# MATHEMATICAL MODELLING OF THE RECOMBINATION CAPACITY OF SYSTEM OF ENZYMES ACTING ON DNA

## (PEMODELAN MATEMATIK BAGI KAPASITI PENGGABUNGAN SEMULA SUATU SISTEM ENZIM-ENZIM YANG BERTINDAK KE ATAS DNA)

**NOR HANIZA SARMIN**
**TAHIR AHMAD**
**FAHRUL ZAMAN HUYOP**
**SITI MARIYAM SHAMSUDDIN**
**FONG WAN HENG**

**Jabatan Matematik**
**Fakulti Sains**
**Universiti Teknologi Malaysia**

**2007**

**UNIVERSITI TEKNOLOGI MALAYSIA**

**BORANG PENGESAHAN
LAPORAN AKHIR PENYELIDIKAN**

TAJUK PROJEK :  MATHEMATICAL MODELLING OF THE RECOMBINATION

CAPACITY OF SYSTEM OF ENZYMES ACTING ON DNA

Saya _____NOR HANIZA SARMIN_____
**(HURUF BESAR)**

Mengaku membenarkan **Laporan Akhir Penyelidikan** ini disimpan di Perpustakaan    Universiti Teknologi Malaysia dengan syarat-syarat kegunaan seperti berikut :

1.  Laporan Akhir Penyelidikan ini adalah hakmilik Universiti Teknologi Malaysia.

2.  Perpustakaan Universiti Teknologi Malaysia dibenarkan membuat salinan untuk tujuan rujukan sahaja.

3.  Perpustakaan dibenarkan membuat penjualan salinan Laporan Akhir Penyelidikan ini bagi kategori TIDAK TERHAD.

4.  * Sila tandakan ( / )

☐  SULIT  (Mengandungi maklumat yang berdarjah keselamatan atau Kepentingan Malaysia seperti yang termaktub di dalam AKTA RAHSIA RASMI 1972).

☐  TERHAD  (Mengandungi maklumat TERHAD yang telah ditentukan oleh Organisasi/badan di mana penyelidikan dijalankan).

☐ /  TIDAK TERHAD

_____
TANDATANGAN KETUA PENYELIDIK

_____
Nama & Cop Ketua Penyelidik

Tarikh : _____

CATATAN :    *Jika Laporan Akhir Penyelidikan ini SULIT atau TERHAD, sila lampirkan surat daripada pihak berkuasa/organisasi berkenaan dengan menyatakan sekali sebab dan tempoh laporan ini perlu dikelaskan sebagai SULIT dan TERHAD.

# MATHEMATICAL MODELLING OF THE RECOMBINATION CAPACITY OF SYSTEM OF ENZYMES ACTING ON DNA

## (PEMODELAN MATEMATIK BAGI KAPASITI PENGGABUNGAN SEMULA SUATU SISTEM ENZIM-ENZIM YANG BERTINDAK KE ATAS DNA)

NOR HANIZA SARMIN
TAHIR AHMAD
FAHRUL ZAMAN HUYOP
SITI MARIYAM SHAMSUDDIN
FONG WAN HENG

Jabatan Matematik
Fakulti Sains
Universiti Teknologi Malaysia

2007

# ACKNOWLEDGEMENT

# ABSTRACT

## MATHEMATICAL MODELLING OF THE RECOMBINATION CAPACITY OF SYSTEM OF ENZYMES ACTING ON DNA

*(Keywords: Modelling, DNA, formal language theory, splicing systems)*

This research initiates the connection between formal language theory and the study of informational macromolecules. The modelling of a biological splicing system has been done mathematically through formal language theory, which is a branch of applied group theory and theoretical computer science. The accuracy of how a splicing works has been verified through the corresponding biological splicing system.

In this research, different kinds of languages that result from the action of restriction enzymes on strings of DNA molecules have been studied. Examples of strictly locally testable languages are given. Besides, some theorems related to different combination of restriction enzymes in a strictly locally testable language are presented. A language is local if every string of a certain length is a constant relative to it. Some theorems related to concepts of constant and local are also given. A maximal firm subword of a word is discussed, where automata concept is used to illustrate the simple splicing system. This research also gives few actual molecular considerations for the molecules that will arise from a given initial sets of strings using some chosen restriction enzymes.

This research has brought together the communities of mathematical scientists and bio-molecular scientists since each of these sciences can make a contribution to the other. The most specific benefit of this research is the application of mathematical analysis of questions of which DNA bio-molecules can potentially arise in a test tube from the action of specific sets of enzymes acting on the specific sets of DNA molecules.

**Key researchers** :

Assoc. Prof. Dr. Nor Haniza Sarmin (Head)
Assoc. Prof. Dr. Tahir Ahmad
Assoc. Prof. Dr. Fahrul Zaman Huyop
Assoc. Prof. Dr. Siti Mariyam Shamsuddin
Fong Wan Heng

**E-mail** :     nhs@fs.utm.my
**Tel. No.** :   07-5534266
**Vote No.** :   74259

# ABSTRAK

## PEMODELAN MATEMATIK BAGI KAPASITI PENGGABUNGAN SEMULA SUATU SISTEM ENZIM-ENZIM YANG BERTINDAK KE ATAS DNA

*(Katakunci: Pemodelan, DNA, teori bahasa formal, sistem pemotongan dan pencantuman)*

Penyelidikan ini mencetuskan kaitan di antara teori bahasa formal dan pembelajaran molekul makro yang berinfomasi. Pemodelan bagi sistem pemotongan dan pencantuman secara biologi telah dilakukan secara bermatematik melalui teori bahasa formal, iaitu satu cabang dari teori kumpulan berpenggunaan dan sains komputer berteori. Kejituan pemotongan dan pencantuman pula telah disahkan melalui sistem pemotongan dan pencantuman secara biologikal.

Dalam penyelidikan ini, beberapa jenis bahasa yang wujud daripada aksi enzim pembatas ke atas jujukan molekul DNA telah diselidiki. Contoh-contoh bahasa bolehuji setempat yang tegas ada dimuatkan. Selain itu, beberapa teorem tentang gabungan berlainan bagi enzim pembatas dalam bahasa bolehuji setempat yang tegas juga diberikan. Suatu bahasa dikatakan tempatan jika setiap jujukan dengan kepanjangan tertentu yang relatif kepadanya adalah pemalar. Beberapa teorem berkenaan konsep pemalar dan tempatan telah dinyatakan berserta pembuktiannya. Konsep automata juga digunakan untuk menggambarkan sistem pemotongan dan pencantuman mudah. Penyelidikan ini juga memberi beberapa pertimbangan bermolekul sebenar untuk molekul yang boleh dihasilkan daripada suatu jujukan permulaan dengan menggunakan beberapa enzim pembatas yang dipilih.

Penyelidikan ini telah menemukan bersama komuniti saintis dari bidang matematik dan biomolekul. Ini disebabkan setiap bidang tersebut dapat memberi manafaat antara satu sama lain. Faedah khusus yang utama terhasil dari penyelidikan ini ialah aplikasi analisis matematik berkenaan biomolekul DNA yang akan terhasil dalam tabung uji akibat dari aksi set enzim pembatas tertentu yang bertindak ke atas set molekul DNA yang dipilih.

**Penyelidik utama** :

Prof Madya Dr. Nor Haniza Sarmin (Ketua)
Prof Madya Dr. Tahir Ahmad
Prof Madya Dr. Fahrul Zaman Huyop
Prof Madya Dr. Siti Mariyam Shamsuddin
Fong Wan Heng

**Mel-e** : nhs@fs.utm.my
**No Telefon** : 07-5534266
**No Vot** : 74259

# TABLE OF CONTENTS

## LIST OF SYMBOLS

| | | |
|---|---|---|
| a | - | [A/T] |
| *A* | - | Alphabet |
| *A\** | - | Strings |
| *B* | - | The set of all patterns associated with enzymes that either produce 5′ overhangs or produce blunt ends |
| c | - | [C/G] |
| *c* | - | Constant |
| *cxd* | - | Site |
| *C* | - | The set of all patterns associated with enzymes that produce 3′ overhangs |
| g | - | [G/C] |
| *I* | - | A set of initial strings |
| *L* | - | Language |
| *L(S)* | - | Language generated by splicing system *S* |
| *r* | - | Splicing rule |
| *R* | - | A set of splicing rules |
| *S* | - | Splicing system |
| t | - | [T/C] |
| *x* | - | Crossing |
| (*c*, *x*, *d*) | - | Triple |
| $\varnothing$ | - | Empty set |
| $\lambda$ | - | Empty string |
| ▼, ▲ | - | Cutting sites |
| \ | - | Not in |
| $\not\subset$ | - | Not subset |
| $\cup$ | - | Union |

# CHAPTER 1

# INTRODUCTION

## 1.1    Introduction

Every living organism has DNA that makes the organism unique.  There are more than 200 types of readily available restriction enzymes as listed in the New England Biolabs catalog.  These restriction enzymes can cut strings of DNA molecules at specific places, resulting in molecules with sticky ends.  New molecules then arise when molecules previously cut by restriction enzymes are pasted together by a ligase.  Splicing system was defined to model the recombinant action of restriction enzyme and a ligase on DNA molecules.  The language which results from a splicing system is called a splicing language.  This language contains the initial strings of DNA molecules and is closed under the application of splicing rules. This splicing language is further studied using formal language theory, which is a branch of applied discrete mathematics and theoretical computer science.  It concerns with sets of strings called languages and different mechanism for generating and recognizing them.

## 1.2    Research Background

This research initiates the connection between formal language theory and the study of informational macromolecules.  Previously, these two branches of studies are independent of each other.  However, when splicing system is introduced, the

generative capacity of system of enzymes acting on a set of DNA molecules is established formally using formal language theory. Different languages can result from this recombinant behaviours and are analyzed using concept of languages in formal language theory.

## 1.3    Problem Statement

To apply as many restriction enzymes in different splicing languages and to introduce new concepts related to splicing language.

## 1.4    Research Objectives

The objectives of this research are:

1) To study the different concepts in splicing systems and to find examples to illustrate those concepts.

2) To study the features of sets of restriction sites of the restriction enzymes that will allow formal descriptions of their generative capacity.

3) To study the sites in DNA molecules at which the restriction enzymes react.

4) To introduce new concepts in splicing system to determine recombinant behaviour of system of enzymes on DNA molecules.

5) To provide mathematical proofs for concepts related to splicing system.

**1.5** **Scope of the Study**

This research will involve the study of the sites in DNA molecules at which some 200 types of readily available restriction enzymes act and determine features of sets of these sites that allow transparent formal descriptions of their generative capacity using the concept of formal language theory. Splicing system, which is used to model the recombinant action of restriction enzymes and a ligase on DNA molecules, will be studied. New concepts related to splicing language will also be introduced.

