ELSEVIER Discrete Applied Mathematics 87 (1998) 139–147

DISCRETE APPLIED MATHEMATICS

# Splicing representations of strictly locally testable languages

Tom Head[*]

*Department of Mathematical Sciences, State University of New York-Binghamton, Binghamton, NY 13902-6000, USA*

Received 26 March 1997; received in revised form 16 February 1998; accepted 9 March 1998

**Abstract**

The relationship between the family *SH* of simple splicing languages, which was recently introduced by Mateescu et al. and the family *SLT* of strictly locally testable languages is clarified by establishing an ascending hierarchy of families $\{S_iH\colon i \geqslant -1\}$ of splicing languages for which $SH = S_1H$ and the union of the families is the family *SLT*. A procedure is given which, for an arbitrary regular language $L$, determines whether $L$ is in *SLT* and, when $L$ is in *SLT*, specifies constructively the smallest family in the hierarchy to which $L$ belongs. Examples are given of sets of restriction enzymes for which the action on DNA molecules is naturally represented by splicing systems of the types discussed. © 1998 Elsevier Science B.V. All rights reserved.

*Keywords*: Splicing systems; *H* systems; DNA computing; Local testability; Regular languages; Restriction enzymes

## 1. Introduction

The splicing system concept was introduced in [6] as a formal device for the generation of languages and as a formal model of specific forms of DNA recombination. The definition was created in close imitation of the recombinant behavior of double-stranded DNA molecules in the presence of specific restriction enzymes and a ligase. See [9, 10] for an exposition of this new field of splicing theory and for an explanation of its origin in molecular biology. The use of splicing concepts as a basis of universal computing is being investigated extensively now as exemplified in [4, 5, 10, 14–16, 19]. Our interest continues, since [6, 7] , to center on models of the generative dynamics of molecular soups, as exemplified by [11].

We are concerned here with subclasses of the splicing systems that in [6] were called *null context splicing systems* (called here NCH systems). The present note was stimulated by the recent paper [12] which introduced a subclass of the null context splicing systems called *simple H systems*. It was observed in [6] that the family of

[*] E-mail: tom@math.binghamton.edu.

languages generated by null context splicing systems coincides with the family *SLT* of *strictly locally testable languages* which is a subfamily of the regular languages that was introduced in [13]. Languages generated by splicing systems in the original sense of [6, 7] are always regular [1, 2, 17, 18].

In Section 2 the required formal definitions are given. This includes the definition of a sequence of language families $\{S_iH\colon i \geqslant -1\}$ beginning with the finite languages, having $S_1H = SH$, and having the class of all *SLT* languages as union.

In Section 3 we study the problem of generating languages with *NCH* systems. We provide a procedure which determines, for any regular language $L$, whether $L$ is in *SLT*. When $L$ is in *SLT* the procedure continues and calculates the least integer $k \geqslant -1$ for which $L$ is in $S_kH$.

In Section 4 examples of sets $R$ of restriction enzymes are given for which, for each such $R$, the set of DNA molecules generated (by $R$ and a ligase) from a finite initial set of DNA molecules is predicted to be the molecular embodiment of a language of type $S_kH$ for specified $k$.

## 2. Definitions and examples

Let $A$ be a fixed finite set to be used as an alphabet for the free monoid $A^*$ that consists of all strings of symbols in $A$, including the null string 1. The languages we discuss will be subsets of $A^*$.

**Definition.** A *null context splicing system* (*NCH* system) $G = (A, I, R)$ with alphabet $A$ consists of a finite subset $I$ of $A^*$ called the *initial language* and a finite subset $R$ of $A^*$ called the *rule set*. The *language* $L(G) = L(A, I, R)$ *generated by* $G$ is the smallest language $L$ in $A^*$ that contains $I$ and has the property that whenever strings $wrx$ and $yrz$ are in $L$, with $r$ in $R$, the strings $wrz$ and $yrx$ are also in $L$. A language $L$ is called a *null context splicing language* (*NCH* language) if there exists a null context splicing system $G = (A, I, R)$ for which $L = L(G)$.

