Fundamenta Informaticae XXI (2001) 1001–1017 1001
IOS Press

$5' \to 3'$ Watson-Crick Automata With Several Runs

Peter LEUPOLD*
Fachbereich Elektrotechnik/Informatik, Fachgebiet Theoretische Informatik
Universität Kassel, Germany
eMail: Peter.Leupold@web.de

Benedek NAGY†
Department of Computer Science, Faculty of Informatics
University of Debrecen
4032 Debrecen, Egyetem tér 1., Hungary
eMail: nbenedek@inf.unideb.hu

Abstract. $5' \to 3'$ WK-automata are Watson-Crick automata whose two heads start on opposite ends of the input word and always run in opposite directions. One full reading in both directions is called a run. We prove that the automatas' power increases with every run more they can make, both for deterministic and non-deterministic machines. This defines two incomparable infinite hierarchies of language classes between the regular and the context-sensitive languages. This hierarchy is complemented with classes defined by several restricted variants of $5' \to 3'$ WK-automata like stateless automata. Finally we show that several standard problems are undecidable for languages accepted by $5' \to 3'$ WK-automata in only one run, for example the emptiness and the finiteness problems.

Keywords: Automata, Formal Languages, Natural Computing, Watson-Crick Automata

Address for correspondence: Fachbereich Elektrotechnik/Informatik, Fachgebiet Theoretische Informatik, Universität Kassel, Wilhelmshher Allee 71-73, 34121 Kassel, Germany

*This work was done in part, while Peter Leupold was funded as a post-doctoral fellow by the Japanese Society for the Promotion of Science under grant number P07810, and in part, while he was funded by the Alexander von Humboldt-Stiftung as a return fellow.

†This work was done in part during a visit of Benedek Nagy in Kyoto under a project funded by the Hungarian Science and Technology Foundation (TéT) and under a visit in Kassel under a project funded by a bilateral project funded by the Hungarian Scholarship Board (Balassi Institute) and DAAD; the project is also funded by the National Office for Research and Technology (NKTH), Hungary.

1. Introduction

In recent years, much of the work in Formal Language Theory has stood in some relation to biochemistry. The origin of this were hopes of actually building biocomputers based on the theoretical models that have been developed. So far, however, it has been mainly the theoretical side that has profited from this meeting by the way of numerous new formal mechanisms and models that were inspired by processes observed in nature or in laboratories. A very rich arsenal of devices has been defined. For just a glimpse one may consult for example the book by Păun et al. [11] or an article by Dassow et al. [1] with a specific focus on Formal Language Theory. One class of this type of new devices are Watson-Crick automata, first introduced by Freund et al. [2].

The main difference to conventional automata is that Watson-Crick automata have two reading heads. The motivation for this comes from the fact that DNA-molecules consist of two parallel strands. Their contents are closely related via the so-called Watson-Crick complementarity; it says that each type of base in one strand always aligns with the same type of base in the other strand, with its complement. Now, one can imagine the two heads of the automaton as running not along the same string, but one on either strand of the DNA molecule. Though they do not read the same word, they read very closely related, namely complementary words. If the compelementarity relation is known, this does not increase the amount of information present in the input.

In reality, there exist molecules that actually move along DNA strands in the manner of reading heads, namely enzymes acting on these strands. However, DNA strands are not without direction. The bases are not symmetrical molecules but have so-called $5'$ and $3'$ ends that always bond one with the other. Equal ends never bond. Thus at one end of the strand, there is a $5'$ side, on the other end a $3'$ side of a base. Two complementary strands can only align with their opposite ends next to each other, i.e., a $5'$ and a $3'$ end on either side. Thus proteins that have a structure that enables them to move along such a double strand are likely to do so in opposite directions, if they are on opposite sides of the double strand. This suggests that the two heads of a Watson-Crick automata would be more probable to do exactly that; one can assume that these heads would be very similar if not identical molecules.

This is what inspired the introduction of $5' \to 3'$ Watson-Crick automata by Nagy [8]. There the two heads start on opposite ends of the strand. This means they actually read in the same biochemical direction, but in the opposite physical direction. As is common in abstract models inspired by biochemistry, we disregard some practical problems like how the two heads could pass each other, when they meet. Further, we do not let the automata read double strands or complementary words, but the same string, just in different directions. As Kuske and Weigel [6] observed, the complementarity does not really play any significant role in the operation of conventional Watson-Crick automata. All language classes of WK-automata are the same, even when the complementarity relation can only be the identity relation. Their argumentation will carry over to all types of automata defined here. Thus this simplification to plain strings does not really take us farther away from the motivation for these automata, but makes notations and definitions much less complicated.

The idea of letting the two heads run in opposite ways was already mentioned by Păun et. al in the book on DNA Computing [11] and also in a later article [10], where these devices were called *reverse WK-automata*. However, their power was not investigated in more detail. Finally, let us note that conventional automata with several heads have been studied before extensively, one of the earliest works is by Ibarra [5] and research on the topic is still going on. However, it seems that no model equivalent to the one defined in this article has been defined, i.e., automata where the two heads run in opposite

directions and can turn only after reading the entire word. Finally, it should be mentioned that the heads in our model do not sense each other. This means they cannot detect the step in which they are at the same position. For conventional Watson-Crick automata the power of this feature was investigated, and Nagy [9] explores it for $5' \to 3'$ WK-automata as they are used here.

After the formal definitions, the first part of this article focuses on discerning the classes of languages accepted by different versions of $5' \to 3'$ Watson-Crick automata. Namely we establish that allowing the automata to run over the input word several times increases their power with every additional run. Thus an infinite hierarchy of language classes is established. Restricted classes, for example with only one state or with only final states are put into relation to this hierarchy. A second part is dedicated to resolving questions of decidability for the language classes defined by $5' \to 3'$ Watson-Crick automata. Most importantly, we show that the emptiness problem is undecidable already for languages accepted in one run.

This work extends a contribution to the Workshop on Non-Classical Models of Automata and Applications 2009 in Wroclaw [7].

2. $5' \to 3'$ WK-automata

The reader is assumed to be familiar with basic concepts and notations of Formal Language Theory like finite automata, formal language, regular and context-free languages. Otherwise she is advised to consult any standard textbook like the ones by Harrison [4] or Salomaa [12] to resolve any kind of doubt. These books also contain the essential prerequisites for the decidability results in Section 5. We denote the empty word by λ.

We now proceed to define the central notion of this article, $5' \to 3'$ WK-automata in several variants. The intuitive description of their mode of operation has been given in the Introduction already. To contrast this variant to prior ones, they carry the term *full reading* in their name, because the new feature is that the heads do not stop when they meet, but rather move on to the respective ends of the word.