**1.6** **Significance of Findings**

The very general long-term benefit of this research will be the mutual stimulation provided to the communities of mathematical scientists and biomolecular scientists since this area of research is fairly new in Malaysia. Each of these sciences can then make a contribution to each other. Algorithms for new concepts of splicing language will be provided. Research papers regarding this research will be published in national and international journals and proceedings.

# CHAPTER 2

# LITERATURE REVIEW

## 2.1    Literature Review

The potential effect of sets of restriction enzymes and a ligase that allow DNA molecules to be cleaved and reassociated to produce further molecules can be found in [1].  The associated languages are analyzed by means of a new generative formalism called a splicing system.  A new relationship between formal language theory and the study of informational macromolecules was thus initiated.  Formal language theory is a branch of applied discrete mathematics and theoretical computer science that is devoted to the study of sets of finite strings (called languages) of symbols chosen from a prescribed finite set (called an alphabet).  The set of double-stranded DNA molecules that may arise from an initial set of DNA molecules in the presence of specified enzyme activities is represented as a language over the four-symbol alphabet of deoxyribonucleotide pairs.

There are three different splicing models namely Head splicing system [1], Paun splicing system [2] and Pixton splicing system [3].  Head's splicing language is always Paun's splicing language, while Paun's splicing language is always Pixton's splicing language.  These different splicing models were studied in [4].

Splicing language generated with one sided context is introduced in [5], which describes reflexive and symmetric splicing language.  Besides, it gives molecular consideration of a splicing system via splicing rules of the full

recombinant capacity of a restriction enzyme accompanied by a ligase. Examples of reflexive and symmetric languages are also given in [6].

Null context splicing system and strictly locally testable (SLT) languages are described in [7]. 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.

An example of a regular language which is not a splicing language is given by Gatterdam in [8]. This papers shows that not all splicing languages are strictly locally testable and hence not persistent. Local and semilocal languages are introduced in [9]. Local language is a splicing language and a splicing language is a regular language. Other types of splicing languages such as simple, semi-simple and semi-null splicing languages are defined by Laun in [10].

## 2.2    Preliminaries

Some main definitions used in this research are listed below.

**Definition 2.1 (splicing system)** [1]**:**

A splicing system $S = (A, I, B, C)$ consists of a finite alphabet $A$, a finite set $I$ of initial strings in $A^*$, and finite sets $B$ and $C$ of triples $(c, x, d)$ with $c$, $x$ and $d$ in $A^*$. Each such triple in $B$ or $C$ is called a pattern. For each such triple the string $cxd$ is called a site and the string $x$ is called a crossing. Patterns in $B$ are called left patterns and patterns in $C$ are called right patterns. The language $L = L(S)$ generated by $S$ consists of the strings in $I$ and all strings that can be obtained by adjoining to $L$ $ucxfq$ and $pexdv$ whenever $ucxdv$ and $pexfq$ are in $L$ and $(c, x, d)$ and $(e, x, f)$ are patterns of the same hand. A language $L$ is a splicing language if there exists a splicing system $S$ for which $L = L(S)$.

**Definition 2.2 (persistent)** [1]**:**

Let $S = (A, I, B, C)$ be a splicing system. Then $S$ is persistent if for each pair of strings *ucxdv*, and *pexfq*, in $A^*$ with $(c, x, d)$ and $(e, x, f)$ of the same hand: If $y$ is a subsegment of *ucx* (respectively *xfq*) that is the crossing of a site in *ucxdv* (respectively *pexfq*) then this same subsegment $y$ of *ucxfq* contains an occurrence of the crossing of a site in *ucxfq*.

**Definition 2.3 (null context splicing system)** [1]**:**

A null context splicing system is a splicing system $S = (A, I, B, C)$ for which each cleavage pattern in $B$ and each in $C$ has the form $(1, x, 1)$.

**Definition 2.4 (constant)** [1]**:**

With respect to a language over $A$, a string $c$ in $A^*$ is a constant if, whenever *ucv* and *pcq* are in the language, *ucq* and *pcv* are also in the language.

**Definition 2.5 (strictly locally testable)** [7]**:**

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.

**Definition 2.6 (uniform splicing system)** [1]**:**

A uniform splicing system is a null context splicing system $S = (A, I, X, X)$ for which there is a positive integer $P$ such that $X = A^P$. A language $L$ is a uniform splicing language if there is a uniform splicing system $S$ for which $L = L(S)$.

**Definition 2.7 (*n*-local, local):**

A language $L$ is *n*-local, $n$ is a non-negative integer, if every string of length $n$ is a constant relative to $L$.  $L$ is a local language if it is *n*-local for some non-negative integer $n$.

The following are some propositions and theorems related to this research.

**Theorem 2.1 (De Luca & Restivo)** [1]**:**

If for a language $L$ over $A$ all the strings in $A^P$ are constants then $L$ is $(P+1)$-strictly locally testable.

Note: For a $P$-strictly locally testable language $L$ over an alphabet $A$, all strings in $A^P$ are constants.

**Theorem 2.2** [1]**:**

The following conditions on a language $L$ over an alphabet $A$ are equivalent:
(i)      $L$ is a persistent splicing language;
(ii)     $L$ is a strictly locally testable language;
(iii)    The set of constants for $L$ contains $A^P$ for some $P$;
(iv)    $L$ is a uniform splicing language.

**Proposition 2.3:**

Let $A$ be a finite alphabet and $L$ in $A^*$ a language.  Then $L$ is a null-context splicing language if and only if $L$ is local.

**Proof.**

Suppose that *L* is local.  Let *n* be a non-negative integer for which *L* is *n*-local.  Let *R* = {*w* in *A*\*: *w* has length *n*}.  Let *I* = {*z* in *L* : no string *w* of length *n* occurs as many as three times as a factor of *z* }.  We confirm that *L* = *L*(*A*,*R*,*I*): *L* ⊇ *L*(*A*,*R*,*I*) since *L* ⊇ *I* and every *w* in *R* is a constant with respect to *L* since *L* is *n*-local.  Suppose now that *L* ⊄ *L*(*A*,*R*,*I*) and let *z* be *a string of least length in L \ L*(*A*,*R*,*I*).  Then *z* is not in *I*, and there must be at least one string *w* of length *n*, for which *w* occurs in *z* at least three times.  Let the specified occurrences of *w* in the three factorizations, *z* = *swt* = *uwv* = *xwy*, be the first, second and third occurrences of *w* in *z*.  Thus *sw* is a *proper* prefix of *uw* and *uw* is a *proper* prefix of *xw*.  Let *uw* = *swp* and *xw* = *uwq*.  Since *w* is a constant with respect to *L*:  (1) from *swt* and *uwv* in *L* we have *swv* in *L*; (2) from *uwv* and *xwy* in *L* we have *uwy* in *L*.  Each of *swv* and *uwy* is *shorter* than *z* and therefore each is in *L*(*A*,*R*,*I*).   Since both *uwy* and *swv* are in *L*(*A*,*R*,*I*) and *w* is in *R*, the contradiction *uwv* = *z* is in *L*(*A*,*R*,*I*) arises.  Thus *L* ⊆ *L*(*A*,*R*,*I*) and *L* = *L*(*A*,*R*,*I*) is confirmed.

Suppose now that *L* = *L*(*A*,*R*,*I*) where both *R* and *I* are finite subsets of *A*\*.  (We do not assume that all strings in *R* have the same length.)  We confirm that *L* is *n*-local where *n* is the length of the longest string in *I* plus twice the length of the longest string in *R*:  Let *L*(0) = *I* and, for each *j* ≥ 1, let *L*(*j*+1) = *L*(*j*) ∪{ *z* in *A*\* \ *L*(*j*) : *z* = *uwy*, where *w* has length *n* and there exist *v* & *x* for which *uwv* & *xwy* are in *L*(*j*)}.  Then *L* = ∪ {*L*(*j*) : *j* ≥ 0}.  Suppose that *L* is *not n*-local. Note that *L*(0) *is n*-local (in the vacuous sense).  It follows that there is a greatest non-negative integer *k* for which *L*(*k*) is *n*-local.   Then *L*(*k*+1) is *not n*-local and there is string *z* in *L*(*k*+1)\*L*(*k*) in which a non-constant factor of length *n* occurs.  Then there are strings *uwv* & *xwy* in *L*(*k*) with *w* in *R* and *z* = *uwy*.  A contradiction arises as follows:  For each occurrence of a factor *s* of length *n* in *z*, one of the following three possibilities holds.  (1) *s* is a factor of *uw* and is therefore a constant because *uwv* is in *L*(*k*) which is *n*-local.  (2) *s* = *qwp* where *q* is a suffix of *u* and *p* is a prefix of *y* and is therefore a constant since *w* is a constant.  (3) *s* is a factor of *wy* and is therefore a constant because *xwy* is in *L*(*k*) which is *n*-local.  From this contradiction it follows that *L is n*-local as asserted.

# CHAPTER 3

## STRICTLY LOCALLY TESTABLE LANGUAGE

### 3.1    Introduction

This chapter highlights some examples of not strictly locally testable languages.  Theorems related to different combination of restriction enzymes in strictly locally testable languages are proved using definitions, theorems and propositions listed in Chapter 2.

### 3.2    Not Strictly Locally Testable Language

Two examples of languages that are not strictly locally testable is given below.  The first example differs from the second one by the length of the restriction sites of the enzymes.  Different restriction enzymes and initial strings are used for both the examples to illustrate the cut and paste activities of restriction enzymes and a ligase that act on a set of DNA molecules, and the new strings which will arise from them.  In these two examples, g, a, t and c denotes [G/C], [A/T], [T/A] and [C/G] respectively.  The lines in these two examples refer to the cutting sites by the respective restriction enzymes.

**Example 3.1:**

Let $S = (D, I, B, \varnothing)$ be a splicing system where $I = \{$ggtacc tctagc tgtaca, gctagc tgtacc tctaga$\}$ is the set consisting of two initial strings. The set $B = \{$(G, GTAC, C), (T, GTAC, A), (G, CTAG, C), (T, CTAG, A)$\}$ is the set of cleavage patterns for enzymes *Acc*65I, *Bsr*GI, *Nhe*I and *Xba*I respectively. These patterns all leave 5' overhangs.

Considering the first initial string, that is ggtacc tctagc tgtaca. Using the enzyme *Acc*65I, the following recombination can be seen, where new molecules (3.1) and (3.2) will arise.

