Learning of Context-Sensitive Languages Described by Augmented Regular Expressions

René Alquézar(*) and Alberto Sanfeliu(**)
(*)*Dept. LSI, Universitat Politècnica de Catalunya, Diagonal 647, 8a, 08028 Barcelona (Spain)*
(**)*Institut de Robòtica i Informàtica Industrial, UPC-CSIC, Barcelona*
alquezar@lsi.upc.es, sanfeliu@ic.upc.es

Abstract

Recently, Augmented Regular Expressions (AREs) have been proposed as a formalism to describe and recognize a non-trivial class of context-sensitive languages (CSLs) [1, 2]. AREs augment the expressive power of Regular Expressions (REs) by including a set of constraints, that involve the number of instances in a string of the operands of the star operations of a RE. Although it has been demonstrated that not all the CSLs can be described by AREs, the class of representable objects includes planar shapes with symmetries, which is important for pattern recognition tasks. In this paper, a general method to infer AREs from string examples is presented. The method consists of a regular grammatical inference step, aimed at obtaining a regular superset of the target language, followed by a constraint induction process, which reduces the extension of the inferred language attempting to discover the maximal number of context relations. Hence, this approach avoids the difficulty of learning context-sensitive grammars.

1. Introduction

In order to extend the potential of application of the syntactic approach to pattern recognition, the efficient use of models capable of describing context-sensitive languages (CSLs) is needed. Moreover, learning such models from examples is interesting both for theoretical and practical purposes. Context-sensitive grammars [3] are not a good choice, since their parsing is computationally expensive and there is a lack of methods to learn them. Augmented Transition Networks (ATNs) are powerful models that have been used in natural language processing, but which are very difficult to infer [4]. Pattern languages permit the repetition of variable substrings along the strings of a language, and there are methods to infer them from examples [5], but their expressive power is clearly insufficient even to describe simple context-sensitive structures such as rectangles.

The problem of learning formal languages is traditionally referred to as *grammatical inference* (GI). The most part of GI research has been devoted to the theory and methods for learning regular languages (or finite-state automata) [6, 7]. In addition, some GI methods have been suggested to learn proper subclasses of context-free languages (CFLs), such as the even linear languages [8], or general context-free grammars from positive structural examples [9]. However, there is a need for GI methods capable of inferring CSLs, specially for pattern recognition tasks in computer vision, where objects usually contain structural relationships that are not describable by CFLs. Unfortunately, work on CSL learning is extremely scarce in the literature. Recently, Takada has shown that a hierarchy of language families that are properly contained in the family of CSLs can be learned using regular GI algorithms [10].

More recently, *Augmented Regular Expressions* (AREs) have been proposed by us as a formalism to describe and recognize a class of CSLs, that covers planar shapes with symmetries [1, 2]. An ARE $\tilde{R}$ is formed by a regular expression (RE) R, in which the stars are replaced by natural-valued variables, called *star variables*, and these are related through a finite number of constraints (linear equations). Hence, REs are reduced to AREs with zero constraints among the star variables. A general approach to infer AREs from string examples is proposed here, that is based on a regular GI step followed by an inductive process which tries to discover the maximal number of context constraints.

2. Augmented Regular Expressions (AREs)

Let $\Sigma = \{a_1, ..., a_m\}$ be an *alphabet* and let λ denote the *empty string*. The *regular expressions (REs) over* Σ and the languages that they describe are defined recursively as follows: $\emptyset$ and λ are REs that describe the empty set and the set $\{\lambda\}$, respectively; for each $a_i \in \Sigma$ $(1 \leq i \leq m)$, a_i is an RE that describes the set $\{a_i\}$; if P and Q are REs describing the languages L_P and L_Q, respectively, then $(P+Q)$, (PQ), and (P^*) are REs that describe the languages

1015-4651/96 $5.00 © 1996 IEEE
Proceedings of ICPR '96

$L_P \cup L_Q$, $L_P L_Q$ and L_P^*, respectively. By convention, the precedence of the operations in decreasing order is $*$ (star), (concatenation), $+$ (union). This precedence together with the associativity of the concatenation and union operations allows to omit many parentheses in writing an RE. The language described by an RE R is denoted $L(R)$. Two REs P and Q are said to be *equivalent*, denoted by $P = Q$, if they describe the same language. REs and finite-state automata (FSA) are alternative representations of the class of regular languages, and there are algorithms to find an RE equivalent to a given FSA and viceversa [11].

