Some Remarks on the Formal Operations Inspired by the Gene Assembly in Ciliates

Victor Mitrana
Universidad Politecnica de Madrid and National Institute for R&D for Biological Sciences
Department of Information Systems and Bioinformatics Department
Calle Alan Turing s/n, 28031, Madrid, Spain
victor.mitrana@upm.es

Andrei Păun, Mihaela Păun
National Institute for R&D for Biological Sciences
Bioinformatics Department
296 Independenței Bd., Bucharest 060031, Romania
{apaun@gmail.com, mihaela.paun@incdsb.ro}

Jose-Ramón Sánchez-Couso
Universidad Politecnica de Madrid
Department of Information Systems
Calle Alan Turing s/n, 28031, Madrid, Spain
joseramon.sanchezcouso@upm.es

Abstract. *We continue here the theoretical study initiated approximately twenty years ago on the possibility of using living cells for computing. In this paper, we reconsider the formal operations inspired by the intramolecular DNA rearrangements in the evolution of the macronucleus from the micronucleus in a group of ciliates. After introducing the concept of a valid string, we propose an efficient algorithm for checking this property for a given string. Then we investigate which of the considered operations preserve the property of a string to be valid. We also show that just one of the operations can be simulated by a finite transducer. The important problem regarding the order of applying the operations is then investigated showing that one operation can commute with the other two. Finally, we introduce the iterated variants and investigate a few properties. A sort of a "normal form" for the gene assembly in ciliates is obtained. The paper ends by a short discussion about open problems and further directions of research.*

Keywords. Ciliate; gene assembly, valid string, intramolecular operations, finite transducer.

1 Introduction

Do the cells "compute"? This seems to be one of the main questions in the theory of computing, especially nowadays when more and more complex computational problems arise in different areas. Some directions of research are aimed to find exact algorithms using current hardware architectures, while other approaches enable approximate, uncertain, tolerant solutions which are mainly based on other architectures than the classic ones. Many directions of research (Machine learning, Fuzzy sets, Evolutionary computing, Artificial neural networks, Expert systems, etc.) have been emerged and vividly developed under the umbrella of *Soft Computing*. Other approaches have eventually formed the *Unconventional Computing* paradigm that includes: *Natural Computing (DNA and cellular (in vivo) computing, biochemical networks, etc.) and Quantum Computing*, see (Kari & Rozenberg, 2008). This work is a contribution to the area of the so-called "in-vivo" computing or the computation with living cells. If the cells really compute, are we able to understand the way in which they do it and which are the problems "solved" by them? It is commonly accepted that, most likely, they do this by reading and modifying DNA sequences all the time. Among many computational models based on the manipulation of DNA, see, eg., (Păun, et al., 1998) and (Rozenberg et al., 2012), there is a model of a rather different type that is based on the way in which some microorganisms, calls *ciliates*, are copying some protein-coding genes from their micronucleus to their macronucleus (Ehrenfeucht et al., 2004). Ciliates are unicellular eukaryotes having two types of nuclei: a macronucleus and a micronucleus (germline). The macronucleus, which is somatically active, is formed from the inactive micronucleus after sexual reproduction. It seems that ciliates have "invented" very ingenious solutions to complex problems million years ago.

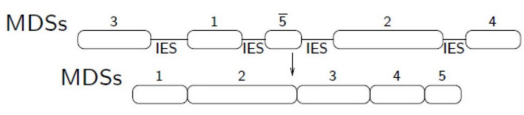

Figure 1: The arrangement of genes in the micronucleus (above) and their arrangement in the macronucleus (below).

A functional copy of the genes in the macronucleus is obtained by eliminating some DNA sequences, and/or changing some gene order, and/or reverting some inverted genes in the micronucleus. We illustrate this situation in Figure 1, where $\bar{5}$ means that the fifth gene is inverted. In this note, we consider the formal variants of three intramolecular operations which were proposed in (Ehrenfeucht et al., 2004) as formal operations for the unscrambled gene assembly in ciliates, more precisely in a group of ciliates called *Stichotrichs*. These operations called *ld, hi, dlad*, formally define the DNA recombination based on folds that are aligned by pointers, some relatively short nucleotide sequences at the intersection of MDSs (macronuclear destined sequences) and noncoding IESs (internally eliminated sequences). It is worth mentioning that the operations considered here are intramolecular operations; however, intermolecular operations have also been defined and investigated, see, e.g., (Kari & Rozenberg, 2008), (Brijder et al., 2012).