<div align="center">

G|<u>GTAC</u>C TCTAGC TGTACA
CCATG|G AGATCG ACATGT

TGTACA GCTAGA G|<u>GTAC</u>C
ACATGT CGATCT  CCATG|G

</div>

GGTACC                                                     (3.1)
CCATGG

TGTACA GCTAGA GGTACC TCTAGC TGTACA             (3.2)
ACATGT CGATCT CCATGG AGATCG ACATGT

Using the enzyme *Bsr*GI, the following recombination can be seen, where new molecules (3.3) and (3.4) will arise.

<div align="center">

GGTACC TCTAGC T|<u>GTAC</u>A
CCATGG AGATCG ACATG|T

T|<u>GTAC</u>A GCTAGA GGTACC
ACATG|T CGATCT  CCATGG

</div>

TGTACA                                                     (3.3)
ACATGT

GGTACC TCTAGC TGTACA GCTAGA GGTACC            (3.4)
CCATGG AGATCG ACATGT CGATCT  CCATGG

Using both the enzymes *Acc*65I and *Bsr*GI, the following recombination can be seen, where new molecules (3.6), (3.7), (3.8) and (3.9) will arise.

$$\begin{array}{l} \text{G|\underline{GTACC} TCTAGC TGTACA} \\ \text{CCATG|G AGATCG ACATGT} \end{array}$$

$$\begin{array}{ll} \text{GGTACC TCTAGC T|\underline{GTACA}} & (3.5) \\ \text{CCATGG AGATCG ACATG|T} \end{array}$$

GGTACA                                                      (3.6)
CCATGT

G|<u>GTACC</u>  TCTAGC TGTACC TCTAGC  TGTACA            (3.7)
CCATG|G AGATCG ACATGG AGATCG ACATGT

GGTACC TCTAGC  TGTACC                                     (3.8)
CCATGG AGATCG ACATGG

TGTACC  TCTAGC TGTACA                                     (3.9)
ACATGG AGATCG ACATGT

Moreover, when the enzymes *Acc*65I and *Bsr*GI are applied to the molecules marked (3.5) and (3.7), the following new string of molecule will arise:

GGTACC TCTAGC  TGTACC  TCTAGC  TGTACC TCTAGC  TGTACA
CCATGG AGATCG ACATG G AGATCG ACATGG AGATCG ACATGT.

Continuing in this manner, the molecules which will result from the first initial string using the enzymes *Acc*65I and *Bsr*GI can be summarized as

$$\left\{ \begin{array}{l} \text{GGTACC} \\ \text{CCATGG} \end{array} \left( \begin{array}{l} \text{TCTAGCTGTACC} \\ \text{AGATCGACATGG} \end{array} \right)^n \begin{array}{l} \text{TCTAGCTGTACA} \\ \text{AGATCGACATGT} \end{array} : n \ge 0 \right\}.$$

Now, considering the second initial string, that is gctacg tgtacc tctaga. Using the enzyme *Nhe*I, the following recombination can be seen, where new molecules (3.10) and (3.11) will arise.

$$\begin{array}{l} \text{G|\underline{CTAG}C TGTACC TCTAGA} \\ \text{CGATC|G ACATGG AGATCT} \end{array}$$

TCTAGA GGTACA G|<u>CTAG</u>C
AGATCT CCATGT CGATC|G

GCTAGC                                                                                    (3.10)
CGATCG

TCTAGA GGTACA GCTAGC TGTACC TCTAGA                       (3.11)
AGATCT CCATGT CGATCG ACATGG AGATCT

Using the enzyme *Xba*I, the following recombination can be seen, where new molecules (3.12) and (3.13) will arise.

GCTAGC TGTACC T|<u>CTAG</u>A
CGATCG ACATGG AGATC|T

T|<u>CTAG</u>A GGTACA GCTAGC
AGATC|T CCATGT CGATCG

TCTAGA                                                                                    (3.12)
AGATCT

GCTAGC TGTACC  TCTAGA GGTACA GCTAGC                        (3.13)
CGATCG ACATGG AGATCT CCATGT CGATCG

Using both the enzymes *Nhe*I and *Xba*I, the following recombination can be seen, where new molecules (3.15), (3.16), (3.17) and (3.18) will arise.

G|<u>CTAG</u>C TGTACC TCTAGA
CGATC|G ACATGG AGATCT

GCTAGC TGTACC T|<u>CTAG</u>A                                        (3.14)
CGATCG ACATGG AGATC|T

GCTAGA                                                                                    (3.15)
CGATCT

G|<u>CTAG</u>C TGTACC  TCTAGC TGTACC TCTAGA             (3.16)
CGATC|G ACATGG AGATCG ACATGG AGATCT

TCTAGC TGTACC  TCTAGA                                               (3.17)
AGATCG ACATGG AGATCT

GCTAGC TGTACC TCTAGC                                                         (3.18)
CGATCG ACATGG AGATCG

Moreover, when the enzymes *Nhe*I and *Xba*I are applied to the molecules marked (3.14) and (3.16), the following new string of molecule will arise:

GCTAGC TGTACC TCTAGC  TGTACC  TCTAGC  TGTACC TCTAGA
CGATCG ACATGG AGATCG ACATGG AGATCG ACATGG AGATCT.

Continuing in this manner, the molecules which will result from the second initial string using the enzymes *Nhe*I and *Xba*I can be summarized as

$$\left\{ \begin{array}{l} \text{GCTAGCTGTACC} \\ \text{CGATCGACATGG} \end{array} \left( \begin{array}{l} \text{TCTAGCTGTACC} \\ \text{AGATCGACATGG} \end{array} \right)^{n} \begin{array}{l} \text{TCTAGA} \\ \text{AGATCT} \end{array} : n \geq 0 \right\}.$$

None of the strings in the infinite set (tctagc tgtacc)$^n$ is a constant since each such string occurs as a segment of a string in each of the two sets. Since if such a string were a constant, a string beginning with GG and ending with GA would lie in *L*. For this splicing language, there does not exist a positive integer *P* for which all strings of length *P* are constant. Thus the language which result from this splicing system using the four enzymes *Acc*65I, *Bsr*GI, *Nhe*I and *Xba*I with initial string *I* = {ggtacc tctagc tgtaca, gctagc tgtacc tctaga} is not strictly locally testable.

**Example 3.2:**

Let *S* = (*D*, *I*, *B*, ∅) be a splicing system where *I* = {ccgg ttag tcga, ctag tcgg ttaa} is the set consisting of two initial strings. The set *B* = {(C, CG, G), (T, CG, A), (C, TA, G), (T, TA, A)} is the set of cleavage patterns for enzymes *Hpa*II, *Taq*$^{\alpha}$ I, *Bfa*I and *Mse*I respectively. These patterns all leave 5' overhangs.

Considering the first initial string, that is ccgg ttag tcga. Using the enzyme *Hpa*II, the following recombination can be seen, where new molecules (3.19) and (3.20) will arise.

C|<u>CG</u>G TTAG TCGA
GGC|C AATC AGCT


TCGA CTAA C|<u>CG</u>G
AGCT GATT GGC|C


CCGG            (3.19)
GGCC

TCGA CTAA CCGG TTAG TCGA       (3.20)
AGCT GATT GGCC AATC AGCT


Using the enzyme $Taq^\alpha$ I, the following recombination can be seen, where new molecules (3.21) and (3.22) will arise.


CCGG TTAG T|<u>CG</u>A
GGCC AATC AGC|T


T|<u>CG</u>A CTAA CCGG
AGC|T GATT GGCC


TCGA            (3.21)
AGCT

CCGG TTAG TCGA CTAA CCGG       (3.22)
GGCC AATC AGCT GATT GGCC


Using both the enzymes $Hpa$II and $Taq^\alpha$ I, the following recombination can be seen, where new molecules (3.24), (3.25), (3.26) and (3.27) will arise.


C|<u>CG</u>G TTAG TCGA
GGC|C AATC AGCT


CCGG TTAG T|<u>CG</u>A       (3.23)
GGCC AATC AGC|T


CCGA            (3.24)
GGCT


C|<u>CG</u>G TTAG TCGG TTAG TCGA       (3.25)
GGC|C AATC AGCC AATC AGCT

TCGG TTAG TCGA                                  (3.26)
AGCC AATC AGCT


CCGG TTAG TCGG                                  (3.27)
GGCC AATC AGCC


Moreover, when the enzymes *Hpa*II and *Taq*$^{\alpha}$ I are applied to the molecules marked (3.23) and (3.25), the following new string of molecule will arise:

CCGG TTAG TCGG TTAG TCGG TTAG TCGA
GGCC AATC AGCC AATC AGCC AATC AGCT.


Continuing in this manner, the molecules which will result from the first initial string using the enzymes *Hpa*II and *Taq*$^{\alpha}$ I can be summarized as

$$\left\{ \text{CCGG} \left( \frac{\text{TTACTCGG}}{\text{AATGAGCC}} \right)^{n} \frac{\text{TTAGTCGA}}{\text{AATCAGCT}} : n \geq 0 \right\}.$$


Now, considering the second initial string, that is ctag tcgg ttaa. Using the enzyme *Bfa*I, the following recombination can be seen, where new molecules (3.28) and (3.29) will arise.


C|<u>TA</u>G TCGG TTAA
GAT|C AGCC AATT


TTAA CCGA C|<u>TA</u>G
AATT GGCT GAT|C


CTAG                                                   (3.28)
GATC


TTAA CCGA CTAG TCGG TTAA                     (3.29)
AATT GGCT GATC AGCC AATT

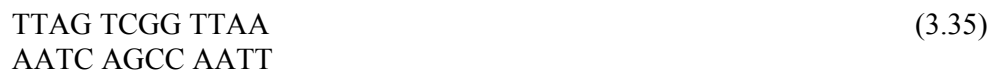
Using the enzyme *Mse*I, the following recombination can be seen, where new molecules (3.30) and (3.31) will arise.


CTAG TCGG T|<u>TA</u>A
GATC AGCC AAT|T

T|<u>TA</u>A CCGA CTAG
AAT|T GGCT GATC

TTAA                                         (3.30)
AATT

CTAG TCGG TTAA CCGA CTAG                     (3.31)
GATC AGCC AATT GGCT GATC

Using both the enzymes *Bfa*I and *Mse*I, the following recombination can be seen, where new molecules (3.33), (3.34), (3.35) and (3.36) will arise.

C|<u>TA</u>G TCGG TTAA
GAT|C AGCC AATT

CTAG TCGG T|<u>TA</u>A               (3.32)
GATC AGCC AAT|T

CTAA                                           (3.33)
GATT

C|<u>TA</u>G TCGG TTAG TCGG TTAA                   (3.34)
GAT|C AGCC AATC AGCC AATT

TTAG TCGG TTAA                              (3.35)
AATC AGCC AATT

CTAG TCGG TTAG                              (3.36)
GATC AGCC AATC

Moreover, when the enzymes *Bfa*I and *Mse*I are applied to the molecules marked (3.32) and (3.34), the following new string of molecule will arise:

CTAG TCGG TTAG   TCGG TTAG TCGG TTAA
GATC AGCC AATC AGCC AATC AGCC AATT.

