho, Q, q_0, F, P)$, where $V = \{a, b, c, d\}$, $\rho = \{(a, b), (b, a), (a, c), (c, a), (a, d), (d, a)\}$, $Q = \{q_0, q_1\}$, $F = \{q_1\}$, and the set P of rewriting rules is

$$P = \left\{ q_0 \begin{pmatrix} a^{2i} \\ b^i c^i \end{pmatrix} \to q_1, \mid 1 \leq i \leq \left\lfloor \frac{k-1}{2} \right\rfloor \right\} \cup \left\{ q_0 \begin{pmatrix} a^{2j-1} \\ b^{j-1} d c^{j-1} \end{pmatrix} \to q_1 \mid 1 \leq j \leq \left\lceil \frac{k-1}{2} \right\rceil \right\}.$$

Then, it is easy to see that $L(\mathcal{M}) = L_k$ and, moreover, $\mathcal{M}$ is deterministic. □

It is only natural now to also ask whether, for the case of non-deterministic Watson–Crick automata, there exists a family of languages with unbounded state complexity; this question was first stated in [10].

We show next that any unary regular language can be recognized by a (non-deterministic) Watson–Crick automaton with only three states. Actually, we prove this property for block non-deterministic finite automata, which can be considered a special case of Watson–Crick automata.

A *block non-deterministic finite automaton* (for short, block-NFA) is a finite automaton $\mathcal{A} = (Q, \Sigma, \delta, q_0, Q_F)$ where δ consists of a finite number of rules (q_1, w, q_2), with $q_1, q_2 \in Q$ and $w \in \Sigma^*$. Clearly, a block-NFA is a special case of a Watson–Crick automaton where the two reading heads are required to always move together.

An arbitrary unary regular language is denoted by a regular expression of the form

$$a^{j_1} + \ldots + a^{j_{r-1}} + a^{j_r}(a^{i_1} + \ldots + a^{i_{s-1}})(a^m)^*, \tag{3}$$

where $0 \leq j_1 < \cdots < j_{r-1} < j_r$, $0 \leq i_1 < \cdots < i_{s_1} < m$, and $r, s \geq 0$. Here the words $a^{j_1}, \ldots, a^{j_{r-1}}$ are usually called the "tail" of the language and the remaining words belong to the cycle of length m.

Theorem 13. *Any unary regular language L can be recognized by a block-NFA $\mathcal{A}$ having three states. Furthermore, $\mathcal{A}$ can be restricted to be unambiguous.*

Proof. Let L be a unary regular language described by a regular expression of the form (3). We construct a block-NFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, Q_F)$ where $Q = \{q_0, q_1, q_2\}$, $Q_F = \{q_1, q_2\}$, and δ contains the rules

- (q_0, a^x, q_1), where $x \in \{j_1, \ldots j_{r-1}\}$,
- (q_0, a^x, q_2) where $x \in \{j_r + i_1, \ldots, j_r + i_{s-1}\}$,
- (q_2, a^m, q_2).

It is easy to see that each word of L is accepted by a unique computation of $\mathcal{A}$. □

Since any block NFA can be seen also as a Watson–Crick automaton where the two heads always move together, the following result is an immediate consequence.

Corollary 14. *Any unary regular language can be recognized by a non-deterministic Watson–Crick automaton with only three states.*

Note that the construction from the previous theorem can be slightly modified such that any unary regular language can be accepted by a deterministic Watson–Crick automaton with a non-injective complementarity relation. Moreover, we can go a step further and prove a more general result.

Theorem 15. *Any block NFA $\mathcal{A}$ with n states can be simulated by a deterministic Watson–Crick automaton with at most n states.*

Proof. Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, Q_F)$ be a block NFA with $|Q| = n$. We can suppose that $\mathcal{A}$ does not contain any λ-transitions, i.e., rules of the form (q, λ, q') with $q, q' \in Q$. Otherwise, we can construct an equivalent block NFA, $\mathcal{A}'$ with $L(\mathcal{A}) = L(\mathcal{A}')$, which does not have λ-transitions and, moreover $\mathcal{A}'$ has at most n states. Let now k be the number of all rules from $\mathcal{A}$. Then, we denote the set of all rules from $\mathcal{A}$ as (q_i, u_i, q_i') with $q_i, q_i' \in Q$ and $u_i \in \Sigma^+$ with $1 \leq i \leq k$.

