Chapter 32

CONTEXT-FREE RECOMBINATIONS

Jarkko Kari
Department of Computer Science
15 MLH
University of Iowa
Iowa City, IA 52242, USA
jjkari@cs.uiowa.edu

Lila Kari
Department of Computer Science
University of Western Ontario
London, ON N6A 5B7, Canada
lila@csd.uwo.ca

Even if the rope breaks nine times, we must splice it back together a tenth time.
— Tibetan proverb

Abstract We address the issue of the computational power of a formal model ([3], [6]) for the guided homologous recombinations that take place during gene rearrangement in ciliates. Results in [7], [5] have shown that a generalization of this model that assumes context-controlled recombinations has universal computational power. Complementing results in [4], [11], [12], [15]; we study properties of context-

*Research partially supported by Grant R2824AO1 of the Natural Sciences and Engineering Research Council of Canada to L.K. and NSF Grant CCR 97-33101 to J.K.

C. Martin-Vide and V. Mitrana (eds.), Where Mathematics, Computer Science, Linguistics and Biology Meet, 361-375.

free recombinations and characterize the languages generated by context-free recombination systems. As a corollary, we obtain context-free recombinations which are computationally weak, being able to generate only regular languages. This is one more indicator that, most probably, the presence of direct repeats does not provide all the information needed for accurate splicing during gene rearrangement.

1. INTRODUCTION AND NOTATION

Ciliates are a diverse group of a few thousand types of unicellular eukaryotes (nucleated cells) that emerged more that 10^9 years ago [13]. Despite their diversity, ciliates still share two common features: the possession of a hair-like cover of cilia used for moving and food capture, and the presence of two nuclei [13]. The *micronucleus* is functionally inert and becomes active only during sexual exchange of DNA, while the active *macronucleus* contains the genes needed for the development of the ciliate. When two cells mate, they exchange micronuclear information and afterwards develop new macronuclei from their respective micronuclei.

In some of the few ciliates studied, the protein-coding segments of the genes (or *MDS*s for macronuclear destined sequences) are present also in the micronucleus interspersed with large segments of non-coding sequences (*IES*s for internally excised sequences). Moreover, these segments are present in a permuted order in the micronucleus. The function of the various eliminated sequences is unknown and moreover they represent a large portion of the micronuclear sequences: the *Oxytricha* macronucleus (average length of 2,200 basepairs per molecule) has $\sim$ 4 % of the DNA sequences present in the micronucleus whereas the *Stylonychia Lemnae* has $\sim$ 2 % ([13]).

As an example, the micronuclear actin I gene in *Oxytricha Nova* is composed of 9 MDSs separated by 8 IESs. The 9 MDSs are present in the permuted order 3-4-6-5-7-9-2-1-8, the proper order being defined by the 1 through 9 arrangement in the functional macronuclear gene [13]. Instructions for unscrambling the micronuclear actin I gene are apparently carried in the gene itself [13]. At the end of each ith MDS ($1 \leqslant i \leqslant 8$) is a sequence of 9 to 13 bp that is identical to a sequence preceding the $(i+1)$th MDS (which occurs somewhere else in the gene). In the model proposed in [13] the homologous recombination between repeats joins the MDSs in the correct order.

In the following we describe a formal system intended to model the guided homologous recombinations that take place during gene rearrangement. Before introducing the formal model, we summarize our notation. An alphabet Σ is a finite, nonempty set. A sequence of letters from Σ is called a string (word) over Σ and in our interpretation corresponds to a linear strand. The length of a word w is denoted by $|w|$ and represents the total number of occurrences of letters in

the word. A word with 0 letters in it is called an empty word and is denoted by λ. The set of all possible words consisting of letters from Σ is denoted by Σ^*, and the set of all nonempty words by Σ^-. We also define circular words over Σ by declaring two words to be equivalent if and only if (iff) one is a cyclic permutation of the other. In other words, w is equivalent to w' iff they can be decomposed as $w = uv$ and $w' = vu$, respectively. Such a circular word $\bullet w$ refers to any of the circular permutations of the letters in w. Denote by $\Sigma^\bullet$ the set of all circular words over Σ.

For a linear word $w \in \Sigma^*$, $\text{Pref}(w) = \{x \in \Sigma^* | \ w = xv\}$, $\text{Suff}(w) = \{y \in \Sigma^* | \ w = uy\}$ and $\text{Sub}(w) = \{z \in \Sigma^* | \ w = uzv\}$.

For a circular word $\bullet w \in \Sigma^\bullet$, we define $\text{Pref}(\bullet w) = \text{Suff}(\bullet w) = \emptyset$ and

$$\text{Sub}(\bullet w) = \{x \in \Sigma^* | \ \bullet w = \bullet uxv, u, v \in \Sigma^*\}$$

as the set of prefixes, suffixes, respectively subwords of $\bullet w$.

For more notions of formal language theory the reader is referred to [14] With this notation we introduce several operations studied in [6], [7] in the context of gene unscrambling in ciliates.

Definition 32.1 *If $x \in \Sigma^+$ is a junction sequence then the recombinations guided by x are defined as follows:*

(i) $uxv + u'xv' \Rightarrow uxv' + u'xv$ (linear/linear), Figure 32.1,

(ii) $uxvxw \Rightarrow uxw + \bullet vx$ (circular/linear), Figure 32.2,

(iii) $\bullet uxv + \bullet u'xv' \Rightarrow \bullet \ uxv'u'xv$ (circular/circular), Figure 32.3.

