A String-based Model for Simple Gene Assembly

Robert Brijder[1], Miika Langille[2], and Ion Petre[2,3]

[1] Leiden Institute of Advanced Computer Science, Universiteit Leiden
Niels Bohrweg 1, 2333 CA Leiden, The Netherlands
rbrijder@liacs.nl
[2] Turku Centre for Computer Science
Department of IT, Åbo Akademi University
Turku 20520 Finland
miika.langille@abo.fi
ion.petre@abo.fi
[3] Academy of Finland

Abstract. The simple intramolecular model for gene assembly in ciliates is particularly interesting because it can predict the correct assembly of all available experimental data, although it is not universal. The simple model also has a confluence property that is not shared by the general model. A previous formalization of the simple model through sorting of signed permutations is unsatisfactory because it effectively ignores one operation of the model and thus, it cannot be used to answer questions about parallelism in the model, or about measures of complexity. We propose in this paper a string-based model in which a gene is represented through its sequence of pointers and markers and its assembly is represented as a string rewriting process. We prove that this string-based model is equivalent to the permutation-based model as far as gene assembly is concerned, while it tracks all operations of the model.

1 Introduction

Gene assembly in ciliates has been subject to extensive combinatorial research in recent years, see [2]. Ciliates are unicellular eukaryotes that organize their genome differently in their two types of nuclei. In micronuclei, genes are split into blocks (called *MDSs*), placed in a shuffled order on the chromosome, separated by non-coding blocks. Moreover, some of the MDSs are even presented in an inverted form. In macronuclei however, genes are contiguous sequences of nucleotides, with all blocks sorted in the orthodox order. The assembly of genes from their micronuclear to their macronuclear form has a definite combinatorial and computational flavor: each MDS M ends with a sequence of nucleotides (called a *pointer*) that is repeated identically in the beginning of the MDS that should follow M in the macronuclear gene.

The exact kinetical mechanisms of gene assembly still remain to be clarified through further laboratory experiments. Two models have been proposed for gene assembly: an intermolecular one, see [7, 8] and an intramolecular one, see [3, 10]. The intramolecular model, that we consider in this paper, consists of

three operations: ld, hi, and dlad. For a detailed discussion of these operations, including their formalization on various levels of abstraction, we refer to [2]. We consider in this paper the simple variant of the model, in which the sequences manipulated by each operation are minimal, see [5] for a detailed discussion. It turns out that although not universal, the simple model is capable of correctly predicting the assembly of all currently available data on genes in stichotrichous ciliates, see [6]. Also, the model has an interesting confluence property that does not hold in the general model, see [9].

The simple model for gene assembly has been investigated in [9] as a process of sorting signed permutations. A major shortcoming of this formalization is that the ld operations are not explicitly modeled and it is only assumed that they take place eventually at some arbitrary moment of time. Although sufficient for the purpose of [9], this formalization does not allow reasoning about parallelism or complexity of assemblies.

Abstracting the gene to its sequence of pointers has been a solution to this problem in the case of the general model. In this way, the gene assembly process is formalized through a process of string rewriting. Doing the same in the case of simple operations does not work, as shown in Example 9 of this paper: the string-based model is not equivalent with the permutation-based model.

We propose in this paper a simple solution to this problem. Rather than representing a gene only through its sequence of pointers, we preserve also the beginning and ending markers (starting the first MDS and ending the last MDS, respectively). We prove that with this simple addition, the string-based model is equivalent with the permutation-based model.

2 Mathematical Preliminaries

For a finite alphabet $A = \{a_1, \ldots, a_n\}$, we denote by A^* the free monoid generated by A and call any element of A^* a *word*. Let $\overline{A} = \{\overline{a}_1, \ldots, \overline{a}_n\}$, where $A \cap \overline{A} = \emptyset$. For $p, q \in A \cup \overline{A}$, we say that p, q have the same *signature* if either $p, q \in A$, or $p, q \in \overline{A}$ and we say that they have *different signatures* otherwise.

We denote $A^{\maltese} = (A \cup \overline{A})^*$. For any $u \in A^{\maltese}$, $u = x_1 \ldots x_k$, with $x_i \in A \cup \overline{A}$, for all $1 \leq i \leq k$, we denote $\|u\| = \|x_1\| \ldots \|x_k\|$, where $\|a\| = \|\overline{a}\| = a$, for all $a \in A$. We also denote $\overline{u} = \overline{x}_k \ldots \overline{x}_1$, where $\overline{\overline{a}} = a$, for all $a \in A$. For two alphabets A, B, a mapping $f : A^{\maltese} \to B^{\maltese}$ is called a *morphism* if $f(uv) = f(u)f(v)$ and $f(\overline{u}) = \overline{f(u)}$.

A *permutation* π over A is a bijection $\pi : A \to A$. Fixing the order relation $(a_1, a_2, \ldots, a_m)$ over A, we often denote π as the word $\pi(a_1) \ldots \pi(a_m) \in A^*$. A *signed permutation* over A is a string $\psi \in A^{\maltese}$, where $\|\psi\|$ is a permutation over A.

