Triangle: Language, Literature, Computation, n. 6, 2011
Publicacions Universitat Rovira i Virgili · ISSN: 2013-939X
https://revistes.urv.cat/index.php/triangle

On Special forms of Splicing on Arrays and Graphs

K.G. Subramanian[1], A. Roslin Sagaya Mary[2], and P. Helen Chandra[3]

[1] School of Mathematical Sciences
Universiti Sains Malaysia
11800 Penang, Malaysia
E-mail: kgsmani1948@yahoo.com
[2] GRLMC
Universitat Rovira I Virgili
1 Plaï£¡a Imperial Tarraco, 43005 Tarragona, Spain
E-mail: anthonath.roslinsagayamary@estudiants.urv.cat
[3] Department of Mathematics
Jayaraj Annapackiam College for Women
Periyakulam, Theni 625 601 India

Summary. Tom Head (1987), in his pioneering work on formal language theory applied to DNA computing, introduced a new operation of splicing on strings, while proposing a model of certain recombination behaviour of DNA molecules under the action of restriction enzymes and ligases. Since then this operation has been studied in great depth giving rise to a number of theoretical results of great interest in formal language theory. Extension of this operation of splicing to higher dimensional structures such as circular words, arrays, trees and graphs have been proposed in the literature. Here we examine the effect of certain specific forms of the splicing operation applied to arrays and graphs

1 Introduction

There has been a lot of interest among researchers in the study of formal language theory applied to DNA computing. A specific model of DNA recombination is the splicing operation which consists of "cutting" DNA sequences and then "pasting" the fragments again, under the action of restriction enzymes and ligases. In [7], Tom Head defined splicing systems motivated by this behaviour of DNA sequences. The splicing systems make use of a new operation, called splicing on strings of symbols. These systems are a new class of generative systems, intended to model certain recombinant behavior of DNA molecules and are of current interest and study. Theoretical investigation of splicing on strings has been extensively done by different researchers [8].

Extension of the splicing operation to graphs has been proposed by Freund [4]. Relationship between graph splicing languages of Freund [4] and Hyperedge replacement graph languages [6] is examined in [3]. Sakakibara and Ferretti [13] have introduced and studied splicing of tree structures. Krithivasn [9] has considered a different kind of splicing of graphs.

On the other hand, in syntactic approaches to generation and recognition of picture patterns, considered as arrays of symbols, several two-dimensional grammars have been proposed and studied [5, 12, 15, 16, 17, 18, 19, 20] extending and generalizing the techniques of formal string language theory. Splicing of arrays structures which could be thought of as graphs on grid structures has also been recently considered [2,10]. A simple but effective method of splicing on images of rectangular arrays is introduced in [2] as an extension of the operation of splicing on strings.

Mateescu etal [11] introduced a special kind of a splicing rule, called simple splicing rule and investigated the effect of this type of rule on words. This notion of simple splicing has been examined in [1] for circular words. As an application of the concept of simple splicing, it is natural to extend this special form of splicing to higher dimensional structures such as graphs, trees and arrays.

Here we introduce and examine the analogue of simple splicing for picture patterns of rectangular arrays in the context of domino splicing rules of [2]. We compare the resulting systems called Simple array splicing systems with two other picture describing models. We then introduce the notion of simple splicing on trees which allows us to obtain a set of trees. The resulting system, called Simple Tree splicing system, describes the derivation

trees of context-free grammars. We also consider the string language consisting of words associated in a standard way with trees . It is known that simple splicing yields only regular languages [11]. It is of interest to note that, in contrast to the string case, the picture language class obtained by the simple array splicing systems is incomparable with the two other regular-like picture classes and in the tree case, the associated string languages are context-free languages.

2 Simple Array Splicing Systems

We first recall the notion of simple splicing on words [11]. For notions of language theory we refer to [14].

Definition 1. *Let V be an alphabet. $\$, \#$ are two special symbols, not in V. A simple splicing rule over V is a string of the form $r = a\# \lambda \$ a\# \lambda$, where $a \in V$. For such a rule r and strings $x, y, z \in V^*$, we write $(x, y) \vdash_r z$ if and only if $x = x_1 a x_2$, $y = y_1 a y_2$, $z = x_1 a y_2$ for some $x_1, x_2, y_1, y_2 \in V^*$. We say that z is obtained by splicing x, y, as indicated by the rule r.*

A ***simple splicing scheme*** is a pair $\sigma = (V, R)$, where V is an alphabet and R is a set of simple splicing rules. A simple splicing scheme is also referred to as a simple H scheme [11]. For a given simple splicing scheme $\sigma = (V, R)$ and a language $L \subseteq V^*$, we define $\sigma(L) = \{z \in V^* \ / \ (x, y) \vdash_r z$ for some $x, y \in L$, $r \in R \}$

We now introduce the notions of simple domino splicing rules and simple array splicing Systems (SAS). These systems are a special kind of H array splicing systems introduced and studied in [2].

