Efficient Recognition of a Class of Context-Sensitive Languages Described by Augmented Regular Expressions

Alberto Sanfeliu[1] and René Alquézar[2]

[1] Institut de Robòtica i Informàtica Industrial, UPC-CSIC
Gran Capità 2-4, Edifici Nexus, 08034 Barcelona, Spain
Dept. LSI, Universitat Politècnica de Catalunya, Barcelona
sanfeliu@ic.upc.es, alquezar@lsi.upc.es

Abstract. Recently, Augmented Regular Expressions (AREs) have been proposed as a formalism to describe, recognize and learn 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 an RE. Although it is 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. Likewise, it is proved that AREs cover all the pattern languages [3]. An efficient algorithm is presented to recognize language strings by means of AREs. The method is splitted in two stages: parsing the string by the underlying regular expression and checking that the resulting star instances satisfy the constraints.

1 Introduction

In order to extend the potential of application of the syntactic approach to pattern recognition [4], the efficient use of models capable of describing context-sensitive structural relationships is needed, since most objects cannot be represented adequately by regular or context-free languages [5]. Moreover, learning such models from examples is interesting to automate as much as possible the development of applications. Context-sensitive grammars [6] are not a good choice, since their parsing is computationally expensive and there is not any available algorithm to learn them automatically. Augmented Transition Networks (ATNs) [7] are powerful models that have been used in natural language processing, but which are very difficult to infer [8]. Pattern languages [3] provide a very limited mechanism to take into account some context influences (namely, the repetition of variable substrings along the strings of the language), and some algorithms have been proposed to infer them from examples and queries [3, 9]. Nevertheless, the expressive power of pattern languages is clearly insufficient to cope with most of the context-sensitive structures (e.g. rectangles).

On the other hand, it is known that controlled (context-free) grammars can generate some context-sensitive languages (CSLs) [6]. By using a recursive

sequence of control sets on universal even-linear grammars, Takada has shown that a hierarchy of language families that are properly contained in the class of CSLs can be learned using regular inference algorithms [10]. Furthermore, an efficient parsing procedure can be devised for each language in any of these families, which is based on parsing successively by a set of universal even-linear grammars [10]. However, the gap in expressive power between each of these language families and the class of CSLs seems to be rather large, and it is not clear what types of context relations can be described by the controlled grammars.

Recently, Augmented Regular Expressions (AREs) have been proposed as a formalism to describe, recognize and learn a class of CSLs, that covers planar shapes with symmetries [1]. AREs are neither the *regular-like expressions* [6], that are known to describe the family of CFLs, nor a type of regulated rewriting [6]. Roughly speaking, 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 variables are related through a finite number of constraints (linear equations). Note that REs are reduced to AREs with zero constraints among the star variables. A general method to learn AREs from examples is described elsewhere [2]. Here, we deal with the problem of recognizing a given string as belonging to the language described by an ARE, and we present an efficient method to solve it.

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 a 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 $L_P \cup L_Q$, $L_P L_Q$ 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, 12].

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) an edge (r, v_i) is created for each $v_i \in V$ that is not *directly-contained* by other star variable; (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 (so v_j is a *son* of v_i). A simple 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.

We say that a star variable $v \in V$ is *instantiated*, during the parsing of a string s by RE R (from which V has been defined), each time the operand of the corresponding star (an RE) is matched zero or some number of consecutive times against a substring of s. The number of repeated matches (*cycles*) of the star operand in an instance of v will be the value of v for that instance. Hence, star variables can only take natural numbers as values. However, we will see that, for computational purposes, it is useful to assign a special value, say -1, to a star variable v, whenever v is not instantiated during a cycle of an instance of its father in $\mathcal{T}$. In this way, all the star variables that are brothers in the star tree $\mathcal{T}$ will have the same structure of instances for a given string. Let us put it more formally.

Let V be the set of star variables associated with an RE R. Given a certain string s belonging to the language $L(R)$, 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:

$$\forall i \in [1, ns]: \quad SI_s(v_i) = (l_1^i \ ... \ l_{nlists(i)}^i) \quad \text{where} \quad nlists(i) \geq 0$$

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

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. Fig.1 shows an example of the star variable instances for a given string and RE, for which the star tree $\mathcal{T}$ has four levels.