Let R be a given RE including ns star symbols ($ns \geq 0$). The set of *star variables* associated with R is an ordered set of natural-valued variables $V = \{v_1, ..., v_{ns}\}$, which are associated one-to-one with the star symbols that appear in R in a left-to-right scan. For $v_i, v_j \in V$, we say that v_i *contains* v_j iff the operand of the star associated with v_i in R includes the star corresponding to v_j; and we say v_i *directly-contains* v_j iff v_i *contains* v_j and there is no $v_k \in V$ such that v_i *contains* v_k and v_k *contains* v_j. The *star tree* $\mathcal{T} = (N, E, r)$ associated with R is a general tree in which the root node r is a special symbol, the set of nodes is $N = V \cup \{r\}$, and the set of edges E is defined by the containment relationships of the star variables: (i) $\forall v_i \in V$: $(\neg\exists v_k \in V,\ v_k$ *contains* $v_i) \implies (r, v_i) \in E$; (ii) $\forall v_i, v_j \in V,\ i \neq j$: v_i *directly-contains* $v_j \implies (v_i, v_j) \in E$. An algorithm to build the *star tree* $\mathcal{T}$ has been reported [1], with a time complexity of $O(|R| \cdot h(R))$, where $h(R)$ is the depth of non-removable parentheses in R.

A star variable v can take as value any natural number, whose meaning is the number of consecutive times (cycles) the operand of the corresponding star (an RE) is instantiated while matching a given string. In such a case, we say that the star variable is *instantiated*. Given a certain string s belonging to the language described by the RE R from which V has been defined, a data structure $SI_s(V) = \{SI_s(v_1), ...SI_s(v_{ns})\}$, called the set of *star instances* (of the star variables in V for s), can be built during the process of parsing s by R. Each member of the set $SI_s(V)$ is a list of lists containing the instances of a particular star variable: $\forall i \in [1..ns]$: $SI_s(v_i) = (l_1^i \ ... \ l_{nlists(i)}^i)$ where $nlists(i) \geq 0$, and $\forall i \in [1..ns]\ \forall j \in [1..nlists(i)]$: $l_j^i = (e_{j1}^i \ ... \ e_{j(nelems(i,j))}^i)$ where $nelems(i,j) \geq 1$.

The star instances stored in $SI_s(V)$ are organized according to the containment relationships described by $\mathcal{T}$. To this end, each list l_j^i is associated with two pointers *father_list*(l_j^i) and *father_elem*(l_j^i) that identify the instance of the father star variable from which the instances of v_i in l_j^i are derived. All the star variables that are brothers in the star tree $\mathcal{T}$ will have the same structure of instances, provided that a special value, say -1, is stored whenever a star variable is not instantiated during a cycle of an instance of its father. Fig.1 shows an example of the set of star instances resulting from the parse of a string by an RE. Two efficient algorithms for *unambiguous** RE parsing that construct the star instances structure have been reported [1].