Definition 2. *Let V be an alphabet. $\#, \$$ are two special symbols, not in V. A column domino over V is of the form $\boxed{\begin{matrix} a \\ b \end{matrix}}$ and a row domino is of the form $\boxed{a}\boxed{b}$ for some $a, b \in V$. Both a, b can be λ or $\#$.*

A ***simple column or row domino splicing rule*** over V is of the form $r = \alpha \ \# \ \lambda \ \$ \ \beta \ \# \ \lambda$ where both α and β are column dominoes or both row dominoes.We refer to α, β as the first and third dominoes of r respectively.

Given two arrays X and Y of sizes $m \times p$ and $m \times q$ respectively,

$$X = \begin{matrix} a_{11} & \cdots & a_{1,j} & \cdots & a_{1p} \\ a_{21} & \cdots & a_{2,j} & \cdots & a_{2p} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{m,j} & \cdots & a_{mp} \end{matrix},$$

$$Y = \begin{matrix} b_{11} & \cdots & b_{1,k} & \cdots & b_{1q} \\ b_{21} & \cdots & b_{2,k} & \cdots & b_{2q} \\ \vdots & \ddots & \cdots & \ddots & \vdots \\ b_{m1} & \cdots & b_{m,k} & \cdots & b_{mq} \end{matrix}$$

$a_{ir}, b_{is} \in V$, for $1 \leq i \leq m$, $1 \leq r \leq p$, $1 \leq s \leq q$. Let $X^{\#}$ and $Y^{\#}$ be bordered arrays of sizes $(m+2) \times (p+2)$, $(m+2) \times (q+2)$ obtained by surrounding X and Y with # symbols, as shown below.

$$X^{\#} = \begin{matrix} \# & \# & \cdots & \# & \cdots & \# & \# \\ \# & a_{11} & \cdots & a_{1,j} & \cdots & a_{1p} & \# \\ \# & a_{21} & \cdots & a_{2,j} & \cdots & a_{2p} & \# \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ \# & a_{m1} & \cdots & a_{m,j} & \cdots & a_{mp} & \# \\ \# & \# & \cdots & \# & \cdots & \# & \# \end{matrix},$$

$$Y^{\#} = \begin{matrix} \# & \# & \cdots & \# & \cdots & \# & \# \\ \# & b_{11} & \cdots & b_{1,k} & \cdots & b_{1q} & \# \\ \# & b_{21} & \cdots & b_{2,k} & \cdots & b_{2q} & \# \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ \# & b_{m1} & \cdots & b_{m,k} & \cdots & b_{mq} & \# \\ \# & \# & \cdots & \# & \cdots & \# & \# \end{matrix}$$

Here we refer to the top and bottom rows of $\#'s$ as the 0^{th} and $m+1^{st}$ row. Similarly for the leftmost and rightmost columns of $\#'s$.
We write $(X, Y) \ |^{\Phi} \ Z$ if there is a sequence of simple column splicing rules $r_0, r_1, r_2, ...r_m$ (not necessarily all different) such that

$$r_i = \left[\begin{matrix} a_{i,j} \\ a_{i+1,j} \end{matrix}\right] \ \# \ \left[\begin{matrix} \lambda \\ \lambda \end{matrix}\right] \ \$ \ \left[\begin{matrix} b_{i,k} \\ b_{i+1,k} \end{matrix}\right] \ \# \ \left[\begin{matrix} \lambda \\ \lambda \end{matrix}\right]$$

and $a_{i,j} = b_{i,k}$

for all $i,\ 0 \leq i \leq m$ and for some $j,k\ 0 \leq j \leq p+1,\ 0 \leq k \leq q+1$ and

$$Z^{\#} = \begin{array}{cccccccc} \# & \# & \cdots & \# & \# & \cdots & \# & \# \\ \# & a_{11} & \cdots & a_{1,j} & b_{1,k} & \cdots & b_{1q} & \# \\ \# & a_{21} & \cdots & a_{2,j} & b_{2,k} & \cdots & b_{2q} & \# \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \# & a_{m1} & \cdots & a_{m,j} & b_{m,k} & \cdots & b_{mq} & \# \\ \# & \# & \cdots & \# & \# & \cdots & \# & \# \end{array}$$

In other words, we can imagine that a 2×1 window is moved down the j^{th} column of $X^{\#}$. The sequence of dominoes collected are the first dominoes of the rules $r_0, r_1, r_2, ..., r_m$ (not all necessarily different). Likewise for the k^{th} column of $Y^{\#}$ except that the dominoes collected are the third dominoes of the rules. When such rules exist in the system, the simple column splicing of the arrays X and Y amounts to the array $X^{\#}$ being vertically "cut" after j^{th} column and the array $Y^{\#}$ after k^{th} column and the resulting left subarray of $X^{\#}$ "pasted" (column catenated) with the right subarray of $Y^{\#}$ to yield $Z^{\#}$. We now say that Z is obtained from X and Y by ***simple domino column splicing in parallel***, where Z is $Z^{\#}$ with surrounding # symbols deleted.