A string $v \in \Sigma^*$ over an (unsigned) alphabet Σ is a *double occurrence string* if every pointer $a \in \mathrm{dom}(v)$ occurs exactly twice in v. A signing of a non-empty double occurrence string is a *legal string*.

Two legal strings, u and v over alphabets A and B, respectively, are said to have the same *structure* if there is a morphism $f : A^{\maltese} \to B^{\maltese}$ such that $f(u) = v$, and is denoted by $u \equiv v$.

3 Permutations and Strings

Simple operations have previously been defined in terms of signed permutations, see [9]. This formalism was useful for observing simple operations applied sequentially, with the arrangement of explicit MDSs being clearly visible. However, note that with signed permutations the ld operation is always assumed to occur at some point in the assembly. This is not sufficient for studying some aspects of simple gene assembly, and so we now task ourselves with providing a formal framework for performing simple gene assembly on *legal strings*. Micronuclear and intermediate genes are represented as a sequence of pointers and the operations defined as a *string pointer reduction system*. Moreover, we will show that simple gene assembly on legal strings is equivalent to that on signed permutations.

Legal strings have been used before to formalise the general operations in [2]. This definition is missing one crucial piece of information, necessary to model simple operations in particular. Namely, the absence of markers indicating the beginning and end of the macronuclear gene make it necessary to extend the definition.

Let us now briefly describe the signed permutations, followed then by the extended definition of legal strings and the string pointer reduction system.

Signed Permutations

Genes may be represented as a sequence of MDSs. Using the alphabet $\Pi_n = \{1, 2, \ldots, n\}$ to denote MDSs, where the numbering is given by the order in which MDSs are assembled in the macronuclear gene. Thus, a micronuclear gene will be a signed permutation over Π_n and a macronuclear gene will be a sorted signed permutation over Π_n.

Definition 1 *We say that a signed permutation π is sorted if either:*

i. $\pi = \overline{i(i+1)\ldots n1\ldots(i-1)}$, *or*
ii. $\pi = \overline{(i-1)\ldots\overline{1}\overline{n}\ldots\overline{(i+1)}\overline{i}}$,

for some $1 \leq i \leq n$. If $i = 1$ we say that π is a linear permutation. We call π circular otherwise. In case i. we say that π is sorted in the orthodox order, while in case ii. we say that π is sorted in the inverted order.

The term circular in the above definition refers to a gene that gets assembled, say in the form $i\ldots n1\ldots(i-1)$, and then gets excised from the chromosome by an ld operation applied on the pointer in the beginning of i and its identical copy at the end of $(i-1)$.

Given this definition for genes, we may now consider gene assembly as a sorting of signed permutations. We will define the simple operations as transformation rules for signed permutations in such a way that a simple operation is applicable on a gene pattern if and only if the corresponding rule is applicable on the associated signed permutation.

As mentioned previously, when using signed permutations to model gene assembly, the ld operation is ignored. Indeed, ld only combines two MDSs i and $(i+1)$ already placed next to each other into a bigger composite MDS. To avoid renaming the alphabet after each ld operation, we will consider that when i and $(i+1)$ are placed next to each other, the operation joining them is already accomplished.

Definition 2 *The molecular model of simple* hi *and simple* dlad *can be formalized as follows.*

i. For each $p \geq 1$, sh_p *is defined as follows:*

$$\mathsf{sh}_p(xp\dots(p+i)(\overline{p+k})\dots(\overline{p+i+1})y) = xp\dots(p+i)(p+i+1)\dots(p+k)y,$$
$$\mathsf{sh}_p(x(\overline{p+i})\dots\overline{p}(p+i+1)\dots(p+k)y) = xp\dots(p+i)(p+i+1)\dots(p+k)y,$$
$$\mathsf{sh}_p(x(p+i+1)\dots(p+k)(\overline{p+i})\dots\overline{p}y) = x(\overline{p+k})\dots(\overline{p+i+1})(\overline{p+i})\dots\overline{p}y,$$
$$\mathsf{sh}_p(x(\overline{p+k})\dots(\overline{p+i+1})p\dots(p+i)y) = x(\overline{p+k})\dots(\overline{p+i+1})(\overline{p+i})\dots\overline{p}y,$$

where $k > i \geq 0$ *and* x, y, z *are signed strings over* Π_n*. We denote* $\mathsf{Sh} = \{\mathsf{sh}_i \mid 1 \leq i \leq n\}$.

ii. For each p, $2 \leq p \leq n-1$, sd_p *is defined as follows:*

$$\mathsf{sd}_p(x\,p\dots(p+i)\,y\,(p-1)\,(p+i+1)\,z) = xy(p-1)p\dots(p+i)(p+i+1)z,$$
$$\mathsf{sd}_p(x\,(p-1)(p+i+1)yp\dots(p+i)z) = x(p-1)p\dots(p+i)(p+i+1)yz,$$

where $i \geq 0$ *and* x, y, z *are signed strings over* Π_n*. We also define* $\mathsf{sd}_{\overline{p}}$ *as follows:*

$$\mathsf{sd}_{\overline{p}}(x(\overline{p+i+1})(\overline{p-1})y(\overline{p+i})\dots\overline{p}z) = x\,(\overline{p+i+1})(\overline{p+i})\dots\overline{p}(\overline{p-1})yz,$$
$$\mathsf{sd}_{\overline{p}}(x(\overline{p+i})\dots\overline{p}y(\overline{p+i+1})(\overline{p-1})z) = xy(\overline{p+i+1})(\overline{p+i})\dots\overline{p}(\overline{p-1})z,$$

where $i \geq 0$ *and* x, y, z *are signed strings over* Π_n*. We denote* $\mathsf{Sd} = \{\mathsf{sd}_i, \mathsf{sd}_{\overline{i}} \mid 1 \leq i \leq n\}$.