We construct a deterministic Watson–Crick automaton $\mathcal{M} = (V, \rho, Q, q_0, Q_F, P)$ such that $L(\mathcal{M}) = L(\mathcal{A})$ as follows. First, we take $V = \Sigma \cup V_k$, where $V_k = \{a_1, \ldots, a_k\}$ and $V_k \cap \Sigma = \emptyset$ and the complementarity relation $\rho = \iota_\Sigma \cup \Sigma \times V_k \cup V_k \times \Sigma$, where ι_Σ is the identity relation on Σ. Then, for each rule (q_i, u_i, q_i') from $\mathcal{A}$, we introduce in $\mathcal{M}$ a transition rule of the form $q_i \begin{pmatrix} u_i \\ v_i a_i \end{pmatrix} \to q_i'$ where $a_i \in V_k$ and $v_i \in \Sigma^*$ is obtained from u_i by cutting the last character, i.e., $v_i = pref_{|u_i|-1}(u_i)$.

Note that for each rule from $\mathcal{A}$, we used a different character a_i on the lower strand of the corresponding rule in $\mathcal{M}$. Thus, the Watson–Crick automaton $\mathcal{M}$ is deterministic and, in addition, it has the same number of states as $\mathcal{A}$. Also, since the rules of $\mathcal{M}$ simulate the behavior of the block automaton $\mathcal{A}$, it is easy to see that $L(\mathcal{A}) = L(\mathcal{M})$. □

As illustrated by Theorem 10 and Corollary 14, some infinite hierarchies collapse when switching from the finite automata to the Watson–Crick automata recognizing them. However, in both cases, this is mainly due to the fact that Watson–Crick automata can read blocks of letters at each step. Thus, we investigate next what happens with infinite hierarchies of languages accepted by block-NFA's when we consider the Watson–Crick automata recognizing them.

Theorem 16. *For any $k \geq 1$, there exists a regular language L_k such that any block-NFA recognizing L_k needs more than k states.*

Proof. Let $L_k = (10^*)^{k+1}1$ and assume that L_k is recognized by a block-NFA $\mathcal{A}$ having k states. Let N be the length of the longest word appearing in rules of $\mathcal{A}$ and take $w_N = (10^N)^{k+1}1$. Since $w_N \in L_k$, $\mathcal{A}$ has an accepting computation C on w_N. By the choice of N, for each $i = 1, \ldots, k+1$, between the ith and $(i+1)$st occurrence of 1 we have at least one applicable transition in the computation C. Let q_i be a state occurring in computation C between the ith and $(i+1)$st occurrence of 1, $1 \leq i \leq k+1$. By the pigeon-hole-principle there exist $1 \leq j_1 < j_2 \leq k+1$ such that $q_{j_1} = q_{j_2}$. Thus $\mathcal{A}$ accepts also words having $k + 2 + l(j_2 - j_1)$ occurrences of 1 for any $l \geq -1$; clearly, some of them are not in L_k. □

Considering the state complexity of languages, the previous result shows the existence of an infinite hierarchy. Although block-NFA's are a particular type of Watson–Crick automata, we show next that this hierarchy collapses when we look at the Watson–Crick automata accepting them.

Theorem 17. *For any $k \geq 1$, the language $L_k = (10^*)^{k+1}1$ can be recognized by a Watson–Crick automaton with three states.*

Proof. Let $\mathcal{M}_k = (V, \iota, \{q_0, q_1, q_2\}, q_0, F, P)$ be a Watson–Crick automaton, where $V = \{0, 1\}$, $F = \{q_2\}$, and P contains the following productions:

- $q_0 \begin{pmatrix} \lambda \\ 1u \end{pmatrix} \to q_1$, for any word $u \in \{0, 1\}^k$,
- $q_1 \begin{pmatrix} 0 \\ x \end{pmatrix} \to q_1$, where $x \in \{0, 1\}$,
- $q_1 \begin{pmatrix} 1 \\ \lambda \end{pmatrix} \to q_1$, and $q_1 \begin{pmatrix} 1 \\ 1 \end{pmatrix} \to q_2$.

With the first production, we advance the lower head $k+1$ characters, independently of what we have on the tape. Then, the upper head will catch up this advance only after reading $k+1$ times the character 1. Then, the automaton enters the final state after reading the last character 1. Since a word is accepted only when both reading heads have completely parsed the input word, the language recognized by $\mathcal{M}$ is L_k. □

The results of this section illustrate the fact that there are many complex languages which can be accepted by Watson–Crick automata with a bounded number of states. Although we do not include a formal proof in this paper, another such complex family of languages accepted by Watson–Crick automata with only a small number of states is $L_k = 10^+1^20^+ \ldots 0^+1^{2^k}$, for any $k \geq 2$. However, it remains open whether, for all $k > 1$, there exists a language L_k such that any non-deterministic Watson–Crick automaton accepting L_k needs at least k states [10].

