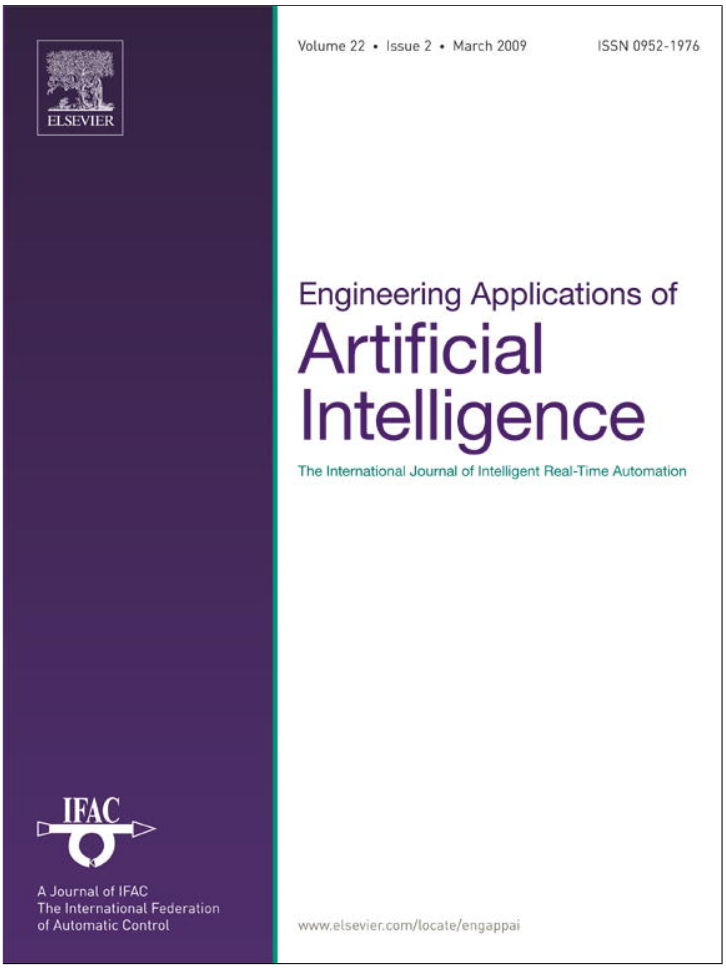

Engineering Applications of Artificial Intelligence 22 (2009) 192–200

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai

From controlled dynamical systems to context-dependent grammars: A connectionist approach

J.F. Martins [a], J.A. Dente [b], A.J. Pires [a], R. Vilela Mendes [c,d,*]

[a] CTS/UNINOVA, Campus da FCT/UNL, Monte de Caparica, 2829-516 Caparica, Portugal
[b] DEEC—Energia, Instituto Superior Técnico, Av. Rovisco Pais 1, 1049-001 Lisboa, Portugal
[c] Centro de Fusão Nuclear, Instituto Superior Técnico, Av. Rovisco Pais 1, 1049-001 Lisboa, Portugal
[d] CMAF, Complexo Indisciplinar, Universidade de Lisboa, Av. Gama Pinto 2, 1649-003 Lisboa, Portugal

ARTICLE INFO

Article history:
Received 23 April 2007
Received in revised form
9 May 2008
Accepted 13 May 2008
Available online 13 August 2008

Keywords:
Formal languages
Context-dependent grammars
Grammatical inference
Learning automata
Networks

ABSTRACT

Formal language techniques may be used to study controlled dynamical systems and lead to a formulation in terms of context-dependent grammars. Using automata as basic units we obtain an on-line implementation of the grammars. The implementation has a hierarchical structure, the lowest level dealing with the identification of alphabet symbols (terminal and non-terminal), and the other with the establishment of the grammar productions. The method provides an automatic construction of the automata network from the experimental data. In particular, the parallel-processing capabilities of automata networks improves the performance of grammar-based models for control and anomaly detection in electromechanical devices of industrial relevance. One such example is presented.

1. Introduction

Formal language theory may be used to model not only autonomous but also controlled dynamical systems (Martins et al., 2001). In both cases, the dynamical system is considered a language-generating source and a grammatical inference algorithm extracts the grammar productions. In an autonomous system, the nature of the grammar depends on the complexity of the system, regular languages being in general sufficient for practical modeling purposes. In contrast, in controlled systems, the need to distinguish between internal state symbols and control symbols leads naturally to the use of context-dependent grammars.

The advantage of language modeling over other information-processing techniques lies in the fact that the structure of the productions is not postulated in advance. Therefore, no prior knowledge is assumed about the dynamical laws. The productions are established from the words generated by the source and, in the end, the resulting grammar becomes a model of the dynamical system (Fu and Booth, 1975). A particular situation where the flexibility of language modeling is quite appropriate is when the system has different memory ranges in different regions of its state space. This is the situation that requites in mathematical modeling variable-length Markov processes (Bühlmann and Wyner, 1999) or chains with complete connections (Berbee, 1987; Iosifescu, 1990). The language modeling adjusts nicely to the variable memory length needed in each region by varying the length of its productions accordingly.