We say that a signed permutation π *over the set of integers* $\{i, i+1, \dots, i+l\}$ *is sortable if there are operations* $\phi_1, \dots, \phi_k \in \mathsf{Sh} \cup \mathsf{Sd}$ *such that* $(\phi_k \circ \dots \circ \phi_1)(\pi)$ *is a sorted permutation. We say that* π *is blocked if neither an* sh *operation, nor an* sd *one is applicable to* π *and* π *is not sorted.*

Let $\phi = \phi_k \circ \dots \circ \phi_1$, $\phi_i \in \mathsf{Sh} \cup \mathsf{Sd}$, *for all* $1 \leq i \leq k$*. We say that* ϕ *is a strategy for* π *if* $\phi(\pi)$ *is either sorted, or blocked. In the former case we say that* ϕ *is a sorting strategy, while in the latter case we say that* ϕ *is an unsuccessful strategy for* π*.*

If $\phi = \phi_k \circ \dots \circ \phi_1$ *is a sorting strategy for* π*, we say that* π *is* Sh*-sortable if* $\phi_1, \dots, \phi_k \in \mathsf{Sh}$ *and we say that* π *is* Sd*-sortable if* $\phi_1, \dots, \phi_k \in \mathsf{Sd}$*.*

Example 1 *i. The permutation* $\pi_1 = \overline{2}1435$ *is sortable. It has the following sorting strategies:* $\mathsf{sh}_1 \circ \mathsf{sd}_4 \circ \mathsf{sh}_1(\pi_1) = 12345$ *and* $\mathsf{sh}_1 \circ \mathsf{sh}_1 \circ \mathsf{sd}_4(\pi_1) = 12345$.

ii. The permutation $\pi_2 = 25\overline{3}14$ *is blocked as no operations are applicable and it is not sorted.*

iii. The permutation $\pi_3 = 35124$ *has two assembly strategies which lead to different results:* $\mathsf{sd}_3(\pi_3) = 51234$ *and* $\mathsf{sd}_4(\pi_3) = 34512$.

Legal Strings

Using signed permutations, we hold the information of explicit MDSs. It is sufficient, however, to consider the gene as a sequence of pointers. Moving to legal strings, we remove all notation pertaining to explicit MDSs, and represent the gene as a string of pointers, with each pointer occurring twice in the gene. Though some information is lost about the gene, the notation is elegant and the characteristics and rules of the model are maintained.

To represent a micronuclear or intermediate gene as a legal string, we need a set of pointers, $\Delta_k = \{2, 3, \ldots, k\}$, and the set of markers $M = \{b, e\}$. Note that each marker will occur only once in the string, and each pointer will occur twice. As with signed permutations, k represents the length of the gene, and we shall assume that it will be clear from the context. Let this be our alphabet $\Sigma = \Delta_k \cup M$.

The definition for legal strings given in [2] did not include the markers, but as it turns out, they are essential to being able to formalise simple operations on legal strings. Additionally, not including the markers meant that it was no longer possible to uniquely obtain a signed permutation from a legal string, as too much information was lost.

Let $a \in \Sigma \cup \overline{\Sigma}$ and let $u \in \Sigma^{\maltese}$ be a legal string. If u contains both substrings a and $\overline{a}$ then a is *positive* in u; otherwise, a is *negative* in u.

Example 2 *i. Consider the signed string* $u_1 = b23423\overline{e}\overline{4}$ *over* Σ_4. *Clearly,* u_1 *is legal. Pointer 4 is positive in* u_1, *while 2 and 3 are negative in* u_1.

ii. The string $u_2 = \overline{43}b234$ *is not legal, since 2 has only one occurrence in* u_2 *and the end marker does not occur at all.*

Let $u = a_1 a_2 \ldots a_n \in \Sigma^{\maltese}$ be a legal string over Σ, where $a_i \in \Sigma \cup \overline{\Sigma}$ for each i. For each letter $a \in \mathrm{dom}(u)$, there are indices i and j with $1 \leq i < j \leq n$ such that $\|a_i\| = a = \|a_j\|$, excepting of course the two markers present. The substring $u_{(a)} = a_i a_{i+1} \ldots a_j$ is the *a–interval* of u. Two different letters $a, b \in \Sigma$ are said to *overlap in* u if the a–interval and the b–interval of u overlap: if $u_{(a)} = a_{i_1} \ldots a_{j_1}$ and $u_{(b)} = a_{i_2} \ldots a_{j_2}$, then either $i_1 < i_2 < j_1 < j_2$ or $i_2 < i_1 < j_2 < j_1$. Moreover, for each letter a, we denote by

$$O_u(a) = \{b \in \Sigma \mid b \text{ overlaps with } a \text{ in } u\} \cup \{a\}.$$

