International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 1 Issue 8, October - 2012

Learning Simple Splicing Grammar Systems

Sindhu J Kumaar[1], P. J. Abisha[2] and D. G. Thomas[3]
[1]Department of Mathematics, B. S. Abdur RahmanUniversity,
Chennai – 600 048, Tamil Nadu, India.
[2, 3]Department of Mathematics, Madras Christian College,
East Tambaram, Chennai – 600 059, Tamil Nadu, India.

Abstract

Pattern language learning algorithms within the inductive inference model and query learning setting have been of great interest. Angluin's [3] pattern languages motivated Dassow et al [8] to modify the way of describing these languages and define pattern grammars. Sindhu et al [17] introduced learning algorithms for two kinds of parallel communicating grammar systems. Dersanambika et al [9] introduced a new type of grammar system called simple splicing grammar system (SSGS) in which four types of splicing rules namely < 1, 3 > , < 1, 4 > , < 2, 3 > and < 2, 4 > are discussed. A model of splicing grammar systems, with pattern grammar as components was introduced by Sindhu et al [18]. In this paper we provide a learning algorithm for a subclass of simple splicing grammar system.

1. Introduction

Inductive inference introduced by Gold [10] is a model that identifies the unknown concept in the limit. Inferring a pattern common to all words in a given sample is a typical instance of inductive inference. A pattern is a finite string of constant and variable symbols and the pattern language introduced by Angluin [3] is the set of all strings obtained by substituting any non-null constant string for each variable symbol in the pattern. In recent years, there has been an increased interest in the problem of learning pattern languages using queries and examples. Motivated by the study of Angluin, a generative device called pattern grammar is defined by Dassow et al [7]. The idea here is to start from a finite set A of axioms which are over an alphabet of constants; given a set P of patterns which are strings over an alphabet of constants and variables, replace uniformly and in parallel, the variables in a given pattern by axioms and continue the process with the current set of strings, obtained by such operations. All strings generated in this way constitute the associated language called a pattern language.

The theory of grammar systems [8] is an interesting and a deeply investigated area of formal language theory. A parallel communicating grammar system consists of several grammars. For solving a task, the components work simultaneously and are allowed to communicate. A communication is done by request: a component can request the whole word generated by another component. A minimal synchronization is assumed: in each time unit every component carries out a rewriting step or the system performs communication.

The behavior of DNA under the influence of restriction enzymes and ligases was studied by Head et al and an overview of this can be seen in [12]. Head defined splicing systems that make use of a new operation called splicing on strings of symbols. Mateescu et al [13] have considered simple splicing systems that make use of splicing rules that are as simple as possible.

A splicing grammar system [6] can be viewed as a set of grammars working in parallel on their own sentential forms (exactly as in parallel communicating grammar systems) and, from time to time, exchanging to each other segments of their sentential forms, determined by given splicing rules. Dersanambika et al [9] examined splicing grammar systems by requiring the simple splicing rules in the sense of Mateescu et al [13]. Various properties of the resulting simple splicing grammar systems are obtained by considering different component grammars.

Sindhu et al [17] have given a polynomial time algorithm to learn a subclass of PCPPL (Parallel Communicating Pure Pattern Language) using restricted subset queries and restricted superset queries.

Sindhu et al [18] have considered a model of splicing grammar systems, with pattern grammar in the components. For this model the master component is a regular grammar. In this paper we present learning algorithms of this model of grammar systems using prefix queries and restricted subset queries.

2. Pattern Grammars

We first recall the definition of pattern grammar given by Dassow et al [8].

Definition 2.1: [7] A pattern grammar (PG) is a 4 – tuple $G = (\sum, X, A, P)$ where $\sum$ is an alphabet whose elements are called constants, X is an alphabet whose elements are called variables. A $\subseteq \sum^*$ is a finite set of elements of $\sum^*$ called axioms and P $\subseteq (\Sigma \cup X)^+$ is a finite set of words called patterns where each word contains atleast one variable. The rewriting is done as follows:

$$P(A) = \left\{ \begin{array}{l} u_1\, x_{i_1}\, u_2\, x_{i_2} \ldots u_k\, x_{i_k}\, u_{k+1} / u_1\, \delta_{i_1}\, u_2\, \delta_{i_2} \ldots u_k\, \delta_{i_k}\, u_{k+1} \in P, \\ u_i \in \sum^*, 1 \leq i \leq k+1,\ x_{i_j} \in A,\ \delta_{i_j} \in X,\ 1 \leq j \leq k \end{array} \right\}$$

