Local splicing grammar systems

Adrian ATANASIU
Faculty of Mathematics
Bucharest University
Str. Academiei 14, 70700 București, Romania
E-mail: aadrian@pcnet.ro

Abstract

A new type of splicing is defined, namely the interchange of two fixed subwords, without moving the sequences situated after them. Idea of splicing grammar systems is introduced by Dassow and Mitrana, ([2]); in this paper only the derivation type is changed. The generative power is the same (are necessarily only two modules to generate all recursive enumerable languages). Two variants of derivation are considered more detailed (when at every step the rules used have the same left part, and an asynchronous derivation - when it is possible to apply rules only in a subset of components). The generative power remains the same but some normal forms are possible to be defined.

1 Introduction, preliminary results

We assume the basic notations in the Formal Languages theory are known ([4]). By REG, CF and RE we denote the families of regular, context-free and recursively enumerable languages respectively.

For an alphabet \( V \), we denote by \( V^* \) the set of all words over \( V \), and by \( \lambda \) the empty word. In [2], J. Dassow and V. Mitrana defines a new type of generative device – splicing grammars. Because we shall use the same terms, we rewrite that construction:

Definition 1.1 A splicing grammar system (SGS) of degree \( n \), is a construct

\[
\Gamma = (N, T, (S_1, P_1), \ldots, (S_n, P_n), M)
\]

where:

1. \( N, T \) are disjoint finite alphabets (their elements are called 'nonterminals' and 'terminals' respectively);

2. \( P_i \ (1 \leq i \leq n) \) are finite sets of rules over \( N \cup T \);
3. $M$ is a finite subset of $(N \cup T)^*\#(N \cup T)^*\#(N \cup T)^*$, where $\#$, $\#$ are distinct symbols which are not in $N \cup T$.

The sets $P_i$ are called components of $\Gamma$; in order to put in evidence grammars as components, we can consider grammars of the form $(N, T, S_i, P_i)$, $1 \leq i \leq n$.

By a configuration, one means a $n$-tuple consisting of words over $N \cup T$.

Definition 1.2 For two configurations

\[
x = (x_1, x_2, \ldots, x_n), \quad x_i \in (N \cup T)^*N(N \cup T)^*, 1 \leq i \leq n
\]

\[
y = (y_1, y_2, \ldots, y_n), \quad y_i \in (N \cup T)^*, 1 \leq i \leq n,
\]

we define $x \Rightarrow_\Gamma y$ iff one of the following two conditions holds:

i) $\forall i, \ (1 \leq i \leq n), \ x_i \Rightarrow_{P_i} y_i$,

ii) $\exists i, j, \ (1 \leq i, j \leq n)$ such that

\[
x_i = x'\alpha\beta x'', \ x_j = y'\gamma\delta y'', \ y_i = x'\alpha\delta y'', \ y_j = y'\gamma\beta x''
\]

for $\alpha \# \beta \# \gamma \# \delta \in M$.

In Definition 1.2, (i) defines a rewriting step, whereas (ii) defines a splicing step. Note here that there is no priority of any of these operations over the other.

Two languages can be associated to a splicing grammar system. One of them is the language generated by a single component and, because no component is distinguished in a way or another, we may choose always the language generated by the first component. This language will be called the individual language of the system.

Formally, the language generated by the $i$-th component is defined by

\[
L_i(\Gamma) = \{w \in T^*|(S_1, \ldots, S_n) \Rightarrow \ldots \Rightarrow (w_1, \ldots, w_n), w_j \in (N \cup T)^*, j \neq i\}
\]

The second language associated will be a total language, namely

\[
L_t(\Gamma) = \bigcup_{1 \leq i \leq n} L_i(\Gamma).
\]

Also, we denote by

- $I_{sgs} L_n(X)$ – the family of individual languages generated by splicing grammar systems of degree $n$, with components of type $X$;

- $T_{sgs} L_n(X)$ – the family of total languages generated by splicing grammar systems of degree $n$, with components of type $X$,

where $X \in \{REG, CF\}$.

