Pergamon

Pattern Recognition, Vol. 30, No. 1, pp. 163–182, 1997
Copyright © 1996 Pattern Recognition Society. Published by Elsevier Science Ltd
Printed in Great Britain. All rights reserved
0031-3203/97 $17.00+.00

PII:S0031-3203(96)00056-8

RECOGNITION AND LEARNING OF A CLASS OF CONTEXT-SENSITIVE LANGUAGES DESCRIBED BY AUGMENTED REGULAR EXPRESSIONS

RENÉ ALQUÉZAR[a] and ALBERTO SANFELIU[b]

[a]Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya
Diagonal 647 8a, 08028 Barcelona, Spain
[b]Instituto de Robótica e Informática Industrial, Universitat Politècnica de Catalunya-CSIC
Edifici Nexus, Gran Capità 2, 08034 Barcelona, Spain

(*Received* 22 *August* 1995; *in revised form* 15 *March* 1996; *received for publication* 15 *April* 1996)

Abstract—In this paper, a new formalism that permits to represent a non-trivial class of context-sensitive languages, the Augmented Regular Expressions (AREs), is introduced. 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 an RE. An efficient algorithm is given to recognize language strings by AREs. Also a general learning method to infer AREs from examples is presented, that consists of a regular grammatical inference step, a DFA to RE transformation, an RE parsing of the examples, and a constraint induction process. Copyright © 1996 Pattern Recognition Society. Published by Elsevier Science Ltd.

Context-sensitive languages Finite automata Formal languages
Grammatical inference Learning Parsing Regular expressions
Syntactic pattern recognition

1. INTRODUCTION

One of the causes for the limited use of the syntactic approach to pattern recognition[1–3] has been the lack of efficient representations and related methods to deal with the context-sensitive structure of the patterns that appear in most real-world problems, either in vision, speech recognition, or natural language processing. Context-sensitive grammars[4,5] are not a good choice, since their parsing is computationally expensive, there is no available learning algorithm to infer them from examples and/or queries, and (less important) the represented language can hardly be imagined from the observation of the grammar rules. Augmented Transition Networks (ATNs)[6] are powerful models that have been used in natural language applications, but which are very difficult to infer automatically.[7] Pattern languages,[8] though not comparable to Chomsky's hierarchy of languages, provide a limited mechanism to take into account some context influences, namely, the repetition of variable substrings along the strings of a language. The inductive inference of pattern languages has been studied and some learning algorithms proposed.[9] Nevertheless, the expressive power of pattern languages is clearly insufficient to cope with many important context-sensitive structures (e.g. symmetric planar shapes).

In this paper, a new yet simple formalism that permits to describe, recognise and learn a class of non-trivial context-sensitive languages, the Augmented Regular Expression (ARE), is introduced. AREs are neither the *regular-like expressions*,[4] that are known to describe the family of context-free languages, nor a type of regulated rewriting[4] (although there is a certain resemblance between them). Roughly speaking, an ARE is formed by a regular expression in which the stars are replaced by natural-valued variables (called star variables), and these variables are related through a finite number of linear equations.[1] Figure 1 displays some patterns that can be represented by AREs.

After recalling some basic definitions and properties of regular expressions (Section 2), AREs and the components which form them are formally defined in Section 3. Likewise, the relationships among the classes of languages represented by context-sensitive grammars, AREs, and pattern languages are discussed. In Section 4, an efficient method to recognize a string as belonging or not belonging to the language represented by an ARE is presented. The method is split in two stages. In the former, the string is parsed with respect to the underlying RE (optionally with the help of an equivalent DFA) to yield a data structure containing instances of the star variables for that string. In the latter, the satisfaction of the constraints included in the ARE is checked on the star instances resulting from a previous successful parsing. In Section 5, a practical approach to learning AREs from examples is proposed. In this case, four main steps are involved. The first one consists of a regular grammatical inference step, aimed at obtaining a DFA that generates a regular superset of

[1]Note that regular expressions are reduced to AREs with zero equations among the star variables.

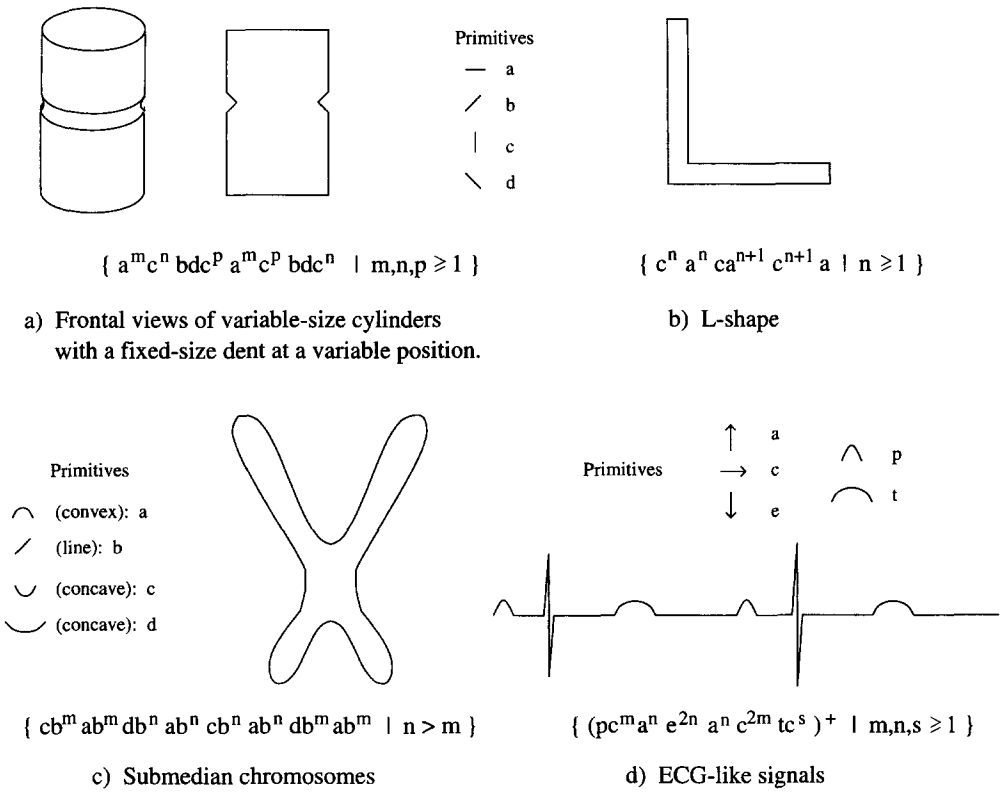


Fig. 1. Some patterns that can be described by AREs.

the target language. Then, an RE equivalent to the inferred DFA is selected as the basic component of the ARE. Afterwards, the star instances corresponding to the example strings are determined by parsing the strings. Finally, the constraints of the ARE are induced by analysing and solving a tree of linear systems formed with the registered instances of the star variables.

2. REGULAR EXPRESSIONS

2.1. Definition and fundamentals

Definition 2.1. Let $\Sigma = \{a_1, \ldots, a_m\}$ be an *alphabet* (a finite set of symbols) and let λ denote the *empty string*. The *regular expressions* over Σ and the languages that they describe are defined recursively as follows.

1. $\emptyset^*$ is a regular expression and describes the empty set.
2. λ is a regular expression and describes the set $\{\lambda\}$.
3. For each $a_i \in \Sigma (1 \leq i \leq m), a_i$ is a regular expression and describes the set $\{a_i\}$.
4. If P and Q are regular expressions describing the languages L_p and L_Q, respectively, then $(P+Q)$, (PQ), and (P^*) are regular expressions that describe the languages $L_P \cup L_Q$ (their *union*), $L_p L_Q$ (their *concatenation*) and L_p^* (the *closure* of L_P), respectively.
5. No other expressions are regular unless they can be generated in a *finite* number of applications of the above rules.

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 omittance of many parentheses in writing a regular expression.

A language is said to be *regular* if and only if it can be described by a regular expression (RE). We write $L(R)$ for the language described by RE R. Two regular expressions P and Q are said to be *equivalent*, denoted by $P = Q$, if they describe the same language. The following are some basic equivalence rules that involve the star operation:

$$\lambda^* = \lambda \tag{1}$$
$$\emptyset^* = \lambda \tag{2}$$
$$R^* R^* = R^* \tag{3}$$
$$RR^* = R^* R \tag{4}$$
$$(R^*)^* = R^* \tag{5}$$
$$\lambda + RR^* = R^* \tag{6}$$
$$(PQ)^* P = P(QP)^* \tag{7}$$
$$(P+Q)^* = (P^* Q^*)^* = (P^* + Q^*)^* \tag{8}$$
$$(P+Q)^* = P^*(QP^*)^* = (P^* Q)^* P^* \tag{9}$$

It is well known (Kleene's theorem[5]) that every language accepted by a finite-state automaton (FSA) can be represented by a regular expression and every language denoted by a regular expression can be recognized by an FSA. Given an FSA A, there can be

many equivalent REs R such that $L(A) = L(R)$. Several algorithms have been proposed to find a regular expression that describes the language accepted by a given FSA.[5,10] By selecting a specific algorithm, a deterministic mapping ψ can be established from FSA to REs, this is, a canonical RE R can be chosen for each FSA A, $R = \psi(A)$.

2.2. *A basic method to find an RE describing the language accepted by a given FSA*

A basic method proposed by Arden[10,11] is recalled here. This method is used with some modifications to derive REs from FSA in learning AREs from examples (see Section 5). The following theorem, the proof of which can be found in Kohavy,[10] is behind Arden's method.

Theorem 2.1. Let Q, P and R be regular expressions over a finite alphabet. Then, if $L(P)$ does not contain λ, the equation $R = Q + RP$ has a unique solution given by $R = QP^*$.

Let $A = (\Sigma, Q, \delta, q_0, F)$ be an FSA, where Σ is a finite set of input symbols (alphabet), Q is a finite set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is a set of final states, and $\delta : (Q \times \Sigma) \to 2^Q$ is a state transition function. Let us assume that Q has n states and a total order ($<$) is established among them, which can be arbitrary except that the first element is the initial state q_0, i.e. $Q = (q_0, \ldots, q_{n-1})$. This order can also be applied to the final states, i.e. $F = (q_{f_1}, \ldots, q_{f_{|F|}})$, where $0 \le f_1 < \cdots < f_{|F|} \le n-1$.

Let α_{ij}^l be the RE that denotes the set of strings from Σ^* that take the automaton from state q_i to state q_j without passing through a state q_k with $k < l$; α_{ij}^n will include only the direct transitions from q_i to q_j, whereas α_{ij}^0 will be the whole set of strings that lead from q_i to q_j. Let R_j be a synonym for α_{0j}^0, the RE that describes the set of strings that take the automaton from the initial state q_0 to state q_j It is clear that a valid RE for $L(A)$ is given by

$$R = R_{f_1} + \cdots + R_{f_{|F|}} \tag{10}$$

Hence, a procedure that determines all the R_j, for $0 \le j < n$, may be used to yield R. Such a procedure is given by solving the following system of symbolic equations:

$$\begin{array}{lllllllllll}
(e_0^n): & R_0 & = & R_0\alpha_{00}^n & + & R_1\alpha_{10}^n & + & \cdots & + & R_{n-1}\alpha_{(n-1)0} & +\lambda \\
(e_1^n): & R_1 & = & R_0\alpha_{01}^n & + & R_1\alpha_{11}^n & + & \cdots & + & R_{n-1}\alpha_{(n-1)1}^n & \\
\cdot\cdot & \cdot\cdot & & \cdot\cdot & & \cdot\cdot & & \cdots & & \cdot\cdot & \\
(e_{n-1}^n): & R_{n-1} & = & R_0\alpha_{0(n-1)}^n & + & R_1\alpha_{1(n-1)}^n & + & \cdots & + & R_{n-1}\alpha_{(n-1)(n-1)}^n &
\end{array}$$

where e_j^n are labels to identify each equation. This system can be solved in n steps. At each step from $j = n-1$ down to $j = 0$, equation (e_j^{j+1}) is processed using the rule of Theorem 2.1, $\lambda \notin L(P) \Rightarrow (R_j = Q + R_jP \Leftrightarrow R_j = QP^*)$, and R_j is substituted into the rest of equations $(e_i^{j+1}), i \ne j$. Whenever Theorem 2.1 is not applicable, the right hand side of the equation can be directly used to replace R_j.