$R = (a(b(ce^*c + df^*d)^*)^*)^*$
$R(V/*) = (a(b(ce^{v_1}c + df^{v_2}d)^{v_3})^{v_4})^{v_5}$
$s = abccdffdcecbddbdfdceecabceeec$
$SI_s(v_5) = ((2)^{\{-1,-1\}})$
$SI_s(v_4) = ((3\ 1)^{\{1,1\}})$
$SI_s(v_3) = ((3\ 1\ 2)^{\{1,1\}}\ (1)^{\{1,2\}})$
$SI_s(v_1) = ((0\ -1\ 1)^{\{1,1\}}\ (-1)^{\{1,2\}}\ (-1\ 2)^{\{1,3\}}\ (3)^{\{2,1\}})$
$SI_s(v_2) = ((-1\ 2\ -1)^{\{1,1\}}\ (0)^{\{1,2\}}\ (1\ -1)^{\{1,3\}}\ (-1)^{\{2,1\}})$

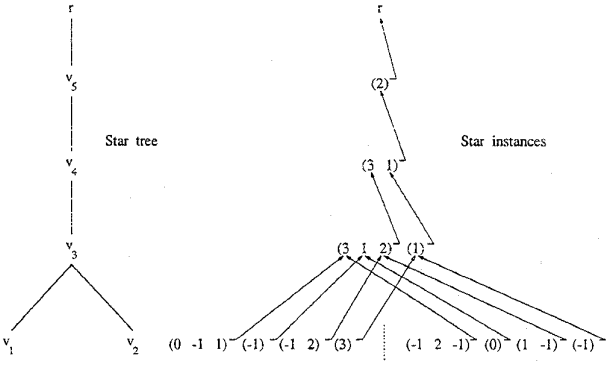

Fig. 1 *An example of star instances data structure.*

Given a star tree $\mathcal{T}$, a set of star instances $SI_s(V)$ for a certain string s, and two nodes $v_i, v_j \in V$, we say that v_i is a *degenerated ancestor* of v_j (for s) iff v_i is an ancestor of v_j in $\mathcal{T}$ and for each instance of v_i in $SI_s(v_i)$ all the values of the instances of v_j in $SI_s(v_j)$ that are derived from it are constant. By definition, the root r is a non-degenerated ancestor of any other node v_j. Let $v_i \in V \cup \{r\}$, $v_j \in V$; we say that v_i is the *housing ancestor* of v_j (for s) iff v_i is the *nearest non-degenerated ancestor* of v_j (for s). When a node is not housed by its father, then its redundant instances can be collapsed into the same list structure of its father, and this step may be done several times until the housing ancestor is reached.

An *Augmented Regular Expression* (or ARE) is a four-tupla $(R, V, \mathcal{T}, \mathcal{L})$, where R is a regular expression over an alphabet Σ, V is its associated set of *star variables*, $\mathcal{T}$ is its associated *star tree*, and $\mathcal{L}$ is a set of independent linear relations $\{l_1, ..., l_{nc}\}$, that partition the set V into two subsets V^{ind}, V^{dep} of independent and dependent star variables, respectively; this is, for $1 \leq i \leq nc$,

$$l_i \text{ is } v_i^{dep} = a_{i1}v_1^{ind} + .. + a_{ij}v_j^{ind} + .. + a_{i(ni)}v_{ni}^{ind} + a_{i0},$$

where ni and nc are the number of independent and dependent star variables, respectively, $ns = nc + ni$, and the

*An RE R is *ambiguous* if there exists a string $s \in L(R)$ for which more than one parse of s by R can be made.

coefficients a_{ij} are always rational numbers.

Given a set of star instances $SI_s(V)$, a star tree $\mathcal{T}$, and a set of linear equations $\mathcal{L}$, let rewrite each constraint $l_i \in \mathcal{L}$ by removing all the terms of independent variables with coefficient zero in the right hand sides of the equations, i.e. l_i is $v_i^{dep} = a_{i1}v'_1 + .. + a_{ik_i}v'_{k_i} + a_{i0}$, for $1 \leq i \leq nc$, such that $\forall j \in [1, k_i] : a_{ij} \neq 0$. Let $v_c \in V$ be the *deepest common ancestor* in $\mathcal{T}$ of the nodes $\{v_i^{dep}, v'_1, ..., v'_{k_i}\}$. Then, we say that $SI_s(V)$ *satisfy* a constraint $l_i \in \mathcal{L}$ iff

i) the *housing ancestors* (for s) of the nodes $\{v_i^{dep}, v'_1, ..., v'_{k_i}\}$ are either v_c or an ancestor of v_c, or they satisfy a strict equality constraint, and

ii) the linear relation l_i is met by the corresponding instances of $\{v_i^{dep}, v'_1, ..., v'_{k_i}\}$ that are derived for each instance of v_c.