Note that all recombinations in Definition 32.1 are reversible, i.e., the operations can also be performed in the opposite directions.

For example, operation (ii) models the process of intramolecular recombination. After x finds its second occurrence in $uxvxw$, the molecule undergoes a strand exchange in x that leads to the formation of two new molecules: uxw and a circular DNA molecule $\bullet vx$. Intramolecular recombination accomplishes the deletion of either sequence vx or xv from the original molecule $uxwxv$ and the positioning of w immediately next to ux. This implies that (ii) can be used to rearrange sequences in a DNA molecule thus accomplishing gene unscrambling.

The above operations are similar to the "splicing operation" introduced by Head in [2] and "circular splicing" and "mixed splicing" ([3], [15], [10], [11], [12]). [9], [1] and subsequently [16] showed that some of these models have the computational power of a universal Turing machine. (See [4] for a review.)

In [7] the above strand operations were generalized by assuming that homologous recombination is influenced by the presence of certain contexts, i.e., either the presence of an IES or an MDS flanking a junction sequence. The observed dependence on the old macronuclear sequence for correct IES removal

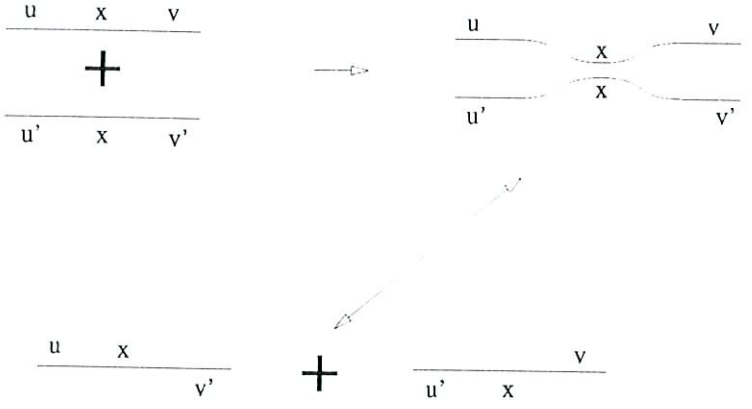

Figure 32.1 Linear/linear recombination.

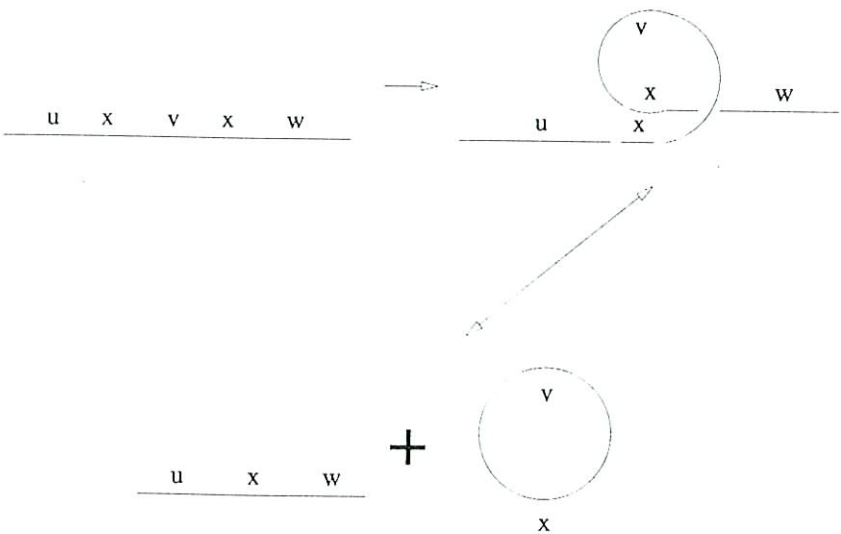

Figure 32.2 Linear/circular recombination.

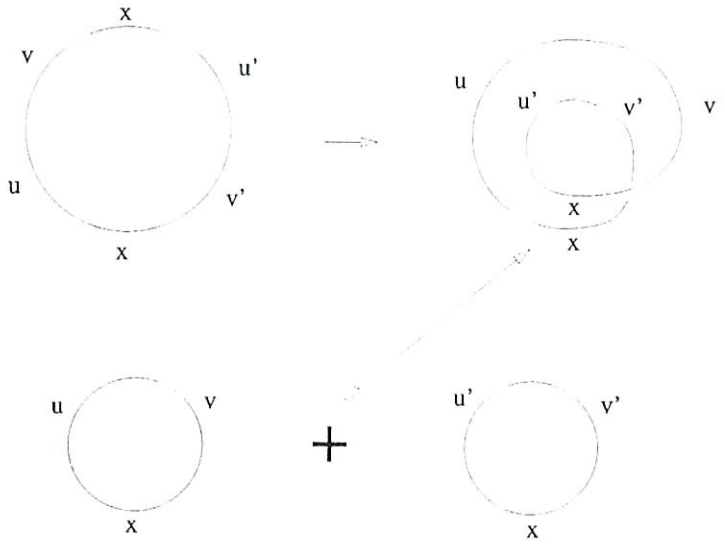

Figure 32.3 Circular/circular recombination.

in *Paramecium* suggests that this is the case ([8]). This restriction captures the fact that the guide sequences do not contain all the information for accurate splicing during gene unscrambling. In particular, in [7] we defined the notion of a *guided recombination system* based on operation (ii) and proved that such systems have the computational power of a Turing machine, the most widely used theoretical model of electronic computers.