The concept of a constant, as introduced by Schutzenberger [20], is a valuable conceptual tool for splicing theory: A string $c$ in $A^*$ is a *constant for a language* $L$ over an alphabet $A$ if, whenever $wcx$ and $ycz$ are in $L$, both $wcz$ and $ycx$ are also in $L$. Note that when $c$ is a constant for $L$, so are each of the strings in $A^*cA^*$. Recall that a string $y$ is a *factor of a string* $w$ if $w = xyz$ for some $x, z$ in $A^*$ and that $y$ is a *factor of a language* $L$ if $y$ is a factor of some string in $L$. A string $c$ that is not a factor of a language $L$ vacuously satisfies the definition of a constant for $L$. Concern with constants is often restricted to those that are also factors. Our main concern will be with the yet more restricted class of constants that we call primes: We say that a constant factor $p$ of a language is a *prime constant factor* (briefly, a *prime*) for $L$ if $p$ is a constant factor for $L$ and for any factorization $p = xyz$, with $xz$ not null, $y$ is not a constant. Note that, when the null string 1 is a constant for a language $L$, it is necessarily the unique prime for $L$. Each constant factor for a language $L$ has one or more factors that are primes for $L$. Each rule of an *NCH* system $G=(A,I,R)$ is necessarily a constant for the language $L(G)$.

**Definition.** Let $k$ be an integer $\geqslant -1$. An $S_k$ *splicing system* ($S_kH$ system) is an *NCH* system $G=(A,I,R)$ for which, for each string $r$ in $R$, length $r \leqslant k$. A language $L$ is called an $S_k$ *splicing language* ($S_kH$ language), if there exists an $S_kH$ system $G=(A,I,R)$ for which $L=L(G)$. The family of $S_k$ splicing languages is denoted by $S_kH$.

Every $S_kH$ language is an *NCH* language and every *NCH* language is an $S_kH$ language for some positive integer $k$. *Thus the union of the families* $S_kH$, $k \geqslant -1$, *is the family of NCH languages.*

The $S_{-1}H$ systems are precisely the systems for which $R$ is empty. Consequently, for an $S_{-1}H$ system $G=(A,I,R)$ we have $L(G)=I$. Thus, the $S_{-1}H$ languages are precisely the finite languages. It is therefore decidable, whether a given regular language $L$ is in $S_{-1}H$; and, when it is in $S_{-1}H$, we have the representation $L=L(G)$ where $G=(A,L,R)$ with $R$ empty.

The $S_0H$ systems are those for which $R$ is either empty or $R=\{1\}$. An $S_0H$ system that generates an infinite language must have the form $G=(A,I,\{1\})$ with $I$ not empty. Let $B=\{b$ in $A$: $b$ occurs in some string in $I\}$. It is then easily confirmed, using the fact that 1 is a constant, that $L(G)=B^*$. Conversely, for any subset $B$ of $A$, $L(G)=B^*$ for $G=(A,B,\{1\})$. Thus, the $S_0H$ languages are precisely the finite languages together with the languages $B^*$, where $B$ is a subset of $A$. It is therefore decidable whether a regular language is in $S_0H$; and, when it is in $S_0H$, we obtain a representation $L=L(G)$ where either $G=(A,L,R)$ with $R$ empty or $G=(A,B,\{1\})$. Note that for any *NCH* system $G=(A,I,R)$, if $R$ contains the null string then, for $G'=(A,I,\{1\})$, we have $L(G)=L(G')$ is an $S_0H$ language.

The $S_1H$ systems and the $S_1H$ languages are precisely the simple $H$ systems (*SH* systems) and simple $H$ languages (*SH* languages) as defined in [12].