Informally speaking, we may say that what ciliates do in gene assembly is excision, inversion and interchange (sorting) of DNA sequences. Therefore, a hypothetical computation with ciliates would mean to encode the given instance of a problem in an abstract micronuclear gene, leaves the ciliate to assemble it into a macronucleus, and then filter and sequence the result. Initially, the formal computational power of gene assembly in ciliates was investigated in comparison with that of Turing machines: (Landweber & Kari, 1998), (Onolt-Ishdorj et al, 2007), yielding to computationally complete models. Later on, there were reported conceptual solutions to some hard problems, like the Hamiltonian Path Problem (Alhazov et al, 2008). Such a solution, if ever implemented in ciliates, would have the advantage that the organisms themselves implement generally time-consuming steps of the procedure.

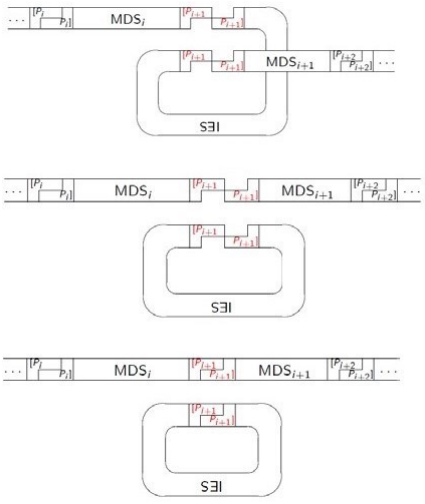

Figure 2: The ld operation.

We now explain the three operations. The operation *ld* excises one circular molecule, which contains noncoding blocks only. In this way, two consecutive MDSs are aggregated into a bigger composite MDS. More precisely, assume that the MDS_{i+1} follows MDS_i, the two being separated by an IES, and the pointer P_{i+1} flanks the IES. Then *ld* makes a fold aligned by pointer P_{i+1}, excises the IES as a circular molecule, and recombines MDS_i and MDS_{i+1} into a longer coding block, as in Figure 2.

In the case of hi, the DNA sequence makes a fold aligned by pointer P_{i+1}, reverts MDS_{i+1}, and rearrange the two MDSs in the correct order, as illustrated in Figure 3.

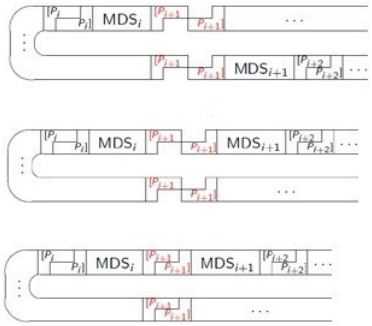

Figure 3: The hi operation.

Finally, in Figure 4, the $dlad$ is illustrated. The molecule is double folded aligned by the pointers P_{i+1} and P_{i+2}, and then a new molecule is obtained by the recombination sequence shown in Figure. 4.

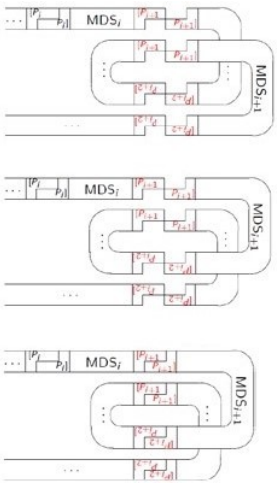

Figure 4: The $dlad$ operation.

We consider here three formal operations on words and language inspired by the operations shown in the three figures above. We first define the *gene assembly in ciliates* (gac)-scheme, the valid string with respect to a gac-scheme, and discuss the algorithm for deciding whether or not a given string is valid with respect to a given gac-scheme. We show that this algorithm is essentially based on the Aho-Corasick algorithm which can be implemented by means of a finite automaton

such that the language of all valid strings with respect to a gac-scheme is regular. Then we consider a few properties of the non-iterated versions of these operations. Thus, the ld operation can be simulated by a finite transducer while this is not possible for the other two operations. All the three operations preserves the property of a string of being valid. A natural and biologically motivated question is whether or not the order of applying these operations does matter. We solve completely the problem by showing that the order does not really matter.

The iterated variants of the three operations are defined and a few properties are investigated. A sort of a "normal form" concerning the order of applying these operations is given. Finally, a few open problems are discussed.

2 Preliminaries

We introduce here the main concepts and definitions which will be used in the rest of the paper. For undefined notions, the reader is referred to (Rozenberg et al., 2012).