Definition 1. A $5' \to 3'$ *full reading finite Watson-Crick-automaton (WK-automaton)* $\mathbf{A}$ is a 7-tuple $(Q, V, s_0, F, \delta, \$, ¥)$ where Q is the finite set of states, V is the input (tape) alphabet, $\$$ and ¥ are the end-markers of the input string ($\$, ¥ \notin V$), $s_0 \in Q$ is the initial state, $F \subset Q$ is the set of final (accepting) states and δ is the state transition, a mapping $\big((Q \times V^* \times V^*) \to 2^Q\big) \cup \big((Q \times (\$, ¥)) \to 2^Q\big)$, such that $\delta(s, w_1, w_2)$ is non-empty only for finitely many triplets.

For describing the way in which such an automaton operates, we first define a notation for its *configurations*. They are given as strings $s\$u_1\overline{x}u_2\underline{y}u_3¥$, where s is the automaton's current state, $\$u_1xu_2yu_3¥$ is the contents of the input tape, $\overline{x}$ indicates that the reading head running to the right is over letter x, i.e. x is the next letter to be read by the head. $\underline{y}$ accordingly is the position of the head running to the left. The initial configuration for an input word xwy is $s_0\$\overline{x}w\underline{y}¥$.

A normal step of the automaton is done as follows. For a given transition such that $s' \in \delta(s, xu, vy)$ with $x, y \in V, u, v \in V^*$ to be applicable, the head running to the right must be over a letter x followed by ua, the head running to the left must be over a letter y preceded by bv. Then the first head moves over a reading xu, the second head moves over b reading vy and the new state of the automaton is s' for some $a, b \in V \cup \{\$, ¥\}$.

When both heads have reached their respective end markers, i.e., we have a configuration $s\underline{\$}w\overline{¥}$, then transitions from the set $(Q \times (\$, ¥)) \to 2^Q$ are applicable and they change the role of the two heads, i.e., the head running left will now change direction and run to the right and vice versa leading to the new configuration $s'\$\overline{x}w\underline{y}¥$. The automaton's state can also be changed during this type of transition. Every time such a transition is executed, we say that the automaton starts a new *run*.

The relation induced in this way by the transition function on the set of configurations is called the *transition relation* and is denoted by $\Rightarrow$. Its reflexive and transitive closure is $\Rightarrow^*$. An input word xwy is accepted by $\mathbf{A}$ in m runs if and only if $s_0\$\overline{x}w\underline{y}¥ \Rightarrow^* s_f\underline{\$}xwy\overline{¥}$ for a final state $s_f \in F$ after exactly m runs.

We now proceed to define the classes of languages that can be accepted by such automata with different restrictions. The restrictions defined below are the same ones that were investigated for the original WK-automata.

Definition 2. The class of languages accepted by $5' \to 3'$ full reading finite Watson-Crick automata in m runs is denoted by $\overrightarrow{=}_m\mathbf{WK}$. Such automata are called

N: *stateless* if they have only one state, i.e., $Q = F = \{s_0\}$;

F: *all-final* if they have only final states, i.e., $Q = F$;

S: *simple* if at most one head is moving in every step, i.e.,
$\delta \subset ((Q \times V^* \times \{\lambda\}) \cup (Q \times \{\lambda\} \times V^*) \cup (Q \times (\$, ¥))) \times 2^Q$;

1: *1-limited* if exactly one letter is read in every step, i.e.,
$\delta \subset ((Q \times V \times \{\lambda\}) \cup (Q \times \{\lambda\} \times V) \cup (Q \times (\$, ¥))) \times 2^Q$;

D: *deterministic* if for all possible configurations c there is at most one configuration c' such that $c \Rightarrow c'$.

The corresponding classes of languages are denoted by $U\overrightarrow{=}_k\mathbf{WK}$ where U is one of the symbols associated to the variants in the enumeration above. For denoting combinations of these variants, U can also be a sequence of such symbols in the order of the enumeration.

Notice that for 1-limited automata, determinism is most easily achieved by defining δ as a function, i.e. with just one possible result for any given combination, which is not sufficient in the case where strings are read.

Occasionally we will use the names of these language classes also as names for the corresponding classes of automata, even including the number of runs. Thus we might use formulations like "L can be accepted by an automaton from $\mathbf{FD}\overrightarrow{=}_{\mathbf{4}}\mathbf{WK}$" instead of "$L$ can be accepted by a deterministic $5' \to 3'$ full reading finite Watson-Crick-automaton with only final states in 4 runs." Further, the number of runs m can be omitted if this parameter can be arbitrary. Some of the syntactically possible language classes are not of any interest. For example, there are no interesting deterministic **N1** and **NS** machines. Since only one of the heads is allowed to move these automata cannot do anything.

For conventional Watson-Crick automata it is known that it is equivalent whether they are defined to read strings or single letters in one step. The same is true for $5' \to 3'$ WK-automata in general. However, for restricted versions, namely stateless automata, this equivalence does not hold. This is why we chose the wider definition above. Nonetheless, the next proposition can easily be proved in an analogous way as for the conventional case.

Proposition 3. For all $k > 0$ we have $\overrightarrow{\underset{k}{}}\mathbf{WK} = \mathbf{S}\overrightarrow{\underset{k}{}}\mathbf{WK} = \mathbf{1}\overrightarrow{\underset{k}{}}\mathbf{WK}$ and $\mathbf{D}\overrightarrow{\underset{k}{}}\mathbf{WK} = \mathbf{SD}\overrightarrow{\underset{k}{}}\mathbf{WK} = \mathbf{1D}\overrightarrow{\underset{k}{}}\mathbf{WK}$.

In what follows, all automata will be assumed to be 1-limited if not explicitly stated or defined otherwise. Because we treat only full reading automata and, in fact, no other version has been defined for $5' \to 3'$ WK-automata, we will in general leave away this part of their name.

3. An Infinite Hierarchy of Language Classes

After the definition of automata that can do several runs, the most obvious question is whether they are more powerful than those with only one run; that is whether the repeated runs are useful or not. It is known that one-head finite automata do not gain any power by two-way and repeated runs, they can accept only regular languages. Before answering this question for our automata, we need to analyze the ways in which the two heads can be coordinated to recognize certain non-regular structures to exceed the power of one single head.

Lemma 4. Let A be a deterministic $5' \to 3'$ WK-automaton accepting the language $L := \{ba^n ba^n b : n > 0\}$ in one run. For large enough n the computation for an input word $ba^n ba^n b$ goes through a configuration where one head is in the first factor a^n while the other is in the second.

Proof:
Without loss of generality we disregard the endmarkers \$ and ¥ that are read only at the end. In the initial configuration, both heads are over b. Let the upper one be the one that moves first, so it enters the first block of a. If the other head moves before this one reaches the central b, then the lemma holds. Otherwise, for big enough n, roughly speaking larger than the number of states, at least one state has occurred twice. Therefore there is an integer $k > 0$ such that all factors $a^{n+i\cdot k}$ for $i \geq 0$ would have lead to a corresponding configuration, i.e. the same state while the upper head is on the central b and the lower one in its initial position. Let us call this configuration α.