**Example.** For this example let $A=\{a,b\}$. For each $k \geqslant 1$, let $L_k = a^k(b^ka^{(k-1)}b^ka^k)^*$. Observe that $L_k=L(A,I,R)$, where $I=\{a^kb^ka^{(k-1)}b^ka^k\}$ and $R=\{a^k\}$. Thus, $L_k$ is a member of the family $S_kH$. Careful consideration reveals that no string of length $<k$ is a constant factor of $L_k$ and therefore $L_k$ does not lie in $S_jH$ for any $j<k$.

The examples above establish the assertion that *the sequence* $S_kH$, $k \geqslant -1$, *is a strictly ascending infinite hierarchy of language families* for alphabets of two or more symbols. For a singleton alphabet the *NCH* languages are merely the finite and cofinite sets.

**Definition** ([13] modified using [3]). A language $L$ is *strictly locally testable* (*SLT*) if there is a positive integer $k$ for which every factor of $L$ of length $k$ is constant.

That the family of *SLT* languages coincides with the family of *NCH* languages is a consequence of the more general Theorem given in [6]. However, the proof of the Theorem of Section 3 subsumes a proof of this equivalence. *Thus the union of the families* $S_k$, $k \geqslant -1$, *is the family of SLT languages.*

## 3. Splicing and strict local testability

Our objective here is to determine which languages can be represented in the form $L = L(A,I,R)$ and to construct concise representations whenever representations exist. In Section 2 we have already treated the languages $L$ that admit representations $L = L(A,I,R)$ in which either $R$ is empty or $R$ contains the null string 1.

In the first lemma we note that in constructing rule sets to represent a language $L$, we need only look among the prime constants factors of $L$.

**Lemma 1.** *Let* $L = L(A,I,R)$. *For each* $r$ *in* $R$ *that is a factor of* $L$, *let* $c(r)$ *be a prime constant factor of* $r$. *Then* $L = (A,I,R')$ *where* $R' = \{c(r)\text{: } r \text{ in } R\}$.

The next two lemmas tell us not to attempt a representation $L = L(A,I,R)$ unless $L$ is *SLT*.

**Lemma 2.** *Let* $L = L(A,I,R)$ *and let* $k$ *exceed by one the length of the longest string in* $I$. *Then every factor of* $L$ *of length* $k$ *possesses a string in* $R$ *as a factor. Consequently, every string of length* $k$ *in* $A^*$ *is a constant for* $L$.

**Proof.** The set $S$ of all strings in $L$ for which every factor of length $k$ possess a string in $R$ as a factor has the two properties: (1) $S$ contains $I$ since no string in $I$ contains a factor of length $k$; and (2) $S$ is closed under splicing by rules in $R$. Consequently $S = L$ directly from the definition of $L = L(A,I,R)$. □

**Lemma 3.** *For every* $L = L(A,I,R)$, $L$ *is SLT.*

**Proof.** In [3] the *SLT* languages are characterized as those languages for which there is a positive integer $k$ for which every string in $A^*$ of length $k$ is a constant. We have taken this characterization as our definition. Lemma 2 concludes the proof. □

The next three lemmas provide, for each language $L$, intrinsically associated sets $I(L)$ and $R(L)$ for which if $L$ has *any* representation of the desired form at all, then $L = L(A, I(L), R(L))$. In fact, we shall see that $L$ has a representation of the desired form if and only if both $I(L)$ and $R(L)$ are finite. As will be observed in the theorem below, when $L$ is regular $R(L)$ is regular and, when $R(L)$ is finite, $I(L)$ is regular. For $L$ regular, the regularity of $R(L)$ and the conditional regularity of $I(L)$ allow us to decide whether both sets are finite, and consequently, whether $L$ is representable in the desired form.

**Definition.** For each finite language $L$, let $R(L)$ be empty. For each infinite language $L$ let $R(L)$ be the set of all prime constant factors of $L$.