We can similarly define simple row splicing operation of two arrays U and V of sizes $p \times n$ and $q \times n$, using simple domino row splicing rules to yield an array W. We write $(U, V) \ |\!\underline{\Theta}\ W$. As done for the column splicing of arrays, we can imagine 1×2 windows being moved over respective rows. The row splicing of the arrays U and V can be thought of as $U^{\#}$ being horizontally "cut" below the r^{th} row and $V^{\#}$ below s^{th} row and the upper subarray of $U^{\#}$ "pasted" (row catenated) to the lower subarray of $V^{\#}$ to yield $W^{\#}$. We now say that W is obtained from U and V by ***simple domino row splicing in parallel***, where W is $W^{\#}$ with # symbols deleted.

We now introduce the main notion of Simple Array Splicing Systems.

Definition 3. *A **Simple array splicing scheme** is a triplet $\Gamma = (V, R_c, R_r)$ where V is an alphabet, R_c = a finite set of simple domino column splicing rules, and R_r = a finite set of simple domino row splicing rules.*
For a given Simple array scheme $\Gamma = (V, R_c, R_r)$ and a language $L \subseteq V^{**}$, we define

$\Gamma(L) = \{Z \in V^{**} / (X^{\#}, Y^{\#}) \ |^{\Phi} \ Z^{\#}$ *or* $(X^{\#}, Y^{\#}) \ |^{\Theta} \ Z^{\#}$ *for some* $X, Y \in L\}$. *In other words,* $\Gamma(L)$ *consists of arrays obtained by column or row splicing any two arrays of L using the simple domino column or row splicing rules.*

A ***Simple array splicing system*** (SAS) is defined by $S = (\Gamma, I)$ where $\Gamma = (V, R_c, R_r)$ and I is a finite subset of V^{**}. The language of S is defined by $L(S) = \Gamma^*(I)$ and we call it a ***Simple array splicing language*** $(SASL)$ and denote the class of such languages by $\mathcal{L}(SASL)$.

We illustrate with an example.

Example 1. Let $V = \{x, a\}$, $I = \left\{ \begin{matrix} x\ x\ x\ x \\ x\ a\ a\ x \\ x\ a\ a\ x \\ x\ x\ x\ x \end{matrix} \right\}$,

$$R_c = \left\{ p_1 : \begin{matrix} x \\ a \end{matrix} \ \# \ \begin{matrix} \lambda \\ \lambda \end{matrix} \ \$_c \ \begin{matrix} x \\ a \end{matrix} \ \# \ \begin{matrix} \lambda \\ \lambda \end{matrix} \right.$$

$$p_2 : \begin{matrix} a \\ a \end{matrix} \ \# \ \begin{matrix} \lambda \\ \lambda \end{matrix} \ \$_c \ \begin{matrix} a \\ a \end{matrix} \ \# \ \begin{matrix} \lambda \\ \lambda \end{matrix}$$

$$p_3 : \begin{matrix} a \\ x \end{matrix} \ \# \ \begin{matrix} \lambda \\ \lambda \end{matrix} \ \$_c \ \begin{matrix} a \\ x \end{matrix} \ \# \ \begin{matrix} \lambda \\ \lambda \end{matrix}$$

$$p_4 : \begin{matrix} \# \\ x \end{matrix} \ \# \ \begin{matrix} \lambda \\ \lambda \end{matrix} \ \$_c \ \begin{matrix} \# \\ x \end{matrix} \ \# \ \begin{matrix} \lambda \\ \lambda \end{matrix}$$

$$p_5 : \begin{matrix} x \\ \# \end{matrix} \ \# \ \begin{matrix} \lambda \\ \lambda \end{matrix} \ \$_c \ \begin{matrix} x \\ \# \end{matrix} \ \# \ \begin{matrix} \lambda \\ \lambda \end{matrix} \left. \right\},$$

$$R_r = \left\{ q_1 : \begin{matrix} x & a \end{matrix} \ \# \ \begin{matrix} \lambda & \lambda \end{matrix} \ \$_r \ \begin{matrix} x & a \end{matrix} \ \# \ \begin{matrix} \lambda & \lambda \end{matrix} \right.$$