The first condition above implies structural similarity of instance lists, while the second one requires the satisfaction of the equation. The star instances $SI_s(V)$ *satisfy* $\mathcal{L}$ iff $SI_s(V)$ satisfy each constraint $l_i \in \mathcal{L}$, for $1 \leq i \leq nc$.

Finally, let $\tilde{R} = (R, V, \mathcal{T}, \mathcal{L})$ be an ARE over Σ, the language $L(\tilde{R})$ represented by $\tilde{R}$ is defined as $L(\tilde{R}) = \{\alpha \in \Sigma^* \mid \alpha \in L(R)$ and there exists a parse of α by R in which $SI_\alpha(V)$ *satisfy* $\mathcal{L}\}$.

The recognition of a string s as belonging to a language $L(\tilde{R})$ can be clearly divided in two steps: parsing s by R, and if success, checking the satisfaction of constraints $\mathcal{L}$ by the star instances $SI_s(V)$ that result from the parse. If R is *unambiguous*, a unique parse and set of star instances $SI_s(V)$ is possible for each $s \in L(R)$, and therefore a single satisfaction problem must be analysed to test whether $s \in L(\tilde{R})$. An algorithm for constraint testing has been reported [2], that runs in $O(|\mathcal{L}| \cdot height(\mathcal{T}) \cdot |V| \cdot I(SI_s(V)))$, where $I(SI_s(V)) = \max_{i=1,|V|} \sum_{j=1}^{nlists(i)} nelems(i,j)$ is the maximal number of instances of a star variable yielded by parsing s.

AREs permit to describe a class of context-sensitive languages by defining a set of constraints that reduce the extension of the underlying regular language. A very simple example is the language of rectangles described by the ARE $\tilde{R}_1 = (R_1, V_1, \mathcal{T}_1, L_1)$, with $R_1(V_1/*) = aa^{v_1}bb^{v_2}aa^{v_3}bb^{v_4}$ and $L_1 = \{v_3 = v_1, v_4 = v_2\}$. However, quite more complex languages with an arbitrary level of star embedment in the RE can be described as well by the ARE formalism. It has been demonstrated that not all the CSLs can be described by an ARE [2]. On the other hand, AREs cover all the pattern languages, but the size of an ARE describing a pattern language over Σ is exponential in $|\Sigma|$ [2].

3. Inference of AREs from string examples

Now, let us consider the problem of learning AREs from examples. This is, given a sample (S^+, S^-) of an unknown language L, *infer* an ARE $\tilde{R} = (R, V, \mathcal{T}, \mathcal{L})$ such that $S^+ \subseteq L(\tilde{R})$ and $S^- \cap L(\tilde{R}) = \emptyset$, and $\tilde{R}$ is determined through some *heuristic* bias. A possible approach is to split the process in two main stages: inferring the underlying RE R, and afterwards, inducing the constraints $\mathcal{L}$. A nice property of this approach is that if the target RE R_T is identified in the first stage, then the target unknown ARE $\tilde{R}_T$ (with $L(\tilde{R}_T) = L$) can be identified in the second stage (in the limit) by inducing the maximal number of constraints satisfied by the positive examples.

On the other hand, in order to infer the RE R correctly, we should be able to partition the negative sample S^- (if any) in the two subsets S_0^- and $(S^- - S_0^-)$, characterized by $S_0^- \cap L(R) = \emptyset$ and $(S^- - S_0^-) \subseteq L(R)$. Supplying the entire S^- to the regular GI method will typically cause an overfitting of the positive examples. Thus, unless a teacher is available to partition the negative examples, or it is guaranteed that $S^- = S_0^-$, a heuristic method based only on the positive examples should be preferred for the regular inference step.