Continuing in this manner, the molecules which will result from the second initial string using the enzymes *Bfa*I and *Mse*I can be summarized as

$$\left\{ \begin{matrix} \text{CTAGTCGG} \\ \text{GATCAGCC} \end{matrix} \left( \begin{matrix} \text{TTAGTCGG} \\ \text{AATCAGCC} \end{matrix} \right)^{n} \begin{matrix} \text{TTAA} \\ \text{AATT} \end{matrix} : n \geq 0 \right\}.$$

None of the strings in the infinite set (ttag tcgg)$^n$ is a constant since each such string occurs as a segment of a string in each of the two sets. Since if such a string were a constant, a string beginning with CC and ending with AA would lie in *L*. For this splicing language, there does not exist a positive integer *P* for which all strings of length *P* are constant. Thus the language which result from this splicing system using the four enzymes *Hpa*II, *Taq*$^\alpha$ I, *Bfa*I and *Mse*I with initial string *I* = {ccgg ttag tcga, ctag tcgg ttaa} is not strictly locally testable.

## 3.3    Different Combination of Restriction Enzymes in a Strictly Locally Testable Language

This section discusses some theorems related to different combination of restriction enzymes in strictly locally testable (SLT) languages.

**Theorem 3.1:**

Suppose *L* = *L*(*S*) where *S* = (*D, I, B, C*). If only one of the restriction enzymes is used for either *B* or *C*, the language generated must be SLT disregard of the initial language *I* (i.e., the initial set of molecular varieties).

**Proof:**

If there is only one restriction enzyme present, only the string of DNA which contains the restriction site of that particular enzyme can be altered by cutting at the specified sequence of restriction enzyme, and later pasting it back. Since there is a positive integer *k* for which every factor of *L* of length *k* is a constant, *L* is a SLT language. □

**Theorem 3.2:**

Suppose $L = L(S)$ where $S = (D, I, B, C)$. If exactly one enzyme having a left overhang and exactly one enzyme having a right overhang is used to give $B$ and $C$, the language generated must be SLT disregard of the initial language $I$.

**Proof:**

If there is only one enzyme in each $B$ and $C$, only the string of DNA which contains the restriction site of any one or both of those particular enzymes can be altered, and since one of the enzyme has a left overhang and the other enzyme a right overhang, it is not possible for them to be cut and pasted together. This is because there is no possibility that a 5' overhang can combine with a 3' overhang. Since there is a positive integer $k$ for which every factor of $L$ of length $k$ is a constant, $L$ is a SLT language. $\square$

**Theorem 3.3:**

Given a set consisting of two restriction enzymes: *Age*I and *Mse*I. Since there are two enzymes in the set, there are $2^2 = 4$ subsets of this set of two enzymes. For all of these four subsets, the language generated by this splicing system will be SLT for every possible choice of $I$.

**Proof:**

These four subsets are $\varnothing$, { *Age*I }, { *Mse*I }and { *Age*I, *Mse*I }. This is due to the crossings of the restriction sites are disjoint as shown below:

Restriction site for the enzyme *Age*I:
5′…A▾CCGG T…3′
3′…T GGCC▴A…5′

Restriction site for the enzyme *Mse*I:

5′...T▼TA A...3′
3′...A AT▲T...5′

Therefore, since the crossings of the restriction sites are disjoint and there is a positive integer $k$ for which every factor of $L$ of length $k$ is a constant, $L$ is a SLT language. □

**Theorem 3.4:**

Given a set consisting of three restriction enzymes: *Age*I, MseI and *Hpa*II. Since there are three enzymes in the set, there are $2^3 = 8$ subsets of this set of three enzymes. For all of these eight subsets, the language generated by this splicing system will be SLT for every possible choice of $I$.

**Proof:**

These eight subsets are $\varnothing$, { *Age*I }, { *Mse*I }, { *Hpa*II }, { *Age*I, *Mse*I }, { *Age*I, *Hpa*II }, { *Mse*I, *Hpa*II } and { *Age*I, *Mse*I, *Hpa*II }. This is due to the crossings of the restriction sites are disjoint as shown below:

Restriction site for the enzyme *Age*I:

5′…A▼CCGG T…3′
3′…T GGCC▲A…5′

Restriction site for the enzyme *Mse*I:

5′...T▼TA A...3′
3′...A AT▲T...5′

Restriction site for the enzyme *Hpa*II:

5′…C▼CG G…3′
3′…G GC▲C…5′

Therefore, since the crossings of the restriction sites are disjoint and there is a positive integer $k$ for which every factor of $L$ of length $k$ is a constant, $L$ is a SLT language. $\square$

**Theorem 3.5:**

Given a set consisting of four restriction enzymes: *Bam*HI, *Bgl*II, *Bcl*I and *Dpn*II. Since there are four enzymes in the set, there are $2^4 = 16$ subsets for this set of four enzymes. Let $S$ be any subset of these sets that contain *Dpn*II. For every such set $S$, $\varnothing$, { *Bam*HI }, { *Bgl*II } and { *Bcl*I }, the language generated by this splicing system is SLT for every possible choice of $I$.

**Proof:**

The restriction sites for enzymes *Bam*HI, *Bgl*II, *Bcl*I and *Dpn*II are shown below:

Restriction site for the enzyme *Bam*HI:

5′…G▼GATC C…3′
3′…C CTAG▲G…5′

Restriction site for the enzyme *Bgl*II:

5′…A▼GATC T…3′
3′…T CTAG▲A…5′

Restriction site for the enzyme *Bcl*I:

5′…T▼GATC A…3′
3′…A CTAG▲T…5′

Restriction site for the enzyme *Dpn*II:

5′…▼GATC …3′
3′… CTAG▲…5′

For a language $L$ generated by sets $\varnothing$, {*Bam*HI}, {*Bgl*II} and {*Bcl*I}, the set of constants for $L$ contains $A^P$ for some $P$, that is $P = 4$ in this case. Therefore $L$ is a SLT language.

For any splicing system with sets consisting of *Dpn* II, that splicing system is a null context splicing system where $P = 4$. Such a splicing system is also a uniform splicing system, thus the language generated is a uniform splicing language, implying that it is a SLT language also. □

**Theorem 3.6:**

Given a set consisting of four restriction enzymes: *Dpn*I, *Dpn*II, *Bam*HI and *Bcl*I. Since there are four enzymes in the set, there are $2^4 = 16$ subsets for this set of four enzymes. Let $S$ be any subset that contains at least one of { *Dpn*I, *Dpn*II }. For every such set $S$, $\varnothing$, { *Bam*H I } and { *Bcl* I }, the language generated by this splicing system is SLT for every possible choice of $I$.

**Proof:**

The restriction sites for enzymes *Dpn*I, *Dpn*II, *Bam*HI and *Bcl*I are shown below:

Restriction site for the enzyme *Dpn*I:
5′…GA▼TC…3′
3′…CT▲AG…5′

Restriction site for the enzyme *Dpn*II:
5′…▼GATC …3′
3′… CTAG▲…5′

Restriction site for the enzyme *Bam*HI:
5′…G▼GATC C…3′
3′…C CTAG▲G…5′

Restriction site for the enzyme *Bcl*I:

5′…T$^{\blacktriangledown}$GATC A…3′
3′…A CTAG$_{\blacktriangle}$T…5′

For a language $L$ generated by sets $S$, $\varnothing$, {*Bam*HI} and {*Bcl*I}, the set of constants for $L$ contains $A^{P}$ for some $P$, that is $P = 4$ in this case. Strings that contain sites are constants. There is a positive integer $k = 4$ for which every factor of $L$ of length 4 is a constant. Therefore $L$ is a SLT language. $\square$

## 3.4   Conclusion

This chapter shows two examples of a language that is not strictly locally testable. Besides, several theorems regarding different combination of restriction enzymes in a strictly locally testable language are also discussed.

# CHAPTER 4

## CONCEPTS OF CONSTANT AND LOCAL

### 4.1 Introduction

In this section, some theorems and proofs on the concepts of constant and local are included. Recall that a string $x$ is a constant relative to a language $L$ if for any two words $pxq$ and $uxv$ in $L$, it follows that $pxv$ is also in $L$. Also, a language $L$ is $n$-local, where $n$ is a non-negative integer, if every string of length $n$ is a constant relative to $L$. Furthermore, $L$ is a local language if it is $n$-local for some non-negative integer $n$.

### 4.2 Theorems on Concepts of Constant and Local

Below are six theorems regarding the concepts of constant and local in a language together with their proofs.

**Theorem 4.1:**

If $x$ is a constant relative to $L$ and $pxq$ and $uxv$ are in $L$, then $uxq$ is in $L$.

**Proof:**

Let $x$ be a constant relative to $L$. For $pxq$ and $uxv$ in $L$, by definition, $pxv$ is in $L$. Thus for $uxv$ and $pxq$ in $L$, by definition also, $uxq$ is in $L$. □

**Theorem 4.2:**

If $L$ is $n$-local then it is $k$-local for every $k > n$.

**Proof:**

Suppose $L$ is $n$-local. Every string of length $n$ is a constant relative to $L$. If $px_1x_2\ldots x_{n-1}x_nq$, $ux_1x_2\ldots x_{n-1}x_nv \in L$, then $px_1x_2\ldots x_{n-1}x_nv \in L$ and $ux_1x_2\ldots x_{n-1}x_nq \in L$ since $x_1x_2\ldots x_{n-1}x_n$ is a constant. Suppose $px_1x_2\ldots x_{n-1}x_nx_{n+1}\ldots x_{k-1}x_kq$, $ux_1x_2\ldots x_{n-1}x_nx_{n+1}\ldots x_{k-1}x_kv \in L$. From $px_1x_2\ldots x_{n-1}x_nq \in L$ and $ux_1x_2\ldots x_{n-1}x_nx_{n+1}\ldots x_{k-1}x_kv \in L$, $px_1x_2\ldots x_{n-1}x_nx_{n+1}\ldots x_{k-1}x_kv \in L$ also since $x_1x_2\ldots x_{n-1}x_n$ is a constant. Similarly, from $ux_1x_2\ldots x_{n-1}x_nv \in L$ and $px_1x_2\ldots x_{n-1}x_nx_{n+1}\ldots x_{k-1}x_kq \in L$, $ux_1x_2\ldots x_{n-1}x_nx_{n+1}\ldots x_{k-1}x_kq \in L$ also since $x_1x_2\ldots x_{n-1}x_n$ is a constant. Therefore, if $L$ is $n$-local, then it is $k$-local for every $k > n$. □

**Theorem 4.3:**

If $L = xv'(vyzu)^*u'x$, then $L$ is local and $n$-local for $n \geq 2$.