$$q_2 : \begin{matrix} a & a \end{matrix} \ \# \ \begin{matrix} \lambda & \lambda \end{matrix} \ \$_r \ \begin{matrix} a & a \end{matrix} \ \# \ \begin{matrix} \lambda & \lambda \end{matrix}$$

$$q_3 : \begin{matrix} a & x \end{matrix} \ \# \ \begin{matrix} \lambda & \lambda \end{matrix} \ \$_r \ \begin{matrix} a & x \end{matrix} \ \# \ \begin{matrix} \lambda & \lambda \end{matrix}$$

$$q_4 : \begin{matrix} \# & x \end{matrix} \ \# \ \begin{matrix} \lambda & \lambda \end{matrix} \ \$_r \ \begin{matrix} \# & x \end{matrix} \ \# \ \begin{matrix} \lambda & \lambda \end{matrix}$$

$$q_5 : \begin{matrix} x & \# \end{matrix} \ \# \ \begin{matrix} \lambda & \lambda \end{matrix} \ \$_r \ \begin{matrix} x & \# \end{matrix} \ \# \ \begin{matrix} \lambda & \lambda \end{matrix} \left. \right\}$$

L is the language consisting of pictures of the form in Figure 1, where white square is interpreted as a and black as x.

Fig. 1. Rectangles of a's surrounded by x's

Example 2. Let $V = \{x, .\}$, $I = \left\{ \begin{matrix} x\ .\ . \\ x\ .\ . \\ x\ x\ x \end{matrix} \right\}$,

$$R_c = \left\{ p_1 : \begin{bmatrix}\# \\ .\end{bmatrix} \# \begin{bmatrix}\lambda \\ \lambda\end{bmatrix} \$_c \begin{bmatrix}\# \\ .\end{bmatrix} \# \begin{bmatrix}\lambda \\ \lambda\end{bmatrix} \right.$$
$$p_2 : \begin{bmatrix}. \\ .\end{bmatrix} \# \begin{bmatrix}\lambda \\ \lambda\end{bmatrix} \$_c \begin{bmatrix}. \\ .\end{bmatrix} \# \begin{bmatrix}\lambda \\ \lambda\end{bmatrix}$$
$$p_3 : \begin{bmatrix}. \\ x\end{bmatrix} \# \begin{bmatrix}\lambda \\ \lambda\end{bmatrix} \$_c \begin{bmatrix}. \\ x\end{bmatrix} \# \begin{bmatrix}\lambda \\ \lambda\end{bmatrix}$$
$$\left. p_4 : \begin{bmatrix}x \\ \#\end{bmatrix} \# \begin{bmatrix}\lambda \\ \lambda\end{bmatrix} \$_c \begin{bmatrix}x \\ \#\end{bmatrix} \# \begin{bmatrix}\lambda \\ \lambda\end{bmatrix} \right\}$$

$$R_r = \left\{ q_1 : \begin{bmatrix}\#|x\end{bmatrix} \# \begin{bmatrix}\lambda|\lambda\end{bmatrix} \$_r \begin{bmatrix}\#|x\end{bmatrix} \# \begin{bmatrix}\lambda|\lambda\end{bmatrix} \right.$$
$$q_2 : \begin{bmatrix}x|.\end{bmatrix} \# \begin{bmatrix}\lambda|\lambda\end{bmatrix} \$_r \begin{bmatrix}x|.\end{bmatrix} \# \begin{bmatrix}\lambda|\lambda\end{bmatrix}$$
$$q_3 : \begin{bmatrix}.|.\end{bmatrix} \# \begin{bmatrix}\lambda|\lambda\end{bmatrix} \$_r \begin{bmatrix}.|.\end{bmatrix} \# \begin{bmatrix}\lambda|\lambda\end{bmatrix}$$
$$\left. q_4 : \begin{bmatrix}.|\#\end{bmatrix} \# \begin{bmatrix}\lambda|\lambda\end{bmatrix} \$_r \begin{bmatrix}.|\#\end{bmatrix} \# \begin{bmatrix}\lambda|\lambda\end{bmatrix} \right\}$$

L is the picture language consisting of all $m \times n$ arrays describing token L of $x's$.