If the lower head still does not move until the upper one reaches the final b, then they run independently in the sense that the computation can be seen as follows: the upper head represents one finite automaton that reads the entire word, the lower head represents another finite automaton whose initial state is determined by the first automaton's computation. We obtain the intersection of two regular languages, again a regular language. This cannot consist of the infinitely many words $ba^{n+i\cdot k}ba^{n+i\cdot k}b$.

So we look at the phase in which the lower head advances till the central b. If the upper head does not move, then both will be on the central b. There will have been another repetition of states, such that infinitely many second blocks from a^+ will lead to the same state in this configuration of heads. Now one head must run to its end of the tape while the other does not move, otherwise the lemma holds. Then the other must run to its end. For this phase we can repeat the reasoning for the first phase, where also one head read an entire block of a at a time. We obtain that if a word $ba^n ba^n b$ are accepted, then there must be integers $k, m > 0$ such that for all $i, j \geq 0$ the word $ba^{n+i\cdot k}ba^{n+j\cdot m}b$ is accepted, too. This is not a subset of L which contradicts our assumption.

The remaining case is where the upper head advances while the lower head runs to the central b. Before the lower head enters the first block of a, the upper head must reach the final b, otherwise the lemma holds. So there must be a configuration with the upper head on the last b, the lower head on

the central b; we call this type of configuration β. From the configurations of type α described above, infinitely many must lead to ones of type β with the same state. This means that starting from this state infinitely many blocks of the form $a^{n+i\cdot k}$ must lead to acceptance if the second block has the same length, to rejection in all other cases. This is impossible, because the starting configuration of type β is always equivalent, no matter the length of the final factor that has already been read by both heads.

So we see that the lemma must hold if the upper head moves first. If the computation starts with the lower head moving left, symmetric considerations hold. If both heads move in the first step, then the lemma holds. □

This observation that restricts the possible positions of the two heads for languages of the given structure now allows us to distinguish classes of languages by the number of runs that $5' \to 3'$ WK-automata need to recognize them.

Theorem 5. For every $m > 0$ the class of languages accepted by $5' \to 3'$ deterministic WK-automaton in m runs is properly contained in the class of languages accepted in $m + 1$ runs, i.e., $\mathbf{D}^{\rightleftharpoons}_{m}\mathbf{WK} \subsetneq \mathbf{D}^{\rightleftharpoons}_{m+1}\mathbf{WK}$.

Proof:
The inclusion is trivial, because automata with one run more can simulate those with one run less by doing their last run without changing the decision about acceptance or non-acceptance. To separate the classes, we define the following languages with parameter m:

$$L'_m := \{a^{n_1}b^{n_2}a^{n_3}b^{n_4}\dots a^{n_{2m-1}}b^{n_{2m}} : n_i > 0 \text{ for } 1 \leq i \leq 2m\}.$$

From this we derive the languages $L_m := \{ww : w \in L'_m\}$.

Generalizing Lemma 4 we see that an automaton must run through some configuration, where the two heads are inside of factors a^{n_i} or b^{n_i} whose length must correspond for every n_i according to the language's definition. We do not elaborate details of this, because it would be very technical and at the same time quite straight-forward; thus it would not be enlightening at all.

For any given set of three such pairs of factors u, v, and w their relative position is as in $uvwuvw$. A simple exhaustive analysis of possible head positions within one run shows the following: the two heads can be in two corresponding factors at most for two such pairs within one run. This means that for accepting L_m with $2m$ corresponding pairs of factors at least m runs are necessary to be able to run through all the necessary head positions. So no automaton can accept this language in less runs.

On the other hand, it is rather straightforward to construct an automaton that accepts L_m in m runs. First the lower head advances through $m - 1$ factors of only a or only b. Then both heads match the factors a^{n_1} letter by letter. If the numbers are equal, then the upper head advances to the second block b^{n_2} while the lower head advances to the first of these two blocks. Then these are matched. In the second run the blocks number three and four are matched and so on. □

Intuitively, the different runs are rather independent. In fact, it is sufficient to transmit one pair of pieces of information: whether all previous runs have been successful (and how successful they were), and how many of the fixed number of possible runs have been done already. This observation has the following consequence.

Corollary 6. For every $m > 0$ the class of languages accepted by m run automata is not closed under intersection.

Proof:
The language L_{m+1} from the proof of Theorem 5 cannot be accepted in m runs. At the same time even the language L_{2m} can be obtained by the intersection of two languages that can be accepted in m runs. The two languages to intersect are:

$$L_{m_1} := \{ww_1ww_2 : w, w_1, w_2 \in L'_m\} \text{ and } L_{m_2} := \{w_1ww_2w : w, w_1, w_2 \in L'_m\}.$$

□

Another consequence derives from the observation that the argumentations in the proofs of Lemma 4 and Theorem 5 do not make use of the automata's determinism at any point. So the same kind of hierarchy exists for non–deterministic machines.

Corollary 7. For every $m > 0$ the class of languages accepted by $5' \to 3'$ non-deterministic WK-automaton in m runs is properly contained in the class of languages accepted in $m+1$ runs, i.e., $\overrightarrow{\overleftarrow{m}}\mathbf{WK} \subsetneq \overrightarrow{\overleftarrow{m+1}}\mathbf{WK}$.

This raises the question whether the two hierarchies are possibly the same, i.e. are deterministic and non-deterministic machines of equal power as in the case of finite automata or Turing Machines? And if they are different, can additional runs in a deterministic machine compensate the absence non-determinism? The non-deterministic class with m runs could just be between the deterministic ones with m and $m + 1$ runs. However, the answers to both questions are negative, The two hierarchies are not comparable in a smooth way.

Theorem 8. Let $\ell > m > 0$ be two natural numbers. Then the classes $m\overrightarrow{\overleftarrow{}}\mathbf{WK}$ and $\mathbf{D}^{\ell}_{\overrightarrow{\overleftarrow{}}}\mathbf{WK}$ are incomparable with respect to set-theoretic inclusion.

Proof:
The languages L_m separating the classes in the proof of Theorem 5 also separate the non-deterministic classes in Corollary 7. Therefore L_ℓ is in $\mathbf{D}^{\ell}_{\overrightarrow{\overleftarrow{}}}\mathbf{WK}$ but not in $m\overrightarrow{\overleftarrow{}}\mathbf{WK}$.