The cardinality of a finite set A is denoted by $card(A)$. An alphabet V is a finite set of symbols/letters, a string over V is a finite sequence of symbols from V, the empty string is denoted by λ, the set of all strings over V is denoted V^*, while the set of all nonempty strings over V is denoted by V^+. Further on, the length of a string x is denoted by $|x|$. For a string $w \in V^*$, $Sub(w) = \{x \in V^+ \mid w = uxv, u, v \in V^*\}$ is the set of all substrings of w. If u or v is not empty in this definition, then x is a proper substring of w. Furthermore, w^R denotes the mirror image of the string w, that is $w^R = a_n a_{n-1} \dots a_1$, provided that $w = a_1 a_2 \dots a_n$, where each a_i, $1 \leq i \leq n$, is a symbol. For a set of strings (language) L we set $L^R = \{w^R \mid w \in L\}$.

A *finite transducer* is a 6-tuple $M = (Q, V, U, \delta, q_0, F)$, where Q, V, U are finite and non-empty sets called states set,input alphabet and output alphabet, respectively, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and δ is the transition-and-output function from $Q \times (V \cup \{\lambda\})$ to finite subsets of $Q \times U^*$. This function may be extended in a natural way to $Q \times V^*$. Each finite transducer M as above defines a *finite transduction*

$$g_M(x) = \{y \in U^* | (q, y) \in \delta(q_0, x), q \in F\}.$$

The finite transduction g_M can be extended to a language $L \subseteq V^*$ as follows

$$g_M(L) = \bigcup_{x \in L} g_M(x).$$

If M reads exactly one symbol at each transition and U is ignored (i.e. M has no output), we have a *finite automaton*. Moreover, M is called deterministic, if δ is a function from $Q \times V$ into $Q \times U^*$.

A gac-scheme (gene assembly in ciliates) is a pair $\sigma = (V, P)$, where V is an alphabet and P is a finite subset of V^+, whose elements are called pointers, such that $P^R = P$.

We now introduce the three formal operations on strings suggested by the gene assembly in ciliates illustrated in Figures 2, 3, and 4.

Given a gac-scheme $\sigma = (V, P)$ and a string $w \in V^+$, we formally define the ld operation as follows.

$$\begin{aligned} ld_\sigma(w) &= \{x\alpha z \mid w = x\alpha y\alpha z,\ x, z \in V^*, \\ &\quad y \in V^+,\ \alpha \in P, Sub(y) \cap P = \emptyset\}. \end{aligned}$$

We consider here that the circular molecule obtained in Figure 2 is no longer the micronuclear precursor for another assembly, so we keep only the linear molecule. Note that, as mentioned before, the segment y must not contain any pointer.

Given a gac-scheme $\sigma = (V, P)$ and a string $w \in V^+$, we formally define the hi operation as follows.

$$\begin{aligned} hi_\sigma(w) &= \{x\alpha y^R\alpha^R z \mid w = x\alpha y\alpha^R z, x, z \in V^*, \\ &\quad y \in V^+, \alpha \in P\}. \end{aligned}$$

The $dlad$ operation is applicable to those strings which have an alternating direct repeat pair of pointers. This string can be folded into two loops, each of which is aligned by one pair of pointers, as illustrated in Figure 4. Formally, given a gac-scheme $\sigma = (V, P)$ we define

$$\begin{aligned} dlad_\sigma(w) &= \{u\alpha y\beta x\alpha v\beta z \mid w = u\alpha v\beta x\alpha y\beta z, \\ &\quad v, x, y \in V^+, u, z \in V^*, \alpha, \beta \in P\}. \end{aligned}$$

It is worth noting that simpler variants have been considered in (Harju et al., 2006), too. This means the followings: in the case of hi and dld each fold, and consequently each rearranged sequence, contains just one MDS. It is obvious that there are sequences such that no simple operation is applicable while the variants considered here can be applied. For instance, take the string

$$w = x_0\underline{32}x_1\underline{10}x_2\underline{67}x_3\underline{89}x_4\underline{45}x_5\underline{67}x_6\underline{23}x_7\underline{45}x_8,$$

where all the pointers are underlined, and none of the substrings x_i, $0 \leq i \leq 8$ contains any digit. Clearly, w is a valid string but no simple operation is applicable to this string. On the other hand, both hi and dld can be applied. Indeed,

$$x_0\underline{32}x_6^R\underline{76}x_5^R\underline{54}x_4^R\underline{98}x_3^R\underline{76}x_2^R\underline{01}x_1^R\underline{23}x_7\underline{45}x_8 \in hi(w),$$

and

$$x_0\underline{32}x_1\underline{10}x_2\underline{67}x_6\underline{23}x_7\underline{45}x_5\underline{67}x_3\underline{89}x_4\underline{45}x_8 \in dld(w).$$

3 Results

Given a *gac*-scheme $\sigma = (V, P)$, we say that a string $w \in V^*$ is *valid* with respect to σ, if w admits a unique decomposition as follows:

$$w = w_0\alpha_1 w_1 \alpha_2 \dots w_{k-1}\alpha_k w_k,$$

with $k \geq 0, \alpha_i \in P, 1 \leq i \leq k$, and $Sub(w_j) \cap P = \emptyset$, for all $0 \leq j \leq k$.