In [3] is proved that

\[
RE = Y_{sgs} L_2(CF), \ Y \in \{I, T\}
\]

It remains as an open problem the relation between $I_{sgs} L_2(X)$ and $T_{sgs} L_2(X)$. 

2
2 Local splicing grammar systems

In the following we define a new variant of splicing grammar systems: *local splicing grammar systems* (LSGS). The name shows that the interchange of words is made only local – near the cut – so no other part of the words (their suffixes for examples) are involved.

The definition of grammars is the same with Definition 1.1; it differs only in the second part of Definition 1.2 (the derivation mode).

Namely:

(ii) \( \exists i, j, 1 \leq i, j \leq n \) so that \( x_i = x'\alpha\beta x'', x_j = y'\gamma\delta y'', y_i = x'\alpha\delta x'', y_j = x'\gamma\beta y'' \) for \( \alpha \# \beta \# \gamma \# \delta \in M \).

So, for a rule \( \alpha \# \beta \# \gamma \# \delta \in M \), we have

\[
(..., x'\alpha\beta x'', ..., y'\gamma\delta y'', ...) \Rightarrow (... x'\alpha\delta y'', ..., y'\gamma\beta x'', ...)
\]

in general splicing grammar systems,

\[
(..., x'\alpha\beta x'', ..., y'\gamma\delta y'', ...) \Rightarrow (... x'\alpha\delta x'', ..., y'\gamma\beta y'', ...)
\]

in local splicing grammar systems.

**Remark 2.1** The operations of insertion and deletion used in Formal Language Theory, can be defined in local splicing terms as follows:

- \( \lambda \# \alpha \# \lambda \) – for the deletion \( \alpha \rightarrow \lambda \), and
- \( \lambda \# \lambda \# \alpha \) – for the insertion \( \lambda \rightarrow \alpha \).

Let us denote – like in [2] – with \( Y_{\text{lsgs}} L_n(X), Y \in \{I, T\}, X \in \{\text{REG, CF}\} \) the class of languages of type \( Y \) (individuals or totals) generated by local splicing grammar systems with \( n \) components of type \( X \).

**Theorem 2.1** \( I_{\text{lsgs}} L_2(CF) = \text{RE} \).

**Proof:** Let \( L \) be a recursive enumerable language and \( G = (N, V, S, P) \) be a grammar of type 0 with \( L(G) = L \). We shall construct a LSGS with two components as follows:

\[ \Gamma = (N \cup \{X, Y\}, T, S, X, P_1, P_2, M) \]

where:

- \( X, Y \not\in N \cup T \),
- \( P_1 = \{A \rightarrow \alpha | A \rightarrow \alpha \in P\} \cup \{A \rightarrow A | A \in N\} \),
- \( P_2 = \{X \rightarrow \alpha Y X | AB \rightarrow \alpha \in P\} \cup \{X \rightarrow X\} \),
- \( M = \{\lambda \# AB \# \lambda \# \alpha | AB \rightarrow \alpha \in P\} \).

Let us prove the equality \( L_1(\Gamma) = L \).

"\( \subseteq \)" : At every derivation step, one of the following possibilities can arise:
1. In the first component a rule $A \rightarrow \alpha$ or $A \rightarrow A$ is used; doesn’t matter what rule is used for the second component.

2. A local splicing is used; so, is accomplished a pass of the form

$$(xABy, x'\alpha Y y'X) \Rightarrow (x\alpha y, x'ABY y'X),$$

where $x, y \in (N \cup T)^*$, $x', y' \in (N \cup T \cup \{Y\})^*$ and $\lambda \# AB \# \lambda \# \alpha \in M$.

Based on this derivation in $\Gamma$, a derivation in $G$ can be constructed, using the rules:

i) $A \rightarrow \alpha$ for each application of $A \rightarrow \alpha$ in the first component; any applications of a rule of type $A \rightarrow A$ is ignored;

ii) $AB \rightarrow \alpha$, if a splicing with rule $\lambda \# AB \# \lambda \# \alpha$ was used in $\Gamma$.