To find a language that is not in $\mathbf{D}^{\ell}_{\overrightarrow{\overleftarrow{}}}\mathbf{WK}$ but in $m\overrightarrow{\overleftarrow{}}\mathbf{WK}$ we start from looking again at L_m. A different way of writing it is

$$L_m = \{a^{n_1}b^{n_2}a^{n_3}b^{n_4}\ldots a^{n_{2m-1}}b^{n_{2m}}a^{k_1}b^{k_2}a^{k_3}b^{k_4}\ldots a^{k_{2m-1}}b^{k_{2m}} : k_i, n_i > 0 \text{ for } 1 \leq i \leq 2m \\ \text{and } n_1 = k_1 \wedge n_2 = k_2 \wedge \cdots \wedge n_{2m} = k_{2m}\}.$$

If we change the logical conjunction of terms $n_1 = k_1$ etc. into a disjunction we obtain the language

$$L_{m\vee} = \{a^{n_1}b^{n_2}a^{n_3}b^{n_4}\ldots a^{n_{2m-1}}b^{n_{2m}}a^{k_1}b^{k_2}a^{k_3}b^{k_4}\ldots a^{k_{2m-1}}b^{k_{2m}} : k_i, n_i > 0 \text{ for } 1 \leq i \leq 2m \\ \text{and } n_1 = k_1 \vee n_2 = k_2 \vee \cdots \vee n_{2m} = k_{2m}\}.$$

For a deterministic $5' \to 3'$ WK-automaton the only way to decide this language is checking all the possible matching pairs until one is found. Thus the languages $L_{m\vee}$ separate the classes according to runs

in just the same manner as do the languages L_m. On the other hand, for every $L_{m\vee}$ a nondeterministic $5' \to 3'$ WK-automaton can be constructed that accepts the language in just a single run. It works as follows: in every computation it checks just one of the possible matching pairs and the general structure of the input (correct number of blocks of a and b). On the other hand, for each such pair it has a computation checking it. If the pair matches, then the automaton accepts, otherwise it rejects the input. If the input word has a matching pair, then the corresponding computation will accept it, which means that it will be accepted by the automaton. If none of the pairs match, then there is no accepting computation, and the input is rejected. □

So the only inclusion relations that hold are the trivial ones. More runs include less runs, and non-determinism includes determinism.

It is worth noting that conventional Watson-Crick automata, whose heads run in the same direction, can accept all languages L_m and also $L_{m\vee}$ in just one run. For them, this kind of repetitive structure is relatively easy, since the heads can scan two repetitive parts in parallel. For our model, these pose a problem, while palindromic structures are very apt to be recognized. For example, the language $\{ww^R : w \in L'_m\}$ can be accepted in a single run for any m.

With the restriction to have all states final the number of runs also distinguishes different classes building a hierarchy.

Proposition 9. For all $m > 0$ the inclusion $\overrightarrow{=}_m\mathbf{WK} \subseteq \mathbf{F}\overrightarrow{=}_{m+1}\mathbf{WK}$ holds. Further, also $\mathbf{D}\overrightarrow{=}_m\mathbf{WK} \subseteq \mathbf{F1D}\overrightarrow{=}_{m+1}\mathbf{WK}$.

Proof:
The lack of distinction between final and non-final states can be compensated by the additional run in the following way: any language from $\overrightarrow{=}_m\mathbf{WK}$ or $\mathbf{D}\overrightarrow{=}_m\mathbf{WK}$ is decided by an automaton of the respective type at the end of its m-th run. This automaton can easily be modified to do another complete run if and only if the configuration after the m-th run is an accepting one. Making all the states of the modified device final, we obtain that the language belongs also to $\mathbf{F}\overrightarrow{=}_{m+1}\mathbf{WK}$, respectively $\mathbf{FD}\overrightarrow{=}_{m+1}\mathbf{WK}$. Moreover the new automaton can be 1-limited (as in Proposition 3), since the final acceptance will be at the end of the additional run. □

In the non-deterministic case, actually a stronger statement than $\overrightarrow{=}_m\mathbf{WK} \subseteq \mathbf{F}\overrightarrow{=}_{m+1}\mathbf{WK}$ can be obtained, by a modifications of the proof method used by Kuske and Weigel [6] for showing that conventional WK automata are equivalent to their **FS** versions.

Proposition 10. For all $m > 0$ the equality $\overrightarrow{=}_m\mathbf{WK} = \mathbf{FS}\overrightarrow{=}_m\mathbf{WK}$ holds.

Proof:
It is clear that the first $m-1$ run of an $\overrightarrow{=}_m\mathbf{WK}$ machine can be simulated by as in an $\mathbf{S}\overrightarrow{=}_m\mathbf{WK}$ machine by adding new states. All these states can be final without any problem, since the decision will be after the m-th run. For the final (i.e., the m-th) run we should care about the steps, because our **F** machine accepts the word if both heads finished it. The idea of the construction of the last run is that the first head will always rest on odd positions while the second head's moves ensure that it only visits even positions (except the very first and very last configuration). This can be done by an initialization (first step), lookahead and simulation, and finally by acceptance (possible last steps). For technical details we refer the reader to [6]. □

Corollary 11. The classes $\mathbf{U}^{\rightrightarrows}_{m}\mathbf{WK}$ are the same with $\mathbf{U} \in \{$ **FS**, **F**, **S**, **1**, $\lambda\}$.

Proof:
The statement is a combination of Propositions 3 with Proposition 10. □

Proposition 12. The inclusion $\mathbf{F1D}^{\rightrightarrows}_{m}\mathbf{WK} \subsetneq \mathbf{F1}^{\rightrightarrows}_{m}\mathbf{WK}$ is strict.

Proof:
For the proof of this fact consider the following languages:

$$W_m = \bigcup_{j=1}^{2m} (\{a^n b^{jn} | n \in \mathbb{N}\} \cup \{a^n b a^k | n, k \in \mathbb{N}, (j-1)n < k \leq jn\}) \text{ for } m \in \mathbb{N}.$$

A deterministic machine needs at least as many runs as half of the number of languages in the union (i.e. m) to distinguish these cases. In contrast to this fact, for non-deterministic machine one run is enough for any of these languages. Also these languages can be accepted by **F1** machines. □

By definition the chain of inclusions

$$\mathbf{F1D}^{\rightrightarrows}_{m}\mathbf{WK} \subseteq \mathbf{FSD}^{\rightrightarrows}_{m}\mathbf{WK} \subseteq \mathbf{FD}^{\rightrightarrows}_{m}\mathbf{WK} \subseteq \mathbf{D}^{\rightrightarrows}_{m}\mathbf{WK}$$

holds, and we have already seen above that $\mathbf{D}^{\rightrightarrows}_{m}\mathbf{WK} \subseteq \mathbf{F1D}^{\rightrightarrows}_{m+1}\mathbf{WK}$. At the same time, $\mathbf{F1D}^{\rightrightarrows}_{m+1}\mathbf{WK}$ is a proper superset of $\mathbf{F1D}^{\rightrightarrows}_{m}\mathbf{WK}$. It remains unclear which of the containments in this chain are strict. This is an open problem concerned to the deterministic hierarchy.