Example 3 *Let* $u = b2342\overline{3}\overline{5}\overline{4}\overline{e}\overline{6}56$ *be a legal string. The 5–interval of* u *is the substring* $u_{(5)} = \overline{5}\overline{4}\overline{6}5$, *which contains only one occurrence of the pointers 4 and*

6, but either two or no occurrences of 2 and 3 and so $O_u(5) = \{4, 5, 6\}$. *Note that the marker is not included in the interval. Similarly,*

$$\begin{aligned} u_{(2)} &= 2342, && \textit{and } 2 \textit{ overlaps with } 3 \textit{ and } 4, \\ u_{(3)} &= 3423, && \textit{and } 3 \textit{ overlaps with } 2 \textit{ and } 4, \\ u_{(4)} &= 423\overline{54}, && \textit{and } 4 \textit{ overlaps with } 2, 3 \textit{ and } 5, \\ u_{(6)} &= \overline{6}56, && \textit{and } 6 \textit{ overlaps with } 5. \end{aligned}$$

We will now consider the transformation from signed permutations to legal strings, carried out by replacing each MDS with the pointers at either end of it. When transforming from signed permutations to legal strings, there is a problem with how to deal with sorted blocks of pointers. Therefore, when doing the transformation, the signed permutation must be assumed to be micronuclear, i.e., no MDSs have been joined together. This allows for the following morphism from a signed permutation π to a legal string u. $\zeta : \Pi_n^{\Phi} \to \Sigma_n^{\Phi}$ is applied to each letter in π, and is defined as follows

i. $i \to i(i+1)$,
ii. $1 \to b2$,
iii. $n \to ne$,

where $1 \leq i \leq n$.

Example 4 *i. The signed permutation* $\pi = 1\overline{2}$, *is equivalent to the following legal string* $\zeta(\pi) = b2\overline{e}\overline{2}$.
ii. The signed permutation $\pi = 2314\overline{7}6\overline{5}$, *is equivalent to the following legal string* $\zeta(\pi) = 2334b245\overline{e}\overline{7}67\overline{6}\overline{5}$.

We say that a legal string u is *realistic* if there exists a signed permutation π such that $u = \zeta(\pi)$. It is important to note, however, that not all legal strings are realistic. The following example illustrates this point.

Example 5 *The string* $u = be22$ *is legal but it is not realistic, since it has no "realistic parsing", i.e., there is no signed permutation which would transform to this legal string.*

Let us first give our full definition of legal strings and the simple operations, and then we will show why the markers are necessary to the abstraction.

Simple Operations on Signed Strings

Recall that in signed permutations gene assembly was simulated by joining together MDSs to obtain a (circularly) sorted sequence, the macronuclear gene. With the move to legal strings we have now removed all details of individual MDSs, leaving us with a string of pointers. The goal remains the same though, to remove all pointers by matching them together, consequently building the

macronuclear gene. We shall now introduce the *string pointer reduction system* for simple operations to formalise the three molecular operations ld, hi and dlad.

A realistic legal string is considered sorted, once all pointers are removed, leaving us with one of the following possible sorted permutations: be, $\overline{e}\overline{b}$, eb and $\overline{b}\overline{e}$. The first two cases represent linear sortings, and the latter two are considered circular.

In the following we shall consider only realistic legal strings, that is, the strings are from $\Sigma_k^{\maltese}$ for some $k \geq 2$. Each of the rules below is a function that maps legal strings to legal strings.

i. The *string negative rule* snr for a pointer $p \in \Delta_k$ ($k \geq 2$) is the equivalent of the ld operation. As with ld, the string negative rule is always simple, and as such, it is the same for the general and simple model. It can only be applied when the pointers are adjacent, i.e., not separated by any other pointers. It can be formalised in the following way

$$\mathsf{snr}_p(u_1 p p u_2) = u_1 u_2,$$

$$\mathsf{snr}_p(p u_3 p) = u_3,$$

where $u_1, u_2 \in \Sigma^{\maltese}$ and u_3 contains only markers (boundary case). Let

$$\mathsf{Snr} = \{\mathsf{snr}_p \mid p \in \Delta_k, k \geq 2\}$$

be the set of all simple string negative rules.

ii. The *simple string positive rule* sspr for a pointer $p \in \Delta_k (k \geq 2)$ is the equivalent of the sh operation. Recall that in the simple hi operation the pointers may only be separated by one MDS, and thus only a single pointer or marker. This gives us the following formalisation

$$\mathsf{sspr}_p(u_1 p u_2 \overline{p} u_3) = u_1 \overline{u}_2 u_3,$$

where $|u_2| = 1$ and $u_1, u_2, u_3 \in \Sigma^{\maltese}$. Let

$$\mathsf{Sspr} = \{\mathsf{sspr}_p \mid p \in \Delta_k, k \geq 2\}$$

be the set of all simple string positive rules.

iii. The *simple string double rule* ssdr for pointers $p, q \in \Delta_k (k \geq 2)$ is the equivalent of the sd operation. Recall that in the simple dlad operation the first occurrences of p and q must be adjacent, with the same condition applied to the second occurrences. This gives us the following formalisation

$$\mathsf{ssdr}_{p,q}(u_1 p q u_2 p q u_3) = u_1 u_2 u_3,$$

where $u_1, u_2, u_3 \in \Sigma^{\maltese}$. Let

$$\mathsf{Ssdr} = \{\mathsf{ssdr}_{p,q} \mid p, q \in \Delta_k, k \geq 2\}$$

be the set of all simple string double rules.