The above procedure yields all the REs $\alpha_{ij}^l (1 \le l \le n, 0 \le j < n, 0 \le i < l)$, although some of them may be empty. A subset of $2n^2$ of these REs can be used to parse a string by an RE efficiently with the help of the source FSA A, as shown in Alquézar and Sanfeliu.[12] However, it must be noted that the time complexity of Arden's algorithm is exponential $O(2^n)$ in the number of states of the given FSA in the worst case, due to the fact that the length of the returned equivalent RE might be exponential in n. This occurs, for example, when the FSA is fully connected (i.e. its state transition diagram is a clique). Nevertheless, in many cases, when the given FSA presents some limitations on the connectivity and degree of circuit embedment in its state transition graph, a run-time polynomial in n can be achieved in practice (e.g. for FSA equivalent to REs of the form $a_1^*a_1 \cdots a_i^*a_i \cdots a_n^*a_n (a_i \in \Sigma)$, a run-time cubic in n is experimentally obtained). Indeed, a best-case complexity of $\Omega(n^3)$ can be shown by realizing that a number cubic in n of REs α_{ij}^l are yielded (see above).

3. AUGMENTED REGULAR EXPRESSIONS

In order to define the Augmented Regular Expressions (AREs) some preliminary concepts are needed, which are introduced in the following subsections.

3.1. *Star variables and star tree of a regular expression*

Definition 3.1. Let R be a given RE and let us say that R includes ns star symbols ($ns \ge 0$). The set V of *star variables* associated with R is an ordered set of natural-valued variables $\{v_1, \ldots, v_{ns}\}$, which are associated one-to-one with the star symbols that appear in R in a left-to-right scan.

Let $pos(V,i)$ be a function[2] that returns the position in R of the star symbol associated with the star variable v_i; moreover, if p is the position in R of a star symbol, then $i = pos^{-1}(V,p)$ gives the index of the corresponding star variable v_i. The function pos can be used to order the set $V : v_iv_j \Leftrightarrow pos(V,i) < pos(V,j)$.

Definition 3.2. For $v_i, v_j \in V$, we say that v_i *contains* v_j if and only if the operand of the star associated with v_i in R

[2]Actually, it can be represented as an attribute of each v_i.

includes the star corresponding to v_j; and we say v_i *directly contains* v_j if and only if 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.

Definition 3.3. Given an RE R, its associated *star tree* $\mathcal{T} = (N, E, r)$ 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 in the following manner:

i) for all the star variables $v_i \in V$ that are not *directly contained* by other star variables, an edge (r, v_i) is created (v_i is said to be a *son* of r), and therefore v_i is located in the first level of the tree;[3]

ii) for all $v_i, v_j \in V$, if v_i *directly contains* v_j, then an edge (v_i, v_j) is created (and v_j is said to be a *son* of v_i).

Furthermore, let us assign an integer identifier to each node of the star tree $\mathcal{T}$: let the identifier of the root r be 0, and let the identifier of any other node be the index i of the star variable v_i corresponding to the node ($1 \leq i \leq ns$). A simple algorithm to build the *star tree* $\mathcal{T}$ associated with a given RE R has been reported,[12] 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 a 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 substring. In such a case, we say that the star variable is *instantiated*, and sometimes we refer to its value as an *actual instance*. For computational purposes, we will see that it is useful to assign a special value, say -1, to a star variable v when its father in the star tree $\mathcal{T}$ is instantiated but v is not during the matching process.

3.2. Star instances data structure

Since an RE R describes a language $L(R)$ of strings (over Σ), it is convenient to parse a given string $s \in \Sigma^*$ with respect to R. A parsing algorithm must return *yes* or *not* depending on whether $s \in L(R)$ or not, and in the first case, it must also return a kind of "instance" of R that just describes s (something similar to a derivation tree in parsing a string by a grammar). An RE R is *ambiguous* if there exists a string $s \in L(R)$ for which more than one "instance" of R can be built. Next, a data structure is presented which is designed to store the information of the instances of the star variables that occur in parsing a string s by an *unambiguous* RE R. This structure can be regarded as a partial representation of the "instance" of R for s, since the matched subexpressions themselves are not recorded.

When a string s is parsed by an *unambiguous* RE R, the associated star variables $v_i \in V (1 \leq i \leq ns)$ will be instantiated zero, one, or more times, depending both on the instances of the star variables that *directly contain* them and on which terms of the union-type REs in R are selected to parse a substring of s. If the operand of a star in R consists of a union of two or more REs (called terms), which term is used for each match of the operand can be traced. Consequently, for each cycle, only the star variables that are located in the matched term can be instantiated, while the special value -1 can be given to the rest of star variables that are *directly contained* by the same father. In this way, all the star variables that are brothers in the star tree $\mathcal{T}$ will have the same structure of potential instances for a given string, whether they are actually instantiated or not. Let us put it more formally.

Let $SI_s(V) = \{SI_s(v_1), \ldots, SI_s(v_{ns})\}$ be the set of instances of the star variables in V resulting from the parsing of a string s by the RE R (from which V has been defined). Each member of the set is a list of lists containing instances of a particular star variable:

$$\forall i \in [1..ns]: \quad SI_s(v_i) = (l^i_1, \ldots, l^i_{nlists(i)}) \quad \text{where } nlists(i) \geq 0$$

$$\forall i \in [1..ns] \forall j \in [1..nlists(i)]: \quad l^i_j = (e_{j1}, \ldots, e_{j(nelems(i,j))}) \quad \text{where } nelems(i,j) \geq 1$$

The instances stored in the lists are organized according to the containment relationships of the star variables described by the star tree $\mathcal{T}$. This is carried out by defining for each list l^i_j two pointers $father_list(l^i_j)$ and $father_elem(l^i_j)$ that identify the instance of the father star variable from which the instances of v_i in l^i_j are derived.

For all the star variables that are in the first level of $\mathcal{T}$, the following structure arises:

$$\forall v_i, (r, v_i) \in \mathcal{T} \Rightarrow SI_s(v_i) = (l^i_1) \wedge l^i_1 = (e^i_{11}) \wedge father_list(l^i_1) = -1 \wedge father_elem(l^i_1) = -1$$

i.e. $nlists(i) = 1$ and $nelems(i, 1) = 1$; furthermore, if v_i is not instantiated in parsing s then $e^i_{11} = -1$ else $e^i_{11} \geq 0$ is the number of matches of the star operand in the only instance of v_i. Otherwise, let v_f be the father of v_i in $\mathcal{T}$. For all the star variables that are in the second level of $\mathcal{T}$, the list of instance lists is either empty (when $e^f_{11} \leq 0$) or its structure is:

$$\forall v_i, (r, v_f), (v_f, v_i) \in \mathcal{T} \wedge e^f_{11} > 0 \Rightarrow SI_s(v_i) = (l^i_1) \wedge l^i_1 = (e^i_{11}, \ldots, e^i_{1(e^f_{11})}) \wedge father_list(l^i_1) = 1 \wedge father_elem(l^i_1) = 1$$

i.e. $nlists(i) = 1$ and $nelems(i, 1) = e^f_{11}$; and, if v_i is not instantiated in the k-th match of the star operand of v_f then $e^i_{1k} = -1$, else $e^i_{1k} \geq 0$ is the number of matches of the star operand of v_i in the k-th cycle. Finally, for all the star variables v_i that are in the higher levels of $\mathcal{T}$ (with father v_f), we have the following general rule:[4]

[3] The root r is at level 0.

[4] This rule is also met, in fact, by the star variables in the second level of $\mathcal{T}$.

$$nlists(i) = \#\{e^f_{jk} \mid e^f_{jk} > 0\} \wedge \forall j \in [1..nlists(i)] : nelems(i,j) = e^f_{j'k'} \wedge father_list(l^i_j) = j' \wedge father_elem(l^i_j) = k'$$

and e^i_{jk} is either a natural (the instance of v_i in the k-th cycle of the instance of v_f identified by the pointers $\{j', k'\}$) or -1 (if v_i is not instantiated in such a cycle). Fig. 2 shows an example of the star instances for a given string and RE, in which the star tree $\mathscr{T}$ has four levels. Two algorithms for *unambiguous* RE parsing that build the star instances structure have been reported.[12]

3.3. Definition and expressive power of AREs

Definition 3.4. An *Augmented Regular Expression* (or ARE) is a four-tupla $(R, V, \mathscr{T}, L)$, where R is a regular expression over an alphabet Σ, V is its associated set of *star variables*, $\mathscr{T}$ is its associated *star tree*, and L is a set of independent linear relations $\{l_1, \ldots, l_{nc}\}$ each one involving the variables in V, that is

$$l_i \equiv a_{i1}v_1 + \cdots + a_{ij}v_j + \cdots + a_{i(ns)}v_{ns} + a_{i0} = 0 \quad \text{for } 1 \le i \le nc$$

where ns is the number of star variables and nc $(0 \le nc \le ns)$ is the number of relations (or constraints). A preferred equivalent formulation of the set L is given by partitioning the set of star variables V into two subsets V^{ind}, V^{dep} of independent and dependent star variables, respectively, and expressing the latter as linear combinations of the former:

$$l_i \equiv v_i^{dep} = a'_{i1}v_1^{ind} + \cdots + a'_{ij}v_j^{ind} + \cdots + a'_{i(ni}v_{ni}^{ind} + a'_{i0}, \text{ for } 1 \le i \le nc$$

where ni and nc are the number of independent and dependent star variables, respectively.

The definition of V restricts the allowed values for the star variables to natural numbers, $\forall k \in [1, ns] : v_k \in \mathscr{N}$. Consequently, the set of linear relations L is only well-defined when the involved variables take natural numbers as values. This also implies that some of the star variables may be implicitly constrained to a smaller range inside the natural numbers (e.g. $v_k \ge z, z \in \mathscr{N}; v_k$ always odd; v_k always even; etc.). Moreover, the coefficients a_{ij}(or a'_{ij}) of the linear relations will always be rational numbers.

Definition 3.5. Let $\tilde{R} = (R, V, \mathscr{T}, L)$ be an ARE, the language $L(\tilde{R})$ represented by $\tilde{R}$ is the set of strings α from Σ^* such that $\alpha \in L(R)$ and there exists a parsing of α by R in which the *star instances* $SI_\alpha(V)$ *satisfy all the linear constraints in* L (let the predicate *satisfy* $(SI_\alpha(V), L)$ denote this condition).

$$R = (a(b(ce^*c + df^*d)^*)^*)^* \qquad R(V/*) = (a(b(ce^{v_1}c + df^{v_2}d)^{v_3})^{v_4})^{v_5}$$
$$V = \{v_1, v_2, v_3, v_4, v_5\} \qquad \mathscr{T} = (V \cup r, \{(r, v_5), (v_5, v_4), (v_4, v_3), (v_3, v_1), (v_3, v_2)\}, r)$$

$$s = abccdf\,fdcecbddbdf\,dceecabceeec$$

$$SI_s(v_5) = ((2)^{\{-1,-1\}}) \qquad SI_s(v_1) = ((0\ \ -1\ \ 1)^{\{1,1\}}(-1)^{\{1,2\}}(-1\ \ 2)^{\{1,3\}}(3)^{\{2,1\}})$$
$$SI_s(v_4) = ((3\ \ 1)^{\{1,1\}}) \qquad SI_s(v_2) = ((-1\ \ 2\ \ -1)^{\{1,1\}}(0)^{\{1,2\}}(1\ \ -1)^{\{1,3\}}(-1)^{\{2,1\}})$$
$$SI_s(v_3) = ((3\ \ 1\ \ 2)^{\{1,1\}}(1)^{\{1,2\}})$$

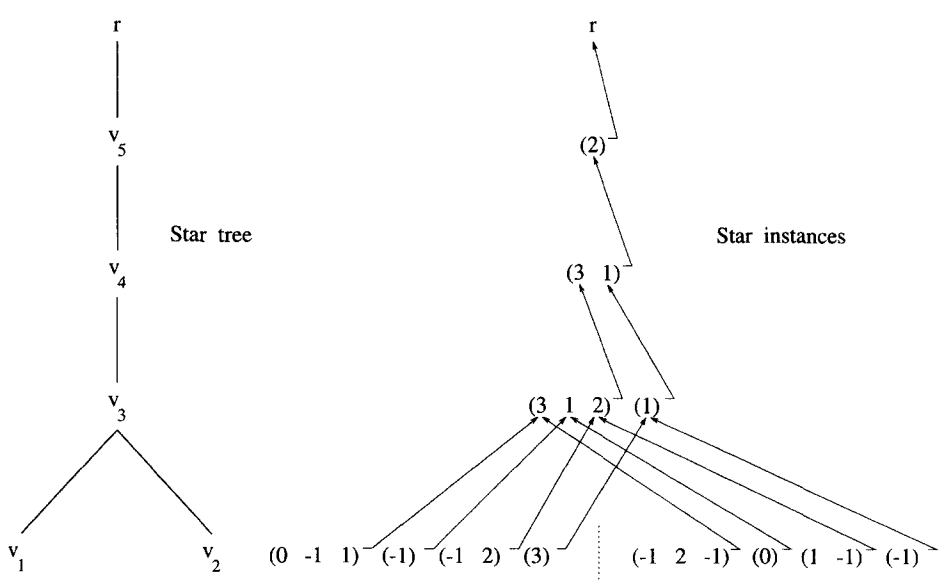


Fig. 2. An example of star instances data structure.