**Proof:**

$L$ is a local language if it is $n$-local for some non-negative integer $n$, that is, every string of length $n$ is a constant relative to $L$. Since $vyzu$ of length 4 is a constant relative to $L$ and every string of length 4 is a constant relative to $L$, $L$ is 4-local.

Suppose $x$ is a constant and $xv'u'x \in L$. However, $xv'u'xv'u'x \notin L$. Thus $L$ is not 1-local. But every string of length 2 is a constant relative to $L$. Therefore $L$ is $n$-local for $n \geq 2$. $\square$

**Theorem 4.4:**

If $L' = xz'(zuvy)^*y'x$, then $L'$ is local and $n$- local for $n \geq 2$.

**Proof:**

$L'$ is a local language if it is $n$-local for some non-negative integer $n$, that is, every string of length $n$ is a constant relative to $L'$. Since *zuvy* of length 4 is a constant relative to $L'$ and every string of length 4 is a constant relative to $L'$, $L'$ is 4-local.

Suppose $x$ is a constant and $xz'y'x \in L'$. However, $xz'y'xz'y'x \notin L'$. Thus $L'$ is not 1-local. But every string of length 2 is a constant relative to $L'$. Therefore $L'$ is $n$-local for $n \geq 2$. $\square$

**Theorem 4.5:**

If $L = xv'(vyzu)^*u'x$ and $L' = xz'(zuvy)^*y'x$, then $L'' = L \cup L' = xv'vyz(uvyz)^*uu'x \cup xz'z(uvyz)^*uvyy'x \cup \{xv'u'x, xz'y'x\}$.

**Proof:**

$$
\begin{aligned}
L'' \quad &= L \cup L' \\
&= xv'(vyzu)^*u'x \cup xz'(zuvy)^*y'x \\
&= xv'u'x \cup xv'vyz(uvyz)^*uu'x \cup xz'y'x \cup xz'z(uvyz)^*uvyy'x \\
&= xv'vyz(uvyz)^*uu'x \cup xz'z(uvyz)^*uvyy'x \cup \{xv'u'x, xz'y'x\}. \quad \square
\end{aligned}
$$

**Theorem 4.6:**

If $L" = xv'vyz(uvyz)*uu'x \cup xz'z(uvyz)*uvyy'x \cup \{xv'u'x, xz'y'x\}$, then $L"$ is *not* local.

**Proof:**

$L"$ is not 1-local since there exist string of length 1 that is not a constant relative to $L"$: $xv'u'x, xz'y'x \in L"$ but $xv'u'xz'y'x \notin L"$. $L"$ is not 2-local since $xv'vyzuu'x, xz'zuvyy'x \in L"$ but $xv'vyy'x \notin L"$. $L"$ is not 3-local since $xv'vyzuu'x$, $xz'zuvyzuvyy'x \in L"$ but $xv'vyzuvyy'x \notin L"$. $L"$ is not 4-local since $xv'vyzuvyzuu'x$, $xz'zuvyzuvyy'x \in L"$ but $xv'vyzuvyzuvyy'x \notin L"$.

Similarly, no string beginning with $xv'$ and ending with $y'x$ would lie in $L"$. There is a string of length $n$ that is not a constant relative to $L"$. $L"$ is not $n$-local and thus not local. $\square$

**4.3    Theorems on Conditions of Splicing Rules**

This section presents three theorems showing conditions of splicing rules which are equivalent to each other.

**Theorem 4.7:**

The following four conditions on the pair consisting of a word $w$ in $A*$ and a language $L \in A*$ are equivalent:

  (a) $w$ is a constant with respect to $L$.
  (b) $L$ is closed under the splicing rule $(w, 1, w, 1)$, i.e., under the splicing rule $w$.
  (c) $L$ is closed under the splicing rule $(1,w, 1, w)$.
  (d) $L$ is closed under a splicing rule $(u, v, u, v)$ where $uv = w$.

**Proof:**

**(a) ⇒ (b):** Suppose *w* is a constant with respect to *L*. Thus, if *pwq*, *rws* ∈ *L*, then *pws* ∈ *L* and *rwq* ∈ *L*. Using splicing rule (*w*, 1, *w*, 1), *pwq* = *p-w*, 1-*q* ∈ *L* and *rws* = *r-w*, 1-*s* ∈ *L*. This splicing rule (*w*, 1, *w*, 1) yields *p-w*,1-*s* = *pws* ∈ *L* and *r-w*,1-*q* = *rwq* ∈ *L*. Therefore, *pwq*, *rws*, *pws*, *rwq* ∈ *L* using rule (*w*, 1, *w*, 1) and *L* is closed under this splicing rule.

**(b) ⇒ (c):** Suppose *L* is closed under the splicing rule (*w*, 1, *w*, 1). Thus, *pwq*, *rws*, *pws*, *rwq* ∈ *L* using (*w*, 1, *w*, 1). Using splicing rule (1, *w*, 1, *w*), *pwq* = *p*-1, *w-q* ∈ *L* and *rws* = *r*-1,*w-s* ∈ *L*. This splicing rule (1, *w*, 1, *w*) further yields *p*-1, *w-s* = *pws* ∈ *L* and *r*-1, *w-q* = *rwq* ∈ *L*. Therefore, *pwq*, *rws*, *pws*, *rwq* ∈ *L* using rule (1, *w*, 1, *w*) and *L* is closed under this splicing rule.

**(c) ⇒ (d):** Suppose *L* is closed under the splicing rule (1, *w*, 1, *w*). Thus, *pwq*, *rws*, *pws*, *rwq* ∈ *L* using (1, *w*, 1, *w*). Using splicing rule (*u*, *v*, *u*, *v*) where *uv* = *w*, *pwq* = *puvq* = *p-u*, *v-q* ∈ *L* and *rws* = *ruvs* = *r-u*, *v-s* ∈ *L*. This splicing rule (*u*, *v*, *u*, *v*) further yields *p-u*, *v-s* = *puvs* = *pws* ∈ *L* and *r-u*, *v-q* = *ruvq* = *rwq* ∈ *L*. Therefore, *pwq*, *rws*, *pws*, *rwq* ∈ *L* using rule (*u*, *v*, *u*, *v*) where *uv* = *w* and *L* is closed under this splicing rule.

**(d) ⇒ (a):** Suppose *L* is closed under the splicing rule (*u*, *v*, *u*, *v*) where *uv* = *w*. Thus, *pwq*, *rws*, *pws*, *rwq* ∈ *L* using (*u*, *v*, *u*, *v*). Since *pwq*, *rws* ∈ *L* and *pws*, *rwq* ∈ *L*, then *w* is a constant with respect to *L*. □

**Theorem 4.8:**

The following four conditions on a language *L* ∈ *A*\* are equivalent:
(a) *L* is *n*-local.
(b) *L* is closed under the action of each splicing rule (*w*, 1, *w*, 1) for which *w* has length *n*, i.e., under the splicing rule *w*, where *w* has length *n*.

(c) $L$ is closed under the action of every splicing rule $(1, w, 1, w)$ with $w$ of length $n$.

(d) $L$ is closed under the action of each splicing rule $(u, v, u, v)$ for which length $uv$ is $n$.

**Proof:**

**(a) $\Rightarrow$ (b):** Suppose $L$ is $n$-local, that is, every string of length $n$ is a constant relative to $L$. Thus, if $pw_1w_2\ldots w_nq$, $rw_1w_2\ldots w_ns \in L$, then $pw_1w_2\ldots w_ns \in L$. Using splicing rule $(w, 1, w, 1)$, $pw_1w_2\ldots w_nq = pw_1w_2\ldots w_n$, $1\text{-}q \in L$ and $rw_1w_2\ldots w_ns = rw_1w_2\ldots w_n$, $1\text{-}s \in L$ yields $pw_1w_2\ldots w_n$, $1\text{-}s = pw_1w_2\ldots w_ns \in L$ and $rw_1w_2\ldots w_n, 1\text{-}q = rw_1w_2\ldots w_nq \in L$. Therefore, $pw_1w_2\ldots w_nq$, $rw_1w_2\ldots w_ns$, $pw_1w_2\ldots w_ns$ and $rw_1w_2\ldots w_nq \in L$ using rule $(w, 1, w, 1)$ and $L$ is closed under this splicing rule.

**(b) $\Rightarrow$ (c):** Suppose that $L$ is closed under the splicing rule $(w, 1, w, 1)$. Thus, $pw_1w_2\ldots w_nq$, $rw_1w_2\ldots w_ns$, $pw_1w_2\ldots w_ns$ and $rw_1w_2\ldots w_nq \in L$ using rule $(w, 1, w, 1)$. But using splicing rule $(1, w, 1, w)$, $pw_1w_2\ldots w_nq = p\text{-}1$, $w_1w_2\ldots w_n - q \in L$ and $rw_1w_2\ldots w_ns = r\text{-}1$, $w_1w_2\ldots w_n\text{-}s \in L$. This splicing rule further yields $p\text{-}1$, $w_1w_2\ldots w_n\text{-}s = pw_1w_2\ldots w_ns \in L$ and $r\text{-}1$, $w_1w_2\ldots w_n\text{-}q = rw_1w_2\ldots w_nq \in L$. Therefore, $pw_1w_2\ldots w_nq$, $rw_1w_2\ldots w_ns$, $pw_1w_2\ldots w_ns$ and $rw_1w_2\ldots w_nq \in L$ using rule $(1, w, 1, w)$ and $L$ is closed under this splicing rule.

**(c) $\Rightarrow$ (d):** Suppose that $L$ is closed under the splicing rule $(1, w, 1, w)$. Thus, $pw_1w_2\ldots w_nq$, $rw_1w_2\ldots w_ns$, $pw_1w_2\ldots w_ns$ and $rw_1w_2\ldots w_nq \in L$ using rule $(1, w, 1, w)$. But using splicing rule $(u, v, u, v)$ where $uv = w_1w_2\ldots w_n$, $pw_1w_2\ldots w_nq = puvq = p\text{-}u$, $v\text{-}q \in L$ and $rw_1w_2\ldots w_ns = ruvs = r\text{-}u$, $v\text{-}s \in L$. This splicing rule further yields $p\text{-}u$, $v\text{-}s = puvs = pw_1w_2\ldots w_ns \in L$ and $r\text{-}u$, $v\text{-}q = ruvq = rw_1w_2\ldots w_nq \in L$. Therefore, $pw_1w_2\ldots w_nq$, $rw_1w_2\ldots w_ns$, $pw_1w_2\ldots w_ns$ and $rw_1w_2\ldots w_nq \in L$ using rule $(u, v, u, v)$ where $uv = w_1w_2\ldots w_n$ and $L$ is closed under this splicing rule.