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

$$\forall v_i, \ (r, v_i) \in \mathcal{T} \ \Rightarrow \ SI_s(v_i) = (l_1^i) \ \wedge \ l_1^i = (e_{11}^i) \ \wedge \ father\_list(l_1^i) = -1 \ \wedge \ father\_elem(l_1^i) = -1$$

i.e. $nlists(i) = 1$ and $nelems(i, 1) = 1$; furthermore, if v_i is not instantiated in parsing s then $e_{11}^i = -1$ else $e_{11}^i \geq 0$ 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 or higher levels of $\mathcal{T}$, we have the following general rule:

$$
\begin{aligned}
R &= (a(b(ce^*c \;+\; df^*d)^*)^*)^* \\
V &= \{v_1, v_2, v_3, v_4, v_5\} \\
R(V/*) &= (a(b(ce^{v_1}c \;+\; df^{v_2}d)^{v_3})^{v_4})^{v_5} \\
\mathcal{T} &= (V \cup r, \{(r, v_5), (v_5, v_4), (v_4, v_3), (v_3, v_1), (v_3, v_2)\}, r) \\
s &= abccdffdcecbddbdfdceecabceeec \\
SI_s(v_5) &= (\ (2)^{\{-1,-1\}}\ ) \\
SI_s(v_4) &= (\ (3\ 1)^{\{1,1\}}\ ) \\
SI_s(v_3) &= (\ (3\ 1\ 2)^{\{1,1\}}\ \ (1)^{\{1,2\}}\ ) \\
SI_s(v_1) &= (\ (0\ \text{-}1\ 1)^{\{1,1\}}\ \ (\text{-}1)^{\{1,2\}}\ \ (\text{-}1\ 2)^{\{1,3\}}\ \ (3)^{\{2,1\}}\ ) \\
SI_s(v_2) &= (\ (\text{-}1\ 2\ \text{-}1)^{\{1,1\}}\ \ (0)^{\{1,2\}}\ \ (1\ \text{-}1)^{\{1,3\}}\ \ (\text{-}1)^{\{2,1\}}\ )
\end{aligned}
$$

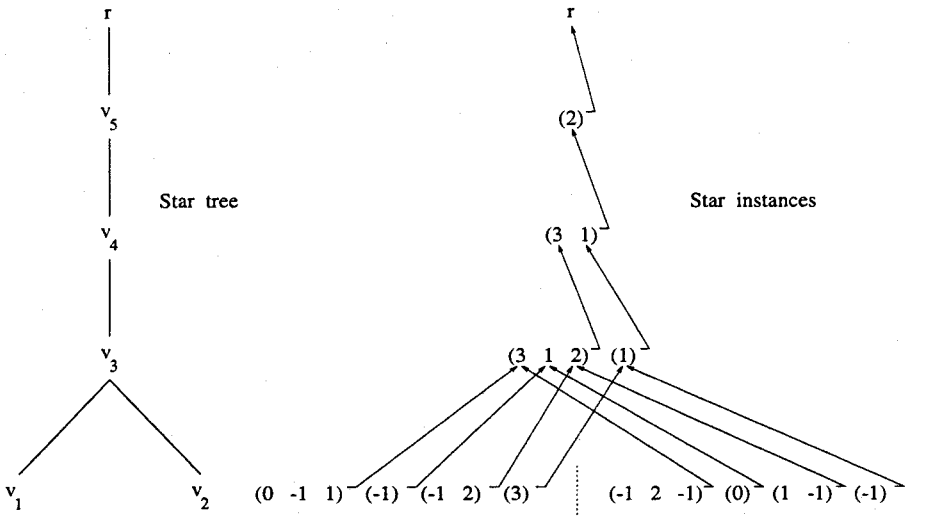


Fig. 1 *An example of star instances data structure.*

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

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 cycle). Two efficient algorithms for *unambiguous*[1] RE parsing that construct the star instances structure have been reported [1].

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

[1] 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.

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

$$l_i \text{ is } v_i^{dep} = a_{i1}v_1^{ind} + .. + a_{ij}v_j^{ind} + .. + a_{i(ni)}v_{ni}^{ind} + a_{i0}, \text{ for } 1 \leq i \leq nc$$

where ni and nc are the number of independent and dependent star variables, respectively (and $ns = nc + ni$). The equations in $\mathcal{L}$ are only well-defined for natural values of the involved variables. Moreover, the coefficients a_{ij} of the linear relations will always be rational numbers.

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 the star instances $SI_\alpha(V)$ *satisfy* $\mathcal{L}\}$. The formal definition of the predicate $satisfy(SI_\alpha(V), \mathcal{L})$ is stated in the next section, together with the description of the proposed method for string recognition.