We now consider the case where all types of recombinations (linear/linear, linear/circular, circular/circular) are allowed and moreover no context restrictions apply. We study properties of such recombinations. This study complements results obtained in [10], [11], [12], [15], [4] on linear splicing, circular splicing, self-splicing and mixed splicing. However, while Theorem 32.2 may follow from a result in [4] on the closure of an AFL under all splicings, Theorem 32.1 characterizes the language $L(R)$ of an arbitrary context-free recombination system with a possibily infinite set of junction sequences and arbitrary axiom sets.

Theorem 32.2 shows that the resulting rewriting systems are computationally weak having only the power to generate regular languages. This is one more indicator that, most probably, the presence of direct repeats does not provide all the information needed for accurate splicing during gene rearrangement.

2. CONTEXT-FREE RECOMBINATIONS

In our intuitive image of context-free recombinations we can view strings as cables or "extension cords" with different types of "plugs". Given a set of junction sequences J, each $x \in J$ defines one type of "plug". Strings, both linear and circular can then be viewed as consisting of "elementary" cables that only have plugs at their extremities. (A circular strand consists of elementary cables connected to form a loop.) A recombination step amounts to the following operations: take two connections using identical plugs (the connections can be in two different cables or in the same cable); unplug them; cross-plug to form new cables.

In view of Lemma 32.4 to be proved later, we will assume, without loss of generality, that all sets of plugs J are subword-free.

Definition 32.2 *Let $J \subseteq \Sigma^+$ be a set of plugs. We define the set of elementary cables (respectively left elementary cables and right elementary cables) with plugs in J as*

$E_J = (J\Sigma^+ \cap \Sigma^+ J) \setminus \Sigma^+ J \Sigma^+,$
$L_J = \Sigma^* J \setminus \Sigma^* J \Sigma^+,$
$R_J = J\Sigma^* \setminus \Sigma^+ J \Sigma^*.$

Note that an elementary cable in E_J is of the form $z_1 u = v z_2$ where $z_1, z_2 \in J$ are plugs. In other words, an elementary cable starts with a plug, ends with a plug, and contains no other plugs as subwords. The start and end plugs can overlap.

A left elementary cable is of the form wz, where $z \in J$ is a plug and wz does not contain any other plug as a subword. In other words, if we scan wz from left to right, z is the first plug we encounter.

Analogously, a right elementary cable is of the form zw where $z \in J$ is a plug and wz does not contain any other plug as a subword.

Definition 32.3 *For a set of plugs $J \subseteq \Sigma^+$ and a linear word $w \in \Sigma^+$, the set of elementary cables with plugs in J occurring in w is defined as*

$$E_J(w) = E_J \cap \mathit{Sub}(w),$$

while the set of left, respectively right, elementary cables occurring in w is

$$L_J(w) = L_J \cap \mathit{Pref}(w)$$

$$R_J(w) = R_J \cap \mathit{Suff}(w)$$

Note that $L_J(w)$ and $R_J(w)$ are the empty set or singleton sets.

Examples: If $\Sigma = \{a, b\}$ and $J = \{b\}$ then $L_J(aba) = ab$, $R_J(aba) = ba$, $E_J(aba) = \emptyset$. Also $L_J(ab) = ab$, $R_J(ab) = b$, $E_J(ab) = \emptyset$ and $L_J(ba) = b$, $R_J(ba) = ba$, $E_J(ba) = \emptyset$.

Definition 32.4 *For a set of plugs $J \subseteq \Sigma^+$ and a circular word $\bullet w \in \Sigma^\bullet$ we define the elementary cables occurring in $\bullet w$ as follows:*

(i) If $\exists x \in J \cap Sub(\bullet w)$ the elementary cables with plugs in J occurring in $\bullet w$ are defined as $E_J(\bullet w) = E_J(www)$, $L_J(\bullet w) = R_J(\bullet w) = \emptyset$,

(ii) If $J \cap Sub(\bullet w) = \emptyset$ then $E_J(\bullet w) = L_J(\bullet w) = R_J(\bullet w) = \emptyset$.

Examples. If $\Sigma = \{a, b\}$ and $J = \{aba, baa\}$ then $E_J(\bullet aba) = \{abaa, baaba\}$. If $\Sigma = \{a, b, c, d\}$ and $J = \{abc, bcdab\}$ then $E_J(\bullet abcd) = \{abcdab, bcdabc\}$.

From the above examples we see that in circular words start and end plugs are allowed to overlap.

In the definitions for elementary cables, left and right elementary cables can be easily generalized to languages. For a language $L \subseteq \Sigma^* \cup \Sigma^\bullet$,

$$E_J(L) = \bigcup_{w \in L} E_J(w), L_J(L) = \bigcup_{w \in L} L_J(w), R_J(L) = \bigcup_{w \in L} R_J(w)$$

The following two lemmas introduce some properties of elementary cables.