As an upper bound for the hierarchy resulting from Theorem 5 we can use finite automata with two heads that are free to move in either direction in every step. Commonly these are called two-way automata. With two heads, they can simulate any type of $5' \to 3'$ Watson-Crick automaton in a very straight-forward manner: at the beginning of a computation, let one of the two heads run to the end of the tape; from this point on let the two heads act just as the heads of the automaton to be simulated. The language class defined by two-way automata is known to be strictly included in the class of context-sensitive languages, thus all of the language classes we have defined here are context-sensitive, too.

We summarize the previous hierarchy results of this section in Figure 1, which contains the well-known class of context-sensitive languages (**CS**). Below this class we find the infinite hierarchy that results from Theorem 5. As mentioned above, we believe that the best-known language classes in this range, linear and context-free languages, are incomparable to the language classes generated by the classes of $\mathbf{D}^{\rightrightarrows}_{m}\mathbf{WK}$-automata. In the figure double arrows represent proved strict inclusions, while for the inclusions represented by single arrows their strictness is open.

In addition, the number of heads in two-way automata induces another infinite hierarchy inside the context-sensitive languages as shown by Ibarra [5]; the hierarchy induced by the number of runs of $5' \to 3'$ WK-automata lies exactly between the first and second level of that hierarchy, that is between regular languages (finite automata with one head) and those accepted by two-head two-way automata. Indeed, regular languages are properly included in a class of languages accepted by rather restricted $5' \to 3'$ Watson-Crick automata.

Theorem 13. $REG \subsetneq \mathbf{F1D}^{\rightrightarrows}_{1}\mathbf{WK}$.

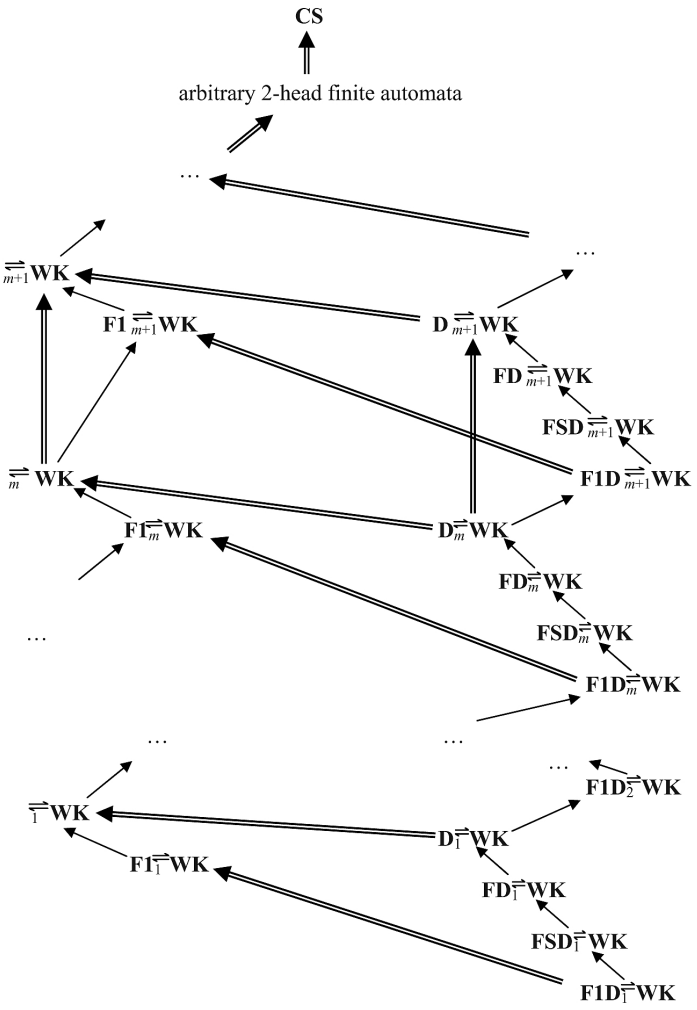

Figure 1. An infinite hierarchy induced by $5' \to 3'$ Watson-Crick automata.

Proof:
A finite automaton M can be simulated by an automaton W from $\mathbf{F1D}^{\rightleftharpoons}_{1}\mathbf{WK}$ in the following way. W's upper head, starting from the left end of the input word, simulates M's head by changing the states accordingly. The lower head will be one step behind (in numbers, not in absolute position, because it starts at the opposite end of the word) and catch up only if the current state is final. Only in such a case W will accept. So as an invariant we have that when the upper head has read i letters, the lower one has read $i-1$ letters. If the current state is final in the original automaton, then first the lower head makes a step, then the upper one makes one step and changes the state. If the state is not final, then this order is reversed. The invariant is restored in either case, and the entire input will be read by both heads iff the original finite automaton accepts.

The inclusion is strict, since it is easy to construct a $\mathbf{F1D}^{\rightleftharpoons}_{1}\mathbf{WK}$ automaton which accepts the language $\{a^n b^n | n \in \mathbb{N}\}$. The upper head counts the a one by one, while the lower head simultaneously

(strictly speaking: always one step behind) counts the b: In the initial state the first head can read only an a reaching state 2 or it can read a b reaching state 3. From state 2, the second head reads a b going the initial state. From state 3 there is a transition reading an a by the second head to state 4. From state 4 there is a transition reading a b by the first head to sate 3. The machine checks the number of a's and b's by the transitions between the initial state and state 2. The a's should be before the b's in accepted computation. When the first head reaches the first b, it can read only b's while the other head can read only a's. □

4. Restricted $5' \to 3'$ Watson-Crick Automata

Below the hierarchy depicted in Figure 1 we will now investigate inclusion relations between the language classes defined by some restricted classes of $5' \to 3'$ Watson-Crick automata. In this section, all occurrences of m as the number of runs shall have the range $m > 0$ if not specified otherwise.

At the hierarchy's very bottom we find the the stateless variants of $5' \to 3'$ Watson-Crick automata, this restriction is shown to be more strict, as the already detailed all-final one. It is easy to see that there is no hierarchy corresponding to the one from Theorem 5 for them.

Proposition 14. For all $m > 0$ we have $\mathbf{N}^{\rightleftharpoons}_{m}\mathbf{WK} = \mathbf{N}^{\rightleftharpoons}_{1}\mathbf{WK}$ and further $\mathbf{NS}^{\rightleftharpoons}_{m}\mathbf{WK} = \mathbf{NS}^{\rightleftharpoons}_{1}\mathbf{WK}$, $\mathbf{N1}^{\rightleftharpoons}_{m}\mathbf{WK} = \mathbf{N1}^{\rightleftharpoons}_{1}\mathbf{WK}$. The corresponding equalities also hold for the deterministic variants.