Acknowledgments

This research was supported by Natural Sciences and Engineering Research Council of Canada Discovery Grant, UWO Faculty of Science Grant, and Canada Research Chair Award to L.K. and Natural Sciences and Engineering Research Council of Canada Discovery Grant to K.S.

Appendix

Theorem 18. *It is undecidable whether a given non-deterministic Watson–Crick automaton is weakly deterministic.*

Proof. Let $T = (Q, \Gamma, B, \Sigma, \delta, q_0, F)$ be a deterministic Turing machine, where Q is a finite set of states, Γ is a finite set of tape symbols, $B \in \Gamma$ is the blank symbol (the only symbol allowed to occur on the tape infinitely often at any step during the computation), $\Sigma \subseteq \Gamma \setminus \{B\}$ is the set of input symbols, $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, N\}$ is a partial function called the transition function, where L is left shift, R is right shift, N is no shift, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of final states.

We construct a Watson–Crick automaton $M = (V, \iota, Q', r_0, F', P)$ where $V = Q \cup \Gamma \cup \{\sharp\}$, $Q' = \{r_0, r_v, r_n, r_f, r^{\sharp_l}, r^{\sharp_u}\} \cup \{r_n^{x_u}, r_n^{x_l} \mid x \in \Gamma\} \cup \{r_{\sharp xy} \mid x, y \in \Gamma \cup Q \cup \{\sharp\}\} \cup \{r_{xyz} \mid x \in \Gamma, y \in \Gamma \cup Q, z \in \Gamma \cup Q \cup \{\sharp\}\}$, and $F = \{r_f\}$.

We present, below, a list of rewriting rules and a short description of their role. However, our constructed Watson–Crick automaton contains only a subset of these rules according to the transitions of the Turing machine T.

The initial rule of M is of one of the following forms depending on the structure of the initial transition of the Turing machine.

- $r_0 \begin{pmatrix} \sharp q_0 \sharp q_1 Ba \\ \sharp q_0 \sharp q_1 B \end{pmatrix} \to r_{\sharp q_1 B}$ if $\delta(q_0, B) = (q_1, a, L)$
- $r_0 \begin{pmatrix} \sharp q_0 \sharp q_1 a \\ \sharp q_0 \sharp q_1 a \end{pmatrix} \to r_{\sharp q_1 a}$ if $\delta(q_0, B) = (q_1, a, N)$
- $r_0 \begin{pmatrix} \sharp q_0 \sharp a q_1 \\ \sharp q_0 \sharp a q_1 \end{pmatrix} \to r_{\sharp a q_1}$ if $\delta(q_0, B) = (q_1, a, R)$

With an initial rule of either of these forms, we simulate the first transition of T and we embed a memory of the previous three symbols from the lower strand. Using them, we can check that the sequence indeed contains consecutive configurations.

The following group of rules is applied after the lower head has read the symbol $\sharp$ which separates two consecutive configurations. Moreover, if the following symbol is a state of T, then we also check that a valid production was applied.

- $r_{\sharp xy}\begin{pmatrix}\sharp\\ z\end{pmatrix} \to r_{xyz}$, for all $x \in \Gamma, y, z \in \Gamma \cup Q$
- $r_{\sharp xy}\begin{pmatrix}\sharp qBa\\ \lambda\end{pmatrix} \to r_v$ if $x \in Q, \delta(x, y) = (q, a, L)$, and $q \notin F$
- $r_{\sharp xy}\begin{pmatrix}\sharp qBa\\ \lambda\end{pmatrix} \to r_n$ if $x \in Q, \delta(x, y) = (q, a, L)$, and $q \in F$
- $r_{\sharp xy}\begin{pmatrix}\sharp qa\\ \lambda\end{pmatrix} \to r_v$ if $x \in Q, \delta(x, y) = (q, a, N)$, and $q \notin F$
- $r_{\sharp xy}\begin{pmatrix}\sharp qa\\ \lambda\end{pmatrix} \to r_n$ if $x \in Q, \delta(x, y) = (q, a, N)$, and $q \in F$
- $r_{\sharp xy}\begin{pmatrix}\sharp aq\\ \lambda\end{pmatrix} \to r_v$ if $x \in Q, \delta(x, y) = (q, a, R)$, and $q \notin F$
- $r_{\sharp xy}\begin{pmatrix}\sharp aq\\ \lambda\end{pmatrix} \to r_n$ if $x \in Q, \delta(x, y) = (q, a, R)$, and $q \in F$

The state r_v is reached only after we checked that we have a valid transition in T not involving a final state. Then, we only need to check that the rest of the input tape is the same in the two consecutive configurations. We do this using the following group of rewriting rules.