Lemma 32.1 *Given a set of plugs $J \subseteq \Sigma^+$,*

(i) If $u, v \in \Sigma^*$ and $u \in Sub(v)$ then $E_J(u) \subseteq E_J(v)$.

(ii) If $u \in Pref(v)$ then $L_J(u) \subseteq L_J(v)$.

(iii) If $u \in Suff(v)$ then $R_J(u) \subseteq R_J(v)$.

Proof. (i) $E_J(u) = E_J \cap \mathrm{Sub}(u) \subseteq E_J \cap \mathrm{Sub}(v) = E_J(v)$ as $\mathrm{Sub}(u) \subseteq \mathrm{Sub}(v)$.

(ii) and (iii) are proved analogously. □

Lemma 32.2 *If $x \in J$ is a plug then*

(i) $E_J(uxv) = E_J(\{ux, xv\})$,

(ii) $E_J(\bullet ux) = E_J(xux)$,

(iii) $L_J(uxv) \subseteq L_J(ux)$,

(iv) $R_J(uxv) \subseteq R_J(xv)$.

Proof. (i) Let $e = z_1u_1 = u_2z_2$ be an elementary cable in $E_J(uxv)$, $z_1, z_2 \in J$. If e is a subword of ux or of xv then $e \in E_J(ux)$ or $E_J(xv)$, respectively.

If e is not a subword of either ux or xv but e is a subword of uxv then necessarily x is a subword of e. Because x is a plug and e is an elementary cable, e must be a suffix of ux or a prefix of xv. In either case, e is a subword of ux or xv.

(ii) By definition, $E_J(\bullet ux) = E_J(uxuxux)$. By (i), $E_J(uxuxux) = E_J(\{ux, xuxux\}) = E_J(\{ux, xux, xux\})$, which by Lemma 32.1 equals $E_J(xux)$.

(iii) ux is a prefix of uxv therefore $L_J(ux) \subseteq L_J(uxv)$ by Lemma 32.1. Because x is a plug, both $L_J(ux)$ and $L_J(uxv)$ are nonempty and therefore they are equal singleton sets.

(iv) xv is a suffix of uxv therefore $R_J(xv) \subseteq R_J(uxv)$ by Lemma 32.1. Because x is a plug, both $R_J(xv)$ and $R_J(uxv)$ are nonempty and therefore they are equal singleton sets. □

The proposition below shows that recombination of cables does not produce additional elementary cables, i.e. the set of the elementary cables of the result strings equals the set of elementary cables of the strings entering recombination.

Proposition 32.1 *If $J \subseteq \Sigma^+$ is a set of plugs and $x \in J$ then*
(i) $E_J(uxvxw) = E_J(uxw) \cup E_J(\bullet vx)$,
(ii) $E_J(\{uxv, u'xv'\}) = E_J(\{uxv', u'xv\})$,
(iii) $E_J(\{\bullet uxv, \bullet u'xv'\}) = E_J(\bullet uxv'u'xv)$,
(iv) $L_J(uxvxw) = L_J(\{uxw, \bullet vx\})$,
(v) $L_J(\{uxv, u'xv'\}) = L_J(\{uxv', u'xv\})$,
(vi) $R_J(uxvxw) = R_J(\{uxw, \bullet vx\})$,
(vii) $R_J(\{uxv, u'xv'\}) = R_J(\{uxv', u'xv\})$.

Proof. (i) By Lemma 32.2, $E_J(uxvxw) = E_J(\{ux, xvxw\}) = E_J(\{ux, xvx, xw\}) = E_J(\{uxw, xvx\}) = E_J(\{uxw, \bullet vx\})$.

(ii) Similarly, $E_J(\{uxv, u'xv'\}) = E_J(\{ux, xv, u'x, xv'\}) = E_J(\{uxv', u'xv\})$.

(iii) $E_J(\bullet uxv, \bullet u'xv'\}) = E_J(\{\bullet xvu, \bullet xv'u'\}) = E_J(\{xvux, xv'u'x\})$ while

$E_J(\bullet uxv'u'xv) = E_J(\bullet xv'u'xvu) = E_J(xv'u'xvux) = E_J(\{xv'u'x, xvux\})$.

(iv) By Lemma 32.2, $L_J(uxvxw) = L_J(ux)$ while $L_J(\{uxw, \bullet vx\}) = L_J(uxw) = L_J(ux)$.

(v) By Lemma 32.2, $L_J(\{uxv, u'xv'\}) = L_J(\{ux, u'x\})$ while $L_J(\{uxv', u'xv\}) = L_J(\{ux, u'x\})$.

(vi) By Lemma 32.2, $R_J(uxvxw) = R_J(xw)$ while $R_J(\{uxw, \bullet vx\}) = R_J(xw)$.

(vii) By Lemma 32.2, $R_J(\{uxv, u'xv'\}) = R_J(\{xv, xv'\})$ while $R_J(\{uxv', u'xv\}) = R_J(\{xv', xv\})$. □

We are now ready to define the notion of a context-free recombination system. This is a construct whereby we are given a starting set of sequences and a list of junction sequences (plugs). New strings may be formed by recombinations among the existing strands: if one of the given junctions sequences is present, recombinations are performed as defined in Section 1. Recombinations are context-free, i.e., they are not dependent on the context in which the junctions sequences appear. The language of the system is defined as the set of all strands

that can thus be obtained by repeated recombinations starting from the initial set.