Fig. 2. Array describing token L Token L of x's

Example 3. Let $V = \{a, b\}$, $I = \left\{ \begin{matrix} a & b \\ b & a \end{matrix} \right\}$,

$$R_c = \left\{ p_1 : \boxed{\begin{matrix} b \\ a \end{matrix}} \# \boxed{\begin{matrix} \lambda \\ \lambda \end{matrix}} \$ \boxed{\begin{matrix} b \\ a \end{matrix}} \# \boxed{\begin{matrix} \lambda \\ \lambda \end{matrix}} \right.$$

$$p_2 : \boxed{\begin{matrix} \# \\ b \end{matrix}} \# \boxed{\begin{matrix} \lambda \\ \lambda \end{matrix}} \$ \boxed{\begin{matrix} \# \\ b \end{matrix}} \# \boxed{\begin{matrix} \lambda \\ \lambda \end{matrix}}$$

$$p_3 : \boxed{\begin{matrix} a \\ \# \end{matrix}} \# \boxed{\begin{matrix} \lambda \\ \lambda \end{matrix}} \$ \boxed{\begin{matrix} a \\ \# \end{matrix}} \# \boxed{\begin{matrix} \lambda \\ \lambda \end{matrix}}$$

$$\left. p_4 : \boxed{\begin{matrix} a \\ b \end{matrix}} \# \boxed{\begin{matrix} \lambda \\ \lambda \end{matrix}} \$ \boxed{\begin{matrix} a \\ b \end{matrix}} \# \boxed{\begin{matrix} \lambda \\ \lambda \end{matrix}} \right\},$$

$$R_r = \left\{ q_1 : \boxed{a}\boxed{b} \# \boxed{\lambda}\boxed{\lambda} \$ \boxed{a}\boxed{b} \# \boxed{\lambda}\boxed{\lambda} \right.$$

$$q_2 : \boxed{b}\boxed{a} \# \boxed{\lambda}\boxed{\lambda} \$ \boxed{b}\boxed{a} \# \boxed{\lambda}\boxed{\lambda}$$

$$q_3 : \boxed{\#}\boxed{b} \# \boxed{\lambda}\boxed{\lambda} \$ \boxed{\#}\boxed{b} \# \boxed{\lambda}\boxed{\lambda}$$

$$\left. q_4 : \boxed{a}\boxed{\#} \# \boxed{\lambda}\boxed{\lambda} \$ \boxed{a}\boxed{\#} \# \boxed{\lambda}\boxed{\lambda} \right\}$$

L is the language consisting of all "chessboards" with even side-length. i.e. pictures of the form in 2.
The picture or pattern of Figure 2 can be represented by an array M, where

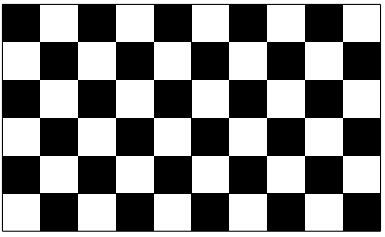

Fig. 3. Chessboard Pattern

$$M = \begin{matrix} a\ b\ a\ b\ a\ b\ a\ b\ a\ b \\ b\ a\ b\ a\ b\ a\ b\ a\ b\ a \\ a\ b\ a\ b\ a\ b\ a\ b\ a\ b \\ b\ a\ b\ a\ b\ a\ b\ a\ b\ a \\ a\ b\ a\ b\ a\ b\ a\ b\ a\ b \\ b\ a\ b\ a\ b\ a\ b\ a\ b\ a \end{matrix}$$

where 'a' stands for black and 'b' for white.

We now compare the generative power of Simple array splicing systems with other picture description models. The class LOC [5] of local picture array languages described by 2×2 "windows" and the class of two-dimensional right-linear languages described by 2RLG [5, 15] are two of the basic classes in the hierarchy of picture array describing grammars.

Theorem 1. *The classes LOC [5] of local array languages and $\mathcal{L}(SASL)$ of Simple array splicing languages are incomparable but not disjoint.*

Proof: The picture language M consisting of all $m \times n$ arrays $(m \geq 2, n \geq 2)$ describing token L of 1's is in LOC [2]. It is also described by a Simple array splicing system $S = (V, R_c, R_r, I)$ as in example 2.

It is known [5] that the picture language L of all rectangular arrays over $V = \{a\}$ with 3 columns is not in LOC. In fact, it is not possible to fix the number of columns only one symbol, as the block $\begin{bmatrix} a\ a \\ a\ a \end{bmatrix}$ can be moved without restriction on the columns. But it is generated by a Simple array splicing system where

$$I = \left\{ \begin{matrix} a\ a\ a \\ a\ a\ a \\ a\ a\ a \end{matrix} \right\},$$

$$R_r = \Big\{ q_1 : \boxed{a}\boxed{a} \quad \# \quad \boxed{\lambda}\boxed{\lambda} \quad \$ \quad \boxed{a}\boxed{a} \quad \# \quad \boxed{\lambda}\boxed{\lambda}$$

$$q_2 : \boxed{\#}\boxed{a} \quad \# \quad \boxed{\lambda}\boxed{\lambda} \quad \$ \quad \boxed{\#}\boxed{a} \quad \# \quad \boxed{\lambda}\boxed{\lambda}$$

$$q_3 : \boxed{a}\boxed{\#} \quad \# \quad \boxed{\lambda}\boxed{\lambda} \quad \$ \quad \boxed{a}\boxed{\#} \quad \# \quad \boxed{\lambda}\boxed{\lambda} \quad \Big\}$$

and $R_c = \emptyset$

It is known [5] that the picture language of square images in which diagonal positions carry symbol 1 but the remaining positions carry symbol 0 is in LOC. But it is not in SAS. Since row and column splicing are independently done, it is clear that arrays with a proportion between rows and columns and in particular pictures with only square size cannot be generated by any SAS.