Proof:
All of these machines are 'forgetting' in the sense that every run starts in the same state, the only state the automaton has. As a consequence, the result of each run will be the same except for possible different paths in non-determinism in the case of non-deterministic variants. Thus a word that can be accepted in a possible m-th run can already be accepted in the first run. □

For stateless $5' \to 3'$ Watson-Crick automata, where only one head can move one step at a time, the second head does not really add anything to the computational power. So they are equivalent to their conventional one-head counterparts.

Proposition 15. The class $\mathbf{N1}^{\rightleftharpoons}\mathbf{WK}$ is equal to the class of languages accepted by stateless Finite Automata.

Proof:
It is straight-forward to construct a stateless Finite Automaton for a language V^* for any finite set V of letters. Since these automata accept any word that they can read completely, they cannot accept languages of any other form. We show that $\mathbf{N1}^{\rightleftharpoons}\mathbf{WK}$ consists only of such languages, too. Actually, a computation of an automaton from $\mathbf{N1}^{\rightleftharpoons}\mathbf{WK}$ can be divided into two phases: first, the first head since runs as far right as it can, then the second head runs as far left as it can. Because transitions depend only on the symbol under the head that moves, these two parts can be considered independently, though in time they might be interleaved. The input is accepted iff both heads reach the end. So this is the intersection of two languages of stateless Finite Automata, which obviously is again a language of a stateless Finite Automaton. □

The machines from the class $\mathbf{NS}^{=}\mathbf{WK}$ are quite similar. However, they can read two (or more) letters in one step. Thus it is easy to construct such an automaton that accepts the language $(aa)^+$. For further examples concerning stateless Finite Automata with several heads, we refer to recent work by Frisco and Ibarra [3].

Corollary 16. The inclusion $\mathbf{N1}^{=}\mathbf{WK} \subsetneq \mathbf{NS}^{=}\mathbf{WK}$ is strict.

For both of the classes $\mathbf{N1}^{=}\mathbf{WK}$ and we can give $\mathbf{NS}^{=}\mathbf{WK}$ other strict inclusions.

Proposition 17. The following inclusions are strict: $\mathbf{NS}^{=}\mathbf{WK} \subsetneq \mathbf{N}^{=}\mathbf{WK}$ and $\mathbf{N1}^{=}\mathbf{WK} \subsetneq \mathbf{ND}^{=}\mathbf{WK}$.

Proof:
The inclusion $\mathbf{NS}^{=}\mathbf{WK} \subsetneq \mathbf{N}^{=}\mathbf{WK}$ follows directly from the definition. While languages of the form V^* (i.e., $\mathbf{N1}^{=}\mathbf{WK}$ languages by the proof of Proposition 15) are trivially accepted by deterministic no-state machines. A separating language is the language of palindromes $\{w | w = w^R\}$. This language can be accepted, even by a deterministic machine, if both heads step together reading corresponding letters. Without states, however, it is impossible to coordinate the two heads in such a way that a **NS** machine can check whether corresponding letters are equal. □

So both $\mathbf{NS}^{=}\mathbf{WK}$ and $\mathbf{ND}^{=}\mathbf{WK}$ strictly include $\mathbf{N1}^{=}\mathbf{WK}$. But we do not obtain an inclusion chain between the three classes here. Rather, we find a fork in our hierarchy.

Proposition 18. The language families accepted by $\mathbf{NS}^{=}\mathbf{WK}$ and by $\mathbf{ND}^{=}\mathbf{WK}$ are incomparable.

Proof:
In one direction, as we mentioned in the proof of Proposition 17, the language of palindromes is accepted by an $\mathbf{ND}^{=}\mathbf{WK}$ reading the same letters by the heads simultaneously, while it is impossible to accept this language by an $\mathbf{NS}^{=}\mathbf{WK}$. On the other hand, the language $((aaa) + (aa) + (bb) + (bbb))^*$ is in $\mathbf{NS}^{=}\mathbf{WK}$. Each of the heads may read doubles or triplets of a's and b's. However this cannot be done by $\mathbf{ND}^{=}\mathbf{WK}$. Since it cannot be predicted how many of a's and/or b's must be read in a step in a word having at least three letters of the same type in both ends, this language is not accepted by a deterministic (stateless) machine. □

It is known from [11], that the reverse **NS** WK automata, i.e., our **NS** machines are equivalent to the classical **NS** WK automata with respect to the accepted languages. Moreover, this class is strictly included in the class of regular languages.

Proposition 19. The following inclusion is strict: $\mathbf{ND}^{=}\mathbf{WK} \subsetneq \mathbf{N}^{=}\mathbf{WK}$.

Proof:
The inclusion follows directly from the definition. On the other hand, the two language classes are separated by the language $((aaa) + (aa) + (bb) + (bbb))^*$ used in the proof of Proposition 18 above. One can easily construct a stateless machine accepting the language (even an **NS** machine). But this language cannot be accepted by deterministic stateless machine. □

Proposition 20. The classes $\mathbf{N}^{=}\mathbf{WK}$ and $\mathbf{F1}_1^{=}\mathbf{WK}$ are incomparable. Also $\mathbf{N}^{=}\mathbf{WK}$ and $\mathbf{F1D}_1^{=}\mathbf{WK}$ are incomparable.

Proof:
Let us consider the **N** machine with the following three transitions:

- while the first head reads an a, the second reads aa, or
- aa is read by the first head while a is read by the second head, or
- b is read by each of the heads.

The above language is not accepted by any 1 run **F1** machine: An **F1** machine accepts the word if both heads made the same number of steps. Let us consider words of the language of the form $a^n b a^m$. Both $a^n b a^{2n}$ and $a^{2n} b a^n$ should be accepted, which means that the difference of the number of steps of the heads can be arbitrarily high (even the first or the second head makes more steps than the other). So their 'distance' cannot be handled by finitely many states.
The language $\{a^n b^n\}$ cannot be accepted by any stateless machine. The order of a's and b's cannot be checked by any **N** machines. Opposite to this, as we presented in the proof of Theorem 13 it is accepted by $\mathbf{F1D}_1^{\rightleftharpoons}\mathbf{WK}$ machine. □

Proposition 21. The strict inclusions $\mathbf{ND}_1^{\rightleftharpoons}\mathbf{WK} \subsetneq \mathbf{FD}_1^{\rightleftharpoons}\mathbf{WK}$ and $\mathbf{N}_1^{\rightleftharpoons}\mathbf{WK} \subsetneq {}_1^{\rightleftharpoons}\mathbf{WK}$ hold.