Definition 32.5 *A context-free recombination system is a triple*

$$R = (\Sigma, J, A)$$

where Σ is an alphabet and $J \subseteq \Sigma^+$ is a set of plugs, while $A \subseteq \Sigma^+ \cup \Sigma^\bullet$ is the set of axioms of the system.

Given a recombination system R, for sets $S, S' \subseteq \Sigma^+ \cup \Sigma^\bullet$ we say that S derives S' and we write $S \Rightarrow_R S'$ iff there exists $x \in J$ such that one of the following situations holds:

(i) $\exists uxv, u'xv' \in S$ such that $uxv + u'xv' \Rightarrow uxv' + u'xv$ and $S' = S \cup \{uxv', u'xv\}$,

(ii) $\exists uxvxw \in S$ such that $uxvxw \Rightarrow uxw + \bullet vx$ and $S' = S \cup \{uxw, \bullet vx\}$,

(iii) $\exists uxw, \bullet vx \in S$ such that $uxw + \bullet vx \Rightarrow uxvxw$ and $S' = S \cup \{uxvxw\}$,

(iv) $\exists \bullet uxv, \bullet u'xv' \in S$ such that $\bullet uxv + \bullet u'xv' \Rightarrow \bullet\, uxv'u'xv$ and $S' = S \cup \{\bullet uxv'u'xv\}$,

(v) $\exists \bullet\, uxv'u'xv \in S$ such that $\bullet uxv'u'xv \Rightarrow \bullet\, uxv + \bullet u'xv'$ and $S' = S \cup \{\bullet uxv, \bullet u'xv'\}$.

Definition 32.6 *The language generated by a context-free recombination system R is defined as*

$$L(R) = \{w \in \Sigma^* \cup \Sigma^\bullet \mid A \Rightarrow_R^* S, w \in S\}$$

Lemma 32.3 *For any context-free recombination systems $R = (\Sigma, J, A)$ there exists a context-free recombination system $R' = (\Sigma, J', A)$ such that J' is subword-free and $L(R) = L(R')$.*

Proof. Let $J' = \{w \in J \mid \nexists u \in J, u \neq w$ such that $w = x'ux'', x', x'' \in \Sigma^*\}$. As $J' \subseteq J$, obviously $L(R') \subseteq L(R)$ as any recombination sequence using a plug in J' is a recombination according to R as well.

Conversely, let $x \in J \setminus J'$. There exist $y \in J', x', x'' \in \Sigma^*, y \neq x$ such that $x = x'yx''$.

Then,

(i) $uxv + u'xv' = ux'yx''v + u'x'yx''v' \Rightarrow_{R'} ux'yx''v' + u'x'yx''v = uxv' + u'xv$,

(ii) $uxvxw = ux'yx''vx'yx''w \Rightarrow_{R'} ux'yx''w + \bullet x''vx'y = uxw + \bullet vx'yx''$ $= uxw + \bullet vx$,

(iii) $uxw + \bullet vx = ux'yx''w + \bullet vx'yx'' \Rightarrow_{R'} ux'yx''vx'yx''w = uxvxw$,

(iv) $\bullet uxv + \bullet u'xv' = \bullet ux'yx''v + \bullet u'x'yx''v' \Rightarrow_{R'} \bullet\, ux'yx''v'u'x'yx''v =$ $uxv'u'xv$,

(v) $\bullet uxv'u'xv = \bullet ux'yx''v'u'x'yx''v \Rightarrow_{R'} \bullet ux'yx''v + \bullet yx''v'u'x' = \bullet uxv + \bullet u'x'yx''v' = \bullet uxv + \bullet u'xv'$.

Consequently, any derivation step in R can be simulated by a derivation step in R', i.e., $L(R) \subseteq L(R')$. □

As a consequence of the preceding lemma we may assume, without loss of generality, that a context-free recombination system has a subword-free set J of plugs. The following Lemma will aid in the proof of our main result.

Lemma 32.4 *Let $R = (\Sigma, J, A)$ be a context-free recombination system. Let $ux = x'u'$ start and end with plugs $x', x \in J$ where $u, u' \neq \lambda$. If ux satisfies $E_J(ux) \subseteq E_J(A)$, then there exist $\alpha, \beta \in \Sigma^*$ such that $\alpha ux\beta$ or $\bullet\alpha ux\beta$ is in $L(R)$.*

Proof. Induction on k, the number of occurrences of plugs in $ux = x'u'$.

Base case: $k = 2$. Then ux is elementary, $ux \in E_J(ux)$, therefore there exists an axiom $a \in A$ such that $ux \in E_J(a)$. If $a \in \Sigma^*$ is a linear word, then $a = \alpha ux\beta \in L(R)$. If $\bullet a \in \Sigma^\bullet$ is circular, as $\bullet a$ contains at least one plug, $\bullet a + \bullet a + \bullet a = \bullet aaa \in L(R)$. By definition $ux \in E_J(aaa)$ which implies $aaa = \alpha ux\beta$ and $\bullet aaa = \bullet\alpha ux\beta \in L(R)$.

Inductive step: Let $ux = x'u' = vyz$ where y is any plug in the middle, e.g. the second last plug.