The AREs permit description of a class of context-sensitive languages by imposing a set of rules that constrain the language of a regular super-set. A very simple example is the language of rectangles $\{a^m b^n a^m b^n \mid m, n \geq 1\}$, which is well known to be context-sensitive (see grammar CSG_1 in Appendix A), and which is described by the ARE $\tilde{R}_1 = (R_1, V_1, \mathscr{T}_1, L_1)$, where

$$\begin{aligned} R_1 &= a^* ab^* ba^* ab^* b \\ V_1 &= \{v_1, v_2, v_3, v_4\} \\ R_1(V_1/*) &= a^{v_1} ab^{v_2} ba^{v_3} ab^{v_4} b \\ \mathscr{T}_1 &= (V_1 \cup r, \{(r, v_1), (r, v_2), (r, v_3), (r, v_4)\}, r) \\ L_1 &= \{v_3 = v_1, v_4 = v_2\} \\ &\quad (\text{i.e. } V^{ind} = \{v_1, v_2\} \;\; V^{dep} = \{v_3, v_4\} \\ &\quad \text{and } nc = 2). \end{aligned}$$

However, more complex languages with an arbitrary level of star embedment and multiple linear constraints (even among stars at different levels of embedment) can be described as well by the ARE formalism. Consider, for instance, the ARE $\tilde{R}_2 = (R_2, V_2, \mathscr{T}_2, L_2)$ with

$$R_2(V_2/*) = (c^{v_1}(d^{v_2} b^{v_3})^{v_4} c^{v_5} a^{v_6} c^{v_7} (b^{v_8} d^{v_9})^{v_{10}} c^{v_{11}} e^{v_{12}})^{v_{13}}$$

and

$$\begin{aligned} L_2 = \{ & v_{11} = v_1 + v_5 - v_7, \\ & v_{12} = v_6, \\ & v_2 = v_4 - 1, \\ & v_3 = v_4 - 1, \\ & v_8 = 0.5 v_{10} + 0.5, \\ & v_9 = 0.5 v_{10} + 0.5\} \end{aligned}$$

The set of constraints L_2 (besides an unambiguity requirement for parsing R_2) implies that $v_2, v_3, v_8, v_9 \geq 1; v_4 \geq 2$; and v_{10} will always be odd. Fig. 3 and Fig. 8 show some examples that belong to the language represented by $\tilde{R}_2$, given an alphabet of graphical primitives $\{\uparrow a, \nearrow b, \rightarrow c, \searrow d, \downarrow e\}$. The context-sensitive grammar CSG_2 in Appendix A generates $L(\tilde{R}_2)$. The reader is encouraged to compare the compact and descriptive representation provided by the ARE $\tilde{R}_2$ with the obscure grammar CSG_2, that comprises 79 rules.

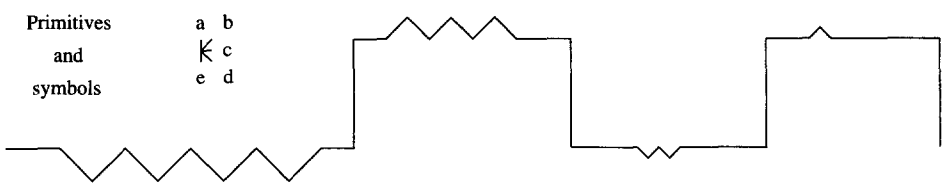
The following question naturally arises: *Can all the context-sensitive languages be represented by AREs?* The answer is in the following theorem.

Theorem 3.1. The Augmented Regular Expressions do not describe all the context-sensitive languages.

Proof. The context-sensitive grammar CSG_3 (see Appendix A) that generates the language $\{a^k \mid k = 2^i \wedge i \geq 1\}$ is a counter example. This language is not describable because AREs can only filter the range of values of the star variables through linear relations, and these relations only involve the star variables but not any external variable (such as i in $L(CSG_3)$). Therefore, there is no ARE $\tilde{R} = (R, V, \mathscr{T}, L)$ such that L can represent the constraint $v_1 = 2^i \wedge i \geq 1$ for $R(V/*) = a^{v_1}$. □

The context-sensitive language $\{a^k \mid k \text{ is a prime}\}$ is another counter example. Indeed, it seems reasonable to expect that a large class of CSLs will not be described by AREs either, due to the limited type of context constraints that can be represented.

Consider now the language $\{xx \mid x \in (0+1)^+\}$ generated by the context-sensitive grammar CSG_4. $L(CSG_4)$ corresponds to the pattern language[8] xx over the binary alphabet $\Sigma = \{0, 1\}$, where the variable x stands for any string in Σ^+. The ARE $(0+1)^{v_1}(0+1)^{v_2}$ with $\{v_2 = v_1\}$ cannot express that the substrings associated with the instances of the operands of the stars denoted by v_1 and v_2 are identical. However, if the equivalence rule $(0+1)^* = (0^*1)^*0^*$ is applied before,



$$\begin{aligned} s_1 &= c^5 d^3 b^3 d^3 b^3 d^3 b^3 d^3 b^3 c^3 a^{10} c^3 b^2 d^2 b^2 d^2 b^2 d^2 c^5 e^{10} c^6 dbdbc^8 a^{10} c^4 bdc^{10} e^{10} \\ [R_2(V_2/*) &= (c^{v_1}(d^{v_2} b^{v_3})^{v_4} c^{v_5} a^{v_6} c^{v_7} (b^{v_8} d^{v_9})^{v_{10}} c^{v_{11}} e^{v_{12}})^{v_{13}}] \\ [SI_{s_1}(v_{13}) &= ((2)^{\{-1,-1\}})] \end{aligned}$$

$$\begin{aligned} SI_{s_1}(v_1) &= ((5\ \ 6)^{\{1,1\}}) & SI_{s_1}(v_7) &= ((3\ \ 4)^{\{1,1\}}) \\ SI_{s_1}(v_5) &= ((3\ \ 8)^{\{1,1\}}) & SI_{s_1}(v_{11}) &= ((5\ \ 10)^{\{1,1\}}) \\ SI_{s_1}(v_6) &= ((10\ \ 10)^{\{1,1\}}) & SI_{s_1}(v_{12}) &= ((10\ \ 10)^{\{1,1\}}) \\ SI_{s_1}(v_4) &= ((4\ \ 2)^{\{1,1\}}) & SI_{s_1}(v_{10}) &= ((3\ \ 1)^{\{1,1\}}) \\ SI_{s_1}(v_2) &= ((3\ 3\ 3\ 3)^{\{1,1\}}(11)^{\{1,2\}}) & SI_{s_1}(v_8) &= ((2\ 2\ 2)^{\{1,1\}}(1)^{\{1,2\}}) \\ SI_{s_1}(v_3) &= ((3\ 3\ 3\ 3)^{\{1,1\}}(11)^{\{1,2\}}) & SI_{s_1}(v_9) &= ((2\ 2\ 2)^{\{1,1\}}(1)^{\{1,2\}}) \end{aligned}$$

Fig. 3. An example of pattern recognized by the ARE $\tilde{R}_2$ with its corresponding star instances.

the ARE $(0^{v_1}1)0^{v_2}0^{v_3}(0^{v_4}1)^{v_5}0^{v_6}$ with $\{v_5 = v_2;\ v_6 = v_3;\ v_4 = v_1\}$ is able to describe the above pattern language.

Theorem 3.2. The Augmented Regular Expressions do cover all the pattern languages, but the size of an ARE describing a pattern language over Σ is exponential with respect to the number of alphabet symbols $|\Sigma|$.

Proof. Let p be a pattern language over $\Sigma = \{a_1, \ldots, a_m\}$ $(m \geq 2)$ including some finite number of variables $\{x_1, \ldots, x_l\}$ $(l \geq 0)$. Each variable $x_i (1 \leq i \leq l)$ can be represented by an RE $R_\Sigma = (a_1 + \cdots + a_m)^*$. By applying repeatedly equation (9) an equivalent RE R'_Σ, without union operators is obtained that contains $2^m - 1$ stars (this is easily shown by induction). Let $\tilde{R}'_\Sigma$, be an ARE with no constraint such that the stars of R'_Σ are replaced by independent star variables. Let $t(i)$ be the number of occurrences of x_i in p. Each occurrence x_{ij} of x_i in p gives rise to a duplicate of $\tilde{R}'_\Sigma$ with new star variables: $\tilde{R}'_{ij}$. An ARE $\tilde{R}'_p$ describing the pattern language p can be stated by letting the star instances of the AREs $\tilde{R}'_{i1}$ be independent and establishing a set L of $(2^m - 1) \cdot \sum_{i=1}^{l}(t(i) - 1)$ equations of the form $v_{ijk} = v_{i1k}$ $(1 \leq i \leq l;\ 2 \leq j \leq t(i);\ 1 \leq k \leq 2^m - 1)$. □

On the other hand, it is obvious that the class of pattern languages[8] does not cover the languages represented by AREs. For example, the language of rectangles $L(\tilde{R}_1)$ and the context-free language $\{0^{v_1}1^{v_2}0^{v_3} \mid v_2 = v_1 + v_3\}$ cannot be described by any pattern language.

4. STRING RECOGNITION THROUGH AUGMENTED REGULAR EXPRESSIONS

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 successful, checking the satisfaction of constraints L by the star instances $SI_s(V)$ that result from the parsing. If R *is unambiguous*, a unique parsing 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 $s \in L(\tilde{R})$.

4.1. Parsing strings by REs to build the star instances

Two algorithms for *unambiguous* RE parsing have been reported[12] which, given a string s and an RE R, respond whether $s \in L(R)$ or not, and in the first case, build the corresponding star instances $SI_s(V)$. The processing of the input string is clearly divided in two phases: the *recognition* and *construction* phases. The first algorithm uses the RE R (alone) for *recognition*. The *construction* phase is a kind of re-run of the *recognition* phase in which it is known in advance that the string will be successfully parsed by the RE, and therefore, the true instances of the star variables can be recorded. To this end, the current star variable that is involved in parsing is tracked, and the value of each new instance is computed by counting the number of consecutive matches of the operand of the related star. The time complexity of the first algorithm is $O(|s| \cdot |R|)$ globally and for both phases.

A more efficient parsing method is attainable if the unambiguous RE R has been obtained from an equivalent DFA A which is also given. This is the case if R has been inferred from examples by applying the DFA-to-RE mapping, described in Section 5, to the result of a DFA learning method. The second parsing algorithm[12] is such an efficient method which uses the DFA A, some of the REs α^l_{ij} yielded by Arden's algorithm, and the *skeleton*[5] of R. The key point is that A (instead of R) is used for *recognition* $O(|s|)$, and that the path of visited states obtained, guides the *construction* of the star instances structure for the input string s. There are two achievements that permit reduction of the time complexity of the *construction* phase also. The former is to locate the substrings of s that are associated with the cycles of the involved star-type REs by finding subpaths of visited states that start and end with the same state without passing through it. The latter is to select directly the term of the involved union-type REs that actually matches the corresponding substring without the need of attempting to parse the non-matched terms.[12] Hence, the second algorithm has a time complexity of $O(\max\{|skel(R)|, n \cdot |s|\})$, due to the *construction* phase, where n is the number of states of A, $|s|$ and $|skel(R)|$ denote the lengths of the input string and the *skeleton* of R, respectively, and $|skel(R)| \leq |R|$.

4.2. Constraint satisfaction

Algorithm 1 (in Appendix B) is proposed to evaluate the predicate *satisfy*$(SI_s(V), L)$ given a set of star instances, previously recorded in a successful parsing, and a set of linear constraints among the star variables. It provides the second step in recognizing a string through an ARE, and its theoretic time complexity is $O(|L| \cdot h(\mathscr{T}) \cdot |V| \cdot I(SI_s(V)))$ where $|L|$ and $|V|$ are the number of constraints and star variables, respectively, $h(\mathscr{T})$ is the height of $\mathscr{T}$, and $I(SI_s(V)) = \max_{i=1,|V|} \sum_{j=1}^{nlists(i)} nelems(i,j)$ is the maximal number of *potential* instances of a star variable (i.e. including those assigned to -1) yielded by parsing s.

It must be noted that every valid constraint can only involve a set of star variables which share a common structure of instances, i.e. the number of instances of each of them is the same, and these can be grouped, one instance of each variable, in rows, one row for each cycle of the instances of a common *selected* ancestor. In principle, this means that the set of related star variables should be brothers in $\mathscr{T}$ (and their father being the common ancestor).