**(d) $\Rightarrow$ (a):** Suppose $L$ is closed under the splicing rule $(u, v, u, v)$ where $uv = w_1w_2\ldots w_n$. Thus, $pw_1w_2\ldots w_nq$, $rw_1w_2\ldots w_ns$, $pw_1w_2\ldots w_ns$ and $rw_1w_2\ldots w_nq \in L$ using

rule $(u, v, u, v)$. Since $pw_1w_2\ldots w_nq$, $rw_1w_2\ldots w_ns \in L$ and $pw_1w_2\ldots w_ns$, $rw_1w_2\ldots w_nq$ $\in L$, then $w_1w_2\ldots w_n$ is a constant with respect to $L$. Since every string of length $n$ is a constant relative to $L$, $L$ is $n$-local. $\square$

**Theorem 4.9:**

The following three conditions on a language $L$ are equivalent:

(a) $L$ is local.

(b) $L$ is a uniform splicing language.

(c) $L$ is a null-context splicing language.

**Proof:**

**(a) $\Rightarrow$ (b):** Suppose $L$ is local. Proposition 2.3 proves that if $L$ is local, then $L$ is a null-context splicing language. Here $R = \{w \in A^*: w$ has length $n\}$. Since $L$ is a null-context splicing language, for which, for some non-negative number $n$, $R$ is the set of all strings in $A^*$ having length $n$, and according to Definition 2.6, $L$ is a uniform splicing language.

**(b) $\Rightarrow$ (c):** Suppose $L$ is a uniform splicing language. By definition, a uniform splicing system is a null-context system for which, for some non-negative number $n$, $R$ is the set of all strings in $A^*$ having length $n$. Thus $R$ is a null-context rule. Thus $L$ is a null-context splicing language.

**(c) $\Rightarrow$ (a):** By Proposition 2.3, $L$ is a null-context splicing language, which implies $L$ is local. $\square$

## 4.4    Conclusion

This chapter presents six theorems on concepts of constant and local. Also presented are three theorems on conditions of splicing rules which are equivalent to each other.

# CHAPTER 5

## SIMPLE SPLICING SYSTEMS

### 5.1 Introduction

This chapter highlights some examples of the maximal firm factors of a word. Besides, simple splicing system is discussed using the automata concept. Another concept used in splicing system called the solid code is also discussed.

### 5.2 Maximal Firm Factors of a Word

Following are some examples of the maximal firm factors of a word.

**Example 5.1.**

Let $A = \{a, b\}$, $B = \{b\}$, $I$ will be left unspecified so far. There is only one rule, namely $r = (b, \lambda, b, \lambda)$. A word $w$ in $A^*$ is firm with respect to $r$ if (and only if) $w$ is in $a^*$. The maximal firm factors of the word *aabaaabaaba* are indicated via underscores: <u>aab</u><u>aaab</u><u>aab</u><u>a</u>.

**Example 5.2.**

Let $A = \{a, b, c, d, e\}$, $B = \{b, d\}$, $I$ will be unspecified so far. There are only two rules, namely $r = (b, \lambda, b, \lambda)$ and $r' = (d, \lambda, d, \lambda)$. These two rules can be abbreviated as $b$ and $d$ respectively. A word $w$ in $A^*$ is firm with respect to $r$ if $w$ in $\{a, c, d, e\}^*$. A word $w$ in $A^*$ is firm with respect to $r'$ if $w$ in $\{a, b, c, e\}^*$. A word $w$ in $A^*$ is firm with respect to $R = \{r, r'\}$ iff $w$ is in $\{a, c, e\}^*$. The maximal firm factors of the word *aabacadcccbceeab* are indicated via underscores: <u>aa</u>b<u>aca</u>d<u>ccc</u>b<u>cee</u>ab.

**Example 5.3.**

Let $A = \{a, b\}$, $B = \{b\}$ and $I = \{aaabaa, aba\}$. The language generated by $(A, B, I)$ is $L = \{aaabaa, aba, aaaba, abaa\}$. The maximal firm factors of words in $L$ are indicated via underscores: $\{\underline{aaa}b\underline{aa}, \underline{a}b\underline{a}, \underline{aaa}b\underline{a}, \underline{a}b\underline{aa}\}$.

**Example 5.4.**

Let $A = \{a, b, c, d, e\}$, $B = \{b, d\}$ and $I = \{aabcedccbee\}$. The language generated by $(A, B, I)$ is $L' = aab(cedccb)^*ee$. The maximal firm factors of words in $L'$ are indicated via underscores: <u>aa</u>b(<u>ced</u>c<u>c</u>b)*<u>ee</u>.

**Example 5.5.**

Let $A = \{a, b, c, d, e\}$, $B = \{b, d\}$ and $I = \{abcde, edcba\}$. The language generated by $(A, B, I)$ is $L'' = \{a(bcdc)^*ba, e(dcbc)^*de, a(bcdc)^*bcde, e(dcbc)^*dcba\}$. The maximal firm factors of words in $L''$ are indicated via underscores: $\{\underline{a}(b\underline{c}d\underline{c})^*b\underline{a}, \underline{e}(d\underline{c}b\underline{c})^*d\underline{e}, \underline{a}(b\underline{c}d\underline{c})^*b\underline{c}d\underline{e}, \underline{e}(d\underline{c}b\underline{c})^*d\underline{c}b\underline{a}\}$.

### 5.3    Simple Splicing System

Following is the definition of a simple splicing system.

**Definition 5.1: (Simple Splicing System)**

A simple splicing system consists of a finite alphabet A, a subset $B$ of $A$ (often called the marked symbols in $A$), a finite set of splicing rules $R = \{(b, \lambda, b, \lambda):$ $b$ in $B\}$, and a finite set $I$ of initial words in $A^*$. Because of the extremely simple nature of the rule set $R$, which is determined completely by listing the elements of the subset $B$, a simple splicing system is denoted in the simplified form $(A, B, I)$. Simple splicing systems are always and automatically reflexive and transitive.

In [7], the simple splicing systems are fitted into a hierarchy that starts off $S_{-1}H$, $S_0H$, $S_1H = SH$, $S_2H$, $S_3H$,... with the union of this nest of language families being the classical family of all strictly locally testable language (SLT) which is equivalent to null context splicing (NCH) language.

$S_{-1}H$ languages are merely the finite languages which can be represented by $L = L (A, L, \text{empty})$. Since there are no 'rules' to do any cutting, each of the words in $L$ can be regarded as a firm word since none can be cut.

$S_0H$ languages are merely the language $B^*$, where $B$ ranges over the subsets of $A$. Thus these languages can be represented by $L = L (A, B, \{\lambda\})$. Thus, no word of length $>= 2$ is firm, since it can certainly be cut (anyplace in the word). As for each individual letter, $b$ in $B$, each such letter is firm, or no word (or letter) is firm. Probably it will be best to say that there are no firm words.

The first family that is worth considering is $S_1H = SH$. Languages in $SH$ are generated like $L = L (A, I, R)$ where $R$ is a subset of $A$. Any word of he form $xry$ where $r$ is in $R$ can be cut as follows: for any $xry$ and $urv$, $xrv$ and $ury$ can be formed. The cutting can be considered as $x|ry$, $u|rv$ or $xr|y$, $ur|v$. We shall standardize that the

cutting goes at the end of the site: *xry* and *urv* get cut into *xr|y*, *ur|v* and then the pieces get pasted to give *xrv* and *ury*.

A word *w* in *A\** that contains no letter in *R* cannot be cut at all, thus it is considered to be firm. If *w = xry* with neither *x* nor *y* null then *w* can be cut into two pieces and should therefore be not firm. For *SH* languages, a word *w* is firm if and only if it contains no element of *R*.

*SH* systems are just the same as splicing systems where all the rules have the form (*a*, $\lambda$, *a*, $\lambda$) which can be written for convenience by merely mentioning that '*a* is in *R*'. $S_2H$ also allows rules of the form (*ab*, $\lambda$, *ab*, $\lambda$) which can be written for convenience by merely mentioning that '*ab* is in *R*'.

For the *SH* language *L* we will say that a word *w* in *L* is firm if it contains no occurrence of a letter in *R*. Recall that by a factor of a word *w* in *A\** is meant any word *y* for which there are words *x* and *z* for which *w = xyz*, where *x*, *z* in *A\**. So a maximal firm subword of a word *w* should be a factor *y* of *w* in which no letter in *R* occurs in *y* but if *w = uayz* with *a* in *A*, then *a* is in *R* and if *w = xybv* with *b* in *A*, then *b* is in *R*.

If *w = uayz* and *a* is not in *R*, then *ay* would be a firm factor of *w* that is one letter longer than *y* and so *y* wouldn't be a maximal firm subword of *w*. If *w = xybv* and *b* is not in *R*, then *yb* would be a firm factor of *w* that is one letter longer than *y* and so *y* wouldn't be a maximal firm subword of *w*. A word *y* is a maximal firm subword of *w* as long as *y* can get, and stay, inside *w*. In other words, a maximal firm subword of *w* is a longest possible factor of *w* that contains no letter in *R*.

Next, the simple splicing system automata concept, known as *SH*-automata concept, is discussed.

## 5.4    *SH*-automata Concept

Below are some examples on maximal firm subwords and the simplest non-deterministic automaton that recognizes (*A*, *I*, *R*):
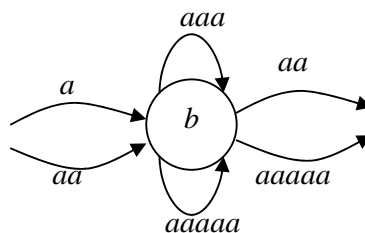
**Example 5.6.**

Suppose $A = \{a, b\}$, $I = \{aabaaaaabaaabaa\}$ and $R = \{b\}$. The maximal firm subwords of the only word in *I* are: *aa*, *aaaaa*, *aaa*.  The simplest non-deterministic automaton that recognizes *L* (*A*, *I*, *R*):
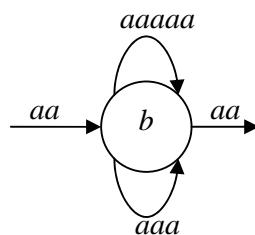


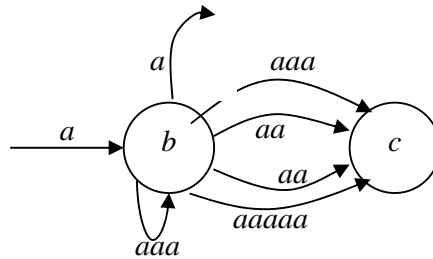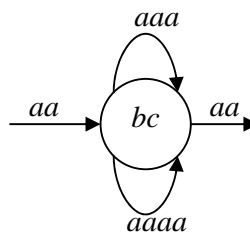The regular expression for this language is *aab*(*aaab*+*aaaaab*)**aa*.