Words vy and yz satisfy the conditions of the claim as they contain fewer than k plugs, so we may apply the inductive hypothesis to both of them. Consequently, $\alpha vy\beta$ or $\bullet\alpha vy\beta$ is in $L(R)$ and $\gamma yz\delta$ or $\bullet\gamma yz\delta$ is in $L(R)$.

One more recombination yields:

$\alpha vy\beta + \gamma yz\delta \Rightarrow \alpha vyz\delta + \gamma y\beta$ or

$\alpha vy\beta + \bullet\gamma yz\delta \Rightarrow \alpha vyz\delta\gamma y\beta$ or

$\bullet\alpha vy\beta + \gamma yz\delta \Rightarrow \gamma y\beta\alpha vyz\delta$ or

$\bullet\alpha vy\beta + \bullet\gamma yz\delta \Rightarrow \bullet\, \alpha vyz\delta\gamma y\beta$, respectively.

Note that in each of the four possible cases the result contains a linear or circular word in $L(R)$ that has $vyz = ux = x'u'$ as a subword, as required. □

The theorem below shows that a context-free recombination system characterized by a set of plugs J and a set of axioms A has the following property. Any cable that consists of elementary cables plugged together one after the other and that is either linear or circular can be obtained from the axioms using cross-plugging. Conversely, no other types of cables can be obtained from the axioms.

Theorem 32.1 *Let $R = (\Sigma, J, A)$ be a context-free recombination system. Then $L(R) = X$ where*

$X = \{w \in \Sigma^* \cup \Sigma^\bullet |$ *either $E_J(w) = L_J(w) = R_J(w) = \emptyset$ and $w \in A$, or $E_J(w), L_J(w), R_J(w)$ are not all empty and $E_J(w) \subseteq E_J(A)$, $L_J(w) \subseteq L_J(A)$, $L_J(w) \subseteq L_J(A)$}$\}$.

Proof. "$X \subseteq L(R)$"

Let $w \in X$. If $E_J(w) = L_J(w) = R_J(w) = \emptyset$ and $w \in A$, then $w \in L(R)$.

Assume now that $w \in \Sigma^*$ is a linear word such that $E_J(w), L_J(w), R_J(w)$ are not all empty and $E_J(w) \subseteq E_J(A)$, $L_J(w) \subseteq L_J(A)$, $R_J(w) \subseteq R_J(A)$.

If w contains only one plug $x \in J$ then $w = uxv$ and $L_J(w) = ux$, $R_J(w) = xv$. As $L_J(w) \subseteq L_J(A)$, there exists an axiom $a_1 \in A \cap \Sigma^*$ such that $a_1 = uxt$. As $R_J(w) \subseteq R_J(A)$, there exists an axiom $a_2 \in A \cap \Sigma^*$ such that $a_2 = sxv$.

We have $a_1 + a_2 = uxt + sxv \Rightarrow uxv + sxt$, which implies that $uxv = w \in L(R)$.

If w contains more than one plug, then $w = u\gamma v$ where $\gamma = xl = rx'$, $x, x' \in J$, $ux = L_J(w) \subseteq L_J(A)$ and $x'v = R_J(w) \subseteq R_J(A)$. Consequently, there exist axioms $a_1, a_2 \in A \cap \Sigma^*$ such that $a_1 = uxt$, $a_2 = sx'v$.

By Lemma 32.4, there exist $\alpha, \beta \in \Sigma^*$ such that $\alpha\gamma\beta$ or $\bullet\alpha\gamma\beta$ is in $L(R)$.

We can then recombine

$uxt + \alpha\gamma\beta + sx'v = uxt + \alpha xl\beta + sx'v \Rightarrow uxl\beta + \alpha xt + sx'v =$
$urx'\beta + \alpha xt + sx'v \Rightarrow urx'v + \alpha xt + sx'\beta = u\gamma v + \alpha xt + sx'\beta$

or, in the circular case,

$uxt + \bullet\alpha\gamma\beta + sx'v = uxt + \bullet\alpha xl\beta + sx'v \Rightarrow uxl\beta\alpha xt + sx'v =$
$urx'\beta\alpha xt + sx'v \Rightarrow urx'v + sx'\beta\alpha xt = u\gamma v + sx'\beta\alpha xt.$

In both cases, $u\gamma v = w \in L(R)$.

If $\bullet w \in \Sigma^\bullet$ is a circular word that contains at least one plug ($E_J(\bullet w) \neq \emptyset$) then $\bullet w = \bullet ux$ for some $x \in J$. The word xux satisfies the conditions of Lemma 32.4 therefore $\alpha xux\beta$ or $\bullet\alpha xux\beta$ is in $L(R)$.

Then we have either $\alpha xux\beta \Rightarrow \bullet ux + \alpha x\beta$ or $\bullet\alpha xux\beta \Rightarrow \bullet ux + \bullet\alpha x\beta$ which both imply that $\bullet ux = \bullet w \in L(R)$.

For the converse inclusion "$L(R) \subseteq X$" note that if $w \in L(R)$, $E_J(w) = \emptyset$, $L_J(w) = \emptyset$, $R_J(w) = \emptyset$, and $w \in A$ then by definition $w \in X$. Otherwise, if some words in $L(R)$ belong to X, the result of their recombinations have the necessary properties that ensure their belonging to X by Proposition 32.1. Therefore, $L(R) \subseteq X$. □

The theorem above leads to the conclusion of our paper after we show that the language X is regular, being accepted by a finite automaton.