This definition appears also in (Vaszil, 2020).

Theorem 1. *Given a gac-scheme $\sigma = (V, P)$, one can algorithmically decide whether or not a given string $w \in V^*$ is valid in $\mathcal{O}(n + m)$, where $n = |w|$ and $m = \sum_{\alpha \in P} |\alpha|$.*

Proof. The algorithm is essentially the algorithm proposed by Aho and Corasick in (Aho & Corasick, 1975) for finding all occurrences of a finite set of strings in a given string. We actually use the algorithm for finding all occurrences of the pointers from P in w. The following conditions must be satisfied in order to validate w:

- At any position i in w, there is at most one pointer that can appear. In other words, there are no two pointers that appear at the same position in w.
- If a pointer α appears at the position i in w, and j is the next position in w, where a pointer appears, $j > i + |\alpha| - 1$.

Note that the complexity of the algorithm is the same to that of Aho-Corasick without the number of the output matches. □

As the Aho-Corasick is based on the construction of a special type of a finite automaton, we may also state:

Corollary 1. *Given a gac-scheme $\sigma = (V, P)$, the language of all valid strings over V with respect to σ is regular.*

3.1 Properties of the Non-iterated Versions

Theorem 2. *Let $\sigma = (V, P)$ be a gac-scheme.*
1. There is a finite transducer M such that $ld_\sigma(w) = g_M(w)$ for every $w \in V^*$.
2. There is no finite transducer M such that $hi_\sigma(w) = g_M(w)$ for every $w \in V^*$.
3. There is no finite transducer M such that $dlad_\sigma(w) = g_M(w)$ for every $w \in V^*$.

Proof. 1. Instead of giving the formal construction of the finite transducer M, we prefer to explain how it works on the input string w. Its computations have four phases:

- *Phase 1.* The transducer scans a prefix of w and writes each symbol read. In a nondeterministic way, it passes to Phase 2.
- *Phase 2.* Now, M checks whether a pointer follows in w and memorize it. If there are more pointers that appears in w at the same position (note that w is not necessarily valid), M chooses one nondeterministically. The pointer is printed out. Then it passes to Phase 3.
- *Phase 3.* Now M checks the presence of the second occurrence of the memorized pointer. If the pointer does not appear anymore, the computation is blocked. If the pointer appears, it checks also the presence of another pointer that does not overlap with the occurrences of the memorized symbol. If this is the case, the computation is blocked again. During this Phase, no symbol is written to output. If the checking process has been successful, the transducer passes to Phase 4, the last one.
- *Phase 4.* M writes the rest of the input string (after the second occurrence of the memorized pointer) to the output.

By these explanations, it immediately follows that $ld_\sigma(w) = g_M(w)$.

2. Assume by contradiction that there exists a finite transducer M such that $hi_\sigma(w) = g_M(w)$ for every $w \in V^*$. Let us consider the linear language $L = \{wcw^Rc \mid w \in \{a, b\}^+\}$ and the *gac*-scheme $\sigma = (V, \{c\})$. By our assumption, $hi_\sigma(L) = \bigcup_{w \in L} hi_\sigma(w) = \{wcw \mid w \in \{a, b\}^+\}$. This implies that $g_M(L) = \{wcw \mid w \in \{a, b\}^+\}$. But this is not possible because the language $\{wcw \mid w \in \{a, b\}^+\}$ is not context-free and the class of context-free languages is closed under finite transducer mappings.

3. A similar argument may be used for the third statement. We consider the context-free language $L = \{a^n cb^n cbcb^m ca^m \mid n, m \geq 1\}$ and the *gac*-scheme $\sigma = (\{a, b, c\}, \{c\})$. Thus, we get $dlad_\sigma(L) = \{a^n cb^m cb^{n+1} a^m \mid n, m \geq 1\}$, which is not context-free anymore. □

As an immediate consequence, it follows that the class of regular languages is closed under the operation ld, while the class of context-free languages is closed under none of the other two operations. Note that a similar result is also proved in (Freund et al., 2002).

One important question is: Which of the three operations preserves the validity property? The next statement answer this question.