Note that the operations have now become very simple indeed, with only pointers being removed. The sspr operation is the only one that affects the remaining permutation, inverting the pointer or marker separating the two occurrences.

Example 6 *Consider the following signed permutation* $\pi = \overline{1}2354$*. It has the following realistic legal string* $u = \zeta(\pi) = \overline{2}\overline{b}23345e45$*, and these simple operations applicable to it.*

i. The snr *operation removes two adjacent pointers with the same signature. E.g.,* $\mathsf{snr}_3(u) = \overline{2}\overline{b}245e45$.

ii. The sspr *operation removes two pointers with different signatures, separated by exactly one pointer or marker, inverting the sequence separating them. E.g.,* $\mathsf{sspr}_2(u) = b3345e45$.

iii. The ssdr *operation removes two overlapping pointers, where the first occurrences are adjacent, as are the second occurrences. E.g.,* $\mathsf{ssdr}_{4,5}(u) = \overline{2}\overline{b}233e$.

Example 7 *Let us consider the actin I gene from Sterkiella nova, which has the following signed permutation* $\pi = 346579\overline{2}18$*. The corresponding legal string would be*

$$u = \zeta(\pi) = 34456756789e\overline{3}\overline{2}b289.$$

It has the following assembly strategy using the simple string pointer reduction system.

$$\begin{aligned}
&\mathsf{sspr}_2(u) = 34456756789e\overline{3}\overline{b}89\\
&\mathsf{ssdr}_{5,6} \circ \mathsf{sspr}_2(u) = 3447789e\overline{3}\overline{b}89\\
&\mathsf{snr}_4 \circ \mathsf{ssdr}_{5,6} \circ \mathsf{sspr}_2(u) = 37789e\overline{3}\overline{b}89\\
&\mathsf{snr}_7 \circ \mathsf{snr}_4 \circ \mathsf{ssdr}_{5,6} \circ \mathsf{sspr}_2(u) = 389e\overline{3}\overline{b}89\\
&\mathsf{ssdr}_{8,9} \circ \mathsf{snr}_7 \circ \mathsf{snr}_4 \circ \mathsf{ssdr}_{5,6} \circ \mathsf{sspr}_2(u) = 3e\overline{3}\overline{b}\\
&\mathsf{sspr}_3 \circ \mathsf{ssdr}_{8,9} \circ \mathsf{snr}_7 \circ \mathsf{snr}_4 \circ \mathsf{ssdr}_{5,6} \circ \mathsf{sspr}_2(u) = \overline{e}\overline{b}
\end{aligned}$$

It is important to note that the naming of simple operations on legal strings does not follow the same standard as for signed permutations. On signed permutations an operation was always named by the smallest MDS in the composite involved. However, on legal strings we have lost the information of MDSs, and only remaining pointers are shown. Thus an operation must be named according to the pointer on which the fold is made. This is illustrated in the following example.

Example 8 *Let* $u = b2452334\overline{e}\overline{5}$ *be a realistic legal string corresponding to the signed permutation* $\pi = 1423\overline{5}$*. They have the following equivalent assembly strategies,*

$$\begin{aligned}
&\mathsf{snr}_3(u) = b24524\overline{e}\overline{5}, && \text{(}\mathsf{Id}\text{ operation is assumed here)}\\
&\mathsf{ssdr}_{2,4} \circ \mathsf{snr}_3(u) = b5\overline{e}\overline{5}, && \mathsf{sd}_2(\pi) = 1234\overline{5},\\
&\mathsf{sspr}_5 \circ \mathsf{ssdr}_{2,4} \circ \mathsf{snr}_3(u) = be, && \mathsf{sh}_1 \circ \mathsf{sd}_2(\pi) = 12345.
\end{aligned}$$

Let us now show that the operations on legal strings are equivalent to those on signed permutations.

Theorem 1 *For any signed permutations π and π', with $\pi' = \phi_m \circ \ldots \phi_1(\pi)$, $\phi_i \in \mathsf{Sh} \cup \mathsf{Sd}$, let $u = \zeta(\pi)$ and $u' = \zeta(\pi')$. Then $\tau_0(u') = \psi_m \circ \tau_m \circ \ldots \psi_1 \circ \tau_1(u)$, where ψ_i is the equivalent of ϕ_i, and τ_i is a composition of* snr *operations.*

Proof. We will only prove the claim for $m = 1$. In its full generality, the claim can be proved by iterating the same argument.