Definition 32.7 *Given a finite automaton $\mathcal{A}$, the circular language accepted by $\mathcal{A}$, denoted by $L(\mathcal{A})^\bullet$, is defined as the set of all words $\bullet w$ such that $\mathcal{A}$ has a cycle labelled by w.*

The circular/linear language accepted by a finite automaton $\mathcal{A}$ is defined as $L(\mathcal{A}) \cup L(\mathcal{A})^\bullet$, where $L(\mathcal{A})$ is the linear language accepted by the automaton $\mathcal{A}$ defined in the usual way.

Definition 32.8 *A circular/linear language $L \subseteq \Sigma^* \cup \Sigma^\bullet$ is called regular if there exists a finite automaton $\mathcal{A}$ such that it accepts the circular and linear parts of L, i.e. that accepts $L \cap \Sigma^*$ and $L \cap \Sigma^\bullet$.*

Theorem 32.2 *Let $J \subseteq \Sigma^*$ be a set of plugs and let $A \subseteq \Sigma^* \cup \Sigma^\bullet$ be a finite axiom set. Then X defined as in Theorem 32.1 equals the circular/linear language accepted by a finite automaton $\mathcal{A}$ and is therefore regular.*

Proof. If the set J is not finite, then we should start by eliminating plugs that do not appear in any elementary cables of A. As the axiom set A is finite, the number of elementary cables is finite, and the set of (useful) plugs is finite as well. Consequently, we can assume, without loss of generality, that the set J is finite.

Let $\mathcal{A} = (S, \Sigma, \delta, s_0, s_f)$ be a finite automaton constructed as follows.

The set of states is

$$S = \{s_x | \ x \in J\} \cup \{s_0, s_f\}$$

and the transition relation δ is defined as follows:

(i) $\delta(s_x, u) = \{s_y | \text{ for each } e = xu = vy \in E_J(A)\}$,

(ii) For each $ux \in L_J(A)$ we have the transition $\delta(s_0, ux) = s_x$,

(iii) For each $xu \in R_J(A)$ we have $\delta(s_x, u) = s_f$.

From the above construction and Theorem 32.1 one can prove that $L(\mathcal{A}) = X$. □

Note that Definition 32.7 of the acceptance of a circular language by an finite automaton and Definition 32.8 of regularity of a circular/linear language overlap but do not coincide with existing definitions. We will therefore conclude with a comparison between various definitions and an argument in favour of our choice. We can define circular languages accepted by automata in two more ways.

Definition 32.9 *Given a finite automaton $\mathcal{A}$, the circular language accepted by $\mathcal{A}$, denoted by $L(\mathcal{A})_1^\bullet$, is defined as the set of all words $\bullet w$ such that $\mathcal{A}$ has a cycle labelled by w that contains at least one final state.*

Definition 32.10 *Given a finite automaton $\mathcal{A}$, the circular language accepted by $\mathcal{A}$, denoted by $L(\mathcal{A})_2^\bullet$ is defined as the set of all words $\bullet w$ such that $w = uv$ and $vu \in L(\mathcal{A})$.*

The circular languages accepted by finite automata using Definition 32.10 coincide with the regular circular languages as introduced by Head in [3], while acceptance of circular languages as in Definition 32.9 will be proven to coincide with the definition in [11]. The following result holds.

Proposition 32.2 *(i) The family of circular languages accepted by finite automata under Definition 32.7 is strictly included in the family of circular languages accepted by finite automata under Definition 32.9. (ii) The family of circular languages accepted by finite automata under Definition 32.9 is strictly included in the family of circular languages accepted by finite automata under Definition 32.10.*

Proof. (i) If $L \subseteq \Sigma^\bullet$ is a language accepted by an automaton A under Definition 32.7, then there exists an automaton B such that L is accepted by B under Definition 32.9. Indeed, one can construct an automaton B from A by making every state a final state. Then every cycle contains a final state.

The inclusion is strict as shown by the two state automaton with transitions $\delta(q, a) = p, \delta(p, a) = q, \delta(p, b) = p$ where q is the only final state. All words $ab^n a$ are accepted, which implies a loop labeled by bs only. This implies that in the sense of Definition 32.7 words $\bullet b^n$ for some n are accepted although they are not accepted by the automaton in the sense of Definition 32.9.

(ii) If $L \subseteq \Sigma^\bullet$ is a language accepted by an automaton B under Definition 32.9 then there exists an automaton C such that L is accepted by C under Definition 32.10. Indeed, we can construct an automaton C from B as follows. For every final state q of B we can construct a copy $B(q)$ of B where only the initial and final states have been changed: In $B(q)$ state q is the only initial and the only final state. Then we take the union of $B(q)$ for all final states q of B. The resulting automaton C accepts the same language in the sense of Definition 32.10 as the original B accepted in the sense of Definition 32.9.

Indeed, let q be an arbitrary final state of B. As q is the only final and initial state of $B(q)$ then the new machine $B(q)$ accepts in the sense of Definition 32.10 all circular words that label a loop that goes through state q. (If w is a label of such a loop, that starts and ends in state p, then $w = uv$ where u labels a path from p to q and v labels a path from q to p. Then vu labels a path that starts and ends in q, i.e., w is accepted by $B(q)$ i.e. of C in the sense of Definition 32.10. Conversely, if w is accepted by C in the sense of Definition 32.10 this means it is accepted by some $B(q)$ in the sense of Definition 32.10 then $w = uv$ where vu labels a path from q to q. That means vu labels a loop that goes through q, and then also $w = uv$ labels such a loop.)