Theorem 3. *Let $\sigma = (V, P)$ be a gac-scheme. If w is valid with respect to σ, then each of the sets $ld_\sigma(w)$, $hi_\sigma(w)$, and $dlad_\sigma(w)$ contains valid strings only.*

Proof. If w is a valid string with $w = x\alpha y\alpha z$, and $Sub(y) \cap P = \emptyset$, then the string u produced by ld, that is $u = x\alpha z$, is also valid. Indeed, if two pointers were overlapping in the prefix $x\alpha$ of u, they would overlap in the same prefix of w, which is not possible. The situation is similar if two pointers were overlapping in

the suffix αz of u, hence u is valid with respect to σ. It follows that ld preserves the validity property.

We now assume that $w = x\alpha y\alpha^R z$ and prove that $u = x\alpha y^R\alpha^R z$ is also valid. Assume the contrary; we distinguish three cases which are to be analyzed in the sequel. Let us assume that $\alpha = \gamma\delta$ and $y = y_1y_2$, such that δy_2^R is a pointer. By the definition of P, $y_2\delta$ must be also a pointer. But this is not possible as w is valid.

Now, let us assume that $\alpha = \gamma\delta$ and $y = y_1y_2$, such that $y_1^R\delta^R$ is a pointer. This implies that δy_1 is a pointer and this contradicts again the validity of w.

Finally, let $\alpha = \gamma\delta$ and $y = y_1y_2$, such that $\delta y^R\delta^R$ is a pointer. It follows that $\delta y\delta^R$ is also a pointer which is also a contradiction. Consequently, the hi operation preserves the validity property.

We now assume that $w = u\alpha v\beta x\alpha y\beta z$ and prove that $u\alpha y\beta x\alpha v\beta z$ is also valid. This immediately follows as soon as we notice that the underlined segments $\underline{u\alpha v\beta x}\underline{\alpha y\beta} z$ in w are interchanged by the $dlad$ operations. □

As we have seen above, during the process of obtaining its macronucleus, the ciliate removes some DNA sequences, and/or change some gene order, and/or reverts some inverted genes in the micronucleus. A natural question is whether or not the order of this evolutionary steps is important. More precisely, is it possible to obtain the same sequences if the order of applying the three operations is changed? This is solved by the following statement.

Theorem 4. *Let $\sigma = (V, P)$ be a gac-scheme, and x be a valid string in V^+. Then the followings hold:*
1. $ld_\sigma(hi_\sigma(x)) = hi_\sigma(ld_\sigma(x))$.
2. $ld_\sigma(dld_\sigma(x)) = dld_\sigma(ld_\sigma(x))$.
3. There are valid strings y such that $dld_\sigma(hi_\sigma(y)) \neq hi_\sigma(dld_\sigma(y))$.

Proof. 1. Let $z \in hi_\sigma(ld_\sigma(x))$; this means that there exists $y \in ld_\sigma(x)$ such that $z \in hi_\sigma(y)$. It follows that $y = x_0\alpha x_2$, provided that $x = x_0\alpha x_1\alpha x_2$, with $\alpha \in P$ and $Sub(x_1) \cap P = \emptyset$. We distinguish three cases depending on the substring of y that is modified by the hi operation:

Case (i). The modified substring is $x_0\alpha$. Two subcases must be considered: (i1) $x_0 = u\beta v\beta^R q$, $\beta \in P$, and (i2) $x_0 = u\alpha^R v$ (note that x is a valid string). In the subcase (i1), $z = u\beta v^R\beta^R q\alpha x_2$ holds. We infer that $z \in ld_\sigma(y')$, where $y' = u\beta v^R\beta^R q\alpha x_1\alpha x_2$, hence $y' \in hi_\sigma(x)$. In the subcase (i2), $z = u\alpha^R v^R\alpha x_2$ holds. We now infer that $z \in ld_\sigma(y')$, where $y' = u\alpha^R v^R\alpha x_1\alpha x_2$, hence $y' \in hi_\sigma(x)$.

Case (ii). The modified substring is αx_2. This may be treated analogously to Case (i).

Case (iii). The pointer α is a proper substring, that is neither a prefix nor a suffix, of the modified string. Formally, $x_0 = u\beta v, x_2 = q\beta^R t$, for some u, v, q, t. It follows that $z = u\beta q^R\alpha^R v^R\beta^R t$. We deduce that $z \in ld_\sigma(y')$, where $y' = u\beta q^R\alpha^R x_1^R\alpha^R v^R\beta^R t$, therefore $y' \in hi_\sigma(x)$. Thus, we have proved that $ld_\sigma(hi_\sigma(x)) \supseteq hi_\sigma(ld_\sigma(x))$ holds.