However, this is not exactly the case. It could happen that the values of the assigned instances of a certain son were constant for each instance of its father. In such a

[5]The *skeleton* of an RE describes R in terms of the languages corresponding to a determined subset of the paths of A and it is formed in a simplifying step after running Arden's algorithm (see Section 5).

case, a unique value might be associated with the father's instance and, furthermore, both values (of father and son instances) might be related by a constraint. In other words, regarding the star instances, the son would be promoted to a lower level in the tree and share with its father the instance structure determined by the instances of its grandfather; we could say that, with respect to the promoted son, the father is a *degenerated* ancestor and the grandfather is the *housing* ancestor. This promotion process could continue until a *non-degenerated* ancestor were found, for which the instances of the promoted variable would not have a common value for each instance of the ancestor, or until a default node were reached as a *housing* ancestor. This default node can be the root node r of the star tree or a selected ancestor (e.g. a *housing* ancestor that is shared with other star variables, as is explained next). Each time a star variable is promoted to a lower level, all of its redundant instances must be collapsed into a single one in order to keep the common structure of instances. The procedure *determine_ancestor_and_instances*,[12] whose cost is $O(h(\mathscr{T}) \cdot I(SI_s(V)))$, implements the process described.

Moreover, even if a common *housing* ancestor is not found, a set of star variables can be related by a constraint whenever all of their *housing* ancestors are related by a strict equality. This fact ensures that a common instance structure is available, even though the star instances are not constant, as it occurs in the AREs describing pattern languages. For example, in the ARE $(0^{v_1}1)^{v_2}0^{v_3}(0^{v_4}1)^{v_5}0^{v_6}$ with $\{v_5 = v_2;\ v_6 = v_3;\ v_4 = v_1\}$, the constraint $v_4 = v_1$ is valid because v_5 and v_2 are the *housing* ancestors of v_4 and v_1, respectively, and $v_5 = v_2$.

In summary, each constraint in L can only be satisfied by the star instances $SI_s(V)$, gathered during the parsing of a string s, if the star variables involved either share a common *housing* ancestor or their *housing* ancestors satisfy a strict instance equality. Let the constraint l_i in $L(1 \leq i \leq nc)$ be expressed as

$$v_i^{dep} = a_{i1}v_1^{ind} + \cdots + a_{il}v_l^{ind} + a_{i0},$$

where the set of independent variables fulfils the restriction that $\forall j \in [1..l] : a_{ij} \neq 0$.

In order to test l_i the instances of the dependent star variable v_i^{dep}, obtained in the parsing of s, are analysed. If there is no actual instance of v_i^{dep}, the constraint is considered to have been met. Otherwise, the deepest common ancestor of the star variables $\{v_i^{dep}, v_1^{ind}, \ldots, v_l^{ind}\}$, i.e. the first common ancestor going from each of these nodes to the root of $\mathscr{T}$, is selected as a candidate to common *housing* ancestor. To check the linear constraint l_i it is mandatory that the instances of all the star variables involved in the relationship can be arranged in the structure of instances caused by the *housing* ancestor of v_i^{dep} (call it v_{hi}). Consequently, if any of them (say v_k^{ind}) has a housing (non-degenerated) ancestor (say v_{hk}) that is deeper than the common *housing* ancestor candidate and the equation $v_{hk} = v_{hi}$ is not met by the instances, then it means that a shared structure of instances is not available for the string s, and therefore, the constraint l_i is considered to be violated. In the particular case of a star variable of an ARE always having a constant value ($v_i^{dep} = a_{i0}$) regardless of its level in $\mathscr{T}$, its housing ancestor will be the root node, and obviously, all actual instances must be collapsed to the value a_{i0} to verify the constraint.

Finally, when the housing ancestor of all the star variables in l_i coincides with their deepest common ancestor or all the housing ancestors are related by strict equality, the constraint is tested on all the *actual instances* of v_i^{dep}. To this end, these instances are arranged in a column vector B, whereas the corresponding instances of the involved independent variables are orderly put as columns in a matrix A, together with an all-1's column associated with the constant term of the constraint. Then, it suffices to test $A \cdot X = B$, where X is the vector of coefficients in the right hand side of the constraint.

Consider the example of Fig. 3. Given the constraints L_2 and the star instances displayed (for the string $s1$), Algorithm 1 would set v_{13} as housed descendant of the root node r, and the rest of star variables of v_2 as housed descendants of v_{13}. In the main loop, the six constraints of L_2 would be checked. The first one, $V_{11} = v_1 + v_5 - v_7$, would lead to the successful test of the equality $A \cdot X = B$ shown in Fig. 4. The rest of constraints would be verified in a similar manner. Therefore, the string s_1 of Fig. 3 would be accepted as belonging to $L(\tilde{R}_2)$.

5. INFERRING AREs FROM STRING EXAMPLES.

Now, let us consider the problem of learning AREs. A possible approach is to split the process in two main stages: inferring the underlying RE, and afterwards, inducing the constraints that bear the context sensitivity of the language. For the first stage, some regular grammatical inference (RGI) method is required. Almost all of the known RGI methods return an FSA, and consequently the RGI step will have to be followed usually by an FSA to RE mapping. For the second stage, the tasks of building the star tree, parsing a set of example strings, and inferring the constraints from the collected star instances, are needed.

$$\begin{array}{r} \text{1st cycle of instance } v_{13} = 2 \text{ in } s_1 \\ \text{2nd cycle of instance } v_{13} = 2 \text{ in } s_1 \end{array} \begin{bmatrix} & v_1 & v_5 & v_7 \\ 1 & 5 & 3 & 3 \\ 1 & 6 & 8 & 4 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} v_{11} \\ 5 \\ 10 \end{bmatrix}$$

Fig. 3. Verification of the constraint $v_{11} = v_1 + v_5 - v_7$ through the matrix product $A \cdot X = B$ (the top row of the displayed A and B is just for labeling purposes).

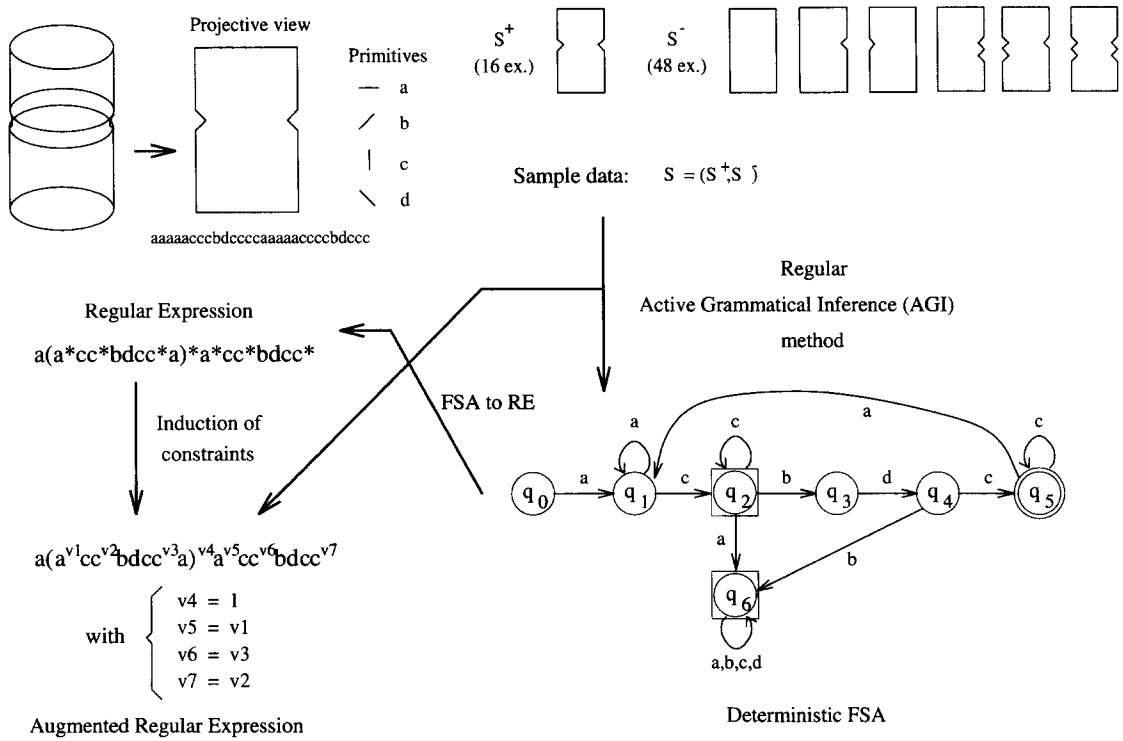


Fig. 5. An example of application of the proposed method for inferring AREs.

5.1. *Description of the overall procedure*

Before proceeding to describe the details of the proposed method for learning AREs, it is worthwhile to give, first, a global picture of it in terms of an actual example. Fig. 5 displays a simple but illustrative case. The problem at hand is 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 language associated with such an object is context-sensitive,[6] and, consequently we cannot expect that a regular or even a context-free language learning algorithm returns a suitable recognizer for this class of objects. 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 fact, our ARE learning method is a rather straightforward (but, as far as we know, unexplored) approach for inferring syntactic descriptions of this kind.

In the case of Fig. 5, a sample $S = (S^+, S^-)$ of 16 positive and 48 negative examples was provided, corresponding to some variable-size instances of the contours shown at the top of the figure. Even though the target language is context-sensitive, one may try to enter this sample into an FSA learning algorithm and analyse the usefulness of the result.[7] The application of the (regular) active grammatical inference method recently reported,[23] to the given S yielded the deterministic FSA displayed in Fig. 5, which accounts for the basic repetitive structure of the model but which over-generalizes a lot, accepting rather arbitrary contours without any length restriction. However, this result is a good starting point to search for a more accurate description that includes the context constraints (i.e. an ARE). The next step is to obtain an equivalent compact representation that facilitates the induction of the constraints, and this turns out to be an RE. In our example, the method explained in next subsection was used yielding the RE shown in Fig. 5. Finally, from the automatic analysis of the star instances of the RE that were produced by *parsing* the positive examples, a set of constraints could be derived that, in conjunction with the regular expression, perfectly described the target language.

The data flow of the process followed to infer an ARE from examples is depicted in Fig. 6. Once the regular expression R is determined from the inferred FSA A, the associated *star variables* V and the *star tree* $\mathcal{T}$ must be obtained. These are used both to build an "array of star instances" *ASI*, containing the information recorded from parsing the examples, and to analyse it in order to induce the set of constraints L. The result of the process can be expressed as the tupla $\tilde{R} = (R, V, \mathcal{T}, L)$. Algorithm 2 (in Appendix B) is a more detailed description of the overall learning procedure. Note that a wide range of algorithms is available to perform the regular grammatical inference step.[17–23] This must not be interpreted, however, as if the choice of the regular GI method were irrelevant. On the contrary, the implicit or explicit biases of the selected method may or may not

[6]The length of the two horizontal segments is the diameter of the cylinder, and the lengths of the vertical segments separated by the dent are obviously the same at each side of the axis.

[7]A strategy that resembles "the drunk searching the keys under the lamp."

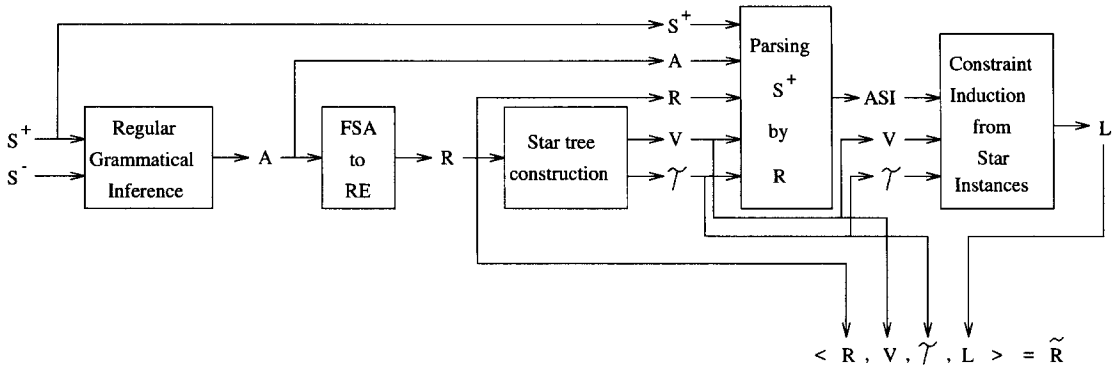


Fig. 6. Flow diagram of main data in the process of learning Augmented Regular Expressions from string examples.

help to reach a "natural" (the simplest in some sense) regular expression for the data that supports the discovery of the underlying context constraints.

A drawback of the preceding approach must be noted: all negative examples (if any) will be rejected by the inferred RE, whereas, in fact, some of them could belong to the language accepted by the RE in the target ARE (provided that they did not satisfy the constraints). Therefore, an alternative learning scheme may be applied such that, initially, the negative examples are not supplied to the RGI step, but if any of them is accepted by the inferred ARE, then the process is restarted incorporating the conflictive negative examples to the RGI step. In this way, several runs could be necessary to reach a consistent ARE; in the worst case, after a finite number of runs (bounded by the number of negative examples), all the negative sample would be given to the RGI step, as in Fig. 6, thus guaranteeing the consistency of the inferred ARE. In any case, it would be helpful to have available an informant who partitioned the negative sample in the two subsets of strings to be accepted and rejected respectively by the RE in the target ARE.