The inclusion is strict as an automaton accepting languages using Definition 32.9 does not accept any finite nonempty languages because any loop can be repeated arbitrarily many times. Automata accepting languages using Definition 32.10 accept finite languages. □

As mentioned above, Definition 32.9 is equivalent to the definition in [11] (the circular language accepted by an automaton is the set of all words that label a loop containing at least one initial and one final state). Indeed, let A be an NFA that accepts a circular language L in the sense of [11], i.e., a word is

accepted if it labels a cycle that contains both initial and final states. To accept the same language in the sense of Definition 32.9 we make two identical copies of A, say A and A', and we make λ-transitions (which we can eliminate later) between the copies as follows: For every initial state i we have a λ-transition from i to i', and for every final state f we make λ -transitions from f' to a new state f'', and from f'' to f. All states f'' are final states, and no other states are final. A cycle that contains a final state must go through some f'', which means it goes from A' to A. Consequently it must go from A to A' as well, i.e. the cycle contains some initial state i also, i.e. the word labels some cycle of the original machine that contains both i and f.

To summarize, Definition 32.7 of circular languages accepted by automata is strictly more restrictive than Definition 32.9, which is in turn strictly more restrictive that Definition 32.10. Definition 32.9 is equivalent to the definition in [11], and Definition 32.10 is equivalent to the definition of regular circular languages proposed in [3]. Note that, as shown in [11], the family of languages accepted by Definition 32.10, which are in addition closed under repetition (if w^n is in the language whenever w is in the language), equals the family of circular languages accepted by automata in the sense of Definition 32.9.

Our preference for Definition 32.7 was motivated by the fact that, under this definition, in Theorem 32.2 we can use the same automaton to accept both the linear and circular components of the language. This makes our definition more natural and Theorem 32.2 stronger that its counterpart following from [4]. (Theorem 5.2, Chapter 5, [4], implies that the result of combined splicing starting from a finite set is regular by showing that the linear and circular components are each regular – using Definition 32.10 – but possibly accepted by different automata.)

However, Theorem 32.2 and Proposition 32.2 show that the language of a context-free recombination system with a finite axiom set is regular under any of the existing definitions.

References

[1] Csuhaj-Varjú, E.; R. Freund; L. Kari & Gh. Păun (1996), DNA computing based on splicing: universality results, in L. Hunter & T. Klein, eds., *Proceedings of the 1st Pacific Symposium on Biocomputing*: 179–190. World Scientific, Singapore.

[2] Head, T. (1987), Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors, *Bulletin of Mathematical Biology*, 49: 737–759.

[3] Head, T. (1991), Splicing schemes and DNA, in G. Rozenberg & A. Salomaa, eds., *Lindenmayer Systems*: 371–383. Springer, Berlin.

[4] Head, T.; Gh. Păun & D. Pixton (1997), Language theory and molecular genetics, in G. Rozenberg & A. Salomaa, eds., *Handbook of Formal Languages*, II: 295–358. Springer, Berlin.

[5] Kari, L.; J. Kari & L. Landweber (1999), Reversible molecular computation in ciliates, in J. Karhumäki; H. Maurer; Gh. Păun & G. Rozenberg, eds., *Jewels are Forever*: 353-363. Springer, Berlin.

[6] Landweber, L.F. & L. Kari (1998), The evolution of cellular computing: nature's solution to a computational problem, in *Proceedings of the 4th DIMACS Meeting on DNA Based Computers*: 3–15, Philadephia. Also in *Biosystems*.

[7] Landweber, L.F. & L. Kari (1999), Universal molecular computation in ciliates, in L. Landweber & E. Winfree, eds., *Evolution as Computation*, Springer, Berlin.

[8] Meyer, E. & S. Duharcourt (1996), Epigenetic Programming of Developmental Genome Rearrangements in Ciliates, *Cell*, 87: 9–12.

[9] Păun, Gh. (1995), On the power of the splicing operation, *International Journal of Computer Mathematics*, 59: 27–35.

[10] Pixton, D. (1995), Linear and circular splicing systems, in *Proceedings of the First International Symposium on Intelligence in Neural and Biological Systems*: 181–188. IEEE Computer Society Press, Los Alamos, Ca.

[11] Pixton, D. (1996), Regularity of splicing languages, *Discrete Applied Mathematics*, 69.1-2: 99–122.

[12] Pixton, D., Splicing in abstract families of languages, ms. in preparation.

[13] Prescott, D.M. (1994), The DNA of ciliated protozoa, *Microbiological Reviews*, 58.2: 233–267.

[14] Salomaa, A. (1973), *Formal Languages*. Academic Press, New York.

[15] Siromoney, R.; K.G. Subramanian & V. Rajkumar Dare (1992), Circular DNA and splicing systems, in *Parallel Image Analysis*: 260–273. Springer, Berlin.

[16] Yokomori, T.; S. Kobayashi & C. Ferretti (1997), Circular Splicing Systems and DNA Computability, in *Proceedings of the IEEE International Conference on Evolutionary Computation'97*: 219-224.