For the converse inclusion, let $z \in ld_\sigma(hi_\sigma(x))$. There exists $y \in hi_\sigma(x)$ such that $z \in ld_\sigma(y)$. We may consider that $y = x_1\beta x_2^R\beta^R x_3$, provided that $x = x_1\beta x_2\beta^R x_3$. It is easy to notice that the substrings of y that can be modified by the ld operation are: (i) $x_1\beta$, (ii) βx_2^R, (iii) $x_2^R\beta^R$, and (iv) $\beta^R x_3$. By the definition of ld which imposes that the loop does not contain any pointer, we immediately conclude that z can be obtained by applying first the ld operation to the substrings of x: $x_1\beta$, βx_2, $x_2\beta^R$, and $\beta^R x_3$, respectively, and then the hi operation guided by the pointers β and β^R.

2. Let $q \in ld_\sigma(dld_\sigma(x))$. We consider that $x = u\alpha v\beta w\alpha y\beta z$; there exists $t = u\alpha y\beta w\alpha v\beta z \in dld_\sigma(x)$ such that $q \in ld_\sigma(t)$. The possible site for the application of the ld operation to t could be one of the following substrings (i) $u\alpha$, (ii) $\alpha y\beta$, (iii) $\beta w\alpha$, (iv) the sub$\alpha v\beta$, and (v) βz. No matter which site is chosen, the two operations can be interchanged. It follows that $ld_\sigma(dld_\sigma(x)) \subseteq dld_\sigma(ld_\sigma(x))$.

Now, let $q \in dld_\sigma(ld_\sigma(x))$. We consider that $x = u\alpha v\alpha w$; there exists $t = u\alpha w \in ld_\sigma(x)$ such that $q \in dld_\sigma(t)$. Obviously, by the definition of ld, any application of dld to t can be applied to the same substring of x with just one exception, namely the substring that includes α, when the substring of x includes $\alpha v\alpha$. Consequently, $ld_\sigma(dld_\sigma(x)) = dld_\sigma(ld_\sigma(x))$ holds.

3. We take the string $y = a12a34a21a56a34a56a$, where the substrings of digits represent pointers. On the one hand, $t = a12a34a56a34a21a56a \in dld_\sigma(x)$. Actually, t is the only string in $dld_\sigma(x)$. Then, $hi_\sigma(t)$ is also a singleton that contains $q = a12a43a65a43a21a56a$. Therefore, $q \in hi_\sigma(dld_\sigma(y)$. On the other hand, the unique string in $hi_\sigma(y)$ is $r = a12a43a21a56a34a56a$, but $dld_\sigma(r) = \emptyset$. Now the proof is complete. □

Clearly, the first two statements of this theorem are valid for the simple variants. If we restrict to the simple variants, the equality $dld_\sigma(hi_\sigma(y)) = hi_\sigma(dld_\sigma(y))$ holds for any valid string y.

3.2 Properties of the Iterated Versions

In this section, all the strings considered are valid strings. Given a *gac*-scheme $\sigma = (V, P)$, and $\tau \in \{ld_\sigma, hi_\sigma, dlad_\sigma\}$ we define:

$$(i) \quad \tau^0(w) = \{w\},$$
$$(ii) \quad \tau^{i+1}(w) = \bigcup_{x\in\tau^i(w)} \tau(x),$$
$$(iii) \quad \tau^*(w) = \bigcup_{i\geq 0} \tau^i(w).$$

Given a gac-scheme $\sigma = (V, P)$, we say that a string w is an ld_σ-macronuclear string if $ld^*_\sigma(w) = \{w\}$. The set of all ld_σ-macronuclear strings evolved from a string x is denoted by $mld_\sigma(x)$ and is defined by

$$mld_\sigma(x) = \{w \mid w \in ld^*_\sigma(x) \text{ and } w \text{ is } ld_\sigma - macronuclear.\}$$

An ld_σ-macronuclear string may be viewed as a formal description of a protein-coding gene structure in a chromosome of a ciliate macronucleus. Before sexual reproduction the genomic copies of these genes in the micronucleus are separated by the presence of internally eliminated non-protein-coding DNA sequences which must be removed during sexual reproduction.

Theorem 5. *Let $\sigma = (V, P)$ be a gac-scheme. There exists a finite transducer M such that $mld_\sigma(w) = g_M(w)$ for every string w over V.*

Proof. We consider a finite transducer M that implements the following nondeterminstic algorithm.
Step 1. M finds the first occurrence of a pointer in P, say α. All symbols read up to the occurrence of α as well as α are left unchanged. The pointer α is memorized in the current state.
Step 2. Nondeterministically, M chooses one of the next two ways for continuing the computation:
Step 2.1. It "guesses" that the next pointer is also α. Then it deletes all the symbols of the input string from the current position of α up to the next one, as well as the found occurrence of α. If its guess was wrong, the computation is blocked. Note that M can do this as it saved α. Now, M goes back to *Step 2.*.
Step 2.2. M "guesses" that the next pointer is different than α. Then M leaves unchanged all the read symbols until the next occurrence of a pointer is met. If this pointer is α, then the computation is blocked. If it is not α, M writes the new pointer on the output tape, replace α in the memory by the new pointer and goes back to *Step 2.*