This means that, P(A) contains words obtained by replacing the variables in the pattern by words from A and different occurrences of the same variable are replaced by the same word. Then the pattern language (PL) generated by G is L(G) = P $\cup$ A $\cup$ P (A) $\cup$ P(P (A)) $\cup$…

Example 2.1: G = ({a, b}, {δ}, {ab}, {aδb}) is a pattern grammar generating the language L(G) = $\{a^nb^n / n \geq 1\}$, as A = {ab}, P(A) = {aabb}, P(P(A)) = {aaabbb} and so on.

We observe that the concept of variable is similar to that of variable in Chomskian grammar. But the rewriting process is different; it is uniform, in the sense that all the occurrences of a variable in a pattern are replaced by the same word and the variables are rewritten in parallel. Still, the pattern grammars generate class of languages which are incomparable with Chomskian languages and Lindenmayer languages.

3. Simple Splicing Grammar System

The splicing operation is a novel operation on strings and languages introduced in [11] in order to model the recombinant behavior of DNA sequences. One gives rules of the form $u_1\# u_2 \$ u_3\#u_4$, called splicing rules, where u_1, u_2, u_3, u_4 are strings over the alphabet we work with; from two strings $x = x_1u_1u_2x_2$, $y = y_1u_3u_4y_2$, for splicing rule $u_1\# u_2 \$ u_3\#u_4$, we produce the string $z = x_1u_1u_4y_2$. We say that z is obtained by the splicing of x, y, which are cut and joined at sites specified by u_1, u_2, u_3, u_4.

We now recall the definition of simple splicing grammar system introduced in [9].

Definition 3.1: [9] A < 1, 3 > - simple splicing grammar system (< 1, 3 > - SSGS) of degree n is a construct

$\Gamma = (N, T, (S_1, P_1), (S_2, P_2), (S_3, P_3), \ldots, (S_n, P_n), M)$ where

i. *N, T are disjoint alphabets and P_i, $1 \leq i \leq n$ are finite sets of production rules over $N \cup T$. Each $S_i \in N$, $1 \leq i \leq n$ is a special symbol called the start symbol.*

ii. *M is a finite subset of $(N \cup T)\ \#\ \$\ (N \cup T)\ \#$, with #, \$ two distinct symbols which are not in $N \cup T$. Each element of M is a < 1, 3 > - simple splicing rule. Here $(N \cup T)\ \#\ \$\ (N \cup T)\ \#$ means $(N \cup T)\ \#\ \{\lambda\}\ \$\ (N \cup T)\ \#\ \{\lambda\}$ where $\{\lambda\}$ is the empty string.*

The sets (S_i, P_i) are called the components of Γ. We can consider grammars of the form $G_i = (N, T, S_i, P_i)$, $1 \leq i \leq n$. By configuration, we mean an n – tuple consisting of words over N $\cup$ T. For two configurations,

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

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

We define $x \Rightarrow_\Gamma y$ if and only if any of the following two conditions hold.

(i) for each $1 \leq i \leq n$, $x_i \Rightarrow_{p_i} y_i$

(ii) there exist $1 \leq i, j \leq n$ such that $x_i = x_i'\ a\ x_i''$, $x_j = x_j'\ a\ x_j''$, $y_i = x_i'\ a\ x_j''$, $y_j = x_j'\ a\ x_i''$, for $a\ \#\ \$\ a\ \# \in M$, and $y_k = x_k$, for $k \neq i, j$

In a derivation $x \Rightarrow_\Gamma y$, in < 1, 3 > - SSGS, (i) defines a rewriting step, but (ii) defines a < 1, 3 > splicing step, corresponding to a communication step in a parallel communicating grammar system. Here if a splicing step is happening between ith and jth components, the other components will not do the rewriting process. In other words, either (i) or (ii) holds at a particular step. There is no priority of any of these operations over the other.

A < 2, 3 > - simple splicing grammar system is analogously defined by choosing < 2, 3 > simple splicing rules $\#\ (N \cup T)\ \$\ (N \cup T)\ \#$. Also < 1, 3 > - SSGS and < 2, 4 > - SSGS are essentially the same. Likewise <1, 4 > and < 2, 4 > simple splicing rules are from the sets $(N \cup T)\ \#\ \$\ \#\ (N \cup T)$ and $\#\ (N \cup T)\ \$\ \#\ (N \cup T)$ respectively.