**Lemma 4.** *For $L = L(A,I,R)$, $R(L)$ is finite and $L = L(A,I,R(L))$.*

**Proof.** These are consequences of the definition of $R(L)$ and of Lemma 1. □

**Definition.** Let $L$ be a language and let $R'$ be a set of strings that are constant relative to $L$. For $R'$ empty we let $I(R') = L$. When $R'$ contains the null string 1 there is a subset $B$ of $A$ for which $L = B^*$ (see Section 2) and we let $I(R') = B$. For $R'$ not empty and 1 not in $R'$, we let $I(R') = L \backslash (\bigcup \{A^*pA^*pA^*pA^*$: $p$ in $R'$).

**Lemma 5.** *Let $L = L(A,I,R)$ and let $R'$ be a finite subset of $R(L)$. Then there exists a finite subset $I'$ of $L$ for which $L = L(A,I',R')$ if and only if $I(R')$ is finite. When $I(R')$ is finite $L = L(A,I(R'),R')$.*

**Proof.** From the discussions of families $S_{-1}H$ and $S_0H$ in Section 2, we know this lemma holds when either $R'$ is empty or $R'$ (hence also $R$) contains the null string. We proceed under the additional hypothesis: $R'$ not empty and 1 not in $R'$.

Suppose first that $I(R')$ is finite. Since $I(R')$ is contained in $L$ and $R'$ is contained in $R(L)$, $L(A,I(R'),R')$ is contained in $L$. If $L$ is not contained in $L(A,I(R'),R')$ then there is a shortest string $s$ in $L \backslash L(A,I(R'),R')$ and it must have the form $s = wpxpypz$ for some $w,x,y,z$ in $A^*$ and some $p$ in $R'$. Using the underscored occurrences of $p$ in $s = w\underline{p}xpypz$ and $s = wpx\underline{p}ypz$ we conclude $u = wpypz$ is in $L$. Likewise from $s = wpx\underline{p}ypz$ and $s = wpxpy\underline{p}z$ we conclude $v = wpxpz$ is in $L$. Since $u$ and $v$ are in $L$ and are shorter than $s$, we conclude $u$ and $v$ are in $L(A,I(R'),R')$. Using the underscored occurrences of $p$ in $v = wpx\underline{p}z$ and $u = w\underline{p}ypz$ we conclude the contradiction $s = wpxpypz$ in $L(A,I(R'),R')$. Thus, when $I(R')$ is finite $L = L(A,I(R'),R')$ as required.

Suppose now that $L = L(A,I',R')$ for a finite subset $I'$ of $L$. Then by the three part definition of $I(R')$ and Lemma 2, $I(R')$ must be finite. □

**Definition.** For each language $L$ let $I(L) = I(R(L))$.

**Lemma 6.** *For $L = L(A,I,R)$, the set $I(L)$ is finite and $L = L(A,I(L),R(L))$.*

**Proof.** From Lemma 4, $L = L(A,I,R(L))$. Apply Lemma 5 with $I' = I$ and $R' = R(L)$ to obtain the finiteness of $I(L) = I(R(L))$ and the equality $L = L(A,I(L),R(L))$. □

Finally, we note that it is precisely the *SLT* languages that have representations of the desired form.

**Lemma 7.** *Let $L$ be SLT. Then $L = L(A,I,A^k)$ where $k$ is a positive integer for which every string in $A^*$ of length $k$ is constant for $L$ and $I = I(A^k) = L \backslash (\bigcup \{A^*pA^*pA^*p$ $A^*$: $p$ in $A^k$).*

**Proof.** The definition of $I$ insures the finiteness of $I$. One may then confirm directly that $L = L(A, I, A^k)$. □

**Proposition.** *A language $L$ has a representation of the form $L = L(A, I, R)$ if and only if $L$ is SLT. Thus NCH = SLT.*