Theorem 2. *The class $\mathcal{L}(SASL)$ of Simple array splicing languages and $\mathcal{L}(2RLG)$ [5,15] of picture languages generated by two dimensional right linear grammars are incomparable but not disjoint.*

Proof: The picture language of "chessboards" with even side-length (Figure 3) is also generated by a SAS (Example 2.3) and is known to be generated by a $2RLG$ [5,15].

The picture language L_1 consisting of arrays describing token H (Figure 4)cannot be generated by any $2RLG$, as the horizontal row of x's cannot be maintained by any $2RLG$ [5,15]. But the language consisting of picture arrays describing token H with three rows and any number of columns can be generated by the following SAS :

Let $V = \{x, .\}$, $\quad I = \left\{ \begin{matrix} x\ .\ .\ x \\ x\ x\ x\ x \\ x\ .\ .\ x \end{matrix} \right\},$

$$R_c = \Big\{ p_1 : \boxed{\begin{matrix}.\\x\end{matrix}} \quad \# \quad \boxed{\begin{matrix}\lambda\\\lambda\end{matrix}} \quad \$ \quad \boxed{\begin{matrix}.\\x\end{matrix}} \quad \# \quad \boxed{\begin{matrix}\lambda\\\lambda\end{matrix}}$$

$$p_3 : \boxed{\begin{matrix}x\\.\end{matrix}} \quad \# \quad \boxed{\begin{matrix}\lambda\\\lambda\end{matrix}} \quad \$ \quad \boxed{\begin{matrix}x\\.\end{matrix}} \quad \# \quad \boxed{\begin{matrix}\lambda\\\lambda\end{matrix}}$$

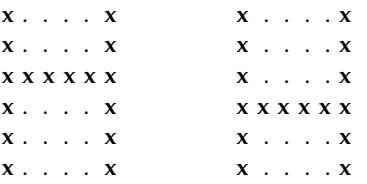

Fig. 4. Token H

$$p_4 : \boxed{\begin{matrix}\#\\.\end{matrix}} \quad \# \quad \boxed{\begin{matrix}\lambda\\\lambda\end{matrix}} \quad \$ \quad \boxed{\begin{matrix}\#\\.\end{matrix}} \quad \# \quad \boxed{\begin{matrix}\lambda\\\lambda\end{matrix}}$$

$$p_5 : \boxed{\begin{matrix}.\\\#\end{matrix}} \quad \# \quad \boxed{\begin{matrix}\lambda\\\lambda\end{matrix}} \quad \$ \quad \boxed{\begin{matrix}.\\\#\end{matrix}} \quad \# \quad \boxed{\begin{matrix}\lambda\\\lambda\end{matrix}} \Bigg\} \text{ and } R_r = \emptyset$$

The picture language $L_2 = \{((ab)^p \cup (ba)^q)_m \;/\; p, q, m \geq 1\}$ cannot be described by any SAS. This is due to the fact that the column splicing of any two arrays $((ab)^p)_m$ and $((ba)^q)_m$ will yield an array which is not in L_2. But it is generated by the following $2RLG$.

$$\begin{aligned}
\Sigma &= \{a, b\}\,; \\
\Sigma_I &= \{A_1, A_2, A_3, A_4\}\,; \\
V_h &= \{S, X\}\,; \\
R_h &= \{S \to A_1A_2X;\ S \to A_3A_4Y;\ X \to A_1A_2X;\ Y \to A_3A_4Y; \\
&\qquad X \to A_1A_2;\ Y \to A_3A_4\}\,; \\
V_v &= \{A_1, A_2, A_3, A_4\}\,; \\
R_v &= \{A_1 \to aA_1;\ A_1 \to a; A_2 \to bA_2;\ A_2 \to b; A_3 \to bA_3;\ A_3 \to b; \\
&\qquad A_4 \to aA_4;\ A_4 \to a\}
\end{aligned}$$

Remark 1. We have the following relationship among $SAS, LOC, 2RLG$ as seen from Theorems 1 and 2.

Note that the picture language of “chessboards" with even side-length is generated by a SAS (Example 2.3) and is known to be generated by a $2RLG$ [5]. It is also a local language.Thus it is a picture language in all the three classes.