The AREs permit to describe 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 described by the ARE $\tilde{R}_1 = (R_1, V_1, \mathcal{T}_1, \mathcal{L}_1)$, with $R_1(V_1/*) = aa^{v_1}bb^{v_2}aa^{v_3}bb^{v_4}$ and $\mathcal{L}_1 = \{v_3 = v_1, v_4 = v_2\}$. However, quite more complex languages with an arbitrary level of star embedment and multiple linear constraints can be described as well by the ARE formalism. Consider, for instance, the ARE $\tilde{R}_2 = (R_2, V_2, \mathcal{T}_2, \mathcal{L}_2)$ 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 $\mathcal{L}_2 = \{v_{11} = v_1 + v_5 - v_7,\ v_{12} = v_6,\ v_2 = v_4 - 1,\ v_3 = v_4 - 1,\ v_8 = 0.5v_{10} + 0.5,\ v_9 = 0.5v_{10} + 0.5\}$. Fig.2 shows an example that belong to $L(\tilde{R}_2)$, given an alphabet of graphical primitives $\{\uparrow a, \nearrow b, \rightarrow c, \searrow d, \downarrow e\}$.

Theorem 1. *The Augmented Regular Expressions does not describe all the CSLs.*

Proof. A counterexample is the language $L_1 = \{a^k \mid k = 2^i \wedge i \geq 1\}$, which is known to be context-sensitive [12]. L_1 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_1). Hence, there is no ARE $\tilde{R} = (R, V, \mathcal{T}, \mathcal{L})$ such that $\mathcal{L}$ can represent the constraint $v_1 = 2^i \wedge i \geq 1$ for $R(V/*) = a^{v_1}$. □

The context-sensitive language $\{a^k \mid k$ is a prime $\}$ is another counterexample. 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 CSL $L_2 = \{xx \mid x \in (0+1)^+\}$ that corresponds to the pattern language xx over the binary alphabet $\Sigma = \{0, 1\}$, where the variable x stands for any string in Σ^+ [3]. 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, the ARE $(0^{v_1}1)^{v_2}0^{v_3}(0^{v_4}1)^{v_5}0^{v_6}$ with $\{v_5 = v_2;\ v_6 = v_3;\ v_4 = v_1\}$ is able to describe L_2.

Theorem 2. *The Augmented Regular Expressions does cover all the pattern languages, but the size of an ARE describing a pattern language over Σ is exponential in $|\Sigma|$.*

Proof. Let p be a pattern language over $\Sigma = \{a_1, ..., a_m\}$ $(m \geq 2)$ including some finite number of variables $\{x_1, ..., x_l\}$ $(l \geq 0)$. Each variable x_i $(1 \leq i \leq l)$ can be represented by an RE $R_\Sigma = (a_1 + ... + a_m)^*$. By applying repeatedly $(P+Q)^* = (P^*Q)^*P^*$, 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 language p can be stated by letting the star instances of the AREs $\tilde{R}'_{i1}$ be independent and defining a set $\mathcal{L}$ 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 does not cover the languages represented by AREs. For example, the language of rectangles $L(\tilde{R}_1)$ and the CFL $\{0^{v_1}1^{v_2}0^{v_3} \mid v_2 = v_1 + v_3\}$ cannot be described by any pattern language.

3 String Recognition Through AREs

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 $s \in L(\tilde{R})$.

3.1 Parsing Strings by REs to Build the Star Instances

Two algorithms for *unambiguous* RE parsing have been reported [1] 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 set of star instances $SI_s(V)$. The processing of the input string is divided in two phases: the *recognition* and *construction* phases. The first algorithm, with a time complexity of $O(|s| \cdot |R|)$, 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 thus, 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 second algorithm is a more efficient parsing method, that can be run if the unambiguous RE R has been obtained from an equivalent DFA A (by

applying a DFA-to-RE mapping [1] based on Arden's algorithm [11]). This will usually be the case if R has been inferred from examples. This algorithm uses, besides the DFA A, some of the REs α_{ij}^{l} yielded by Arden's algorithm [2] and the *skeleton*[3] of R.

The key point is that A (instead of R) is used for *recognition* $O(|s|)$, and that the path of visited states guides the *construction* of the star instances structure for the input string s. There are two achievements that permit to reduce the time complexity of the *construction* phase too. 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 [1]. 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 $|skel(R)| \leq |R|$.

3.2 Constraint Satisfaction.

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).