**Proof.** This is the combination of Lemmas 3 and 7. □

**Theorem.** *For each regular language $L$:*
(1) *$R(L)$ is regular and the finiteness of $R(L)$ can be decided;*
(2) *when $R(L)$ is finite, $I(L)$ is regular and the finiteness of $I(L)$ can be decided;*
(3) *$L$ is NCH if and only if both $I(L)$ and $R(L)$ are finite, in which case $L = L(A, I(L), R(L))$;*
(4) *representability of $L$ in the form $L = L(A, I, R)$ can be decided; and*
(5) *if $L$ is SLT then there is a least integer $k \geqslant -1$ for which $L$ is in $S_kH$ and this $k$ is computable.*

**Proof.** Let $L$ be a regular language and let $M = (A, S, \{q_0\}, F, E)$ be the trimmed minimal automaton recognizing $L$, where: $A$ is the alphabet, $S$ is the set of states, $q_0$ is the initial state, $F$ is the set of final states, $E$ is the set of edges with labels in $A$, and, by *trimmed*, we mean that all states not accessible from $q_0$ have been removed and all states from which $F$ is not accessible have been removed. The language recognized by an automaton $M$ will be denoted $L(M)$.

(1) For each $p$ in $S$ let $M(p) = (A, S, \{q'\}, \{p\}, E')$ where: $q'$ is a newly adjoined state and $E'$ results from adjoining to $E$, for each $q$ in $S$, an edge from $q'$ to $q$ labeled with the null string 1. For each $p$ in $S$, let $C(p) = L(M(p)) \backslash (\bigcup \{L(M(q))$: $q$ in $S \backslash \{p\}\})$. The set of all constant factors for $L$ is $C(L) = \bigcup \{C(p)$: $p$ in $S\}$. The set $R(L)$ of all prime constant factors of $L$ is $R(L) = C(L) \backslash (A^+C(L)A^* \cup A^*C(L)A^+)$. Hence $R(L)$ is regular we can decide whether it is finite.

(2) If $R(L)$ is finite, then $I(L)$ has one of the three forms: $L$, $B$, or $L \backslash \bigcup \{A^*cA^*cA^*cA^*$: $c$ in $R(L)\}$. In each of these three cases, $I(L)$ is regular and we can decide whether $I(L)$ is finite.

(3 and 4) These statements follow from (1), (2) and the Proposition.

(5) Let $L$ be *SLT*. Then $L = L(A, I(L), R(L))$ by Lemmas 7 and 8. We may now apply Lemma 5 to any desired subset $R'$ of $R(L)$. The set $I(R')$ is regular and we can therefore decide whether it is finite. If $I(R')$ is finite then $L = L(A, I(R'), R')$, otherwise there is no $I'$ for which $L = L(A, I', R')$. By applying this technique successively to the subsets $R_i = \{c$ in $R(L)$: length $c \leqslant i\}$ for each $i = -1, 0, 1, 2, \ldots$, we find the least integer $k \geqslant -1$ for which $L$ is in $S_kH$. □

**Remark.** Once a desired $R'$ is obtained, we may then shrink $I(R')$ to any subset $I$ of $I(R')$ for which $L(A, I, R') = L(A, I(R'), R')$ where this latter equality can be decided.

## 4. Molecular considerations

A reader who finds the sketch of biochemical principles included in this Section to be inadequate may consult [9, 10] or [6].

For brevity we will denote double-stranded DNA molecules such as

```
5'-GATCGGTCCAAGTC-3'
3'-CTAGCCAGGTTCAG-5'
```

in the shorter form: *gatcggtccaagtc* (always read in the $5'$ to $3'$ direction). It is not necessary to give the lower strand since *G* pairs only with *C*, *A* with *T*, *T* with *A*, and *C* with *G*. Since an actual molecule is not confined to a line, but rotates freely in space, one should take care to notice that an equally valid representation for the molecule above is: *gacttggaccgatc*. A segment of DNA for which the two representations are identical is said to be *palindromic* (or to have dyadic symmetry). We will not discuss single-stranded DNA molecules. All strings in this section are understood to be over the alphabet $D = \{a, c, g, t\}$ and to represent double-stranded DNA molecules. Notice that each symbol in this alphabet *D* represents a unit consisting of paired deoxyribonucleotides (bound together only by hydrogen bonds): $a = [A/T]$, $c = [C/G]$, $g = [G/C]$, and $t = [T/A]$.