Case 1: $\phi_1 \in \mathsf{Sh}$. Then π is of one of the four forms in Definition 2(i). Assume that $\pi = xp\ldots(p+i)(\overline{p+k})\ldots(\overline{p+i+1})y$, as the other cases are completely similar. Then $u = \zeta(x)p(p+1)(p+1)\ldots(p+i)(p+i)(p+i+1)(\overline{p+k+1})(\overline{p+k})(\overline{p+k})\ldots(\overline{p+i+2})(\overline{p+i+2})(\overline{p+i+1})\zeta(y)$ and

$$\mathsf{sh}_p(\pi) = xp\ldots(p+k)y,$$
$$\zeta(\mathsf{sh}_p(u)) = \zeta(x)p(p+1)(p+1)\ldots(p+k)(p+k)(p+k+1)$$
$$(\mathsf{sspr}_{p+i+1} \circ \mathsf{snr}_{p+1} \circ \mathsf{snr}_{p+i}\,\mathsf{snr}_{p+i+2} \circ \mathsf{snr}\,p+k)(u) = \zeta(x)p(p+k+1)\zeta(y) =$$
$$= (\mathsf{snr}_{p+1} \circ \ldots \circ \mathsf{snr}_{p+k})(\zeta(\mathsf{sh}_p(u))).$$

Case 2: $\phi_1 \in \mathsf{Sd}$. Then π is of one of the forms in Definition 2(ii). Assume that $\pi = x\,p\ldots(p+i)\,y\,(p-1)\,(p+i+1)\,z$, as the other cases are completely similar. Then $u = \zeta(x)p(p+1)(p+1)\ldots(p+i)(p+i)(p+i+1)\zeta(y)(p-1)p(p+i+1)(p+i+2)\zeta(z)$ and

$$\mathsf{sd}_p(\pi) = xy(p-1)p\ldots(p+i)(p+i+1)z,$$
$$\zeta(\mathsf{sd}_p(\pi)) = \zeta(xy)(p-1)pp\ldots(p+i+1)(p+i+1)(p+i+2)\zeta(z),$$
$$(\mathsf{ssdr}_{p,p+i+1} \circ \mathsf{snr}_{p+1} \circ \ldots \circ \mathsf{snr}_{p+i})(u) = \zeta(x)\zeta(y)(p-1)(p+i+2)\zeta(z) =$$
$$= (\mathsf{snr}_p \circ \ldots \circ \mathsf{snr}_{p+i+1})(\zeta(\mathsf{sd}_p(\pi))).$$

Thus we have shown that the operations on legal strings are equivalent to those for signed permutations. □

Now that we have defined legal strings and formalised simple operations on legal strings we will show why it was necessary to extend the definition of legal strings without markers given in [2]. Consider the definition of the sspr operation $\mathsf{sspr}_p(u_1 p u_2 \overline{p} u_3) = u_1 \overline{u}_2 u_3$. The operation sspr_p may only be applied if $|u_2| = 1$, i.e., it contains a single pointer or a single marker. As no markers are recorded in the definition given in [2], u_2 could in fact contain a pointer and a marker, thus making sspr_p inapplicable. The problem is illustrated in the following example.

Example 9 *Let $u = \overline{32}4234$ be a legal string with the markers removed. It would seem that sspr_2 should be applicable to u, as they are separated by only a single pointer. However, u can be obtained from the following signed permutation $\pi = \overline{2}413$. It is clear that sh_1 is not applicable to π because of MDS 4. Since the permutation-based model and the string-based one should be equivalent, this in turn implies that sspr_2 should not be applicable to u.*

Confluent Strategies on Legal Strings

It was shown in [9] that assembly strategies for a given signed permutation using simple operations are confluent: they are either all successful, or all unsuccessful and moreover, they lead to final results having the same structure. We will now show that the same applies on legal strings. Note that we have no need to define structure for legal strings, as they are in fact isomorphic and thus share the same characteristics and the same operations may be applied to both. Note also that the results of this section are simpler to prove and more general than the similar results in [9].

We will now show that assembly strategies on legal strings are confluent. Lemma 1 will show the case where one of the operations is an snr operation, Lemma 2 considers two sspr operations, and Lemma 3 considers two ssdr operations.

Lemma 1 *Let u be a legal string over Σ_n and $\phi, \psi \in \mathsf{Snr} \cup \mathsf{Sspr} \cup \mathsf{Ssdr}$ be two operations applicable to u. If $\phi \in \mathsf{Snr}$ or $\psi \in \mathsf{Snr}$, then $\phi \circ \psi(u) = \psi \circ \phi(u)$.*

Proof. The proof for this is straightforward as the pointers involved in an snr operation must be adjacent, and thus cannot overlap or affect any other pointers. □

Lemma 2 *Let u be a legal string over Σ_n and $\psi, \phi \in \mathsf{Sspr}$ be two operations applicable to u. Then either $\phi \circ \psi(u) = \psi \circ \phi(u)$, or $\psi(u) \equiv \phi(u)$.*

Proof. Let $\psi = \mathsf{sspr}_p$ and $\phi = \mathsf{sspr}_q$, for some $p \neq q$. If $pq\overline{pq} \not\leq u$ and $qp\overline{qp} \not\leq u$, then clearly $\mathsf{sspr}_q \circ \mathsf{sspr}_p(u) = \mathsf{sspr}_p \circ \mathsf{sspr}_q(u)$.