**Example 5.7.**

Suppose $A = \{a, b\}$, $I = \{abaaabaaabaa, aabaaaaabaaaaabaaaaa\}$ and $R = \{b\}$.    Maximal firm subwords of *I*: *a*, *aaa*, *aa*, *aaaaa*.    The simplest non-deterministic automaton that recognizes *L* (*A*, *I*, *R*):



Notice that we will get the same language generated if *I* were {*abaaabaa*, *aabaaaaabaaaaa*} or if *I* were {*abaaaaa*, *aabaaaaabaaabaa*}.    The regular expression for this language is (*a*+*aa*)*b*(*aaab*+*aaaaab*)**(aa*+*aaaaa*).

**Example 5.8.**

Suppose $A = \{a, b, c\}$, $I = \{abaacaaaaaba, abaaabaaacaaba\}$ and $R = \{b, c\}$. Maximal firm subwords of $I$: $a$, $aa$, $aaaaa$, $aaa$. The simplest non-deterministic automaton that recognizes $L(A, I, R)$:



The regular expression for this language is $ab[aaab+(aac+aaac)(aab+aaaaab)]*a$.

**Example 5.9.**

Suppose $A = \{a, b, c\}$, $I = \{abaaabbaabbba\}$ and $R = \{b, c\}$. Maximal firm subwords of $I$: $a$, $aaa$, $aa$. The simplest non-deterministic automaton that recognizes $L(A, I, R)$:
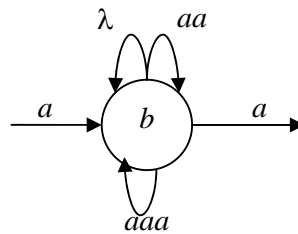


The regular expression for this language is $ab(aaab+b+aab)*a$.

***Note:*** $(a+b)* = A*$

$a*b*$ = those words in which ALL occurrences of an '$a$' come before ALL occurrences of '$b$'.

Following is an example of a $S_2H$ system:

**Example 5.10.**

Suppose $A = \{a, b, c\}$, $I = \{aabcaaabcaaaabcaa\}$, $R = \{bc\}$. This is not really more complicated than an *SH* system. The rule *bc* doesn't occur in any complicated or over-lapping way and in fact neither a *b* nor a *c* even occurs separately. So this $S_2H$ is hardly different from the *SH* system where $A = \{a, d\}$, $I = \{aadaaadaaaadaa\}$, $R = \{d\}$. This language is a homomorphic image of the *SH* language *aad(aaad+aaaad)*aa* under the function $f(a) = a$, $f(d) = bc$. The appropriate $S_2H$ automaton that recognizes $L(A, I, R)$:



The regular expression for this $S_2H$ language is *aabc(aaabc+aaaabc)*aa*.

One trip through the *SH* – automaton is an arbitrary walk through the graph entering via an entrance following any path until exit. The language is an infinite set. We can generate a lot of different words from an *SH*-automaton, but *SH*-automaton defines exactly one language which is exactly the language generated defined by the simple splicing language.

Taking the *SH*-automata concept, which is a short compact way of encoding normal non-deterministic automata – in the special case of *SH* systems, the maximal firm subwords of the initial words of an *SH* systems serve as the labels for the associated *SH*-automaton.

## 5.5    Solid Code

This section discusses on solid code and several examples related to it.

**Definition 5.2 (Solid Code):**

A set $S$ of words in $A^*$ is a solid code if

(1) $w = xyz$ can hold with both $w$ and $y$ in $S$ only when $x$ and $z$ are null.

(2) $xy$ in $S$ and $yz$ in $S$ can hold only if $y$ is null.

Or less formally,

(1) no word in $S$ is a subword of any other word in $S$;

(2) no two distinct words in $S$ overlap non-trivially.

Below are some examples of solid and non solid codes:

**Example 5.11.**

Let $A = \{a, b, c, d, e, f, g\}$. $S = \{a, b, c, d\}$ is a solid code. $S = \{abc, abbc,$ $abbbc, abbbbc, abbbbbc\}$ is solid. $S = ab^*c$ is an infinite solid code. $T = \{cab, ac\}$ is not solid, since $cab$ and $ac$ are overlapping non-trivially. $T = \{ab, ac, abc, dddd\}$ is not solid, since $ab$ is a subword of $abc$. $T = \{abc, cba\}$ is not solid, since $abc$ and $cba$ are overlapping non-trivially. $T = \{abcdde, cd\}$ is not solid, since $cd$ is a subword of $abcdde$.

When $R$ is a solid code, the $S_kH$ system $(A, I, R)$ where $k$ is the length of the longest word in $R$, may be identified with an appropriate $S_1H$ system, and thereby completely understood in terms of the maximal firm words of the $S_1H$ system and the words in the solid code $R$.

**Example 5.12.**

Suppose $A = \{a, b, c, d\}$, $I = \{bbadcddacbcabdcd, dabcaddcddbcab\}$, $R = \{cdd, bca\}$ which is a solid code. $I = \{bbad/cdd/ac/bca/bdcd, da/bca/dd/cdd/bca/b\}$ illustrates how, when $R$ is a solid, words come apart naturally and uniquely into code

words and 'words-in-between'. Since the maximum length of a word in $R$ is three, we have specified an $S_3H$ system. However, due to the nice clean parsing into words in $R$ and words-between, such an $S_3H$ is really pretty much the same as an easily specified $S_1H$ system over a different alphabet.

Suppose that we choose a new letter not in $A$ for each word in $R$. Let us use $p$ and $q$. Then form the $S_1H$ system by using the parsed version of $I$ given above: $A' = \{a, b, c, d, p, q\}$, $I' = \{bbadpacqbdcd, daqddpqb\}$, $R' = \{p, q\}$. This is an $S_1H$ system and we can see that it has 'maximal firm words': $bbad$, $ac$, $bdcd$, $da$, $dd$, $b$. The language $L(A, I, R)$ is the image of the language $L(A', I', R')$ under the homomorphism of $h : (A') \to A^*$ generated by the one-to-one function $h: A' \to \{a, b, c, d, cdd, bca\}$ defined by $h(a) = a$, $h(b) = b$, $h(c) = c$, $h(d) = d$, $h(p) = cdd$, $h(q) = bca$.

**Example 5.13.**

Suppose $A = \{a, b, c, d\}$, $I = \{dabccacdbabcacdb, acacdcabb\}$, $R = \{ab, cacd\}$ which is a solid code. $I = \{d/ab/c/cacd/b/ab/cacd/b, a/cacd/c/ab/b\}$ illustrates how, when $R$ is a solid, words come apart naturally and uniquely into code words and 'words-in-between'. Since the maximum length of a word in $R$ is four, we have specified an $S_4H$ system. However, due to the nice clean parsing into words in $R$ and words-between, such an $S_4H$ is really pretty much the same as an easily specified $S_1H$ system over a different alphabet.

Suppose that we choose a new letter not in $A$ for each word in $R$. Let us use $p$ and $q$. Then form the $S_1H$ system by using the parsed version of $I$ given above: $A' = \{a, b, c, d, p, q\}$, $I' = \{dpcqbpqb, aqcpb\}$, $R' = \{p, q\}$. This is an $S_1H$ system and we can see that it has 'maximal firm words': $d$, $c$, $b$, $a$. The language $L(A, I, R)$ is the image of the language $L(A', I', R')$ under the homomorphism of $h : (A') \to A^*$ generated by the one-to-one function $h: A' \to \{a, b, c, d, ab, cacd\}$ defined by $h(a) = a$, $h(b) = b$, $h(c) = c$, $h(d) = d$, $h(p) = ab$, $h(q) = cacd$.

**Example 5.14.**

Suppose $A = \{a, b, c, d\}$, $I = \{dacbba, bacbbacd\}$ and $R = \{ac, acb\}$. $R$ is not solid since *ac* is a subword of *acb*.

**Example 5.15.**

Suppose $A = \{a, b, c, d\}$, $I = \{abcdddacba, bbcdaadacbb\}$ and $R = \{bc, dacb\}$. $R$ is not solid since *bc* and *dacb* overlap non-trivially.

## 5.6    Conclusion

This chapter discussed the maximal firm factors of a word, the simple splicing system, *SH*-automata concept and concepts of solid code. This will allow languages $L$ $(A, I, R)$ to be reduced to $S_1H$ languages by replacement of the occurrences of words in $R$ by letters of an alphabet of new letters.

# CHAPTER 6

## MOLECULAR CONSIDERATIONS

### 6.1     Introductions

This chapter lists some molecular considerations of a splicing system.  Firm words in a wet-lab experiment are also discussed.

### 6.2     Molecular Considerations

This section lists a few of the actual molecular considerations for the molecules that will arise from a given initial sets of strings using some chosen restriction enzymes.

**Example 6.1:**

List all the sequences of the ds-DNA molecules (without sticky ends) that can arise as ds-DNA molecules having the sequence

5′-C$^{19}$GGATCCC$^{17}$-3′
3′-G$^{19}$CCTAGGG$^{17}$-5′

are added to an aqueous solution containing *Bam*HI and a ligase.