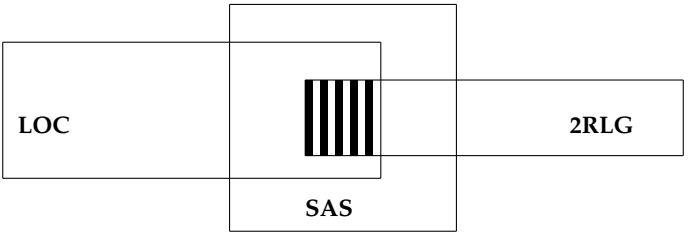

Fig. 5. Relationship among $SAS, LOC, 2RLG$

In fact the set T of 2×2 windows describing this language is:

$$T = \left\{ \begin{array}{l} \# \ \# \ \# \ \# \ \# \ b \ a \ \# \ \# \ \# \ \# \ \# \\ \# \ a, \ b \ \#, \ \# \ \#, \ \# \ \#, \ a \ b, \ b \ a \\ \\ \# \ b \ a \ \# \ a \ b \ b \ a \ a \ b \ b \ a \\ \# \ a, \ b \ \#, \ \# \ \#, \ \# \ \#, \ b \ a, \ a \ b \\ \\ 1 \ 0 \ 0 \ 0 \ \# \ a \ b \ \# \\ 1 \ 1, 1 \ 1, \# \ b, \ a \ \# \end{array} \right\}$$

3 Simple Tree Splicing Systems

We now introduce the notion of Simple splicing on trees as an application of the simple splicing rules introduced in [11]. The splicing of trees considered here is a special form of a general notion of splicing of trees considered in [13].

We consider labelled rooted trees T which are connected cycle-free graphs with a designated node r called the root of the tree and a label $l(v)$ for every node v. Since there is a unique simple path from the root r to any other node v in T, this determines a direction to the edges of the tree and thus tree is viewed as a directed graph with a precedence relationship such that every node has zero or more descendant nodes. A node with a zero descendant is called a leaf and any other node is called an interior node. A subtree of a given tree T is also a labelled rooted tree with its root as an

interior vertex of tree.

It is usual to denote a tree T with root label a and subtrees $T_1, ..., T_n$ with root labels $b_1, ..., b_n$ in linear form as given below $T = a(T_1, ..., T_n)$. We call a labelled rooted tree T with labels in a set V, simply as a tree T over V. The yield of a tree T is a string obtained by reading the labels of the leaves from left to right.

Example 4. Let $V = (a, b, c)$. let T be a tree $T = c(a, c(a, c(a, c, b), b), b)$. The yield of tree T is a^3cb^3.

We now introduce the notion of simple splicing of two trees, by considering simple splicing rules used in the splicing of words[11].

Definition 4. *Let V be an alphabet. Let r be a simple splicing rule* $r = (c \# \lambda \$ c \# \lambda)$ *where* $c \in V$. *Let* T_1 *and* T_2 *be two trees such that c is a root label of a subtree* $T_1^{'}$ *of* T_1 *and a subtree* $T_2^{'}$ *of* T_2. *We say that a tree* T_3 *is obtained by simple tree splicing of* T_1 *with* T_2, *if* T_3 *is the tree obtained from* T_1 *removing the subtree* $T_1^{'}$ *with root v having label c and attaching* $T_2^{'}$ *at the node v.*

A simple tree splicing system $S = (V, A, R)$ where $V$is an alphabet, A is a finite set of trees over V and R is a finite set of simple tree splicing rules. The tree language $T(S)$ consists of all trees obtained from the trees in A by repeatedly applying the tree splicing rules of R. The string language associated with the system S is the set $L(S)$ of words which are the yields of the trees in $T(S)$.

Example 5. Let $V = (a, b, c)$. let T be a tree $T = c(a, c, b)$. Let $S = (V, T, r)$ where $r = c\#\lambda\$c\#\lambda$. On splicing T with itself we obtain a tree $T^{'} = c(a, c(a, c, b), b)$. Repeatedly using the splicing operation, we obtain a set of trees $c(a, c, b), c(a, c(a, c, b), b), c(a, c(a, c(a, c, b)b)b), ...$ which constitutes the tree language $T(S)$. The string language of S is $L(S) = a^n cb^n : n \geq 1$.

We exhibit the relation between the set of derivation trees of a context free grammar and the tree language of a simple tree splicing system.