This derivation simulates a derivation achieved by $\Gamma$ on the first component.

"$\supseteq$": Let be $w \in L$; so, there is a derivation in $G$: $S = \alpha_0 \Rightarrow \alpha_1 \Rightarrow \ldots \Rightarrow \alpha_n = w$.

On this fact, the next derivation can be simulated in $\Gamma$:

• The initial configuration is $(\alpha_0, \beta_0 X)$ where $\alpha_0 = S, \beta_0 = \lambda$.

• Let us suppose that for the local splicing system, the configuration $(\alpha_i, \beta_i X)$ is achieved, and in the derivation of grammar $G$ is used the rule:

1. $A \rightarrow \alpha$ (that is $\alpha_i = xAy \rightarrow x\alpha y = \alpha_{i+1}$);
   Then $(Xay, \beta_i X) \Rightarrow (x\alpha y, \beta_i X)$ (in the second component was applied the rule $X \rightarrow X$);

2. $AB \rightarrow \alpha$ (that is $\alpha_i = xABy \rightarrow x\alpha y = \alpha_{i+1}$);
   Then $(xABy, \beta_i Y) \Rightarrow (1) (xABy, \beta_i \alpha Y X) \Rightarrow (2) (x\alpha y, \beta_i ABY X)$
   $= (\alpha_{i+1}, \beta_{i+1} X)$.

For each component were used:

1. the rules $(A \rightarrow A, X \rightarrow \alpha Y X)$;

2. a local splicing, with $\lambda \# AB \# \lambda \# \alpha$.

The nonterminal $Y$ is necessary in order to assure that the garbage obtained after the using of a local splicing will be isolated and will be not involved in other splicings.

The reciprocal assertion (any language generated by a LSGS is recursive enumerable) is trivial: the system can be simulated by a Turing machine with two tapes. □

**Corollary 2.1** Any individual local splicing grammar system with two components of type $CF$ can be associated to an individual splicing grammar system with two components of type $CF$ and viceversa.
Proof. The assertion results from $I_{\text{sgs}} L_2(CF) = RE = I_{\text{sgs}} L_2(CF)$.\hfill\qed

**Corollary 2.2** $T_{\text{sgs}} L_2(CF) = RE$.

**Proof:** It results from the construction in proof of Theorem 1, that $L_2(\Gamma) = \emptyset$, therefore $L_1(\Gamma) = L_1(\Gamma)$.\hfill\qed

**Example 2.1** Let us consider the language $L = \{a^n b^n c^n | n \geq 1\}$ generated by the grammar with rules

$$S \rightarrow aSBC | ABC, CB \rightarrow BC, Ab \rightarrow ab, bB \rightarrow bb, Bc \rightarrow bc, Cc \rightarrow cc.$$ 

A LSGS with two components which generates the language $L$, is constructed using this grammar and ideas contained in the proof of Theorem 1, as follows:

$$\begin{align*}
\Gamma &= (N, T, S, X, P_1, P_2, M), \\
N &= \{S, B, C, X, Y\}, \quad T = \{a, b, c\}, \\
P_1 &= \{S \rightarrow ASBC | ABC, S \rightarrow S, B \rightarrow B, C \rightarrow C\}, \\
P_2 &= \{X \rightarrow BCY | abY Y, Y | bcY X | ccY X | X\}, \\
M &= \{\lambda \# C B \# B C, \lambda \# A b \# b, \lambda \# B b \# b, \lambda \# B c \# c, \lambda \# C c \# c\}.
\end{align*}$$

Let us see how the word $a^2 b^2 c^2$ is obtained in both these generative devices:

In the grammar $G$:

$$S \Rightarrow ASBC \Rightarrow aaBCBC \Rightarrow aabBCC \Rightarrow aabbCC \Rightarrow aabbcc.$$ 

In LSGS $\Gamma$:

$$(S, X) \Rightarrow (ASBC, abY X) \Rightarrow (aabcbc, abY BCY X) \Rightarrow (aabbcc, abY bcY X) \Rightarrow (aabbCC, aBY CBY bbY X) \Rightarrow (aabbCC, aBY CBY bbY bcY X) \Rightarrow (aabcC, aBY CBY bBY bcY ccY X) \Rightarrow (aabcC, ABY CBY bBY bcY ccY X).$$

A supplementary condition can be required; namely, in a LSGS, at every step of derivation, the rules applied will have the same left side. Then we can write the set of rules $(A \rightarrow \alpha, A \rightarrow \beta)$ under the form $A \rightarrow \alpha, \beta$.