The finite transducer defined above actually applies all the possible iterations of ld_σ to w. □

As the set of all valid strings is regular, by the previous theorem it follows that

Corollary 2. *For any gac-scheme $\sigma = (V, P)$, the set of all ld_σ-macronuclear strings is regular.*

Given a gac-scheme $\sigma = (V, P)$, and a string $w \in V^*$, we define

$$(hi/dlad)_\sigma(w) = hi_\sigma(w) \cup dlad_\sigma(w),$$
$$gac_\sigma(w) = (hi/dlad)_\sigma(w) \cup ld_\sigma(w),$$

and iterate these operations as follows

$$(i) \quad \tau^0_\sigma(w) = \{w\},$$
$$(ii) \quad \tau^{i+1}_\sigma(w) = \bigcup_{x \in \tau^i_\sigma(w)} \tau_\sigma(x),$$
$$(iii) \quad \tau^*_\sigma(w) = \bigcup_{i \geq 0} \tau^i_\sigma(w),$$

where $\tau \in \{gac, (hi/dlad)\}$. As a consequence of Theorem 4, we have this result as a *normal form* of the gene assembly in ciliates.

Theorem 6. *Given a gac-scheme $\sigma = (V, P)$, and a string $w \in V^*$, $gac^*_\sigma(w) = ld^*_\sigma(hi/dlad)^*_\sigma(w)) = (hi/dlad)^*_\sigma(ld^*_\sigma(w))$ holds.*

Proof. Obviously, both inclusions

$$ld^*_\sigma(hi/dlad)^*_\sigma(w)) \subseteq gac^*_\sigma(w)$$

and

$$(hi/dlad)^*_\sigma(ld^*_\sigma(w)) \subseteq gac^*_\sigma(w)$$

hold.

We now prove the inclusion $gac^*_\sigma(w) \subseteq ld^*_\sigma(hi/dlad)^*_\sigma(w))$, while the other inclusion, namely $gac^*_\sigma(w) = (hi/dlad)^*_\sigma(ld^*_\sigma(w))$ may be proved analogously. The argument is the induction on the number of gac steps. More precisely, let $z \in gac^k_\sigma(w)$, for some $k \geq 0$. If $k = 0$, then $z = w$, hence $z \in ld^0_\sigma(hi/dlad)^0_\sigma(w))$. Assume that $gac^k_\sigma(w) \subseteq ld^*_\sigma(hi/dlad)^*_\sigma(w))$ holds, and take $z \in gac^{k+1}_\sigma(w)$. This implies that $z \in gac_\sigma(y)$ for some $y \in gac^k_\sigma(w)$. By the induction hypothesis, $y \in ld^*_\sigma(hi/dlad)^*_\sigma(w))$. We distinguish three cases depending on the operation applied to y in order to get z: ld, hi, and $dlad$.

If $z \in ld_\sigma(y)$, then $z \in ld_\sigma(ld^*_\sigma(hi/dlad)^*_\sigma(w))) \subseteq ld^*_\sigma(hi/dlad)^*_\sigma(w))$.

If $z \in hi_\sigma(y)$, then $z \in hi_\sigma(ld^*_\sigma(hi/dlad)^*_\sigma(w)))$. This means that $z \in hi_\sigma(ld^q_\sigma(hi/dlad)^r_\sigma(w)))$, for some $q, r \geq 0$, but not both being null. If $q = 0$, we are done. Otherwise, by Theorem 4, we can interchange q times the operations hi and ld from left to right and yield that $z \in ld^q_\sigma(hi/dlad)^{r+1}_\sigma(w))$.

As the operations ld and $dlad$ can also be interchaged by Theorem 4, the case $z \in dlad_\sigma(y)$ is treated analogously. □

This theorem says that instead of iteratively applying the three operations in an arbitrary order, one can iteratively apply first the ld operation, which decrease the length of the string, and then the other two operations.

4 Final Remarks

We have reconsidered the three formal operations on strings suggested by the intramolecular gene assembly in ciliates. Properties of both non-iterated and iterated variants have been investigated. We shortly discuss here a few other properties that appear to be interesting to us and were not considered in this paper.

Note that the operations hi and $dlad$ are reversible (but not the ld operation), that is for any gac-scheme $\sigma = (V, P)$ and $\tau \in \{hi_\sigma, dlad_\sigma\}$, $x \in \tau(w)$ if and and only if $w \in \tau(x)$. This leads to the following definition of an equivalence relation $\sim_\tau$:

$$x \sim_\tau y \text{ if } x \in \tau^*(y).$$

It immediately follows that the relation is indeed an equivalence relation, being reflexive, symmetric and transitive.