The language generated by the ith component is defined by

$$L_i(\Gamma) = \{x_i \in T^* / (S_1, S_2, \ldots, S_n) \Rightarrow^* (x_1, x_2, \ldots, x_n), x_j \in (N \cup T)^*, j \neq i\}$$

where $\Rightarrow^*$ is the reflexive and transitive closure of the relation $\Rightarrow$. Two kinds of languages can naturally be associated to a simple splicing grammar system. One of them is the language generated by a single component and because no component is distinguished any way, we may always choose the language generated by the first component. This language is called the individual language of the system. The second associated language will be the total language, namely

$$L_t(\Gamma) = \bigcup_{i=1}^{n} L_i(\Gamma)$$

Example 3.1:

Consider the < 1, 3 > - SSGS with regular rewriting rules. Let
$\Gamma = (N, T, (S_1, P_1), (S_2, P_2), M)$
$N = \{S_1, S_2, A, B\}$
$T = \{a, b, c\}$
$P_1 = \{S_1 \to aA, A \to aA, A \to c\}$
$P_2 = \{S_2 \to cB, B \to bB, B \to b\}$
$M = \{c \# \$ c \#\}$
This system produces languages
$L_1(\Gamma) = \{a^n c b^n / n \geq 1\} \cup \{a^n c \;/ n \geq 1\}$

$L_2(\Gamma) = \{cb^n / n \geq 1\}$
Here the total language is
$L_t(\Gamma) = \{a^n cb^n / n \geq 1\} \cup \{a^n c / n \geq 1\} \cup \{cb^n / n \geq 1\}$

4. Simple Splicing Pattern Grammar System

We recall < 1, 3 > simple splicing pattern grammar system with two components taking the first component to be a regular (or context free) grammar and the second to be a pattern grammar [18].

Definition 4.1

A < 1, 3 > simple splicing RPG grammar system is a construct $\Gamma = (N, X, T, (S, P_1), (A, P_2), M)$, where

(i) N, T are disjoint alphabets. $S \in N$ is a special symbol called start symbol and (N, T, P_1, S) is a regular grammar.

(ii) $A \subseteq T^*$ is a finite set of words called axioms and P_2 is a finite subset of $(X \cup T)^+$ called the set of patterns. Here N, X and T are disjoint alphabets. (T, X, A, P) is a pattern grammar

(iii) M is a finite subset of $(N \cup T) \# \$ (N \cup T) \#$, with # , $ are two distinct symbols which are not in $(N \cup T)$. Each element of M is a < 1, 3 > simple splicing rule.

For two configurations,
$x = (x_1, x_2)$, $x_1 \in (N \cup T)^* N (N \cup T)^*$, $x_2 \in T^*$
$y = (y_1, y_2)$, $y_1 \in (N \cup T)^*$, $y_2 \in T^*$.
We define $x \Rightarrow_\Gamma y$ if and only if any of the following two conditions hold:

(i) $x_1 \underset{P_1}{\Longrightarrow} y_1$ and $y_2 \in P_2(\{x_2\})$

(ii) $x_1 = x_1' a x_1''$, $x_2 = x_2' a x_2''$,
$y_1 = x_1' a x_2''$
$y_2 = x_2' a x_1''$, for $a \# \$ a \# \in M$

In the derivation $x \Rightarrow_\Gamma y$ in <1, 3> simple splicing RPG systems (i) defines a rewriting step, but (ii) defines a splicing step corresponding to a communicating grammar system.

< 2, 3 > , < 1, 4 > and < 2, 4 > simple splicing RPG systems are analogously defined.
The language generated by the 1st component is defined by

$$L_i(\Gamma) = \{x_1 \in T^* / (S_1, w) \Rightarrow^* (x_1, x_2),\ x_1, x_2 \in T^*,\ w \in A\}$$

where $\Rightarrow^*$ is the reflexive and transitive closure of the relation $\Rightarrow$. Two kinds of languages can naturally be associated to a simple splicing grammar system. One of them is the language generated by the individual component. This language is called the individual language of the system. The second associated language will be the total language, namely