Now, if $pq\overline{pq} \leq u$ or $qp\overline{qp} \leq u$, then clearly applying one operation makes the other inapplicable, but if $\psi(u) = qq$ and $\phi(u) = pp$ then $\psi(u) \equiv \phi(u)$. □

Note, however, that after applying one of the sspr operations, an snr operation on the remaining pointer becomes available, giving us the following:

$$\mathsf{sspr}_p(u_1 pq\overline{pq}u_2) = u_1 \overline{qq} u_2,$$
$$\mathsf{sspr}_q(u_1 pq\overline{pq}u_2) = u_1 pp u_2,$$
$$\mathsf{snr}_{\overline{q}} \circ \mathsf{sspr}_p(u_1 pq\overline{pq}u_2) = u_1 u_2 = \mathsf{snr}_p \circ \mathsf{sspr}_q(u_1 pq\overline{pq}u_2),$$

for some legal strings u_1, u_2 over Σ_n.

Lemma 3 *Let u be a legal string over Σ_n and $\psi, \phi \in \mathsf{Ssdr}$ be two operations applicable to u. Then either $\phi \circ \psi(u) = \psi \circ \phi(u)$, or $\psi(u) \equiv \phi(u)$.*

Proof. Let $\psi = \mathsf{ssdr}_{p,q}$ and $\phi = \mathsf{ssdr}_{r,s}$, for some $p, q \neq r, s$. If they have a different signature, then clearly $\phi \circ \psi(u) = \psi \circ \phi(u)$. Assume then that $p, q, r, s \in \Sigma_n$. The cases when one or more of p, q, r, s are in $\overline{\Sigma_n}$ are completely similar.

Also, if $p, q \neq r, s$ then $\phi \circ \psi(u) = \psi \circ \phi(u)$, so let $q = r$ (the case when $p = s$ is similar). Thus either $pqsu_1pqs \leq u$ or $sqpu_1sqp \leq u$. Both operations cannot be applied, but $\mathsf{ssdr}_{p,q}(u)$ and $\mathsf{ssdr}_{q,s}(u)$ arrive at equivalent results. Indeed, $\mathsf{ssdr}_{p,q}(u) = u_2 s u_1 s u_3$, $\mathsf{ssdr}_{q,s}(u) = u_2 p u_1 p u_3$ and so, $\mathsf{ssdr}_{p,q}(u) \equiv \mathsf{ssdr}_{q,s}(u)$. □

We can now extend the results above for strategies using all three operations.

Theorem 2 *Let u be a legal string over Σ_n and $\phi, \psi \in \mathsf{Snr} \cup \mathsf{Sspr} \cup \mathsf{Ssdr}$ be two operations applicable to u. Then either $\phi \circ \psi(u) = \psi \circ \phi(u)$, or $\psi(u) \equiv \phi(u)$.*

Proof. Based on the previous three lemmata, we only need to prove the claim in the case $\phi \in \mathsf{Sspr}$, $\psi \in \mathsf{Ssdr}$. Now, if $\phi = \mathsf{sspr}_p$, $\psi = \mathsf{ssdr}_{q,r}$, and $p \neq q, r$, then $\phi \circ \psi(u) = \psi \circ \phi(u)$.

Assume then that $p = q$. Then in sspr_p, the occurrences of p must have different signatures, and in $\mathsf{ssdr}_{p,r}$, all occurrences of p and r must have the same signature, a contradiction. □

Example 10 *Let $u = b2345623456e$.*

i. Both $\mathsf{ssdr}_{2,3}$ and $\mathsf{ssdr}_{5,6}$ are applicable to u. Moreover, $\mathsf{ssdr}_{2,3} \circ \mathsf{ssdr}_{5,6}$ and $\mathsf{ssdr}_{5,6} \circ \mathsf{ssdr}_{2,3}$ are also applicable to u and

$$\mathsf{ssdr}_{2,3} \circ \mathsf{ssdr}_{5,6}(u) = \mathsf{ssdr}_{5,6} \circ \mathsf{ssdr}_{2,3}(u) = b44e.$$

ii. Both $\mathsf{ssdr}_{2,3}$ and $\mathsf{ssdr}_{3,4}$ are applicable to u. Moreover, applying either operation gives an equivalent result:

$$\sigma(\mathsf{ssdr}_{2,3}(u)) = b456456e \equiv b256256e = \sigma(\mathsf{ssdr}_{3,4}(u)).$$

We now have the necessary information to show that the result for signed permutations described in [9] also applies on legal strings, namely that simple operations on legal strings are confluent.

Theorem 3 *Let u be a legal string over Σ_n and ϕ, ψ be two strategies for u. Then either ϕ and ψ are both sorting strategies for u, or they are both unsuccessful strategies. Moreover, $\phi(u) \equiv \psi(u)$.*

Proof. Assume that the claim of the theorem is not true and consider a legal string u of minimal length such that $\phi = \phi_1 \dots \phi_k$ is a successful strategy for u, while $\psi = \psi_1 \dots \psi_l$ is an unsuccessful one, $\phi_i, \psi_j \in \mathsf{Snr} \cup \mathsf{Sspr} \cup \mathsf{Ssdr}$, $1 \leq i \leq k$, $1 \leq j \leq l$.