5.2. FSA to RE mapping

The method that is suggested to be used for the FSA-to-RE transformation is based on Arden's algorithm described in Section 2, but a final simplifying step and an inner modification of the algorithm are proposed to "improve" the resulting regular expression.

The RE given by equation (10) can be simplified a lot by determining the common factors (prefixes) in the sum terms and applying repeatedly the equivalence rule $PQ + PS = P(Q + S)$. First, let us find a simplified RE for the union of the regular languages R_j for all the states $q_j \in Q$. It is easy to show, by replacing recursively the R_j variables in equations (e_j^j) by the expressions given in equations (e_i^i), for $0 < j < n$ and $0 < i < j$, that the following equivalence holds:

$$\sum_{q_j \in Q} R_j = R_0 + \cdots + R_{n-1} = R_0(\lambda + R_{01}P^Q_{1(n-1)} + \cdots + R_{0(n-1)}P^Q_{(n-1)(n-1)}) = R_0 P^Q_{0(n-1)} \tag{11}$$

where $R_{ij} = \alpha_{ij}^j$, and P^Q_{ik} denotes the set of strings that lead from state q_i to any state $q_j \in Q$ with $i \le j \le k$ without passing through a state q_l with $l \le i$, and it can be defined recursively as

$$P^Q_{ik} = \lambda + \sum_{j=i+1,k}^{R_{ij} \neq \emptyset} R_{ij} P^Q_{jk} \tag{12}$$

Next, we define a relation $q_i \sim q_j$ in the following way:

$$q_i \sim q_j \Rightarrow (j < i) \wedge R_{ij} \neq \emptyset$$

Now, let $Q' \subseteq Q$ be any subset of states, and let $C(Q')$ be the transitive closure of Q' with respect to the relation $\sim$. It can be proved that

$$P^{Q'}_{ik} = \emptyset \Leftrightarrow q_i \notin C(Q'), \forall k \ge i \tag{13}$$

Hence, $P^{Q'}_{ik}$ can be computed recursively using $C(Q')$ for filtering null terms as follows:

$$P^{Q'}_{ik} = \begin{cases} \lambda + \sum_{j=i+1,k}^{R_{ij} \neq \emptyset \wedge q_j \in C(Q')} R_{ij} P^{Q'}_{jk} & \text{if } q_i \in Q' \\ \sum_{j=i+1,k}^{R_{ij} \neq \emptyset \wedge q_j \in C(Q')} R_{ij} P^{Q'}_{jk} & \text{otherwise} \end{cases} \tag{14}$$

Then the union of the regular languages associated with the states of Q' is equivalent to a simplified RE with the extracted common factors:

$$\sum_{q_j \in Q'} R_j = R_0 P^{Q'}_{0(n-1)} \tag{15}$$

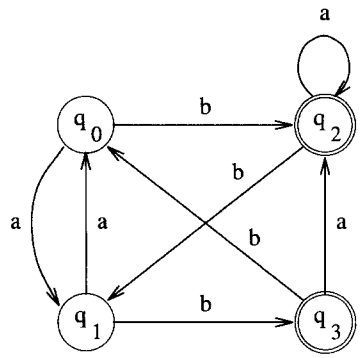
Furthermore, if $Q' \neq \emptyset$ then $P^{Q'}_{0(n-1)} = P^{Q'}_{0z}$, where q_z is the state of Q' with the largest index.

Finally, for the RE equivalent to the given automaton A, we obtain

$$R = \psi(A) = R_0 P^F_{0f_{|F|}} \left(= \sum_{q_f \in F} R_f \right) \tag{16}$$

We refer to the string that is obtained by substituting in R the REs corresponding to R_0 and R_{ij} by the symbols "R0" and "Ri_j" as the *skeleton* of R.

Consider, for example, the 4-state FSA that is displayed in Fig. 7(a). The two equivalent REs, given by equations (10) and (16), after running Arden's algorithm, are shown in Fig. 7 (c) and (d), respectively. The skeleton of the simplified one is displayed in Fig. 7 (e).



(a)

$$R_0 = ((a + ba^*b)(baa^*b)^*(a + bb))^*$$
$$R_1 = ((a + ba^*b)(baa^*b)^*(a + bb))^*(a + ba^*b)(baa^*b)^*$$
$$R_2 = ((a + ba^*b)(baa^*b)^*(a + bb))^*(ba^* + (a + ba^*b)(baa^*b)^*baa^*)$$
$$R_3 = ((a + ba^*b)(baa^*b)^*(a + bb))^*(a + ba^*b)(baa^*b)^*b$$

(b)

$$R_2 + R_3 = ((a + ba^*b)(baa^*b)^*(a + bb))^*(ba^* + (a + ba^*b)(baa^*b)^*baa^*) + ((a + ba^*b)(baa^*b)^*(a + bb))^*(a + ba^*b)(baa^*b)^*b$$

(c)

$$R = R_0 P_{03}^{\{q_2 q_3\}} = ((a + ba^*b)(baa^*b)^*(a + bb))^*((a + ba^*b)(baa^*b)^*(baa^* + b) + ba^*)$$

(d)

$$skel(R) = R0(R0_1(R1_2 + R1_3) + R0_2)$$

(e)

Fig. 7. (a) An example FSA A with 4 states; (b) regular languages associated with its states given by Arden's algorithm; (c) straightforward RE for $A : \sum_{q_f \in F} R_f$; (d) simplified RE for $A : R = \psi(A) = R_0 P_{0f_{|F|}}^{F}$; (e) skelelton of R.

Moreover, in order to be able to describe (and induce) the greatest number of significant context relations using the ARE formalism, the underlying RE should be selected among the REs in its equivalence class according to the following two (somewhat opposite) heuristics:

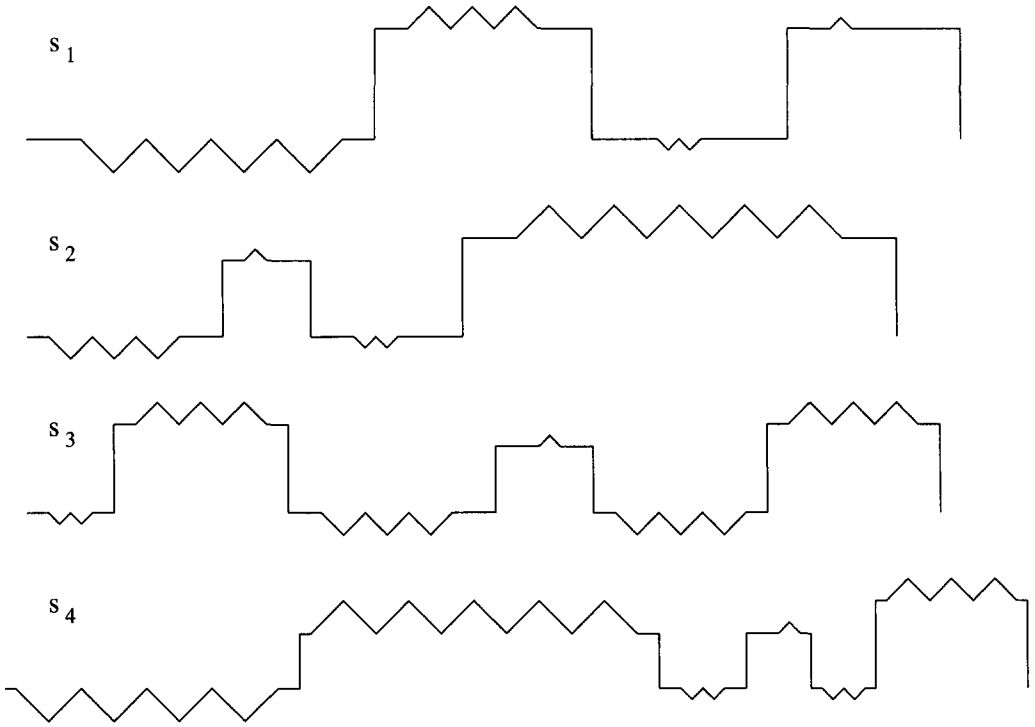
1. Maximize the number of stars.
2. Preserve unambiguity.

The aim of the first heuristic is to increase the potential for inferring context relations from the star instances obtained in parsing the examples by the RE. The aim of the second is to ease both the RE parsing and constraint induction processes. Note that applying any of the equations (3), (5), (8) or (9) (the former two in reverse) to an RE leads to an equivalent RE with a larger number of stars. However, equations (3), (5) and (8) introduce a great deal of ambiguity in the resulting RE. On the other hand equation (9) not only preserves, but enforces unambiguity, since an RE containing a union operation is transformed into an RE containing just concatenation and star operations. For instance, let $(P + Q)^* = (a + b)^*$ and take $a^3ba^3ba^3$ as input string, it results that $(a + b)^{11}$ is the "parse" by $(a + b)^*$ while $(a^3b)^2a^3$ is the "parse" by $(a^*b)^*a^*$.

All the stars in the output RE of Arden's algorithm originate by applying the rule of Theorem 2.1 in solving the equations (e_l^{l+1}), $0 \le l < n$, i.e.

$$R_l = (R_0\alpha_{0l}^{l+1} + \cdots + R_{l-1}\alpha_{(l-1)l}^{l+1}) + R_l\alpha_{ll}^{l+1}$$
$$\Rightarrow R_l = (R_0\alpha_{0l}^{l+1} + \cdots + R_{l-1}\alpha_{(l-1)l}^{l+1})\alpha_{ll}^{l+1^*}.$$

Therefore, if the RE α_{ll}^{l+1} is of the form $(P + Q)$, we may apply equation (9) to the subexpression $\alpha_{ll}^{l+1^*}$ to yield a "better" result in terms of the above heuristics. In the general case, there can be many ways of decomposing α_{ll}^{l+1} into the P and Q union operands. A meaningful decomposition is given by $P = \alpha_{ll}^{n}$ and $Q = \alpha_{ll}^{l+1} - \alpha_{ll}^{n}$, where P denotes the direct transitions from q_l to itself (the *loops* of q_l) and Q denotes the *circuits* starting and ending in q_l that only can traverse states q_k with $k > l$. In this way, loops can be discriminated from the rest of cycles of the given FSA in the resulting equivalent RE. This is important for pattern recognition tasks, where the loops of an FSA model usually account for (indefinite) length or duration of a basic primitive, a meaningful structure in the



$$s_1 = c^5d^3b^3d^3b^3d^3b^3d^3b^3c^3a^{10}c^3b^2d^2b^2d^2b^2d^2c^5e^{10}c^6dbdbc^8a^{10}c^4bdc^{10}e^{10}$$
$$s_2 = c^2d^2b^2d^2b^2d^2b^2c^4a^7c^2bdc^4e^7c^4dbdc^6a^9c^5b^3d^3b^3d^3b^3d^3b^3d^3b^3d^3c^5e^9$$
$$s_3 = c^2dbdbc^2a^8c^2b^2d^2b^2d^2b^2d^2c^2e^8c^3d^2b^2d^2b^2d^2b^2c^4a^6c^4bdc^3e^6c^2d^2b^2d^2b^2d^2b^2c^2a^8c^2b^2d^2b^2d^2b^2d^2c^2e^8$$
$$s_4 = cd^3b^3d^3b^3d^3b^3d^3b^3c^2a^5cb^3d^3b^3d^3b^3d^3b^3d^3b^3d^3c^2e^5c^2dbdbc^2a^5c^3bdce^5cdbdbca^8cb^2d^2b^2d^2b^2d^2ce^8$$

Fig. 8. Four example strings from $L(\tilde{R})_2$ which are used to infer the constraint L_2.

pattern (specially if tools are provided to relate the lengths, or durations, of different parts).

Consider the example of Fig. 5. The application of Arden's algorithm to the FSA of Fig. 5, plus the simplifying step, led to the RE $R = a(a + cc^*bdcc^*a)^* cc^*bdcc^*$, while the above modification on the algorithm yielded $R' = a(a^*cc^*bdcc^*a)^*a^*cc^*bdcc^*$. Although we could further transform R' into the more intuitive description $R'' = aa^*cc^*bdcc^*(aa^*cc^*bdcc^*)^*$ by using equation (7) and equation (4), R' is good enough as an underlying RE to enable the inference of all the contextual constraints displayed by the modelled pattern (see Fig. 5). Note that not all the constraints are inferrable from R.

5.3. *Inducing the constraints of an ARE from recorded star instances*

Once an RE R is inferred, the star variables V and the star tree $\mathscr{T}$ associated with R are easily determined.[12] Then, the aim is to induce an ARE $\tilde{R} = (R, V, \mathscr{T}, L)$ such that L contains the maximal number of (linear) context relations that are met by all the examples provided. In other words, we want $\tilde{R}$ to represent the *smallest* language covering the positive sample that may be accepted by an ARE with the same given RE. To this end, the example 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. Algorithm 3 (in Appendix B) carries out this last process returning a set of linear constraints L among the star variables V. Its time complexity is $O(|V|^3 \cdot I(ASI_{S^+}(V)))$, where $I(ASI_{S^+}(V))$ is the maximal number of *potential* instances of a star variable yielded by parsing the set of strings S^+.