Theorem 3. *Given a context free language $G = (V_N, V_T, P, S)$, there exists a simple tree splicing system S such that the tree language $T(S)$ is exactly the set of derivation trees of G. As a consequence the context free language $L(G)$ generated by G is simply the string language of $L(S)$*

Proof : Assume that the given CFG is in Chomsky Normal Form with rules of the form $A \to BC$ or $A \to a$ where A, B, C are non-terminals and a is a terminal. For each rule $A \to BC$ we associate a tree $A(B, C)$. For each rule $A \to a$ we associate a tree $A(a)$. A corresponding simple tree splicing system S is constructed as follows : $S = (V_N \cup V_T, A', R)$ where A' consists of trees associated with the rules of P. R consists of simple splicing rules of the form $X\#\lambda\$X\#\lambda$ whenever X is the left hand side of a rule in G and is a symbol in the right hand side of a rule in G. It is straight forward to see that $T(S)$ consists of exactly the derivation trees of G.

Remark 2. It is known that the string language of a simple H system [11] is regular. It is of interest to note from the theorem 3.1 that the string language of a simple tree splicing system is context free and thus the generative power of the simple splicing rules is increased when they are applied on the tree structures.

4 Conclusion

In this paper a theoretical study of a special class of H Array Splicing Systems [2] with “Simple" array splicing rules, is made. Although the rules are “simple", the generating power of the picture generating SAS or tree generating STS is reasonably higher. It remains to examine other properties of these systems such as characterizations etc. as in the string case.

Acknowledgement

The first author K.G. Subramanian gratefully acknowledges support for this research by a FRGS grant of the Universiti Sains Malaysia.

References

1. R. Ceterchi, K.G. Subramanian, Simple Circular Splicing Systems, *Romanian Journal of Inform. Sci. and Tech.* 6 (2003) 121-134.
2. P. Helen Chandra, K.G. Subramanian and D.G. Thomas, Parallel Splicing on Images, *Int. J. Pattern Recognition and Artificial Intelligence*, 18(6) (2004) 1071-1091.
3. N.G. David, K.G. Subramanian, D.G. Thomas, A note on Graph Splicing Languages, *Lecture Notes in Comp. Sci.* 2340 (2002) 381-390.
4. R. Freund, Splicing Systems on Graphs, *Proc. Intelligence in Neural and Biological Systems*, IEEE Press (1995) 189-194.
5. D. Giammarresi and A. Restivo, Two-dimensional languages, In *"Handbook of Formal Languages"* Vol.3, Eds. G. Rozenberg and A. Salomaa, Springer Verlag, (1997) 215 - 267.
6. A. Habel, Hyperedge Replacement: Grammars and Languages, *Lecture Notes in Comp. Sci.*, Springer-verlag (1992).
7. T. Head, Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviours, *Bull. Math. Biology* 49 (1987) 737 -759.
8. T. Head, Gh. Păun and D. Pixton, Language theory and molecular genetics: Generative mechanisms suggested by DNA recombination, In *"Handbook of Formal Languages"* Vol.2, Eds. G. Rozenberg and A. Salomaa, Springer Verlag, (1997) 295 - 360.
9. K. Krithivasan, Splicing Systems-The Graph Model, *Proc. of the Workshop on Molecular Computing*,Chennai, (1998) 33-59.
10. K. Krithivasan, V.T. Chakaravarthy and R. Rama, Array splicing systems, In *" New Trends in Formal languages, Control Cooperation and Combinatorics"*, Eds. Gh.Paun and A. Salomaa, *Lecture Notes in Computer Science* 1218, Springer Verlag, 1997, 346 - 365.
11. Mateescu M., Gh. Păun , G. Rozenberg and A. Salomaa, Simple Splicing Systems, *Discrete Applied Math.*, 84 (1998) 145-163.
12. A. Rosenfeld, and R. Siromoney, Picture languages - a survey, *Languages of design* 1 (1993) 229-245.
13. Y. Sakakibara, C. Ferretti Splicing on tree-like structures. *Theoretical Computer Science*, 210(2), pp.227-243, 1999.
14. A. Salomaa, *Formal Languages*, Academic Press, New York, 1973.
15. G. Siromoney, R. Siromoney and K. Krithivasan, Abstract families of matrices and picture languages, *Computer Graphics and Image Processing* 1 (1972) 234 - 307.
16. K.G. Subramanian and R. Siromoney, On Array Grammars and Languages,*Cybernetics and Systems*, 18 (1987) 77-98.
17. K. G. Subramanian, R. Siromoney, V. Rajkumar Dare, A. Saoudi, Basic Puzzle Languages, *International Journal of Pattern Recognition and Artificial Intelligence*, 9 (1995) 763-775.

18. K.G. Subramanian, P Systems and Picture Languages, *Lecture Notes in Computer Science* 4664, Springer (2007) 99-109.
19. K.G. Subramanian, D. L. Van, P. Helen Chandra and N. D. Quyen, Array Grammars with Contextual Operations, *Fundamenta Informaticae*, 83(4) (2008) 411-428.
20. P.S. Wang, *Array Grammars, Patterns and Recognizers*, (Ed) World Scientific Pub. Co., 1989.