It follows from Theorem 2 that either $\phi_k(u) \equiv \psi_l(u)$, or $\phi_k \circ \psi_l(u) = \psi_l \circ \phi_k(u)$. If they are equivalent, then $\phi_k(u)$ or $\psi_l(u)$ would be a smaller counterexample than u contradicting the minimality of u. In the latter case note that due to the minimality of u, it follows that $\phi_k(u)$ has only successful strategies and $\psi_l(\pi)$ has only unsuccessful strategies. Consequently, $\psi_l \circ \phi_k(\pi)$ has both successful and unsuccessful strategies, contradicting the minimality of u □

.

Example 11 *The legal string $u = 45\overline{67}6eb2\overline{32}3456$ has several sorting strategies. some of them are shown below.*

$$\begin{aligned}
\phi_1(u) &= \mathsf{snr}_6 \circ \mathsf{snr}_3 \circ \mathsf{sspr}_2 \circ \mathsf{sspr}_7 \circ \mathsf{ssdr}_{4,5}(u) = eb,\\
\phi_2(u) &= \mathsf{snr}_4 \circ \mathsf{ssdr}_{5,6} \circ \mathsf{sspr}_7 \circ \mathsf{snr}_2 \circ \mathsf{sspr}_3(u) = eb,\\
\phi_3(u) &= \mathsf{snr}_6 \circ \mathsf{ssdr}_{4,5} \circ \mathsf{sspr}_7 \circ \mathsf{snr}_3 \circ \mathsf{sspr}_2(u) = eb,\\
\phi_4(u) &= \mathsf{snr}_4 \circ \mathsf{ssdr}_{5,6} \circ \mathsf{sspr}_7 \circ \mathsf{snr}_3 \circ \mathsf{sspr}_2(u) = eb.
\end{aligned}$$

Example 12 *The legal string* $v = b234678e56782345$ *has several unsuccessful strategies. Some of them are shown below.*

$$\psi_1(v) = \mathsf{ssdr}_{6,7} \circ \mathsf{ssdr}_{2,3}(v) = b48e5845,$$
$$\psi_2(v) = \mathsf{ssdr}_{7,8} \circ \mathsf{ssdr}_{3,4}(v) = b26e5625,$$
$$\psi_3(v) = \mathsf{ssdr}_{6,7} \circ \mathsf{ssdr}_{3,4}(v) = b28e5825,$$
$$\psi_4(v) = \mathsf{ssdr}_{2,3} \circ \mathsf{ssdr}_{7,8}(v) = b46e5645.$$

Note that $\psi_1(v) \equiv \psi_2(v) \equiv \psi_3(v) \equiv \psi_4(v)$.

References

1. Cavalcanti, A., Clarke, T.H., Landweber, L., MDS_IES_DB: a database of macronuclear and micronuclear genes in spirotrichous ciliates. *Nucleic Acids Research* **33** (2005) 396–398.
2. Ehrenfeucht, A., Harju, T., Petre, I., Prescott, D. M., and Rozenberg, G., *Computation in Living Cells: Gene Assembly in Ciliates*, Springer (2003).
3. Ehrenfeucht, A., Prescott, D. M., and Rozenberg, G., Computational aspects of gene (un)scrambling in ciliates. In: L. F. Landweber, E. Winfree (eds.) *Evolution as Computation*, Springer, Berlin, Heidelberg, New York (2001) pp. 216–256.
4. Harju, T., Li, C, Petre, I., and Rozenberg, G., Parallelism in gene assembly. *Natural Computing*, (2006).
5. Harju, T., Li, C, Petre, I., and Rozenberg, G., Complexity Measures for Gene Assembly In: K. Tuyls (Eds.), Proceedings of the Knowledge Discovery and Emergent Complexity in Bioinformatics workshop, Springer, Lecture Notes in Bioinformatics, 4366, 2007.
6. Harju, T., Li, C, Petre, I., and Rozenberg, G., Modelling simple operations for gene assembly. In: Junghuei Chen, Natasha Jonoska, Grzegorz Rozenberg (Eds), Nanotechnology: Science and Computation, 361-376, Springer, 2006.
7. Landweber, L. F., and Kari, L., The evolution of cellular computing: Nature's solution to a computational problem. In: *Proceedings of the 4th DIMACS Meeting on DNA-Based Computers*, Philadelphia, PA (1998) pp. 3–15.
8. Landweber, L. F., and Kari, L., Universal molecular computation in ciliates. In: L. F. Landweber and E. Winfree (eds.) *Evolution as Computation*, Springer, Berlin Heidelberg New York (2002).
9. Langille, M., Petre, I., Simple gene assembly is deterministic. *Fundamenta Informaticae* **72** (2006) 1–12, IOS Press.
10. Prescott, D. M., Ehrenfeucht, A., and Rozenberg, G., Molecular operations for DNA processing in hypotrichous ciliates. *Europ. J. Protistology* **37** (2001) 241–260.