Algorithm 3 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 descendants*. To construct this partition, each ancestor node of $\mathscr{T}$ keeps track of its *housed descendants* that have been determined to be independent. All the variables of $\mathscr{T}$ are visited by levels, and for each one (say v_j), its housing ancestor (say v_k) is found and a vector of its non-redundant instances is formed. Then a matrix is built that contains the instances of the independently housed descendants of v_k. Initially, the number of columns of the matrix is the number of independently housed descendants plus one (an all-1's column) and the number of rows is the number of (non-redundant) actual instances of v_j, which is bounded by

$$
\begin{array}{r}
\text{1st cycle of instance } v_{13} = 2 \text{ in } s_1 \\
\text{2nd cycle} \\
\text{1st cycle of instance } v_{13} = 2 \text{ in } s_2 \\
\text{2nd cycle} \\
\text{1st cycle of instance } v_{13} = 3 \text{ in } s_3 \\
\text{2nd cycle} \\
\text{3rd cycle} \\
\text{1st cycle of instance } v_{13} = 3 \text{ in } s_4 \\
\text{2nd cycle} \\
\text{3rd cycle}
\end{array}
\begin{bmatrix}
 & v_1 & v_4 & v_5 & v_6 & v_7 & v_{10} \\
1 & 5 & 4 & 3 & 10 & 3 & 3 \\
1 & 6 & 2 & 8 & 10 & 4 & 1 \\
1 & 2 & 3 & 4 & 7 & 2 & 1 \\
1 & 4 & 2 & 6 & 9 & 5 & 5 \\
1 & 2 & 2 & 2 & 8 & 2 & 3 \\
1 & 3 & 3 & 4 & 6 & 4 & 1 \\
1 & 2 & 3 & 2 & 8 & 2 & 3 \\
1 & 1 & 4 & 2 & 5 & 1 & 5 \\
1 & 2 & 2 & 2 & 5 & 3 & 1 \\
1 & 1 & 2 & 1 & 8 & 1 & 3
\end{bmatrix}
\quad X = \begin{bmatrix} v_{11} \\ 5 \\ 10 \\ 4 \\ 5 \\ 2 \\ 3 \\ 2 \\ 2 \\ 1 \\ 1 \end{bmatrix}
$$

Fig. 9. The linear system $A \cdot X = B$ for the star variable v_{11} (the top row of the displayed A and B is just for labeling purposes). The solution is $X = [0\,1\,0\,1\,0\;-1\,0]^{\mathrm{T}}$.

the number of parsed strings if v_k is the root node or otherwise by the total sum of the values of the instances 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 column vectors of the matrix. If it is, then the corresponding linear system can be solved to find the constraint coefficients, and the new constraint is appended to L; otherwise, v_j is included in the list of independently housed descendants of v_k.

Let us illustrate the method with the set of strings displayed in Fig. 8, that belong to $L(\tilde{R}_2)$. First, the sons of the root node are processed; in this case, v_{13} is the only one and it is found independent (since its instances $[2\ 2\ 3\ 3]^{\mathrm{T}}$ are not constant). Then, the sons of v_{13} are visited. It turns out that v_{13} is the housing ancestor of all of its sons. The star variables v_1, v_4, v_5, v_6, v_7 and v_{10} are successively found to be independent. At this point, the instances of v_{11} are stored in vector B to be analysed, while the matrix A contains the instances of the independent housing descendants of v_{13} already processed. Fig. 9 displays the corresponding linear system. It turns out that vector B is a linear combination of the columns of A, and solving the system yields $(v_{11} = v_1 + v_5 - v_7)$. This constraint is put into L. Then, the last son v_{12} is visited and the second constraint $(v_{12} = v_6)$ is similarly obtained. Next, v_2 and v_3 (the sons of v_4) are processed, v_{13} is determined as their housing ancestor (since their instances are constant for each instance of v_4), and both are found dependent according to $(v_2 = v_4 - 1,\ v_3 = v_4 - 1)$. Finally, v_8 and v_9 (the sons of v_{10}) are also housed by v_{13} and the analysis of their instances gives rise to the last constraints $(v_8 = 0.5v_{10} + 0.5,\ v_9 = 0.5v_{10} + 0.5)$. In the end, the inferred set of constraints L coincides with the target L_2 of $\tilde{R}_2$.

6. CONCLUSIONS

A powerful new representation class called *Augmented Regular Expressions* (AREs) has been presented to describe, recognize and learn a class of context-sensitive string languages capable of expressing multiple and complex context constraints. Moreover, the string recognition method proposed is efficient (low-polynomial in time). Although it has been demonstrated that not all the context-sensitive languages can be described by AREs, the class of representable objects includes planar shapes with symmetries, which is quite important for pattern recognition tasks.

On the other hand, it has been proved that AREs cover all the pattern languages,[8] but the size of an ARE describing a pattern language is exponential in the number of alphabet symbols. Even though this is not critical for alphabets with few symbols (including the binary case), it is a practical impediment in the rest of cases (say when $|\Sigma| > 10$). The cause of this inefficiency is due to the fact that the constraints in AREs only involve the number of instances of certain subpatterns in a string but not the relations among the subpatterns themselves.

In order to represent pattern languages such as $\{xx \mid x \in \Sigma^+\}$ more efficiently, the ARE concept should be extended by means of the definition of constraints (e.g. equality) among the substrings that result from instantiating the operands of the star-type subexpressions. This would require that the data structure produced by parsing a string by an RE would register, not only the lists of star variable instances, but also the associated lists of matched substrings. At first, this adaptation seems possible, and therefore, a more powerful formalism would be achieved at the expense of increasing the space requirements.

Another extension of the formalism is to allow the definition of (a limited class of) non-linear constraints among the star variables. In this way, a class of non-linear AREs, or NAREs for short, could be defined with a greater expressive power. Note that string recognition by NAREs can still be efficient, since only the equation satisfaction test should be replaced. However, inferring such models from string examples would be extremely hard due to the great number of combinations of variables and terms that could arise; moreover, the probability of inducing artificial constraints (i.e. noise) would grow.

On the other hand, in order to apply AREs to real pattern recognition tasks, it is quite clear that the recognition method presented must be made more flexible to cope with the noisy and incomplete input data and/or imperfect data segmentation that is typical

of practical problems. Generally, there are two ways to enable a parser to process imperfect data: either the model (e.g. grammar) is extended by common errors or the parser is made fault tolerant. In the case of AREs, the latter approach is more easily implementable. A flexible ARE parsing method may be attained by using a regular error-correcting parser for the process of matching the underlying RE in conjunction with a "tolerant" constraint checker that may be based on correlation and linear regression (instead of strict linear equations). Normally, a set of possible parses of the RE would have to be tested for constraint satisfaction in the relaxed sense. The development of the specific tools needed to adapt the ARE formalism to robust parsing are the subject of present and future work. The availability of these tools is required before trying to apply AREs to real-world problems.

Coming back to the given definition of AREs, it must be emphasized that the learning scheme presented is not a method for identifying AREs from examples, which is an open (and probably hard) problem, but a general approach to infer data-consistent AREs, trying to discover the maximal number of context relations. However, the constraint induction algorithm proposed ensures that, if the target RE is identified previously, then the target unknown ARE, which includes it, will be identified in the limit. This property follows from the characteristic of inferring the smallest language containing the examples among the AREs that include the same RE. The effectiveness of the whole procedure depends strongly on the result of the regular grammatical inference (RGI) step. The algorithm to be used in this step should be biased to return preferably small DFA (or better REs directly) with a high level of generalization with respect to the sample. There are two reasons for this demand: to obtain the simplest regular expression for parsing, and to be able to induce in the following phase the context constraints that limit the extension of the inferred language. The constraint induction could be impeded if the starting language, yielded by the RGI step, were too restricted to the given examples (i.e. a kind of sample overfitting).

Two drawbacks of the presented learning scheme must be mentioned, namely, the processing of negative examples, and the efficiency in the worst case. The trouble with negative examples is that, unless an informant is available, the learning algorithm does not know which negative examples should be rejected by the RE to be inferred and which should not. Hence, either all of the negative examples are supplied to the RGI step (thus biasing considerably the induction), or an iterative process is carried out, in which only the negative examples that are accepted by the ARE obtained at the end of an iteration are incorporated into the RGI step in the next iteration, until a consistent ARE is inferred.

Concerning efficiency, all the parts of the learning process run in low-order polynomial time, except the FSA-to-RE transformation, which is exponential in the worst case (e.g. for a fully-connected FSA). Although, fortunately, most of the FSA that are typically induced from object contours or other physical patterns are sparsely connected and present quite a limited degree of circuit embedment, thus allowing a practical computation of the equivalent RE, it is clear that the worst-case behavior may defeat the learning approach. A possible alternative is to obtain the RE from the RGI step to avoid the FSA-to-RE transformation. To this end, an RGI method returning REs should be selected; as far as we know, the uv^kw algorithm by Miclet[13] is the only such method among the range of known RGI methods.[14–16]

It is interesting to locate precisely our work with respect to other methods proposed within the field of inductive inference to approach the problem of learning formal languages from partial information. The reported methods can be classified depending on the class of languages they are able to infer and depending on whether the language is presented by examples or by queries.[17] If the input is restricted to just examples, as in our case, it is usual to distinguish between a *positive* presentation of a language L, this is a set of positive examples $S^+ \subseteq L$, and a (so-called) *complete* presentation, where two disjoint finite sets $S^+ \subseteq L$ (examples) and $S^- \subseteq \bar{L}$ (counter examples) are given. The inference problem associated with the presentation by examples is to find a formal description D of a language L' such that $S^+ \subseteq L'$ and $S^- \subseteq \bar{L}'$, and D satisfy certain restrictions (e.g. D is the smallest acceptor among all candidates). This problem is traditionally referred to as *grammatical inference* (GI).[14,15]

Even though it is well-known that any enumerable class of recursive languages (context-sensitive and below) can be identified in the limit from complete presentation (both positive and negative data),[18][8] most part of the work in GI has been devoted to the problem of inferring regular languages (i.e. finite-state automata).[14,16] Moreover, the majority of the reported regular GI methods are *heuristic* techniques that use only positive examples and some inductive bias,[16,19] while just a few methods have been proposed for regular GI from complete presentation, either in the classical symbolic paradigm[20–22] or through alternative approaches such as recurrent neural networks.[23–25]. In addition, some GI methods have been suggested to learn some proper subclasses of context-free languages, such as the even linear languages, from positive data,[26,27] and a few more algorithms have been proposed to infer general context-free grammars from positive structural examples[28] (or together with negative strings[29]).

However, in order to be used in a wider class of problems and applications, there is a need for GI methods that cope with the issue of learning context-sensitive languages. This requirement is specially significant in syntactic pattern recognition problems

[8]A GI method is said to *identify L in the limit* if for a larger and larger collection of examples, the descriptions D eventually converge to a correct description for L.

for computer vision,[2,3] where many objects contain symmetries and structural relationships that are not describable by context-free languages. Nonetheless, work on learning context-sensitive languages is extremely scarce in the literature. Recently, Takada has shown that a hierarchy of language families that are properly contained in the family of context-sensitive languages can be learned using regular GI algorithms.[30] His approach is based on using control sets on grammars and establishing a recursive sequence of controlling grammars that starts on a regular grammar. However, the class of learnable context-sensitive languages is restricted by the fact that the languages are generated through a sequence of (controlled) universal even linear grammars.

On the other hand, an early work by Chou and Fu[7] discussed the matter of inferring transition networks (TNs) from positive examples. An extension of the (heuristic) *k-tails* regular GI method was proposed for learning Basic TNs, thus covering the inference of context-free languages. Afterwards they gave a derived trial-and-error scheme (guided by a teacher) for the inference of Augmented TNs representing context-sensitive languages, which required some kind of a priori knowledge in the form of transformational rules. In general, however, the step form BTN to ATN learning is a hard one, since it is not clear at all how can the test conditions and register-setting actions typical of ATN arcs be inferred from just string examples. As far as we know, no other paper following this line of research has been reported since then.

In summary, our strategy to infer AREs, although it still presents some drawbacks, is probably the most promising attempt reported so far to learn (caution, we do not mean *identify*) a large class of context-sensitive language descriptors from examples. Moreover, these descriptors, the AREs, can be used indirectly as efficient recognizers, and they provide a compact and intelligible representation.

APPENDIX A

List of context-sensitive grammars referenced in the text

Language $\{a^m b^n a^m b^n \mid m, n \geq 1\}$.