Note that the problem "Does $x \in ld^*_\sigma(y)$ holds, for any given strings x and y?" is decidable in linear time. Indeed, by scanning both strings x and y, when a mismatch is met, try to apply the ld operation to remove the mismatch. If this is possible, continue the scanning process, otherwise the answer is negative.

Proposition 1. *Let $\sigma = (V, P)$ be a gac-scheme and $\tau \in \{hi_\sigma, dlad_\sigma\}$.*
1. The problem of whether $x \sim_\tau y$, for two strings $x, y \in V^+$, is decidable.
2. If hi and dld are simple, then the problem of whether $x \sim_\tau y$, for two strings $x, y \in V^+$, is decidable in linear time.

The proof of the first statement is immediate as any of the equivalence classes defined above are finite. The second statement follows from the fact that a single simultaneous scan of x and y (as that explained above for ld) suffices for taking the decision.

Here a few open problems naturally arise: Is it a more time efficient algorithm for solving the first statement, than the brute force one? What is its complexity? Have both problems (for hi and $dlad$) the same complexity?

Given $\sigma = (V, P)$ a gac-scheme, $x, y \in V^*$, and $\tau \in \{ld_\sigma, (hi/dlad)_\sigma, gac_\sigma\}$ defined above, we define the partial mapping

$$d_\tau(x, y) = \begin{cases} min\{k \mid x \in \tau^k(y) \text{ or } y \in \tau^k(x)\}, \\ \quad \text{if such a } k \text{ exists}, \\ \text{not defined, otherwise.} \end{cases}$$

Clearly, d_τ is computable for any $\tau \in \{ld_\sigma, (hi/dlad)_\sigma, gac_\sigma\}$. While d_{ld_σ} is computable in linear time, the complexity of computing the other functions remains an open problem.

Acknowledgements

This work was performed through the Core Program within the National Research, Development and Innovation Plan 2022-2027, carried out with the support of MRID, project no. 23020101(SIA-PRO), contract no 7N/2022, and project no. 23020301(SAFE-MAPS), contract no 7N/2022.

References

Aho A.V. & Corasick, M.J. (1975). Efficient String Matching: An Aid to Bibliographic Search, *Communications of the ACM*, 18(6), 333–340.

Alhazov A, Petre I, & Rogojin V. (2008). Solutions to Computational Problems Through Gene Assembly. *Nat. Comput.* 7(3), 385–401.

Brijder, R., Daley, M., Harju, T., Jonoska, N., Petre, I., & Rozenberg, G. (2012). Computational Nature of Gene Assembly in Ciliates. In: Rozenberg, G., Bäck, T., Kok, J.N. (Eds) Handbook of Natural Computing. Springer, Berlin, Heidelberg.

Ehrenfeucht, A., Harju, T., Petre, I., Prescott, D.M., & Rozenberg, G. (2004). *Computation in Living Cells. Gene Assembly in Ciliates*, Springer-Verlag Berlin, Heidelberg, New York.

Freund, R., Martín-Vide, C., & Mitrana, V. (2002). On Some Operations on Strings Suggested by Gene Assembly in Ciliates, *New Gener. Comput.* 20(3), 279-294.

Harju, T., Petre, I., Rogojin, V., & Rozenberg, G. (2006). Simple Operations for Gene Assembly. In A. Carbone & N.A. Pierce (Eds.) *Proceedings of the 11^{th} Workshop of DNA Computing (DNA 11)* (pp. 96-111). Springer-Verlag Berlin Heidelberg.

Kari, L. & Rozenberg, G. (2008). The Many Facets of Natural Computing, *Communications of the ACM* 51(10), 72–83.

Landweber, L.F., & Kari, L. (2002). Universal Molecular Computation in Ciliates. In: L.F. Landweber & Winfree, E. (Eds.) *Evolution as Computation* (257–274). Springer, New York.

Onolt-Ishdorj, T., Petre I, & Rogojin V. (2007). Computational Power of Intramolecular Gene Assembly. *Int. J. Found. Comp. Sci.* 18(5), 1123–1136.

Păun, G., Rozenberg, G., & Salomaa, A. (1998). *DNA Computing. New Computing Paradigms*, Springer-Verlag Berlin, Heidelberg, New York.

Rozenberg, G., Bäck, T., & Kok, J.N. (2012). *Handbook of Natural Computing*, Springer-Verlag Berlin, Heidelberg.

Vaszil, G. (2020). Personal communication.