Each of the two strands of a double-stranded DNA molecule is held together by strong covalent bonds. To cut such bonds special enzymes obtained from appropriate strains of bacteria are used. These are the *restriction enzymes* (endonucleases). Such an enzyme adheres at an occurrence of a DNA subsequence called its *site*. It then cuts one covalent bond in each strand at a specific location in the site. We will consider here only enzymes which have palindromic sites. As an example of our method of representing enzyme sites we have, for the enzyme AatII, the site *gacgt^c*. The assertion that *gacgt^c* is the site of AatII means that AatII cuts only at subsegments *gacgtc* of DNA molecules and that the covalent bond in the top strand that is cut is the one indicated by the caret between *t* and *c*. The cut in the bottom strand is also made between the *t* and *c* (recall that the site is palindromic). All this means that, when the enzyme AatII is present, each DNA molecule of the form:

```
5'...XXGACGTCXX...3'  may be cut into  5'...XXGACGT-3'  and  5'-CXX     ...3'
3'...XXCTGCAGXX...5'                   3'...XXC-5'           3'-TGCAGXX...5'.
```

In the AatII site *gacgtc* we refer to the segment *acgt* as the *crossing*, the *g* at the left as the *left context*, and the *c* at the right as the *right context*. The crossing yields the *sticky ends* (also called overhangs) ACGT-$3'$ and $3'$-TGCA protruding from the fragments produced by the cut. Sticky ends that match may join weakly by hydrogen bonding, but may be expected to drift apart unless covalent bonds are established in the two strands. If an enzyme called a *ligase* is present the required covalent bonds may be established to yield a firmly bonded fully double-stranded DNA molecule formed from the two fragments. In this way new *recombinant* DNA molecules may form which have not previously been present. This type of cut and paste activity provides the basis for the technology of gene splicing.

The sites for the three restriction enzymes AatII, AciI, and AseI are *gacgt^c*, *c^cgc*, *gg^cgcgcc*, and *at^taat*, respectively. Since these enzymes have the null string as left context and also as right context, it is entirely natural to model the recombination potential of any subset of this set of three enzymes with a null context splicing system (*NCH* system): For any initial set $I$ of DNA molecules, if a common buffer solution can be found in which these three enzymes and a ligase can function, then the set $L$ of well formed fully double-stranded DNA molecules that may potentially arise is modeled by $L(G)$, where $G=(D,I,R)$ with $R=\{gatc,\ catg,\ aatt\}$. This $G$ is an $S_4H$ system. For any choice of $I$, $L(G)$ is therefore, necessarily an $S_4H$ language, but for some choices of $I$ it could also be $S_kH$ for some $k<4$.

The very elementary structure of the *NCH* splicing languages is due to the fact that site occurrences for such restricted systems are preserved under splicing. In DNA terms, after a site has been used for recombination the site ‘is still there’. This need not be the case with sets of two or more enzymes, like AatII, that have non-null context. See [6] for relevant examples. Of course, if only a single enzyme such as AatII is used then an *NCH* may be used: For initial set $I$ of DNA molecules, the set of well-formed fully double stranded DNA molecules that may arise is modeled by $L(D,I,R)$ with $R=\{gacgtc\}$. As long as the enzymes of a set cannot interact in such a way as to destroy a site then an *NCH* representation may be possible. An example is given in the next paragraph.

The sites for the four restriction enzymes AatII, AciI, AscI, and AseI are *gacgt^c*, *c^cgc*, *gg^cgcgcc*, and *at^taat*. An examination of the four sticky ends of these sites shows that any recombination event must occur between two segments that were cut by the same enzyme. Thus when recombined the site is always restored. Thus we may use the *NCH* system $(D,I,R)$ with $R=\{gacgtc,\ ccgc,\ ggcgcgcc,\ attaat\}$ to model the generative capacity of this set of enzymes when combined with a ligase. The language generated is necessarily in $S_8H$, for any choice of $I$, but for some choices of $I$ it could also be in $S_kH$ for some $k<8$.