Let us illustrate the proposed method using an actual example shown in Fig.2. The problem was to learn a recognizer for the class of contours coming from a frontal view of variable-size cylinders with a fixed-size dent at a variable position along the axis. It is clear that the associated language is context-sensitive, and hence, we cannot expect that a regular or a CFL GI algorithm infers a suitable recognizer. Nevertheless, an adequate description like $a^m c^n bdc^p a^m c^p bdc^n$ should be reachable from a few examples. In the case of Fig.2, a sample $S = (S^+, S^-)$ of 16 positive and 48 negative examples was given, corresponding to some variable-size instances of the contours shown in the top. Assuming $S^- = S_0^-$, the *active* regular GI method [7] was applied to the entire S, and the DFA displayed in Fig.2 was obtained. This DFA accounts for the basic repetitive structure of the model, but it over-generalizes a lot, accepting many invalid contours without any length restriction. From this DFA, an equivalent RE was obtained as base of the ARE (Fig.2). Finally, from the analysis of the star instances produced by *parsing* the positive examples by the RE, a set of constraints could be induced that, in conjunction with the RE, perfectly described the target language.

The data flow of the process followed to infer an ARE from examples is depicted in Fig.3. A wide range of algorithms is available to perform the regular GI step [6, 7], but most of them return an FSA. In these cases, we need to find an RE equivalent to the inferred FSA to build the ARE. Given an FSA A, there can be many equivalent REs R satisfying $L(A) = L(R)$, and several algorithms are applicable to obtain such an RE [11]. By selecting a specific algorithm, a mapping $\psi : FSA \rightarrow REs$ can be established, i.e. a canonical RE R can be chosen for each FSA A, $R = \psi(A)$.

The mapping ψ that we have selected is based on Arden's method [11], but, in order to facilitate the ARE in-