This will simplify the notation of a LSGS: the start symbols remain unchanged and the two sets of rules can be written as a single set:

$$P = \{A \rightarrow \alpha, \beta | A \rightarrow \alpha \in P_1, A \rightarrow \beta \in P_2\}.$$ 

Hence, we can denote a LSGS by $(N, T, S, P, M)$.

It is interesting to remark that this restriction in derivation does not decrease the generative power of LSGS.

**Theorem 2.2** Any LSGS is equivalent with a LSGS with two components of type CF, in which at every derivation step the rules used have the same left side.
Proof. Instead of performing some modifications in a LSGS in order to obtain one or the other form, is more easier to prove this assertion using languages generated.

Let \( L \) be a recursive enumerable language generated by the grammar \( G = (N, T, S, P) \). We construct the local splicing grammar system with restriction required, as follows:

\[
\Gamma = (N \cup \{S', X, Y\}, T, S', P', M) \text{ where:}
\]

\( P' = \{S' \rightarrow S', \alpha XS' | AB \rightarrow \alpha \in P\} \cup \{A \rightarrow [\alpha] | A \rightarrow \alpha \in P\} \cup \{S' \rightarrow S', XS'; S' \rightarrow S, YS\} \)

\( M = \{\lambda\#AB\#X\#\alpha | AB \rightarrow \alpha \in P\} \) (on denote by \([\alpha]\) the word \(\alpha\) in which all terminal characters were erased).

A derivation in this LSGS is performed in two parts:

\[
(S', S') \Rightarrow \ldots \Rightarrow (S', (X\alpha_1)^*(X\alpha_2)^*\ldots(X\alpha_p)^*XS')
\]

where \(\alpha_1, \alpha_2, \ldots, \alpha_p\) are right sides of all rules in \(P\); they are generated in the second component as many we need, in order to use them when a local splicing with such peculiar subword is required.

Passing to a real derivation concerning the first component, this is made by

\[
(S', (X\alpha_1)^*(X\alpha_2)^*\ldots(X\alpha_p)^*XS') \Rightarrow (S, (X\alpha_1)^*(X\alpha_2)^*\ldots(X\alpha_p)^*XYS)
\]

Then, the first component follows the derivation accomplished by grammar \(G\); so:

- if in \(G\) a rule of the form \(A \rightarrow u\) is used, then in \(\Gamma\) is used \(A \rightarrow u, [u]\) which replaces by \([u]\) a nonterminal \(A\) situated after \(Y\).
- if in \(G\) a rule of the form \(AB \rightarrow u\) is used, then in \(\Gamma\) a splicing is activated; apparition of \(X\) in context assures that a subword placed before \(Y\) is involved.

It is easy to see there are not other possibilities to finish the derivation of the first component in \(\Gamma\).

\(\square\)

Remark 2.2 The rule \(S' \rightarrow S', XS'\) from the construction of LSGS can be removed; that was important only for introduce an \(X\) on the first position of the second component. Any other rule from the first set of rules in \(P'\) can write an \(X\) which will have the same role for its following characters; hence we can start with an arbitrary rule, which will not be used in splicing.