Proof:
The inclusions follow directly from the definitions. As we already showed in the proof of Theorem 13 and Proposition 20, the language $\{a^n b^n\}$ is in $\mathbf{FD}_1^{\rightleftharpoons}\mathbf{WK}$ (and so it is in ${}_1^{\rightleftharpoons}\mathbf{WK}$), but not in $\mathbf{N}_1^{\rightleftharpoons}\mathbf{WK}$. Consequently it is not in $\mathbf{ND}_1^{\rightleftharpoons}\mathbf{WK}$ either. □

We summarize this lower part of the hierarchy in Figure 2, which contains the well-known class of regular languages (**Reg**) as a reference point. A further reference are the top-most classes which

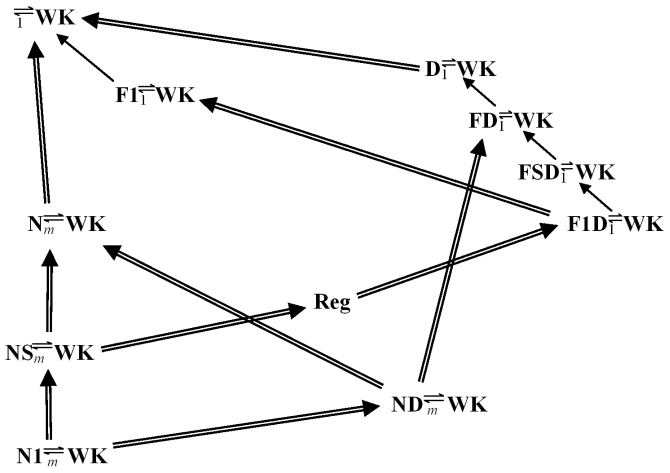

Figure 2. The lower part of the hierarchy induced by $5' \to 3'$ Watson-Crick automata.

constitute the first level of the hierarchy from Figure 1. As mentioned above, we believe that other

well-known language classes in this range, linear and context-free languages, are incomparable to the language classes generated by the classes of $\mathbf{D}_m^{\rightleftharpoons}\mathbf{WK}$-automata. Also here, double arrows represent proved strict inclusions, while for the inclusions represented by single arrows their strictness is open.

5. Decidability

The fact that $5' \to 3'$ Watson-Crick automata accept non-context-free languages even in just one run suggests that most decidability problems related to them will not be solvable. However, since they do not accept all context-free languages, but language classes somewhat orthogonal to the Chomsky Hierarchy, this kind of result cannot be derived directly from well-known undecidability results. Investigating such decidability questions will be the objective of this section.

First we take a look at the non-emptiness problem. This is the question whether the language accepted by a given automaton is empty or not. It is still decidable for context-free languages, but not for 1 run automata, which can be shown using standard techniques. In preceding work [7] this was shown for general deterministic $5' \to 3'$ Watson-Crick automata, and the question was asked how far this undecidability can be lowered towards REG along the inclusion chain $REG \subsetneq \mathbf{F1D}_1^{\rightleftharpoons}\mathbf{WK} \subseteq \mathbf{FSD}_1^{\rightleftharpoons}\mathbf{WK} \subseteq \mathbf{FD}_1^{\rightleftharpoons}\mathbf{WK} \subseteq \mathbf{D}_1^{\rightleftharpoons}\mathbf{WK}$. With just a slight modification of the original proof we can actually establish undecidability all the way down to $\mathbf{F1D}_1^{\rightleftharpoons}\mathbf{WK}$. Since emptiness is decidable for regular languages, this determines exactly the border between decidability and undecidability in this chain.

Theorem 22. For the class $\mathbf{F1D}_1^{\rightleftharpoons}\mathbf{WK}$ the non-emptiness problem is undecidable.

Proof:
For a Turing Machine M we define the corresponding language L_M that contains all of the words $\$w_1\#w_2 \ldots \#w_{i-1}\#\#w_i^R\#w_{i-1}^R \ldots \#w_3^R\#w_2^R$ where $w_1, w_2, \ldots, w_i$ is a sequence of configurations of a computation of M, w_1 is an initial configuration, and w_i is a final and accepting configuration. Configurations are mere strings consisting of the current tape contents with a symbol for the current state inserted in front of the current position of the Turing Machine's head.

As a consequence of its definition, L_M contains one word for every accepting computation of M, and consequently L_M is empty if and only if M does not accept any word. Thus by deciding the non-emptiness problem for the class of languages L_M one could decide the same problem for Turing Machines in general, but this is known to be impossible [12]. What remains to be shown is that the L_M are languages accepted by $5' \to 3'$ WK automata in one run.

Our $5' \to 3'$ WK automaton A_M for a given L_M will work as follows. While the first head reads w_1, the second head simultaneously reads w_2^R. They check whether the strings are configurations and, more importantly, whether w_2 can be reached from w_1 in one step by M. Since one step of a Turing Machine effects only a very local change, rewriting one symbol and moving the head one position, this can be done by our two reading heads and their finite state control. Since w_2^R is read by the second head from right to left, while w_1 is read by the first head from left to right, the two strings are read in the same direction with respect to the Turing Machine representation. When w_1, and w_2^R have been read, i.e., both heads reach the first $\#$ symbol, the procedure is repeated for w_2 and w_3^R, and so on.

Thus after reaching the central $\#\#$ the automaton has checked whether the change from w_j to w_{j+1} corresponds to a step of the Turing Machine M for every $j < i$. Further it knows whether the final configuration is a halting and accepting one; only in this case the automaton will continue, otherwise the

two heads remain over the central $\#\#$, which means that the input will not be accepted. What remains to show is that the two copies of each w_j are really the same. This will be done in the remaining part of the computation, in which the heads run from the central $\#\#$ to the opposite ends of the tape. The head running to the right skips over the first configuration w_i^R, then corresponding configurations are matched letter by letter. In the end, the head running to the left skips over w_1.

If everything has been checked successfully, then and only then the input string is to be accepted by A_M. Since all of its states are final, the only way not to accept a word is to not read it completely. For the checking everything except the initial \$ needs to be read by both heads, the one moving right is already over the last symbol. So the decision over acceptance can be controlled by either reading the initial \$ or not. Further it is clear that the whole computation can be done by moving only one head only one symbol in each computation step. Thus an A_M from $\mathbf{F1D}_1^{\rightleftharpoons}\mathbf{WK}$ can be employed.

Since every accepted input word corresponds to an accepting computation of M, the existence of such a word implies that the language accepted by M is not empty either. □

Closely related to the non-emptiness problem is the finiteness problem, the question whether the language accepted by a given automaton is finite or not. Also this one is undecidable in our case as we can see from a modification of the proof of Theorem 22.

Corollary 23. For the class $\mathbf{F1D}_1^{\rightleftharpoons}\mathbf{WK}$ the finiteness problem is undecidable.