$$L_t(\Gamma) = L_1(\Gamma) \cup L_2(\Gamma)$$

Example 4.1

$\Gamma = (N, T, X, (S, P_1), (A, P_2), M)$
$N = \{S, A\}$, $X = \{\delta\}$
$T = \{a, b, c\}$
$P_1 = \{S \to aA, A \to aA, A \to c\}$
$P_2 = \{\delta b\}$
$A = \{c\}$
If $M = \{c \# \$ c \#\}$, this system produces languages
$L_1(\Gamma) = \{a^n cb^n / n > 1\} \cup \{a^n c / n \geq 1\}$
$L_2(\Gamma) = \{\lambda\} \cup \{cb^n / n \geq 1\}$
Here the total language is
$L_t(\Gamma) = \{a^n cb^n / n > 1\} \cup \{a^n c / n \geq 1\} \cup \{\lambda\} \cup \{cb^n / n \geq 1\}$

5. Learning Regular Sets from Queries and counterexamples

Angluin [4] describes the learning algorithm L^* and show that it efficiently learns an initially unknown regular set from any minimally adequate teacher. Let the unknown regular set be denoted by U and that it is over a fixed known finite alphabet A.

At any given time, the algorithm L^* has information about a finite collection of strings over A, classifying them as members or nonmembers of the unknown regular set U. This information is organized into an observation table, consisting of three things: a non-empty finite prefix-closed set S of strings, a non-empty finite suffix-closed set E of strings, and a finite function T mapping $((S \cup S.A) . E)$ to $\{0, 1\}$. The observation table will be denoted (S, E, T) (A set is prefix-closed if and only if every prefix of every member of the set is also a member of the set. Suffix-closed is defined analogously).

The interpretation of T is that T(u) is 1 if u is a member of the unknown regular set, U. The observation table initially has $S = E - \{\lambda\}$, and is augmented as the algorithm runs. An observation table can be visualized as a two-dimensional array with rows labeled by elements of $(S \cup S.A)$ and columns labeled by elements of E with the entry for row s and column e to T(s. e). If s is an element of $(S \cup S.A)$, then row(s) denotes the finite function f from e to {0, 1} defined by f(e) = T(s. e).

An observation table is called closed provided that for each t in S. A there exists an s in S such that row (t) = row (s). An observation table is called consistent provided that whenever s_1 and s_2 are elements of S such that row (s_1) = row (s_2), for all a in A, row $(s_1. a)$ = row $(s_2. a)$. If (S, E, T) is closed and consistent observation table, then a corresponding acceptor M(S, E, T) over the alphabet A, with state set Q is described. The initial state q_0, accepting states F, and transition function δ is as follows.

$$Q = \{\text{row}(s);\ s \in S\}$$
$$q_0 = \text{row}(\lambda)$$
$$F = \{\text{row}(s);\ s \in S \text{ and } T(s) = 1\},$$
$$\delta(\text{row}(s), a) = \text{row}(s.a)$$

6. Learning Simple Splicing Grammar Systems

Consider the situation where the learning algorithm is allowed to make queries to an oracle. In [4], the notion of "minimally adequate teacher" (MAT) is introduced and the teacher (Oracle) answers membership and equivalence queries in order to construct a learning algorithm for regular sets. In [5], the notions of subset and superset queries are introduced. For a subset (superset) query, the input is a concept C and the output is 'yes' if C is a subset (superset) of the target concept C_* and 'no' otherwise. If the answer is 'no', counter example x from $C - C_*$ $(C_* - C)$ is also returned. Restricted subset queries and restricted superset queries, where no counter example is returned are also introduced in [5].

We recall that a word $u \in T^*$ is a prefix of another word $w \in T^*$, if there exists a word $v \in T^*$, such that w = uv. Thus in a prefix query, the concept to be learnt is usually a word w over the underlying alphabet T. The input is a word $u \in T^*$ and the output is "yes", if u is a prefix of w and "no" otherwise.

We learn a <1, 3> simple splicing grammar system where the master is a regular grammar and the other is a non erasing pattern grammar with a single pattern which is in canonical form. The technique of the algorithm is as follows:

A pattern with k variables is a word over $(T \cup \{x_1, x_2,\ldots, x_n\})$. Elements of T are called constants while $x_i, \in X$ are called variables. A pattern p with k variables is said to be in canonical form if, for each $i \leq k$, the leftmost occurrence of x_i in p occurs to the left of leftmost occurrence of x_{i+1}. For a pattern p with k variables, and a set of k strings $u_1, u_2,\ldots,u_k \in T^*$ let $p[x_1{:}u_1, x_2{:}u_2,\ldots, x_k{:}u_k]$ denote the string obtained by substituting u_1 for each occurrence of x_i in p. The language $\{p[x_1{:}u_1, x_2{:}u_2,\ldots, x_k{:}u_k] \,/\, u_1, u_2,\ldots,u_k \in T^*\}$ generated by using substitutions of this type is the language generated by the pattern p. The class of all k variable patterns is denoted by P_k.