- $r_v\begin{pmatrix}x\\ x\end{pmatrix} \to r_v$, for all $x \in \Gamma$
- $r_v\begin{pmatrix}\lambda\\ \sharp xy\end{pmatrix} \to r_{\sharp xy}$, for all $x, y \in \Gamma \cup Q$

On the other hand, the state r_n is reached after we checked that in T we have a valid transition ending in a final state, i.e., T has accepted the empty string. This is the first moment when the Watson–Crick automaton can choose, in a non-deterministic way, which rules to use in order to verify that the rest of the input tape is the same in these last two configurations. Then, we use the final state r_f to finish parsing the input also with the lower head.

- $r_n\begin{pmatrix}x\\ \lambda\end{pmatrix} \to r_n^{x_u}$ for all $x \in \Gamma$
- $r_n\begin{pmatrix}\lambda\\ x\end{pmatrix} \to r_n^{x_l}$ for all $x \in \Gamma$
- $r_n^{x_u}\begin{pmatrix}\lambda\\ x\end{pmatrix} \to r_n$ for all $x \in \Gamma$
- $r_n^{x_l}\begin{pmatrix}x\\ \lambda\end{pmatrix} \to r_n$ for all $x \in \Gamma$
- $r_n\begin{pmatrix}\lambda\\ \sharp\end{pmatrix} \to r^{\sharp_l}$
- $r_n\begin{pmatrix}\sharp\\ \lambda\end{pmatrix} \to r^{\sharp_u}$
- $r^{\sharp_l}\begin{pmatrix}\sharp\\ \lambda\end{pmatrix} \to r_f$
- $r^{\sharp_u}\begin{pmatrix}\lambda\\ \sharp\end{pmatrix} \to r_f$
- $r_f\begin{pmatrix}\lambda\\ x\end{pmatrix} \to r_f$, for all $x \in \Gamma \cup Q \cup \{\sharp\}$

Using states r_{xyz} which embed a memory of the last three symbols from the lower strand, we search for a symbol $y \in Q$ and then we check that after we apply a valid transition in T, we obtain the next configuration.

- $r_{xyz}\begin{pmatrix}x\\ t\end{pmatrix} \to r_{yzx}$, for all $x, y \in \Gamma, z \in \Gamma \cup Q$, and $t \in \Gamma \cup Q \cup \{\sharp\}$
- $r_{xyz}\begin{pmatrix}qxa\\ \lambda\end{pmatrix} \to r_v$ if $x, z \in \Gamma, y \in Q$, and $\delta(y, z) = (q, a, L)$ where $q \notin F$
- $r_{xyz}\begin{pmatrix}qxa\\ \lambda\end{pmatrix} \to r_n$ if $x, z \in \Gamma, y \in Q$, and $\delta(y, z) = (q, a, L)$ where $q \in F$
- $r_{xyz}\begin{pmatrix}xqa\\ \lambda\end{pmatrix} \to r_v$ if $x, z \in \Gamma, y \in Q$, and $\delta(y, z) = (q, a, N)$ where $q \notin F$
- $r_{xyz}\begin{pmatrix}xqa\\ \lambda\end{pmatrix} \to r_n$ if $x, z \in \Gamma, y \in Q$, and $\delta(y, z) = (q, a, N)$ where $q \in F$
- $r_{xyz}\begin{pmatrix}xaq\\ \lambda\end{pmatrix} \to r_v$ if $x, z \in \Gamma, y \in Q$, and $\delta(y, z) = (q, a, R)$ where $q \notin F$
- $r_{xyz}\begin{pmatrix}xaq\\ \lambda\end{pmatrix} \to r_n$ if $x, z \in \Gamma, y \in Q$, and $\delta(y, z) = (q, a, R)$ where $q \in F$
- $r_{xy\sharp}\begin{pmatrix}xaq\\ t_1t_2\end{pmatrix} \to r_{\sharp t_1t_2}$ if $x \in \Gamma, y \in Q, t_1, t_2 \in \Gamma \cup Q$, and $\delta(y, B) = (q, a, R)$ where $q \notin F$
- $r_{xy\sharp}\begin{pmatrix}xaq\\ \lambda\end{pmatrix} \to r'_n$ if $x \in \Gamma, y \in Q$, and $\delta(y, B) = (q, a, R)$ where $q \in F$
- $r'_n\begin{pmatrix}\sharp\\ \lambda\end{pmatrix} \to r_f$
- $r'_n\begin{pmatrix}\lambda\\ x\end{pmatrix} \to r'_n$ for all $x \in \Gamma \cup Q \cup \{\sharp\}$
- $r_{xy\sharp}\begin{pmatrix}xqa\\ t_1t_2\end{pmatrix} \to r_{\sharp t_1t_2}$ if $x \in \Gamma, y \in Q, t_1, t_2 \in \Gamma \cup Q$, and $\delta(y, B) = (q, a, N)$ where $q \notin F$
- $r_{xy\sharp}\begin{pmatrix}xqa\\ \lambda\end{pmatrix} \to r'_n$ if $x \in \Gamma, y \in Q$, and $\delta(y, B) = (q, a, N)$ where $q \in F$
- $r_{xy\sharp}\begin{pmatrix}qxa\\ t_1t_2\end{pmatrix} \to r_{\sharp t_1t_2}$ if $x \in \Gamma, y \in Q, t_1, t_2 \in \Gamma \cup Q$, and $\delta(y, B) = (q, a, L)$ where $q \notin F$
- $r_{xy\sharp}\begin{pmatrix}qxa\\ \lambda\end{pmatrix} \to r'_n$ if $x \in \Gamma, y \in Q$, and $\delta(y, B) = (q, a, N)$ where $q \in F$ □