Proof:
A deterministic Turing Machine accepts every accepted word w in exactly one computation, the one resulting deterministically from starting with the initial state on the left hand side of w. So if the Turing Machine M in the proof of Theorem 22 is deterministic, then there is a one-to-one correspondence between words accepted by M and by the constructed automaton. This means that the answer to the finiteness problem is the same for both devices. For Turing Machines it is known that also this problem is undecidable. □

Now let us consider other decidability problems, such as equivalence of machines, i.e., the equality of the accepted languages. And also a related question the inclusion problem of the accepted languages. It is known that that both the inclusion and the equality is decidable for any two regular languages (thought there is not known any efficient way to do, i.e., it is PSPACE-complete if the languages are given by, for instance, regular expressions.

Corollary 24. For the class $\mathbf{F1D}_1^{\rightleftharpoons}\mathbf{WK}$ the equivalence problem is undecidable.

Proof:
Let us consider an $\mathbf{F1D}_1^{\rightleftharpoons}\mathbf{WK}$ machine A_1 that cannot move the second head. In this way there is no word that can be read by the second head, consequently it is clear that this machine accepts the empty language. Now let us consider another $\mathbf{F1D}_1^{\rightleftharpoons}\mathbf{WK}$ machine A_2 that checks a Turing computation as it is used in the proof of Theorem 22. Since by the mentioned theorem it is undecidable whether this language is empty, i.e., the same as the language of A_1, the equivalence of the two machines A_1 and A_2 is also undecidable. □

The undecidability of the equivalence problem for a class of languages always implies undecidability of the inclusion problem. We will sketch the argumentation which does not need any reference to the fact that we are talking about the class $\mathbf{F1D}^{\rightleftharpoons}_{1}\mathbf{WK}$. Assume that the inclusion problem is decidable for a class $\mathcal{C}$. Then one can decide whether $L_1 \subset L_2$ and whether $L_2 \subset L_1$ for two languages $L_1, L_2 \in \mathcal{C}$. The languages are equal, if both inclusions hold.

Corollary 25. For the class $\mathbf{F1D}^{\rightleftharpoons}_{1}\mathbf{WK}$ the inclusion problem is undecidable.

Since all the problems treated in this section have turned out to be undecidable, we do want recall that the word problem is decidable for all classes of $5' \to 3'$ Watson-Crick automata treated in this article. This is clear because all corresponding language classes are contained in the context-sensitive languages, and the word problem is decidable for the latter.

From the construction used in the proof of Theorem 22 we can derive another result. Intuitively, the automata there simulate any given Turing Machine and thus generate any recursively enumerable language with some noise added. This noise can be removed by a morphism, and thus we obtain the following result.

Corollary 26. Every recursively enumerable language is a morphic image of a language from the class $\mathbf{F1D}^{\rightleftharpoons}_{1}\mathbf{WK}$.

Proof:
In the proof of Theorem 22, we can write the configuration w_1 with a marked version of the original alphabet. The automaton ignores the marks and treats corresponding letters as equal. Then we apply a morphism that deletes all letters that are not from this marked alphabet to the resulting language. The marked letters, however are mapped to their unmarked versions. Thus application of this morphism leaves only w_1 without the initial state, i.e., exactly the word that is accepted by the Turing Machine M. Therefore the resulting language is exactly the one of the Turing Machine M. □

6. Conclusion

We have investigated Watson-Crick automata whose heads run in opposite directions. Looking at the biochemical reality, this seems a well-motivated modification of the original definition. It is quite intuitive that this variant is very apt to recognize palindromic structures, because its two heads can read corresponding parts of the input at the same time. On the other hand, copy structures of the form uu pose a problem. This is the inverse situation compared to the original Watson-Crick automata, and thus classes with equal restrictions are usually incomparable.

We have established a great number of inclusion relations between different variants of $5' \to 3'$ Watson-Crick Automata. However, every non-double arrow in Figures 1 and 2 represents a question left to be resolved.

As a second open problem, we mention that we believe that $\mathbf{F1D}^{\rightleftharpoons}_{1}\mathbf{WK}$ is incomparable to the class of linear languages. Several examples of non-linear languages accepted by such automata have been presented above. On the other hand, however, we have not been able to give an example of a linear language that cannot be accepted by this class of automata. For automata that can sense the meeting of their two heads [9], every linear language can be accepted in a straight-forward way. Without this ability, however, this might not be possible for all linear languages.

References

[1] Dassow, J., Mitrana, V., Salomaa, A.: Operations and language generating devices suggested by the genome evolution, *Theoretical Computer Science*, **270**(1-2), 2002, 701–738.

[2] Freund, R., Păun, G., Rozenberg, G., Salomaa, A.: Watson-Crick Finite Automata, *Proceedings of the Third Annual DIMACS Symposium on DNA Based Computers*, Philadelphia, 1994.

[3] Frisco, P., Ibarra, O. H.: On Stateless Multihead Finite Automata and Multihead Pushdown Automata, *Developments in Language Theory* (V. Diekert, D. Nowotka, Eds.), 5583, Springer, 2009, ISBN 978-3-642-02736-9.

[4] Harrison, M. A.: *Introduction to Formal Language Theory*, Addison-Wesley, Reading, Massachusetts, 1978.

[5] Ibarra, O. H.: On Two-way Multihead Automata, *Journal of Computer and System Sciences*, **7**(1), 1973, 28–36.

[6] Kuske, D., Weigel, P.: The Role of the Complementarity Relation in Watson-Crick Automata and Sticker Systems, *Developments in Language Theory* (C. Calude, E. Calude, M. J. Dinneen, Eds.), 3340, Springer, 2004, ISBN 3-540-24014-4.

[7] Leupold, P., Nagy, B.: $5' \to 3'$ Watson-Crick Automata with Several Runs, *Non-Classical Models of Automata and Applications (NCMA)* (H. Bordihn, R. Freund, M. Holzer, M. Kutrib, F. Otto, Eds.), 256, Wien, 2009.

[8] Nagy, B.: On $5' \to 3'$ Sensing Watson-Crick Finite Automata, *DNA* (M. H. Garzon, H. Yan, Eds.), 4848, Springer, 2007, ISBN 978-3-540-77961-2.

[9] Nagy, B.: On a Hierarchy of $5' \to 3'$ Sensing WK Finite Automata Languages, *Mathematical Theory and Computational Practice, CiE, Abstract Booklet*, University of Heidelberg, 2009.

[10] Păun, G.: DNA Computing by Matching: Sticker Systems and Watson-Crick Automata, *Pattern Formation in Biology, Vision and Dynamics* (A. Carbone, M. Gromov, P. Prusinkiewicz, Eds.), World Scientific, Singapore, 2000.

[11] Păun, G., Rozenberg, G., Salomaa, A.: *DNA Computing – New Computing Paradigms*, Springer-Verlag, Berlin Heidelberg, 1998.

[12] Salomaa, A.: *Formal Languages*, Academic Press, New York, 1973.