## Acknowledgements

I wish to thank Elizabeth Laun and the two anonymous referees for their unusually careful readings and thoughtful suggestions that have resulted in improvements that appear here. Partial support for this research through NSF CCR-9509831 and DARPA/NSF CCR-9725021 is gratefully acknowledged.

## References

[1] K. Culik II, T. Harju, The regularity of splicing systems and DNA, in: Proc. ICALP ’89, LNCS 372 (1989) 222–233.

[2] K. Culik II, T. Harju, Splicing semigroups of dominoes and DNA, Discrete Appl. Math. 31 (1991) 261–277.

[3] A. DeLuca, A. Restivo, A characterization of strictly locally testable languages and its application to subsemigroups of a free semigroup, Inform. and Control 44 (1980) 300–319.

[4] C. Ferretti, S. Kobayashi, DNA splicing systems and post systems, in: Proceedings of the Pacific Symposium on Biocomputing '96, World Scientific, Singapore, 1995, pp. 288–299.

[5] R. Freund, E. Csuhaj-Varju, F. Watchler, Test tube systems with cutting/recombination operations, in: Proceedings of the Pacific Symposium on Biocomputing '97, World Scientific, Singapore, 1996, pp. 163–174.

[6] T. Head, Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors, Bull. Math. Biol. 49 (1987) 737–759.

[7] T. Head, Splicing schemes and DNA, in: G. Rozenberg, A. Salomaa (Eds.), Lindenmayer Systems – Impacts on Theoretical Computer Science, Computer Graphics, and Developmental Biology, Springer, Berlin, pp. 371–383, 1992; also in: Nanobiology 1 (1992) 335–342.

[8] T. Head, Splicing Languages generated with one sided context, 84 (1998) 145–163.

[9] T. Head, Gh. Paun, D. Pixton, Language theory and molecular genetics: generative mechanisms suggested by DNA recombination, a chapter in: G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, vol. 2, Springer, New York, 1996, pp. 295–360.

[10] L. Kari, DNA computing: arrival of biological mathematics, Math. Intelligencer 19 (1997) 9–22.

[11] E. Laun, K.J. Reddy, Wet splicing systems, in: Proceedings of the 3rd DIMACS Workshop on DNA Based Computers, U. Penn., 1997.

[12] A. Mateescu, Gh. Paun, G. Rozenberg, A. Salomaa, Simple splicing systems, Discrete Appl. Math. in: Biomolecular Computing. Theory and Experiment, Gh. Paun, ed. (Springer), to appear.

[13] R. McNaughton, S. Papert, Counter-Free Automata, MIT Press, Cambridge, MA, 1971.

[14] Gh. Paun, Five (plus two) universal DNA computing models based on the splicing operation, Second DNA Computing Workshop, Princeton, June 1996.

[15] Gh. Paun, G. Rozenberg, A. Salomaa, Computing by splicing, Theoret. Comput. Sci. 168 (1996) 321–336.

[16] Gh. Paun, A. Salomaa, DNA computing based on the splicing operation, Math. Japon. 43 (1996) 607–632.

[17] D. Pixton, Regularity of splicing systems, Discrete Appl. Math. 69 (1996) 101–124.

[18] D. Pixton, Splicing in abstract families of languages, Theoret. Comput. Sci., to appear.

[19] L. Priese, Y. Rogojine, M. Morgenstern, Finite $H$-systems with 3 test tubes are not predictable, in: Proceedings of the Pacific Symposium on Biocomputing '98, World Scientific, Singapore, 1997, 547–558.

[20] M.P. Schutzenberger, Sur certaines operations de fermeture dans les langages rationels, Sympos. Math. 15 (1975) 245–253.