**All molecules:**

1) original molecule, that is

$5'$-$C^{19}$GGATCCC$^{17}$-$3'$
$3'$-$G^{19}$CCTAGGG$^{17}$-$5'$

2) $5'$-$C^{19}$GGATCCG$^{19}$-$3'$
$3'$-$G^{19}$CCTAGGC$^{19}$-$5'$

3) $5'$-$G^{17}$GGATCCC$^{17}$-$3'$
$3'$-$C^{17}$CCTAGGG$^{17}$-$5'$

**Example 6.2:**

Find all the sequences of the ds-DNA molecules that can arise as ds-DNA molecules having the sequence

$5'$-$G^{19}$GGATCCC$^{15}$GGATCCG$^{17}$-$3'$
$3'$-$C^{19}$CCTAGGG$^{15}$CCTAGGC$^{17}$-$5'$
are added to an aqueous solution containing *Bam*HI and a ligase.
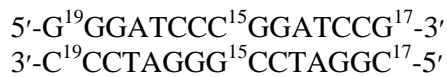
**All molecules:**

1) original molecule, that is

$5'$-$G^{19}$GGATCCC$^{15}$GGATCCG$^{17}$-$3'$
$3'$-$C^{19}$CCTAGGG$^{15}$CCTAGGC$^{17}$-$5'$

2) $5'$-$G^{19}$GGATCCG$^{17}$-$3'$
$3'$-$C^{19}$CCTAGGC$^{17}$-$5'$

3) $5'$-$G^{19}$GGATCCC$^{15}$GGATCCC$^{15}$GGATCCG$^{17}$-$3'$
$3'$-$C^{19}$CCTAGGG$^{15}$CCTAGGG$^{15}$CCTAGGC$^{17}$-$5'$

4) $5'$-$G^{19}$GGATCCC$^{19}$-$3'$
$3'$-$C^{19}$CCTAGGG$^{19}$-$5'$

5) $5'$-$C^{17}$GGATCCG$^{15}$GGATCCC$^{15}$GGATCCG$^{17}$-$3'$
$3'$-$G^{17}$CCTAGGC$^{15}$CCTAGGG$^{15}$CCTAGGC$^{17}$-$5'$

6)   5′-G$^{19}$GGATCCC$^{15}$GGATCCG$^{15}$GGATCCC$^{19}$-3′
     3′-C$^{19}$CCTAGGG$^{15}$CCTAG  GC$^{15}$CCTAGGG$^{19}$-5′


7)   5′-C$^{17}$GGATCCG$^{17}$-3′
     3′-G$^{17}$CCTAGGC$^{17}$-5′


8)   5′-G$^{19}$GGATCCG$^{15}$GGATCCC$^{19}$-3′
     3′-C$^{19}$CCTAGGC$^{15}$CCTAGGG$^{19}$-5′


9)   5′-C$^{17}$GGATCCC$^{15}$GGATCCG$^{17}$-3′
     3′-G$^{17}$CCTAGGG$^{15}$CCTAGGC$^{17}$-5′


and so on, where ALL molecules can be written in recursive form as:


1)   $5'\text{-}G^{19}\left(\dfrac{GGATCCC^{15}}{CCTAGGG^{15}}\right)^{*} \dfrac{GGATCCG^{17}\text{-3'}}{CCTAGGC^{17}\text{-5'}}$


2)   $5'\text{-}C^{17}\left(\dfrac{GGATCCG^{15}}{CCTAGGC^{15}}\right)^{*}\left(\dfrac{GGATCCC^{15}}{CCTAGGG^{15}}\right)^{*} \dfrac{GGATCCG^{17}\text{-3'}}{CCTAGGC^{17}\text{-5'}}$


3)   $5'\text{-}C^{17}\left(\dfrac{GGATCCG^{15}GGATCCC^{15}}{CCTAGGC^{15}CCTAGGG^{15}}\right)^{*} \dfrac{GGATCCG^{17}\text{-3'}}{CCTAGGC^{17}\text{-5'}}$


4)   $5'\text{-}G^{19}\left(\dfrac{GGATCCC^{15}}{CCTAGGG^{15}}\right)^{*}\left(\dfrac{GGATCCG^{15}}{CCTAGGC^{15}}\right)^{*} \dfrac{GGATCCC^{19}\text{-3'}}{CCTAGGG^{19}\text{-5'}}$


5)   $5'\text{-}G^{19}\left(\dfrac{GGATCCC^{15}GGATCCG^{15}}{CCATGGG^{15}CCATGGC^{15}}\right)^{*} \dfrac{GGATCCC^{19}\text{-3'}}{CCATGGG^{19}\text{-5'}}$


**Example 6.3:**


   Find all the sequences of the ds-DNA molecules that can arise as ds-DNA molecules having the sequence
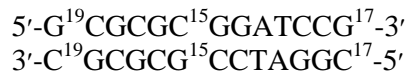
5′-G$^{19}$CGCGC$^{15}$GGATCCG$^{17}$-3′
3′-C$^{19}$GCGCG$^{15}$CCTAGGC$^{17}$-5′
are added to an aqueous solution containing *Bam*HI, *Bst*UI, and a ligase.

**All molecules:**

1)     Original molecule, that is

       5′-G$^{19}$CGCGC$^{15}$GGATCCG$^{17}$-3′
       3′-C$^{19}$GCGCG$^{15}$CCTAGGC$^{17}$-5′

2)     5′-G$^{19}$CGCGC$^{19}$-3′
       3′-C$^{19}$GCGCG$^{19}$-5′

3)     5′-C$^{17}$GGATCCG$^{15}$CGCGC$^{15}$GGATCCG$^{17}$-3′
       3′-G$^{17}$CCTAGGC$^{15}$GCGCG$^{15}$CCTAGGC$^{17}$-5′

4)     5′-G$^{19}$CGCGC$^{15}$GGATCCG$^{15}$CGCGC$^{19}$-3′
       3′-C$^{19}$GCGCG$^{15}$CCTAGGC$^{15}$GCGCG$^{19}$-5′

5)     5′-C$^{17}$GGATCCG$^{17}$-3′
       3′-G$^{17}$CCTAGGC$^{17}$-5′

and so on, where ALL molecules can be written in recursive form as:

1)     5'-G$^{19}$ $\left(\text{CGCGC}^{15}\text{GGATCCG}^{15}\right)^{*}$ CGCGC$^{15}$GGATCCG$^{17}$-3'
       3'-C$^{19}$ $\left(\text{GCGCG}^{15}\text{CCATGGC}^{15}\right)$ GCGCG$^{15}$CCATGGC$^{17}$-5'

2)     5'-C$^{17}$ $\left(\text{GGATCCG}^{15}\text{CGCGC}^{15}\right)^{*}$ GGATCCG$^{17}$-3'
       3'-G$^{17}$ $\left(\text{CCTAGGC}^{15}\text{GCGCG}^{15}\right)$ CCTAGGC$^{17}$-5'

3)     5'-G$^{19}$ $\left(\text{CGCGC}^{15}\text{GGATCCG}^{15}\right)^{*}$ CGCGC$^{19}$-3'
       3'-C$^{19}$ $\left(\text{GCGCG}^{15}\text{CCTAGGC}^{15}\right)$ GCGCG$^{19}$-5'

**Example 6.4:** Give splicing models, with alphabet $A = \{ a, c, g, t \}$, appropriate for
                   Examples 6.2 and 6.3.

For example 6.2, the appropriate splicing model for the representation of the generative activity of *Bam*HI and a ligase acting on molecules having the sequence $\{g^{19}ggatccc^{15}ggatccg^{17}\}$ is $S = (R, I)$ where $R = \{r\}$ and $I = \{m, m'\}$, where

$r = (g, gatcc; g, gatcc)$,

$m = g^{19}ggatccc^{15}ggatccg^{17}$,

$m' = c^{17}ggatccg^{15}ggatccc^{19}$.

For example 6.3, the appropriate splicing model for the representation of the generative activity of *Bam*HI, *Bst*UI and a ligase acting on molecules having the sequence $\{g^{19}cgcgc^{15}ggatccg^{17}\}$ is $S = (R, I)$ where $R = \{r, r_1, r_2, r_3\}$ and $I = \{m, m'\}$, where

$r = (cg, cg; cg, cg)$ ,

$r_1 = (g, gatcc; g, gatcc)$,

$r_2 = (cg, cg; g, gatcc)$,

$r_3 = (g, gatcc; cg, cg)$,

$m = g^{19}cgcgc^{15}ggatccg^{17}$,

$m' = c^{17}ggatccg^{15}cgcgc^{19}$.

## 6.3    Firm Words in Wet-Lab Experiment

There should be four distinct firm words for molecules, since when we deal with ds-DNA molecule that is not a DNA palindrome, we have to list in two ways but rotating one form through 180 degrees.  When we denote dsDNA molecules on a line (as a word or as a string) running from left to right, the dsDNA molecules that these strings represent are free to turn every way in a test tube.

In an experiment, the original molecule can be written on a line as a string in two distinct ways.  When the one molecule (with two string representations) is cut into two pieces each of these pieces has two string representations.   So in the word model (or word encoding) one might get confused if one does not understand this fundamental distinction between a dsDNA molecule and its (usually two distinct)

word representation(s). In fact, to show the splicing operation one view two of the original dsDNA molecules in opposite orientations to see the splice that creates the one long and the one short new dsDNA molecule. Two molecules viewed in the same orientation, when spliced, merely produce the two original molecules, each of the original length.

**6.3    Conclusion**

In this chapter, we showed that the result of a wet-lab splicing system can be predicted using the theoretical splicing model.

# CHAPTER 7

## CONCLUSION AND SUGGESTIONS

### 7.1 Conclusion

In this report, chapter 1 serves as an introduction to this research. Literature review is given in chapter 2, in which the splicing model used in this research is the Head splicing model. In chapter 3, strictly locally testable language is discussed, and theorems regarding to it are proved. In chapter 4, concepts of constant and local are presented, and those theorems related to them are proved. Maximal firm factors of a word are also discussed in chapter 5, which leads to the discussion of simple splicing system. *SH*-automata and the concept of solid code are used to illustrate the different simple splicing systems. Lastly, several molecular considerations are discussed in chapter 6 in which theoretical splicing system can be used to predict the actual biological splicing system.

### 7.2 Suggestions

Splicing systems can be viewed as a simple splicing system using the concept of constant, which can be further showed through various examples. More theorems related to simple splicing system can be proved. Actual wet-lab experiment can be carried out in order to view the result of an actual splicing system as compared to the mathematical model corresponding to it.

# REFERENCES

1. T. Head. Formal language theory and DNA: An analysis of the generative capacity of specific recombinant behaviours. *Bull. Math. Biology.* 1987. 49: 737-759.

2. Gh. Paun, G. Rozenberg and A. Salomaa. Computing by splicing. *Theoret. Comput. Sci.* 1996. 168: 321-336.

3. D. Pixton. Regularity of splicing languages. *Discrete Appl. Math.* 1996. 69: 101-124.

4. P. Bonizzoni, C. Ferretti, G. Mauri and R. Zizza. Separating some splicing models. *Information Processing Letters.* 2001. 79: 255-259.

5. T. Head. Splicing languages generated with one sided context. In: *Computing with Bio-Molecules – Theory and Experiments (G. Paun, ed.).* Singapore: Springer-Verlag. 269-282; 1995.

6. E. Goode and D. Pixton. Recognizing splicing languages: Syntactic monoids and simultaneous pumping.

7. T. Head. Splicing representations of strictly locally testable languages. *Discrete Applied Mathematics.* 1998. 87:139-147.

8. R. W. Gatterdam. Splicing systems and regularity. *International J. Computer. Math.* 1989. 31: 63-67.

9. T. Head. Finitely generated languages and positive data. *Romanian Journal of Information Science and Technology.* 2002. 5(1-2): 127-136.

10. E. Laun. *Constants and splicing systems*. PhD Dissertation. Binghamton University; 1999.