We now present an algorithm that exactly identifies in polynomial time the class P_k.of pattern languages using prefix queries. Let $p = p_1p_2\ldots p_n$ be the pattern to be identified. We begin by checking whether p_1 is a constant. Hence for each $a \in T$ we make a prefix query for a.. If the output is "no" to each of these queries we conclude that p_1 is a variable and since p is in canonical form $p_1 = x_1$.

Suppose at some stage we have discovered that $p_1p_2\, p_3\ldots p_i$ is a prefix of p and $j = \max\{r \,/\, p_s = x_r$ for $1 \leq s \leq i\}$. Again we check whether p_{i+1} is a constant by making prefix query for $p_1\, p_2\, p_3\ldots p_i\, a$, $a \in T$. As before if each of these queries yields a negative answer, we conclude that p_{i+1} is a variable and query whether $p_1\, p_2\ldots p_i x_r$ is a prefix for each $r \leq j + 1$. We conclude that the pattern is complete if each of these queries receives a negative reply.

Now, to learn axiom set A, initially fix $A = \phi$. Arrange the words in $\bigcup_{i=1}^{m} T^i$ (m the maximum length of the axiom is known) are arranged according to increasing order of length and among the words of equal length lexicographically. Let them be $x_1, x_2\ldots x_s$. At the tth step ask the restricted subset query for $(T, A \cup \{x_t\}, p)$. If the answer is 'yes', increment A to $A \cup \{x_t\}$. If the answer is 'no', A is not incremented. The output at the last step is the required PG.

Now as the pattern grammar is known, we assume first, M = {α # $ α #}, which is <1, 3> splicing rule and split the sample word into two using M, where the first part of sample is a subword of the required regular set and the second part is a subword of the required pattern grammar. The regular set is learnt using L^*.

Algorithm
Input:

The alphabets T, N, X, a positive sample w from $L(\Gamma_1)$, $w \in T^+$ of length r, the length ‘n’ of the pattern, the maximum length ‘m’ of the axiom, $r \geq n$, words t_1, $t_2 \ldots$, t_s of $\bigcup_{i=1}^{m} T^i$ given in the increasing length order, among words of equal length according to lexicographic order.

Output:

A Simple Splicing Pattern grammar system $\Gamma' = (N, T, X, (S, P_1), (A, P_2), M)$ with $L(\Gamma') = L(\Gamma_1)$

Procedure (Pattern)

```
Module 1
 i = 0, p = λ, number of characters in the pattern is n
First set
        i = i +1, If i > n stop
        For each a ∈ T
        begin
                Ask prefix query for pa
                  if answer is "yes" then do
                    begin
                        p = pa
                        Go to First set

                    end
        end
Module 2
Second set
        k = 1 to j
        for δ_k ∈ X
        begin
                Ask prefix query for δ_k
                 if answer is "yes" then do
                   begin
                        p = pδ_k
                        Go to First set
                   end
        end
                Ask prefix query for δ_j
                if answer is "yes" then do
                begin
                        p = pδ_j
                        j = j + 1
                        Go to First set
                end
                If i is equal to n
        end
end
```

Procedure (Axiom)

```
Let x_1, x_2, …, x_s be the words in ⋃_{i=1}^{m} T^i arranged in lexicographic order
A = ϕ
        for t = 1 to s do
                begin
                        ask restricted subset query for G = (T, X, A ∪ {x_t}, {p})
                                If 'yes' then A = A ∪ {x} and t = t + 1
                        else output G
                end
Print the pattern grammar (T, X, A, p)
```

Procedure (Master)

```
For each α ∈ T
        Let M = {α # $ α #}
          Cut the sample word w = u α v into two, at the position after reading α
            begin
                        As u α ∈ L (N, T ∪ {α}, P_1, S)
                                run L* using prefixes of u α . If L* gives the
                correct automaton, write the corresponding regular
                                grammar which is equivalent to G_0 = (N, T ∪ {α}, P_1, S)
                                and write the splicing rule M = {α # $ α #}
                                Print Γ' = (N, T, (S, P_1), (A, P_2), M) the Simple Splicing Pattern Grammar system
                else
                M = {α # $ α #}
          end
        end
```