References

[1] E. Czeizler, E. Czeizler, A Short Survey on Watson–Crick Automata, Bull. EATCS 88 (2006) 104–119.
[2] E. Czezler, E. Czeizler, L. Kari, K. Salomaa, Watson–Crick automata: Determinism and state complexity, in: Proc DCFS, 2008.
[3] R. Freund, Gh. Păun, G. Rozenberg, A. Salomaa, Watson–Crick finite automata, in: Proc 3rd DIMACS Workshop on DNA Based Computers, Philadelphia, 1997, pp. 297–328.
[4] J.E. Hopcroft, J.D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley, Reading, MA, 1979.
[5] J. Hromkovic, Descriptional complexity of finite automata: Concepts and open problems, J. Autom. Lang. Comb. 7 (2002) 519–531.
[6] O.H. Ibarra, B. Ravikumar, On partially blind multihead finite automata, Theoret. Comput. Sci. 356 (2006) 190–199.
[7] H. Jürgensen, S. Konstantinidis, Codes, in: G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, vol. I, Springer, 1997, pp. 511–607.
[8] D. Kuske, P. Weigel, The Role of the Complementarity Relation in Watson–Crick Automata and Sticker Systems, DLT 2004, in: LNCS, vol. 3340, 2004, pp. 272–283.
[9] C. Martín-Vide, Gh. Păun, Normal Forms for Watson–Crick Finite Automata, in: F. Cavoto (Ed.), The Complete Linguist: A Collection of Papers in Honour of Alexis Manaster Ramer, Lincom Europa, Munich, 2000, pp. 281–296.
[10] A. Păun, M. Păun, State and transition complexity of Watson–Crick finite automata, in: G. Ciobanu, G. Păun (Eds.), Fundamentals of Computation Theory, FCT'99, in: LNCS, vol. 1684, 1999, pp. 409–420.
[11] G. Păun, G. Rozenberg, A. Salomaa, DNA Computing: New Computing Paradigms, Springer-Verlag, Berlin, 1998.
[12] E. Petre, Watson–Crick ω-Automata, J. Autom. Lang. Comb. 8 (1) (2003) 59–70.
[13] A.L. Rosenberg, On multi-head finite automata, IBM Journal of Research and Development 10 (5) (1966) 388–394.
[14] D. Röthlisberger, et al., Kemp elimination catalysts by computational enzyme design, Nature 453 (2008) 190–195.
[15] E. Shapiro, Y. Benenson, Bringing DNA computers to life, Scientific American 294 (2006) 44–51.
[16] K. Wagner, G. Wechsung, Computational Complexity, D. Reidel Publishing Company, 1986.
[17] S. Yu, State complexity of finite and infinite regular languages, Bull. EATCS 76 (2002) 142–152.