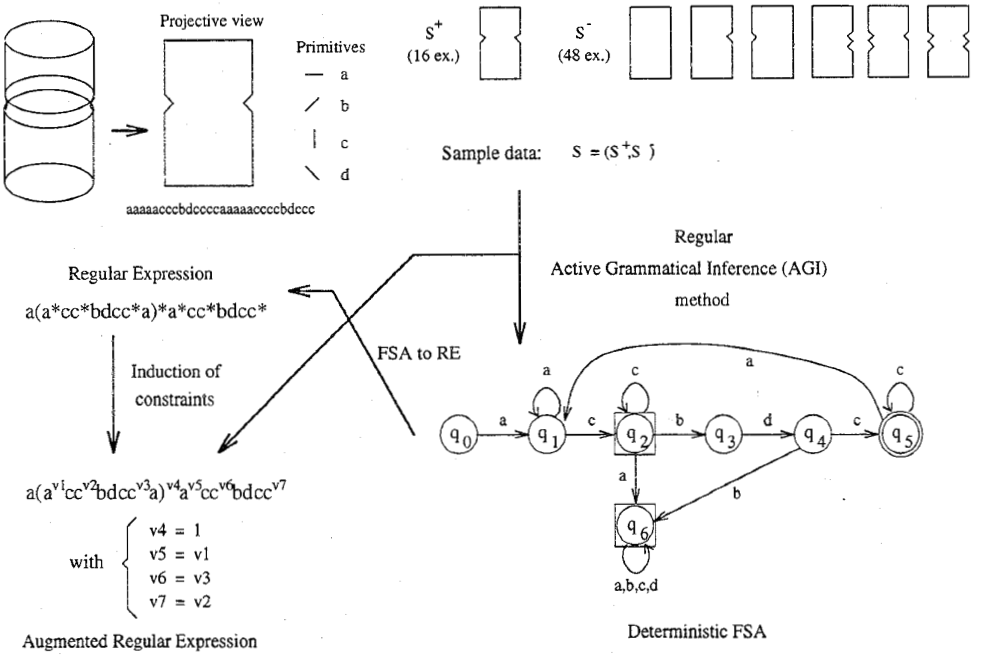

Fig. 2 *An example of application of the proposed method for inferring AREs.*

duction, an inner modification of the algorithm and a final simplifying step have been proposed [2]. Given a DFA, the resulting RE is always unambiguous (thus easing the RE parsing) and such that the star subexpressions due to loops are distinguished from those corresponding to the rest of circuits of the automaton. Since loops may represent indefinite length or duration of a basic primitive, this separation allows for a later induction of constraints relating the lengths or durations of the different parts of a pattern. Moreover, by increasing the number of stars in the RE, while preserving equivalence and unambiguity, through the use of the equivalence rule $(P+Q)^* = (P^*Q)^*P^*$, the potential for inferring constraints that involve the instances of the star operations is also increased.

Once an RE R is inferred, the star variables V and the star tree $\mathcal{T}$ associated with R are easily determined [1]. Then, the aim is to induce an ARE $\bar{R} = (R, V, \mathcal{T}, \mathcal{L})$ such that $\mathcal{L}$ contains the maximal number of (linear) constraints met by all the provided examples. To this end, the positive strings must be parsed by R giving rise to an array of sets of star instances ASI, and those regularities that consistently appear throughout the star instances must be discovered. The complexity of parsing a string s by an RE R is $O(|s| \cdot |R|)$, but if the equivalent DFA A is also available, a more efficient parsing technique can be applied [1].

For the last step, a constraint induction algorithm has been reported [1], that returns a set of linear constraints $\mathcal{L}$ among the star variables V, with a time complexity of $O(|V|^3 \cdot I(ASI_{S^+}(V)))$, where $I(ASI_{S^+}(V))$ is the maximal number of instances of a star variable yielded by parsing the set of strings S^+. This algorithm is based on establishing a tree of linear systems according to the *housing ancestor* concept. Each *housing ancestor* will have its own partition of independent and dependent star variables among its *housed descendents*. To construct this partition, each ancestor node of $\mathcal{T}$ keeps track of its *housed descendents* that have been found independent. All the variables of $\mathcal{T}$ are visited by levels, and for each one (say v_j), its housing ancestor (say v_k) is determined and a vector of its non-redundant instances is formed. Then a matrix is built that contains the instances of the independent housed descendents of v_k. Next, the rank of the matrix is evaluated and any linearly dependent column is removed. Finally, it is determined whether the vector of actual instances of v_j is linearly dependent on the columns of the matrix. If it is, a linear system can be solved to find the constraint coefficients, and the new constraint is appended to $\mathcal{L}$; otherwise, v_j is put in the list of independent housed descendents of v_k.

4. Conclusions

Augmented Regular Expressions (AREs) are compact (and intelligible) descriptions that represent a non-trivial class of context-sensitive languages, including pattern lan-

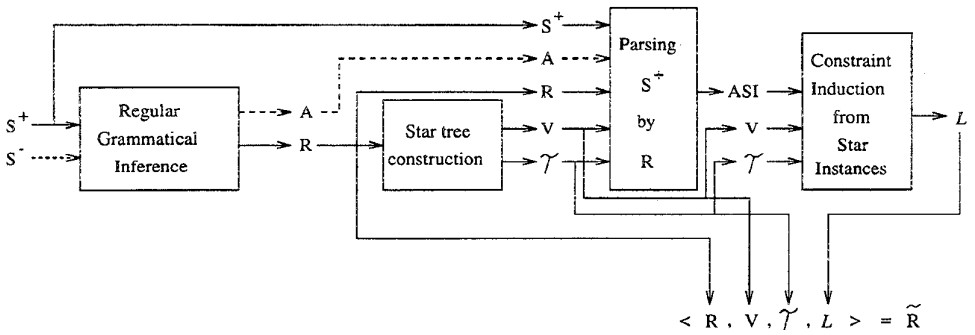

Fig. 3 *Block diagram of a general method to learn Augmented Regular Expressions from string examples.*

guages [2], contours of planar shapes with symmetries, and other complex patterns. Furthermore, the recognition of a string as belonging to the language described by an ARE, which is based on parsing by the underlying RE and checking the constraints, is efficient (to the contrary of parsing by a context-sensitive grammar) [1, 2].