In order to fulfil a constraint of an ARE, it is first required that the involved star variables must share a common structure of instances for the given string s, i.e. the number of instances of each one of them must be the same, and the corresponding instance values can be grouped, one for each variable, in rows, one row for each cycle of the instances of a common ancestor. At first, it would seem that the set of related star variables should be brothers in $\mathcal{T}$ (their father being the common ancestor). However, it should be noted that if the values of the instances of a certain son are always constant for each instance of its father, then a unique value may be associated with it and so, regarding the instance list structure, the son may be promoted to a lower level in the tree. In such a case, we say that, with respect to the promoted son, the father is a *degenerated* ancestor.

---

[2] Let $A = (\Sigma, Q, \delta, q_0, F)$ be a DFA, where Σ is an alphabet, $Q = \{q_0, ..., q_{n-1}\}$ is a finite set of states, $q_0 \epsilon Q$ is the initial state, $F \subseteq Q$ is a set of final states, and δ is a state transition function. Let us assume that an arbitrary order $<$ is established among states (except that the first state is q_0). Then, α_{ij}^{l} is an RE that denotes the set of strings that take the DFA from state q_i to state q_j without passing through a state q_k with $k < l$.

[3] The *skeleton* of an RE R 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 [1].

This promotion process may continue until a *non-degenerated* ancestor is found or a default node (e.g the root node r or a selected ancestor that is shared with other star variables) is reached as *housing* ancestor. 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 fit in the same list structure of the degenerated ancestor.

Moreover, even if a common *housing* ancestor is not found, a set of star variables may meet 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 be 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 $v_4 = v_1$ may be met because v_5 and v_2 can be the housing ancestors of v_4 and v_1, respectively, for a string s, and $v_5 = v_2$ may also be met. Now, let us summarize formally the preceding explanation.

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

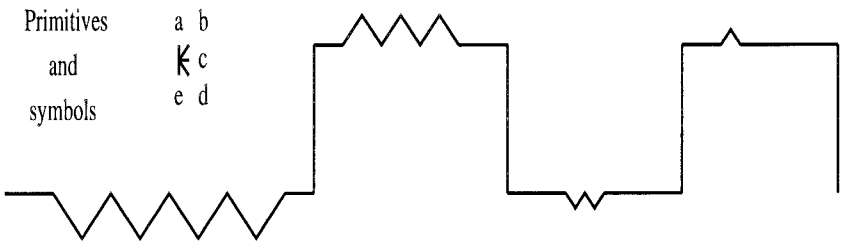
ii) the linear relation l_i is met by the corresponding instances of $\{v_i^{dep}, v'_1, ..., v'_{k_i}\}$.

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 $1 \leq i \leq nc$. Next, an algorithm for constraint testing is described, that evaluates the predicate $satisfy(SI_s(V), \mathcal{L})$ and 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.

In order to test l_i the instances of the dependent star variable v_i^{dep}, obtained in the parse of s, are analysed. If there is no actual instance of v_i^{dep}, the constraint is considered to be met. Otherwise, the deepest common ancestor v_c of the star variables $\{v_i^{dep}, v'_1, ..., v'_{k_i}\}$, i.e. the first common ancestor going from each of these nodes to the root of $\mathcal{T}$, is selected as candidate to common housing ancestor. To verify 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'_j) has a housing ancestor (say v_{hj}) that is deeper than v_c and the equation $v_{hj} = v_{hi}$ 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 case of a star variable of an ARE having always a constant value ($v_i^{dep} = a_{i0}$), no matter its level in $\mathcal{T}$, its

housing ancestor will be the root node, and obviously, all of its 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 v_c 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.2. Given the constraints $\mathcal{L}_2$ and the star instances displayed for the string $s1$, the algorithm would set v_{13} as housed descendent of the root node r, and the rest of star variables of V_2 as housed descendents of v_{13}. In the main loop, the six constraints of $\mathcal{L}_2$ would be checked. The first one, $v_{11} = v_1 + v_5 - v_7$, would lead to the successful test of the system $A \cdot X = B$ shown in Fig.3. The rest of constraints would be verified similarly. Hence, the string s_1 of Fig.2 would be accepted as belonging to $L(\tilde{R}_2)$.



$$s_1 = c^5 d^3 b^3 d^3 b^3 d^3 b^3 d^3 b^3 c^3 a^{10} c^3 b^2 d^2 b^2 d^2 b^2 d^2 c^5 e^{10} c^6 dbdbc^8 a^{10} c^4 bdc^{10} e^{10}$$