Time Analysis: The algorithm given above considers a <1, 3> simple splicing grammar system where the master is a regular grammar and the other component is a non erasing pattern grammar with a single pattern of length n which is in canonical form. Clearly the number of queries needed to identify a pattern $p = p_1 p_2 \ldots p_n$ of length n is bounded by $n(|T| + k)$ since a maximum of $|T| + k$ queries are required to identify each p_i. The running time for L^* is polynomial in the number of states of the minimum acceptor and the maximum length of counter examples. Hence, the implementation time for this algorithm is bounded by quadratic expression in n.

7. Example run for Simple Splicing Pattern Grammar System

From the Splicing language generated by the simple splicing pattern grammar System (example 4.1) $\Gamma = (N, T, (S, P_1), (A, P_2), M)$, let the sample given be aaacbbb. To learn pattern grammar it is enough if we find the pattern p and the axiom set A. Here the length of the pattern is two and maximum length of the axiom is one and the alphabet T = {a, b, c}. Let $p = p_1p_2$. First we check whether p_1 is a constant. Thus for $a \in T$, a prefix query is asked. The answer will be "no" since the pattern is δb Again for $b \in T$, a prefix query is asked and again the answer will be "no". So another prefix query for $c \in T$ is asked and again the answer will be "no". Thus $p_1 = \delta$ is learnt. Now for $a \in T$ we ask a prefix query for δa. The answer will be "no". Again for $b \in T$, a prefix query is asked for δb. The answer will be "yes". Hence the pattern δb is learnt.

Now, to learn axiom set A, initially fix $A = \phi$. The words in T are considered. Let them be a, b, c. Now the restricted subset query for $(T, A \cup \{a\}, p)$ is asked. As the answer is 'no', ask one more subset query $(T, A \cup \{b\}, p)$. Here the teacher answers no. Then we ask subset query $(T, A \cup \{c\} , p)$. Now the teacher answers 'yes', thus the axiom set is learnt which is {c}. As the pattern and axiom are learnt the output is the required pure pattern grammar $\Gamma =$ ({a,b, c}, {c} { δb})

From the sample given we learn the simple splicing pattern grammar system with <1, 3> splicing rule.

As we know the type of the splicing rule we first assume the splicing rule set M = {α # $ α #}. Then for $\alpha \in T$, we divide the sample word in to two at the position after α. In our example first we assume M = {a # $ a #} and split the sample word aaacbbb into two subwords of the form a | aacbb.

Now as the pattern grammar is known, we learn the regular language using L^* having 'a' as a prefix of a word belonging to the required regular language. But as the splicing rule M = {a # $ a #} is not the correct rule L^* will not give the correct automaton. So again we start with M = {b # $ b #} and we learn regular language . Again L^* will not be able to give correct automaton. This process is repeated till we get the correct splicing rule M = {c # $ c #}. Now as the pattern grammar is known, we try to learn the regular grammar using L^* having 'a' as a substring of the required regular grammar. But as the splicing rule M = {a # $ a #} is not the correct rule L^* will not give the correct automaton. So again we start with the rule M = {b # $ b #} and we try to learn the pattern grammar and the regular grammar. Again L* will not be able to the correct automaton. This processes is repeated till we get the correct splicing rule M = {c # $ c #}. Now the sample word is divided into a^3c and b^3. With the subword a^3c, L^* is learnt.

Now as the pattern grammar is known, we try to learn the regular grammar. The learning is as follows:
Let $U = \{a\}^+ c$ be the member of the required regular set, then we define A = {λ, a, c}. To start with take only λ and a in S.

Table 1 is closed and consistent but the language is not accepted. Thus instead of getting a counter example from the teacher we add c to E and table 2 is constructed. Table 2 is closed but not consistent. So we add ac to E and c to S and Table 3 is constructed. But Table 3 is not closed, thus we add ac in S and Table 4 is constructed. Table 4 is closed and consistent; the language is accepted by the teacher. The transition table of acceptor is shown in Table 5.