A general method to learn AREs from examples has been proposed here, which is based on splitting the process in two main stages: inferring the underlying RE and inducing the maximal number of constraints afterwards. This learning strategy is not conceived as an identification method, but a heuristic method in which the inferred ARE strongly depends on the result of the regular GI algorithm used in the former stage. However, if the target RE were identified firstly, the target ARE, which includes it, would also be identified in the limit, since the constraint induction process finds the maximal number of context linear relations met by all the examples (i.e. the ARE with the smallest language containing the examples among the AREs that include the same RE). On the other hand, if negative examples are supplied, then the learning method cannot know which of them should belong to the underlying regular language and which should not, unless an informant classified them in advance.

Since most of the known regular GI methods yield a finite-state automaton (FSA) [6], a specific FSA to RE mapping has been proposed to obtain an equivalent unambiguous RE [2]. Nevertheless, a method providing directly an RE, such as the uv^kw algorithm [6], would be preferred due to the exponential worst-case complexity of the FSA to RE transformation. Likewise, the algorithm to be used in the regular GI step should return preferrably a small RE representing a high generalization with respect to the sample, both to ease the parsing of examples by the RE and to permit the discovery of the target context relations. The constraint induction could be impeded if the regular language were too restricted to the given examples (sample overfitting).

References

[1] R. Alquezar and A. Sanfeliu, "Augmented regular expressions: a formalism to describe, recognize, and learn a class of context-sensitive languages," *Research Report LSI-95-17-R*, Univ. Politecnica de Catalunya, Barcelona, 1995.

[2] R. Alquezar and A. Sanfeliu, "Recognition and learning of a class of context-sensitive languages described by augmented regular expressions," *Pattern Recognition*, in press, 1996.

[3] A. Salomaa, *Formal Languages*, Academic Press, New York, 1973.

[4] S.M. Chou and K.S.Fu, "Inference for transition network grammars," *Proc. 3rd Int. Joint Conf. on Pattern Recognition*, CA, pp.79-84, 1976.

[5] D. Angluin, "Finding patterns common to a set of strings," *J. Comput. Syst. Sci.*, Vol.21, pp.46-62, 1980.

[6] J. Gregor, "Data-driven inductive inference of finite-state automata," *Int. J. of Pattern Recognition and Artificial Intelligence*, Vol.8, No.1, pp.305-322, 1994.

[7] A. Sanfeliu and R. Alquezar, "Active grammatical inference: a new learning methodology," in *Shape, Structure and Pattern Recogniton*, D.Dori and A.Bruckstein (eds.), World Scientific Pub., Singapore, pp.191-200, 1995.

[8] V. Radhakrishnan, G. Nagaraja, "Inference of even linear grammars and its application to picture description languages," *Pattern Recognition*, Vol.21, pp.55-62, 1988.

[9] Y. Sakakibara, "Efficient learning of context-free grammars from positive structural examples," *Information and Computation*, Vol.97, pp.23-60, 1992.

[10] Y. Takada, "A hierarchy of language families learnable by regular language learners," in *Grammatical Inference and Applications*, R.C.Carrasco and J.Oncina (eds.), Springer-Verlag, Lecture Notes in Artificial Intelligence 862, pp.16-24, 1994.

[11] Z. Kohavi, *Switching and Finite Automata Theory*, (2nd edition). Tata McGraw-Hill, New Delhi, 1978.