Example 2.2 Using the language defined in Example 2.1, a LSGS with the rules of this type can be construct:

\[
P' = \{S' \rightarrow S', BCXS'S' \rightarrow S', abXS'S' \rightarrow S', bbXS'S' \rightarrow S', bcXS' \rightarrow S, ccXS'S' \rightarrow S, YSS \rightarrow ASBC, SBCS \rightarrow ABC, BC\}
\]

\(M = \{\lambda\#CB\#X\#BC, \lambda\#Ab\#X\#ab, \lambda\#Bb\#X\#bb, \lambda\#Bc\#X\#bc, \lambda\#Cc\#X\#cc\}\)

Theorem 2.3 (Normal form)

Any LSGS is equivalent with a LSGS which has all rules of the form \(A \rightarrow BC, BC\) or \(A \rightarrow x, y, [x] + [y] \leq 1(|\alpha| \text{ denotes the length of the word } \alpha)\).

Proof. Is similar with construction of the normal form for STOS schemas (see [1]).

\(\square\)
Local splicing grammar systems with asynchronous derivation

In the definition of splicing grammar systems, a restriction in derivation consists in that in one step each component uses exactly one rule. The abandon of this restriction seems to decrease the complexity of languages generated (like \(\{a^n b^n c^n | n \geq 1\}\), \(\{\alpha \alpha | \alpha \in V^*\}\)).

In the following, we will show that – by contrary – the abandon of the synchronization in applying the rules for each component in one derivation step leads to a classification of grammar systems more appropriate to Chomsky classification.

Let us consider only local splicing grammars with two components of type \(X (X \in \{REG, CF\})\). As we shall see, this will be enough for our construction.

Definition 3.1: A local splicing grammar system \(\Gamma = (N, T, S_1, S_2, P_1, P_2, M)\) is called with "asynchronous derivation" if and only if in Definition 1.2 (i) is replaced by:

\((i') \exists I \subseteq \{1, 2\}, I \neq \emptyset \text{ with } x_i \rightarrow y_i \forall i \in I.\)

Therefore, in a derivation step, at least one component is modified by a rule (instead of all components).

Let us denote by ALSGS the class of local splicing grammars with asynchronous derivation. In the following we will refer only to the languages generated by the first component.

This type of grammar can be included obviously in the general type of local splicing grammar systems, but paying the price of modification of generated language (if we introduce in both sets of rules productions of type \(A \rightarrow A \forall A \in N)\).

Let us study the power of this type of generative device.

Theorem 3.1: Any recursive enumerable language can be generated by a local asynchronous splicing system grammar \(\Gamma = (N, T, S_1, S_2, P_1, P_2, M)\) with \(P_1 = \emptyset\).

Proof: Let \(L \in RE\) be a language generated by the grammar \(G = (N', T, S, P)\). We shall construct the following grammar \(\Gamma \in ALSGS:\)

\[N = N' \cup \{X, S_2\}, S_1 = S, P_1 = \emptyset, P_2 = \{S_2 \rightarrow X\beta S_2 | \alpha \rightarrow \beta \in P\},\]

\[M = \{\lambda \# \alpha X \# \beta | \alpha \rightarrow \beta \in P\} .\]

The equality \(L(\Gamma) = L\) results from the next observation concerning the mode in which \(\Gamma\) simulates the derivation in \(G\):

Let \(S = w_0 \Rightarrow w_1 \Rightarrow \ldots \Rightarrow w_n = w\) be the derivation of a word \(w \in L\).

The derivation of this sentence in \(\Gamma\) starts with the configuration \((S, S_2) = (w_0, x_0 S_2)\) (here \(w_0 = S, x_0 = \lambda\)). Let us consider the configuration \((w_i, x_i S_2)\); we suppose that in \(G\) the derivation step \(w_i \Rightarrow w_{i+1}\) was accomplished using the rule \(u \rightarrow v \in \Phi\) (therefore \(S_2 \rightarrow XvS_2 \in P_2, \lambda \# u \# X \# v \in M\)).
Hence, in $\Gamma$ two steps will be considered: first for derivation, the second for splicing:

$$(w_i, x_iS_2) \Rightarrow (w_i, x_iXvS_2) \Rightarrow (w_{i+1}, x_iXuS_2) = (w_{i+1}, x_{i+1}S_2).$$

The reverse inclusion ($L(\Gamma) \subseteq L$) can be proved in a similar way. □

**Remark 3.1** If we consider only local splicing grammar systems with asynchronous derivation constructed in the proof of Theorem, the next remarks follows:

- The writing of this type of grammar can be simplified to $\Gamma = (N, T, S, P, M)$, avoiding all the elements related to the first component and the index of the second component.