From the automaton rules $q_0 \rightarrow aq_1$, $q_0 \rightarrow cq_2$, $q_1 \rightarrow aq_1$, $q_1 \rightarrow cq_3$, $q_2 \rightarrow aq_2$, $q_2 \rightarrow cq_2$, $q_3 \rightarrow aq_2$, $q_3 \rightarrow cq_2$ which generates the required regular set $\{a\}^+c$ are written. Here q_0 is initial state and q_3 is finale state. Finally after learning the regular set and pattern grammar the output is an equivalent simple splicing pattern grammar system. Now as pattern grammar, the regular set and the splicing rule are known we can write a Simple Splicing grammar system $\Gamma' =$ (N, T, (S, P_1), (A, P_2), M) with $L(\Gamma') = L($[illegible]

T_1	λ
Λ	0
A	0
C	0
aa	0
ac	0

Table 1

T_2	λ	c
λ	0	0
a	0	1
c	0	1
aa	0	1
ac	0	0

Table 2

T_3	λ	c	ac
λ	0	0	1
a	0	1	1
c	0	0	0
aa	0	1	1
ac	1	0	0
ca	0	0	0
cc	0	0	0

Table 4

T_3	λ	c	ac
λ	0	0	1
a	0	1	1
c	0	0	0
ac	1	0	0
aa	0	1	1
ca	0	0	0
cc	0	0	0
aca	0	0	0
acc	0	0	0

Table3 3

δ	a	c
q_0	q_1	q_2
q_1	q_1	q_3
q_2	q_2	q_2
q_3	q_2	q_2

Table 5

References

[1] Abisha, P. J., Subramanian, K. G., Thomas, D. G.: Pure Pattern Grammars, Proceedings of international Workshop on Grammar systems. 253 – 262, Austria (2000).

[2] Abisha, P. J., Thomas, D. G., Sindhu J Kumaar.: Learning subclass of Pure Pattern Languages. Proc. of International Colloquium on Grammatical Inference ICGI 2008, LNAI 5278, 80-283, Saint Malo, France (2008).

[3] Angluin, D.: Finding patterns common to a set of strings, Journal of Computer and System Sciences. 21, 46 – 62 (1980),

[4] Angluin, D.: Learning regular sets from queries and counter examples, Information and computation. 75, 87 – 106 (1987).

[5] Angluin, D.: Queries and concept learning, Machine Learning. 2, 319-342 (1988).

[6] Dassow, J., Mitrana. V.: Splicing Grammar Systems, Computers and AI, 15(2 -3), 109 – 122 (1996).

[7] Dassow, J., Paun, Gh., Salomaa, A.: Grammars based on patterns, International Journal of Foundations of Computer Science. 4, 1 – 14 (1993).

[8] Dassow, J., Paun, Gh., Rozenberg, G.: Generating languages in a distributed way: Grammar systems. In: Rozenberg, G., Salomaa, A.(eds.) Handbook of formal languages. Springer, Heidelberg (1997).

[9] Dersanambika, K. S., Krithivasan, K., Subramanian, K. G.: Simple splicing grammar systems. Proceedings of Grammar System Week, 170–178, (2004).

[10] Gold, E. M.: Language identification in the limit, Information and Control. 10, 447-474 (1967).

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

[12] Head, T., Paun, Gh., Pixton, D.: Language theory and molecular genetics: Generative mechanisms suggested by DNA recombination. In: Hand Book of Formal Languages, Rozenberg, R.., Saloma, A. (eds), pp. 295–360.
Springer (1997).

[13] Mateescu, A., Paun, Gh., Rozenberg, G., Salomaa, A: Simple splicing systems. Discrete Applied Mathematics. 84, 145 -163, (1998).

[14] Maurer, H. A., Salomaa, A., Wood, D: Pure grammars. Information and Control. 44, 47 – 72, (1980).

[15] Paun, Gh., Santean, L.: Parallel communicating grammar systems: the regular case. Ann. Univ. Buc., Series Mathem. – Inform. 38, 55 – 63, (1989).

[16] Salomaa, A.: Formal Languages. Academic Press, New York (1973).

[17] Sindhu J Kumaar, Abisha, A. J., Thomas, D. G.: Learning Subclasses of Parallel Communicating Grammar Systems. Proceedings of the conference International Colloquium on Grammatical Inference. LNAI 6339. 301 – 304, (2010).

[18] Sindhu J Kumaar , Abisha, P. J., Thomas, D. G.: Simple Splicing Synchronized Pattern and Pure Pattern Grammar systems. Proceedings of the conference Bio Inspired Computing: Theories and Applications. 220 – 224, (2011).