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

$$SI_{s_1}(v_{13}) = (\ (2)^{\{-1,-1\}}\ )$$

$$SI_{s_1}(v_1) = (\ (5\ 6)^{\{1,1\}}\ ) \qquad SI_{s_1}(v_7) = (\ (3\ 4)^{\{1,1\}}\ )$$

$$SI_{s_1}(v_5) = (\ (3\ 8)^{\{1,1\}}\ ) \qquad SI_{s_1}(v_{11}) = (\ (5\ 10)^{\{1,1\}}\ )$$

$$SI_{s_1}(v_6) = (\ (10\ 10)^{\{1,1\}}\ ) \qquad SI_{s_1}(v_{12}) = (\ (10\ 10)^{\{1,1\}}\ )$$

$$SI_{s_1}(v_4) = (\ (4\ 2)^{\{1,1\}}\ ) \qquad SI_{s_1}(v_{10}) = (\ (3\ 1)^{\{1,1\}}\ )$$

$$SI_{s_1}(v_2) = (\ (3\ 3\ 3\ 3)^{\{1,1\}}\ (1\ 1)^{\{1,2\}} \qquad SI_{s_1}(v_8) = (\ (2\ 2\ 2)^{\{1,1\}}\ (1)^{\{1,2\}}$$

$$SI_{s_1}(v_3) = (\ (3\ 3\ 3\ 3)^{\{1,1\}}\ (1\ 1)^{\{1,2\}} \qquad SI_{s_1}(v_9) = (\ (2\ 2\ 2)^{\{1,1\}}\ (1)^{\{1,2\}}$$

Fig. 2 *An example of pattern recognized by the ARE $\tilde{R}_2$ with its corresponding star instances.*

1st cycle of instance $v_{13} = 2$ in s_1
2nd cycle of instance $v_{13} = 2$ in s_1

$$\begin{bmatrix} & v_1 & v_5 & v_7 \\ 1 & 5 & 3 & 3 \\ 1 & 6 & 8 & 4 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} v_{11} \\ 5 \\ 10 \end{bmatrix}$$

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).*

4 Conclusions

The *Augmented Regular Expressions* (AREs) permit to describe a class of context-sensitive languages (CSLs) capable of expressing multiple and complex constraints (e.g. planar shapes with symmetries). The recognition of a string as belonging to the language described by an ARE, which is based on parsing by the underlying RE and testing the constraints, is efficient, to the contrary of CSG parsing. Moreover, AREs provide a compact and intelligible representation of the associated languages. It has been shown that AREs cover all the pattern languages [3], but the size of an ARE describing a pattern language is exponential in the number of alphabet symbols. It should be remarked that the ARE representation may be extended in several ways. For example, the definition of non-linear constraints could be allowed (e.g. quadratic equations); however, this would highly complicate the ARE learning procedure [2]. To cope with noisy data, a robust recognizer should use an error-correcting regular parser and a tolerant constraint checker, which could be based on correlation and linear regression (instead of strict linear equations).

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*, Universitat Politecnica de Catalunya, Barcelona, Spain (1995).
2. R. Alquezar and A. Sanfeliu: Learning of context-sensitive languages described by augmented regular expressions. *Proc. 13th Int. Conf. on Pattern Recognition*, Aug.1996, Vienna, Austria (1996).
3. D. Angluin: Finding patterns common to a set of strings. *J. Comput. System Science* **21**, 46-62 (1980).
4. H.Bunke and A.Sanfeliu (eds): *Syntactic and Structural Pattern Recognition: Theory and Applications*, World Scientific (1990).
5. E. Tanaka: Theoretical aspects of syntactic pattern recognition. *Pattern Recognition* **28**, 1053-1061 (1995).
6. A. Salomaa: *Formal Languages*, Academic Press, New York (1973).
7. W.A. Woods: Transition networks grammars for natural language analysis. *CACM* **13**, 591-606 (1970).
8. S.M. Chou and K.S.Fu: Inference for transition network grammars. *Proc. Int. Joint Conf. on Pattern Recognition, 3*, CA, 79-84 (1976).
9. A. Marron and K. Ko: Identification of pattern languages from examples and queries. *Information and Computation* **74**, 91-112 (1987).
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, 16-24 (1994).
11. Z. Kohavi: *Switching and Finite Automata Theory*, (2nd edition). Tata McGraw-Hill, New Delhi, India (1978).
12. J.E. Hopfcroft and J.D. Ullman: *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, Reading MA (1979).