Grammar CSG_1:

1	$S \to abab$	2	$S \to aAbB[aC][bD]$
3	$A \to aAE$	4	$A \to aF$
5	$B \to bBG$	6	$B \to bH$
7	$G[aC] \to [aC]b$	8	$Gb \to bb$
9	$G[bD] \to b[bD]$	10	$H[aC] \to [aC]H$
11	$Hb \to bH$	12	$H[bD] \to bb$
13	$Eb \to bE$	14	$Ea \to aa$
15	$E[aC] \to a[aC]$	16	$Fb \to bF$
17	$Fa \to aF$	18	$F[aC] \to aa$

Language $\{(c^{v_1}(d^{v_2}b^{v_3})^{v_4}c^{v_5}a^{v_6}c^{v_7}(b^{v_8}d^{v_9})^{v_{10}}c^{v_{11}}e^{v_{12}})^+ \mid v_{11} = v_1 + v_5 - v_7; v_{12} = v_6;\ \ v_2 = v_4 - 1; v_3 = v_4 - 1; v_8 = 0.5v_{10} + 0.5; v_9 = 0.5v_{10} + 0.5; v_2, v_3, v_6, v_8, v_9 \geq 1\}$.

Grammar CSG_2:

1	$S \to AS$	2	$S \to A$
3	$A \to EF[aG][eH]$	4	$F \to aFI$
5	$F \to J$	6	$I[aG] \to [aG]I$
7	$J[aG] \to aG$	8	$I[eH] \to [eH]e$
9	$G[eH] \to Ge$	10	$E \to [LN]$
11	$E \to [LN][cO]M$	12	$E \to [cK]M[LO]$
13	$[cK] \to c[cK]M$	14	$[cK] \to [cN]$
15	$[cO] \to c[cO]M$	16	$[cO] \to c$
17	$M[LO] \to [LO]M$	18	$[cN][LO] \to c[LN]$
19	$[LO] \to L[cO]M$	20	$[LN]c \to L[cN]$
21	$[cN]c \to c[cN]$	22	$[cN]L \to c[LN]$
23	$Ma \to aM$	24	$[cN]a \to c[aN]$
25	$[LN]a \to L[aN]$	26	$[aN]a \to a[aN]$
27	$G \to [PQR]$	28	$M[PQR] \to M[QR]$
29	$M[PQR] \to [cP][QR]$	30	$[aN][PQR] \to aQ$
31	$M[cP] \to [cP]M$	32	$M[cP] \to [cP]c$
33	$Mc \to cM$	34	$[aN][cP] \to a[cN]$
35	$[aN]Q \to a[QN]$	36	$[cN] \to [QR]cQ$
37	$M[QR] \to Q[Rc]$	38	$MQ \to QM$
39	$M[Rc] \to [Rc]c$	40	$[cN]Q \to c[QN]$
41	$[QN][Rc] \to Qc$	42	$L \to [VT][LW]$
43	$[LW] \to T[UW]$	44	$[LW] \to TL[UW]$
45	$L \to TU$	46	$L \to TLU$
47	$TU \to TXb$	48	$T[UW] \to T[XW]b$
49	$bU \to Xbb$	50	$bX \to Xb$
51	$b[UW] \to [XW]bb$	52	$b[XW] \to [XW]b$
53	$TX \to Ud$	54	$T[XW] \to [UW]d$
55	$dX \to Udd$	56	$dU \to Ud$
57	$d[XW] \to [UW]dd$	58	$d[UW] \to [UW]d$
59	$[VT]U \to [VX]b$	60	$[VT][UW] \to db$
61	$[VX]X \to [dV]X$	62	$[VX][XW] \to [dV][XW]$
63	$[dV]X \to d[dV]$	64	$[dV][XW] \to dd$
65	$Q \to [VT']U'$	66	$Q \to [VT']T'Q'U'$
67	$Q' \to T'T'Q'U'$	68	$Q' \to T'U'$
69	$T'U' \to T'X'd$	70	$dU' \to X'dd$
71	$dX' \to X'd$	72	$T'X' \to U'b$
73	$bX' \to U'bb$	74	$bU' \to U'b$
75	$[VT']U' \to [VX']d$	76	$[VX']X' \to [bV']X'$
77	$[VX']d \to bd$	78	$[bV']X' \to b[bV']$
79	$[bV']d \to bd$		

Rules 3–9
implement the constraint
$v_{12} = v_6$

Rules 10–41
implement the constraint
$v_{11} = v_1 + v_5 - v_7$

Rules 42–64
implement the constraints
$v_2 = v_4 - 1; v_3 = v_4 - 1$

Rules 65–79
implement the constraints
$v_8 = 0.5v_{10} + 0.5; v_9 = 0.5v_{10} + 0.5$

Language $\{a^k \mid k = 2^i \wedge i \geq 1\}$.

Grammar CSG_3:

1 $S \to [ACaB]$	2 $[Ca]a \to aa[Ca]$	1 $S \to 0E[CB]$	2 $S \to 0[FB]$
3 $[Ca][aB] \to aa[CaB]$	4 $[ACa]a \to [Aa]a[Ca]$	3 $S \to 1E[DB]$	4 $S \to 1[GB]$
5 $[ACa][aB] \to [Aa]a[CaB]$		5 $E \to 0EC$	6 $E \to 1ED$
6 $[ACaB] \to [Aa][aCB]$	7 $[CaB] \to a[aCB]$	7 $E \to 0F$	8 $E \to 1G$
8 $[aCB] \to [aDB]$	9 $[aCB] \to [aE]$	9 $C[0B] \to 0[CB]$	10 $C0 \to 0C$
10 $a[Da] \to [Da]a$	11 $[aDB] \to [DaB]$	11 $C[1B] \to 1[CB]$	12 $C1 \to 1C$
12 $[Aa][Da] \to [ADa]a$	13 $a[DaB] \to [Da][aB]$	13 $[CB] \to [0B]$	14 $D[0B] \to 0[DB]$
14 $[Aa][DaB] \to [ADa][aB]$		15 $D0 \to 0D$	16 $D[1B] \to 1[DB]$
15 $[ADa] \to [ACa]$	16 $a[Ea] \to [Ea]a$	17 $D1 \to 1D$	18 $[DB] \to [1B]$
17 $[aE] \to [Ea]$	18 $[Aa][Ea] \to [AEa]a$	19 $F[0B] \to 0[FB]$	20 $F0 \to 0F$
19 $[AEa] \to a$		21 $F[1B] \to 1[FB]$	22 $F1 \to 1F$
		23 $[FB] \to 0$	24 $G[0B] \to 0[GB]$
		25 $G0 \to 0G$	26 $G[1B] \to 1[GB]$
		27 $G1 \to 1G$	28 $[GB] \to 1$

Language $\{xx \mid x \in (0+1)^+\}$.

Grammar CSG_4:

APPENDIX B

Algorithms referenced in the text.

Algorithm 1. *Evaluates the predicate satisfy* $(SI_s(V), L)$.

Inputs: A set V of star variables that are organized in a star tree $\mathcal{T}$.

A set SI containing the lists of instances of the star variables V resulting from the (successful) parsing of a string s.

A set L of linear relations among the star variables in V.

Outputs: A boolean variable *satisfy_constraints* whose value will be TRUE if and only if the star instances SI satisfy all the constraints in L.

begin

$ASI[1] := SI$ {The set SI is placed in the first position of an array ASI just for compatibility with the arguments of the functions determine_ancestor_and_instances and sum_of_star_exponents.}

{Mark the star variables for which the housing ancestor has not been computed}

for *v_id* **in** $[1.. \mid V \mid]$ **do**

  housing_ancestor_of_node [*v_id*]$:= -1$

end_for

{Check the constraints L}

$i := 1$; *satisfy_constraints*$:=$ *TRUE*

while $i \leq L.nc$ and *satisfy_constraints* **do** {check each constraint}

  vdep_id:= element_of_list $(L.^{Vdep}, i)$ {get the ith dependent variable}

  linear_combination[*vdep_id*]:= element_of_list (*L.right_hand_sides*, i)

  X:= *linear_combination*[*vdep_id*].*coefficients*

  list_of_indep_variables:= *linear_combination*[*vdep_id*].*independent_variables*

  common_housing_anc_id:= deepest_common_ancestor ($\mathcal{T}$, *vdep_id*, *list_of_indep_variables*)

  node:= startree_node_identified_by (*vdep_id*)

  determine_ancestor_and_instances ($\mathcal{T}$, *node*, *vdep_id*, *ASI*, 1, *common_housing_anc_id*, *anc_id*, *coef*[*vdep_id*]) {returns *anc_id*, the identifier of the housing ancestor of *node*, and *coef*[*vdep_id*], the values of the instances of $v_{vdep_{id}}$ for each instance of v_{anc_id} in $ASI[1]$}

  housing_ancestor_of_node [*vdep_id*]:= *anc_id*

  if number_of_actual_instances (*coef*[*vdep_id*]) > 0 **then**

  exists_common_housing_ancestor:= *TRUE*

    if *housing_ancestor_of_node* [*vdep_id*] $\neq$ *common_housing_anc_id* **then**

      exists_common_housing_ancestor:= *FALSE*

    end_if

 $j := 1$; $l :=$ *length* (*list_of_indep_variables*)

while $j \leq l$ **and** *satisfy_constraints* **do**

  v_id:= element_of_list (*list_of_indep_variables*, j)

  if *housing_ancestor_of_node* [*v_id*] < 0 **then**

  node:= startree_node_identified_by (*v_id*)

  determine_ancestor_and_instances ($\mathcal{T}$, *node*, *v_id*, *ASI*, 1, *common_housing_anc_id*, *anc_id*, *coef*[*v_id*]) {returns *anc_id*, the identifier of the housing ancestor of *node*, and *coef*[*v_id*], the values of the instances of v_{v_id} for each instance of v_{anc_id} in $ASI[1]$}

housing_ancestor_of_node [*v_id*]:= *anc_id*
end_if
if *housing_ancestor_of_node* [*v_id*] $\neq$ *common_housing_anc_id* **then**
eists_common_housing_ancestor:= *FALSE*
end_if
if not *exsts_common_housing_ancestor* **then**
if not *equal_instances(housing_ancestor_of_node* [*vdep_id*], *housing_ancestor_of_node* [*v_id*], *ASI*, 1) **then**
satisfy_constraints:= *FALSE*
end_if
end_if
$j := j + 1$
end_while
if *satisfy_constraints* **then**
row_max_number:= sum_of_star_exponents (*ASI*, *housing_ancestor_of_node* [*vdep_id*], 1)
column_number:= $l + 1$
build_system_matrix (*vdep_id*, *list_of_indep_variables*, *coef*, *row_max_number*, *column_number*, *row_actual_number*, *B*, *A*) { vector *B* is given by the star instances of $v_{vdep_i d}$, and the system matrix *A* is given by the star instances of the independent variables from which $v_{vdep_i d}$ depends, and after removing the rows for which the instance of $v_{vdep_i d}$ is unassigned.}
satisfy_constraints:= test_linear_system (*A*, *X*, *B*, *row_actual_number*, *column_number*)
end_if
end_if
$i := i + 1$
end_while
$\{satisfy_constraints \equiv satisfy(SI_s(V), L)\}$
end_algorithm

Algorithm 2. *Induction of augmented regular epressions*
Inputs: A sample $S = (S^+, S^-)$ of a context-sensitive (unknown) language $\mathscr{L}$ over an alphabet Σ.
Outputs: A deterministic FSA $A = (\Sigma, Q, \delta, q_0 F)$, consistent with the sample S (i.e. $S^+ \subseteq L(A) \wedge S^- \subseteq \Sigma^* - L(A)$, that is obtained from S through a regular grammatical inference algorithm, and which is assumed to accept a regular superset of $\mathscr{L}$ (i.e. $\mathscr{L} \subseteq L(A)$).
The "canonical" RE R equivalent to A.
The set V of star variables and the star tree $\mathscr{T}$ associated with R.
The array of sets *ASI* with the lists of instances of the star variables V resulting from the parsing of the positive examples ($s \in S^+$) by R.
A set L of linear relations among the star variables in V, that is fully satisfied in the parsing of all the positive examples ($s \in S^+$) by R.
The inferred ARE (*Augmented Regular Expression*): $\tilde{R} = (R, V, \mathscr{T}, L)$.

begin
A:= regular_grammatical_inference_algorithm (S^+, S^-)
n:= number_of_states_of_automaton (A)
run_Arden's_Algorithm $(A, n, R_j, \alpha_{ij}^l)$ {returns R_j and α_{ij}^l for $0 \leq j < n$, $1 \leq l \leq n, 0 \leq i \leq l$}
simplify_equivalent_RE $(A, n, R_j, \alpha_{ij}^l, skelR, R)$ {the "canonical"1 RE R and its skeleton *skelR* are obtained}
run_star_tree_construction (R, V, $\mathscr{T}$) {returns the star variables V and star tree $\mathscr{T}$ associated with R}
number_positive_examples := $|S^+|$
for *example* in [1.. *number_positive_examples*] **do**
s:= read_positive_example $(S^+, example)$ {where s is a string over Σ}
run_RE_parsing_guided_by_FSA (s, A, *skelR*, R, V, $\mathscr{T}$, *parsed*, *path*, *skelR_instance*, *SI*)
{returns *parsed* = TRUE, the *path* of states visited, the skeleton of the example *skelR_instance*, and the most important, the set *SI* of lists of star instances of the star variables V resulting from parsing s.}
ASI[*example*] := *SI*
end_for
run_Algorithm_3 (V, $\mathscr{T}$, *ASI*, *number_positive_examples*, L) {returns L, the induced set of linear constraints among the star variables}
$\tilde{R} := (R, V, \mathscr{T}, L)$ {simply put the components of the ARE together}
end_algorithm

Algorithm 3. *Induces a set of linear relations that hold among the star variables throughout the instances resulting from the parsing of a set of strings.*

Inputs: A set V of star variables, that are organized in a star tree $\mathcal{T}$.
An array of sets *ASI* containing the lists of instances of the star variables V resulting from the (successful) parsing of a set of strings.
The size of the array *ASI*: *number_of_strings*.