  In this situation, the result we made above remains true if we start from a grammar with other notation of the start symbol; if $S'$ is the start symbol in $G$, then we shall add (besides the rules defined by the proof of Theorem 3.1) in $P : S \rightarrow XS'$ and in $M : \lambda \# S \# S'$. 

- One can find the Chomsky classification using the conditions applied to splicing rules. So:

  - $CS$ (the class of context sensitive languages) is generated by the grammar systems from $ALSGS$ with the property:
    
    $[i.f\lambda \# \alpha X \# \beta \in M \text{ then } |\alpha| \leq |\beta|]$ (with the possibly exception $\lambda \# S \# \lambda \in M$) and viceversa;

  - $CF$ is generated by the grammar systems from $ALSGS$ with the property:
    
    $[i.f\lambda \# \alpha X \# \beta \in M \text{ then } \alpha \in N]$ and viceversa;

  - $REG$ is generated by the grammar systems from $ALSGS$ with the property:
    
    $[i.f\lambda \# \alpha X \# \beta \in M \text{ then } \alpha \in N, \beta \in T^*(N \cup \{\lambda\})]$ and viceversa.

- In a "biological" interpretation, from the Theorem 3.1 results that any DNA chain can be obtained from only one genetic surgery – splicing operations – and no generative rules. The asynchronous derivation assures the freezing of developing for some DNA chains (or, possibility to developing chains with varied speeds).

- If we consider as nonterminals only symbols which can appear in the left side of rules from $P_2$, then these rules can be considered to be of the right linear type.

This remarks lead to a simplification of writing for grammar systems $ALSGS$:

**Definition 3.2** $ALSGS$ contains only grammar systems of the form

$\Gamma = (N, C, T, S, P, M)$ where:
N, C, T are finite nonempty sets called nonterminal set, control set and respectively terminal set;

S ∈ N is the start symbol;

P = \{ A → αB | α, A, B ∈ N, α ∈ (C ∪ T)^* \} is the (finite and nonempty) set of rules;

M = \{ x#y$y'|x, x', y, y' ∈ (C ∪ T)^* \} is the finite set of local splicing rules.

In this case, a new normal form for elements of P can be done: A → Ab|a, a ∈ C ∪ T ∪ {λ}.

Example 3.1 For the same language defined in Example 2.1 another grammar system ALSGS can be defined as follows:

N = \{S\}, C = \{A, B, C, X\}, T = \{a, b, c\},
P : S →XA|XaABCS|XaBCS|XBCS vertXabS|XbbS|XbcS|XccS
M : λ#S$λ#Aλ#A$X#AABCλ#A$X#ABCλ#C$X#BC
λ#A$X#abλ#B$b$X#bbλ#B$c$X#bcλ#Cc$X#cc}.

Then, the word aabbcc can be obtained using the next derivation in Γ:

(S, S) ⇒ (S, XA) ⇒ (A, XS) ⇒ (A, XaABCS) ⇒ (AABC, XAS)
⇒ (AABC, XAXaBCS) ⇒ (aabbc, XAXAXBBS)
⇒ (aabbc, XAXAXCBS) ⇒ (aabbc, XAXAXCBXabS)
⇒ (aabbc, XAXAXCBXaBS) ⇒ (aabbc, XAXAXCBXabBS)
⇒ (aabbc, XAXAXCBXaB xbS) ⇒ (aabbc, XAXAXCBXabBXbcS)
⇒ (aabbc, XAXAXCBXabBXbBS)
⇒ (aabbc, XAXAXCBXabBXbBXbcS)
⇒ (aabbc, XAXAXCBXabBXbBXbcXcCS)
⇒ (aabbc, XAXAXCBXabBXbBXbcXcCS)

References