Outputs: A set L of linear relations among the star variables in V, that is fully satisfied in the parsing of all the strings from which *ASI* has been built.

begin
$L.nc := 0$ {Initialize the number of constraints (dependent star variables) in L}
$L.V^{dep} := \emptyset$ {Create an empty set of dependent star variables V^{dep}}
L._right_hand_sides $:= \emptyset$ {Create an empty set of linear combinations associated with V^{dep}}
$\max_{level}$:= deepest_level_of_tree ($\mathcal{T}$) {this is the level of the deepest node in $\mathcal{T}$}
for l **in** [0..*max_level* − 1] **do** {visit the father nodes of $\mathcal{T}$ by levels}
  for i **in** [1.. *nodes_of_level_l* ($\mathcal{T}$, l)] **do**
    father:= node_i_of_level_*l* ($\mathcal{T}$, l, i)
    father_id:= node_identifier_of (*father*)
    number_of_indep_housed_descendants [*father_id*] := 0
    list_of_indep_housed_descendants [*father_id*] := $\emptyset$
    max_number_of_rows_of_system_matrix [*father_id*]:= sum_of_star_exponents (*ASI*, *father_id*, *number_of_strings*)
    for *nson* **in** [1.. nsons_of (*father*)] **do**
    son:= son_of (*father*, *nson*)
    son_id:= node_identifier_of (*son*)
    determine_ancestor_and_instances ($\mathcal{T}$, *son*, *son_id*, *ASI*, *number_of_strings*, 0, *anc_id*, *coef*[*son_id*]) {returns *anc_id*, the identifier of the nearest non-degenerated ancestor of son, and *coef*[*son_id*], the values of the instances of v_{son_id} for each instance of v_{anc_id}} in *ASI*}
    housing_ancestor_of_node [*son_id*] := *anc_id*
    row_max_number := *max_number_of_rows_of_system_matrix* [*anc_id*]
    column_max_number := *number_of_indep_housed_descendants* [*anc_id*] + 1
    build_system_matrix (*son_id*, *list_of_indep_housed_descendants*[*anc_id*], *coef*, *row_max_number*, *column_max_number*, *row_actual_number*, *B*, *A*) {vector B is given by the star instances of v_{son_id}, and the system matrix A is given by the star instances of the independently housed descendants of v_{anc_id}, after removing the rows for which the instance of v_{son_id} is unassigned.}
    if *row_actual_number* > 0 **then**
    rank A:= rank_of_matrix (*A*, *row_actual_number*, *column_max_number*)
    if *rank A* < *column_max_number* **then**
    remove_dependent_columns_from_system_matrix (*A*, *rankA*, *row_actual_number*, *column_actual_number*, *columns_mask*) {matrix A is reduced by removing (in increasing order) those columns that are a linear combination of the previous ones, *column_actual_number* = *rankA*, and *columns_mask* is a binary mask that marks the selected columns}
    else
    column_actual_number := *column_max_number*
    columns_mask := fully_marked_mask (*column_max_number*)
    end_if
    build_extended_system_matrix (*A*, *B*, *row_actual_number*, *column_actual_number*, *AB*) { matrix AB is built by appending to matrix A a column with vector B}
    rank AB:= rank_of_matrix (*AB*, *row_actual_number*, *column_actual_number* + 1)
    if *rank AB* < (*column_actual_number* + 1) **then** {the appended column is a linear combination of the columns of A}
    solve_linear_system (*A*, *B*, *row_actual_number*, *column_actual_number*, *X*) {vector X is obtained by solving the linear system $A \cdot X = B$}
    linear_combination[*son_id*].*coefficients* := X
    linear_combination[*son_id*].*independent_variables*:= extract_masked_list (*list_of_indep_housed_descendants* [*anc_id*],
    column_max_number, *column_actual_number*, *columns_mask*)
    linear_combination[*son_id*] :=remove_indep_variables_with_coef_zero (*linear_combination*[*son_id*])
    $L.nc := L.nc + 1$ {the number of dependent star variables is increased and the new constraint is appended to L}
    $L.V^{dep}$:= append ($L.V^{dep}$, *son_id*) {i.e. $v_{son_id} \in V^{dep}$}
    L._right_hand_sides append (*L.right_hand_sides*, *linear_combination*[*son_id*])
    else {the appended column is linearly independent}

```
        number_of_indep_housed_descendants [anc_id]:= number_of_indep_housed_d escendants [anc_id] + 1
        list_of_indep_housed_descendants [anc_id]:= append (list_of_indep_housed_desc endants [anc_id], son_id)
            end_if
          end_if
        end_for
      end_for
    end_for
end_algorithm
```

REFERENCES

1. E. Tanaka, Theoretical aspects of syntactic pattern recognition, *Pattern Recognition* **28**, 1053–1061 (1995).
2. K. S. Fu, *Syntactic Pattern Recognition and Applications*. Prentice-Hall, New York (1982).
3. H. Bunke and A. Sanfeliu (eds), *Syntactic and Structural Pattern Recognition: Theory and Applications*. World Scientific, Singapore (1990).
4. A. Salomaa, *Formal Languages*. Academic Press, New York (1973).
5. J. E. Hopfcroft and J. D. Ullman, *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, Reading MA (1979).
6. W.A. Woods, Transition networks grammars for natural language analysis, *CACM* **13**, 591–606 (1970).
7. S. M. Chou and K. S. Fu, Inference for transition network grammars, *Proc. Int. Joint Conf. on Pattern Recognition*, **3**, CA, pp. 79–84 (1976).
8. D. Angluin, Finding patterns common to a set of strings, *J. Comput. System Sci.* **21**, 46–62 (1980).
9. A. Marron and K. Ko, Identification of pattern languages from examples and queries, *Inform. Computation* **74**, 91–112 (1987).
10. Z. Kohavi, *Switching and Finite Automata Theory*, (2nd edition). Tata McGraw-Hill, New Delhi, India (1978).
11. D. N. Arden, Delay logic and finite state machines, *Proc. Second Ann. Symp. on Switching Theory and Logical Design*, 133–151 (1961).
12. R. Alquézar and A. Sanfeliu, Augmented regular expressions: a formalism to describe, recognize, and learn a class of context-sensitive languages, *Research Report LSI-95-17R*, Universitat Politecnica de Catalunya, Spain (1995).
13. L. Miclet, Inference of regular expressions, *Proc. 3rd Int. Conf. on Pattern Recognition* 100–105 (1976).
14. L. Miclet, Grammatical inference, in *Syntatic and Structural Pattern Recognition: Theory and Applications*, H. Bunke and A. Sanfeliu, eds. World Scientific, Singapore (1990).
15. R. C. Carrasco and J.Oncina (eds), Grammatical Inference and Applications, *Proc. of the Second Int. Colloquium, ICGI'94, Alicante*, Spain, Springer-Verlag, Lecture Notes in Artificial Intelligence 862, (1994).
16. J. Gregor, Data-driven inductive inference of finite-state automata, *Int. J. of Pattern Recognition and Artificial Intell.* **8**(1), 305–322 (1994).
17. D. Angluin and C.H. Smith, Inductive inference: Theory and methods, *ACM Computing Survey* **15**(3), 237–269 (1983).
18. E.M. Gold, Language identification in the limit, *Inform. Control* **10**, 447–474 (1967).
19. M. Kudo and M. Shimbo, Efficient regular grammatical inference techniques by the use of partial similarities and their logical relationships, *Pattern Recognition* **21**, 401–409 (1988).
20. E.M. Gold, Complexity of automaton identification from given data, *Inform. Control* **37**, 302–320 (1978).
21. J. Oncina and P. Garcia, Identifying regular languages in polynomial time, in *Advances in Structural and Syntactic Pattern Recognition*, H. Bunke, ed. World-Scientific, Singapore, pp. 99–108 (1992).
22. R. Alquézar and A. Sanfeliu, Incremental grammatical inference from positive and negative data using unbiased finite state automata, in *Shape, Structure and Pattern Recogniton, Proc. Int. Workshop SSPR'94*, Nahariya, Israel, D. Dori and A. Bruckstein, eds, World Scientific Pub., Singapore, pp. 291–300 (1995).
23. C.L. Giles, C.B. Miller, D. Chen, H.H. Chen, G.Z. Sun and Y.C. Lee Learning and extracting finite state automata with second-order recurrent neural networks, *Neural Computation* **4**, 393–405 (1992).
24. A. Sanfeliu and R. Alquézar, Active grammatical inference: a new learning methodology, in *Shape, Structure and Pattern Recogniton, Proc. Int. Workshop SSPR'94*, Nahariya, Israel, D. Dori and A. Bruckstein, eds. World Scientific Pub., Singapore, 191–200 (1995).
25. R. Alquézar and A. Sanfeliu, An algebraic framework to represent finite-state machines in single-layer recurrent neural networks, *Neural Computation*, **7**, 931–949 (1995).
26. V. Radhakrishnan and G. Nagaraja, Inference of even linear grammars and its application to picture description languages, *Pattern Recognition* **21**, 55–62 (1988).
27. Y. Takada, Grammatical inference for even linear languages based on control sets, *Inform. Process. Lett.* **28**(4), 193–199 (1988).
28. Y. Sakakibara, Efficient learning of context-free grammars from positive structural examples, *Inform. Computation* **97**, 23–60 (1992).
29. P. Garcia and J. Oncina, Learning general context-free grammars from positive structural samples and negative strings, *DSIC Research Report*, Universidad Politecnica de Valencia, Spain (1993).
30. Y. Takada, A hierarchy of language families learnable by regular language learners, in *Grammatical Inference and Applications, Proc. of the Second Int. Colloquium, ICGI'94, Alicante*, Spain, R. C. Carrasco and J. Oncina, eds. Springer-Verlag, Lecture Notes in Artificial Intelligence, **862**, 16–24 (1994).

About the Author—RENÉ ALQUÉZAR received the licentiate degree in computer science from the Polytechnical University of Catalonia (UPC), Barcelona, Spain, in 1986. From 1987 to 1991 he was with the Spanish company NTE as technical manager of R&D projects on image processing applications for the European Space Agency (ESA). From 1991 to 1994 he was with the "*Institut de Cibernetica*", Barcelona, holding a pre-doctoral research grant from the Government of Catalonia and completing the doctoral courses of the UPC on artificial intelligence. Since 1994, he has been an associate professor in the Department of "*LLenguatges i Sistemes Informàtics*", UPC. His current research interests include syntactic pattern recognition, grammatical inference, recurrent neural networks, computer vision, and artificial intelligence.

About the Author—ALBERTO SANFELIU received his diploma in industrial engineering (specializing in electrical engineering) from the School of Industrial Engineering of Barcelona in 1978, and his Ph.D. degree in industrial engineering from the Polytechnical University of Catalonia (UPC) in 1982. From 1979 to 1981, he was visiting researcher with Prof. K. S. Fu in Purdue University. He joined the UPC in 1981 as associate professor. From 1975 until 1995 he was a researcher in the "*Institut de Cibernètica*", Barcelona, research institute which belonged to the National Council of Scientific Research (CSIC) and to the UPC. He is now a professor in the Department of "*Enginyeria de Sistemes, Automàtica i Informàtica Industrial*", UPC, and a researcher in the "*Instituto de Robótica e Informática Industrial*", CSIC-UPC. Prof Sanfeliu has edited 6 books in the field of Pattern Recognition, mainly in the field of Syntactic and Structural Pattern Recognition, published around 80 articles including book chapters, journal articles and conference papers. He was elected Fellow of the International Association for Pattern Recognition (IAPR) in 1994. He is an Associate Editor of the *International Journal of Pattern Recognition and Artificial Intelligence*, and he has been a Co-guest editor with Horst Bunke in a special issue on Advances in Syntactic Pattern Recognition, which was published in the *Pattern Recognition journal*. He is President of the Spanish Association of Pattern Recognition and Image Analysis. He has been Chairman of the Technical Committee on Syntactic and Structural Pattern Recognition of the IAPR. His research interests include pattern recognition and computer vision mainly applied to robotics.