Electromechanical drives, to be found almost everywhere in industrial applications, are nonlinear controlled dynamical systems for which qualitatively different behavior may occur in distinct working domains. Hence, they are good examples of dynamical systems with a large spectrum of distinct dynamical laws requiring a flexible modeling approach.

Let Eq. (1) define a controlled system, $\mathbf{x}\in\Re^n$ being a state-space variable and $\mathbf{u}\in\Re^p$ and $\mathbf{y}\in\Re$ the input (control) and output variables. $\Phi_i:\Re^n\times\Re^p\to\Re^n$ $(i=1,\ \ldots,\ k)$ and $\Psi:\Re^n\to\Re$ are nonlinear functions, each Φ_i representing different behaviors in distinct input/output subspaces:

$$\begin{cases} \dot{\mathbf{x}}(t)=\Phi_1(\mathbf{x}(t),\mathbf{u}(t)), & \{\mathbf{x},\mathbf{u}\}\in V_1,\\ \vdots & \\ \dot{\mathbf{x}}(t)=\Phi_k(\mathbf{x}(t),\mathbf{u}(t)), & \{\mathbf{x},\mathbf{u}\}\in V_k,\\ \mathbf{y}(t)=\Psi(\mathbf{x}(t)). & \end{cases} \tag{1}$$

A learning-by-examples modeling technique must be able to identify the distinct transfer functions $\Phi_i(\cdot)$. The formal language

* Corresponding author at: CMAF, Complexo Indisciplinar, Universidade de Lisboa, Av. Gama Pinto 2, 1649-003 Lisboa, Portugal. Tel.: +351 914739442.
E-mail address: vilela@cii.fc.ul.pt (R. Vilela Mendes).

0952-1976/$ - see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.engappai.2008.05.005

approach addresses this problem by allowing for a large class of production laws in the algorithm.

A dynamical system may be considered a linguistic source, generating words in a language with a particular grammar G. The grammar is a four-tuple:

$$G = (P, \Sigma_N, \Sigma_T, S). \tag{2}$$

The *terminal alphabet* Σ_T codes the output of the system, the *non-terminal alphabet* Σ_N contains the symbols used to generate the patterns and the *start symbol* S is a special non-terminal symbol used to start the generation of words. The set of *productions* P is a set of rules (in the form $\alpha \rightarrow \beta$, α and β being strings) that determines the generation of words (Saloma, 1985).

The alphabets are established by coding the variables of the dynamical system. In our formalism, coding the output variable—y—defines the terminal alphabet and coding the input variable—u—the non-terminal alphabet:

$$\begin{aligned} y &\underset{\text{coding}}{\longrightarrow} y_j \in \Sigma_T, \\ u &\underset{\text{coding}}{\longrightarrow} U_j \in \Sigma_N, \end{aligned} \tag{3}$$

A learning algorithm must establish a relation between the alphabets (input and output information) in order to generate terminal words representing the evolution of the output variable. *Type-p productions* have the general form:

$$y_1 \ldots y_p U_k \rightarrow y_1 \ldots y_p y_{p+1} \delta. \tag{4}$$

The sequence $y_1 \ldots y_p$, of length p, consists of terminal symbols, U_k is a non-terminal symbol, y_{p+1} a terminal symbol, and δ a special non-terminal symbol that can be replaced by any non-terminal symbol. The p terminal symbols in the left hand side of a *type-p production* represent the coding of the past evolution of the output variable.

The left hand side of the production containing at least one non-terminal symbol classifies this grammar as context-dependent in the Chomsky hierarchy (Chomsky, 1965). Notice that the context-dependent nature of the language is not directly related to the language complexity of the dynamical system, considered as an isolated entity. When isolated the system may even generate a regular language, but the need to code the external control by a non-terminal alphabet leads to a context-dependent formulation. In this sense, the language complexity of the system arises from the undetermined nature of the external control.

Words generated by the source (the dynamical system) are mapped into symbol sequences (in the set of grammar productions). This mapping, performed by a set of edit operations, may be framed as a search problem. Since the dimension of the search space grows with the dimension of the problem, computational resources increase rapidly (Thomason, 1990). To avoid this combinatorial explosion, several authors have proposed to represent logic structures by connectionist structures (Barnden and Pollack, 1991; Gallant, 1993; Honavar and Uhr, 1994; Pinker and Prince, 1988). Notice that connectionist and linguistic systems, as long as they relay on Turing models of computation, can be considered equivalent, in the sense of that whatever is computable in one framework must also be computable in the other (Martins and Vilela Mendes, 2001).

The use of automata for language representation together with distributed and parallel-computing techniques allows for a considerable reduction of the processing time. Automata theory (Tsetlin, 1962) is extremely rich and several techniques have been used in a large variety of application domains such as pattern recognition, encryption algorithms, handwriting recognition, optical character recognition, data compression, operating system analysis, electronic dictionaries, modeling, etc.

An automaton (Narendra and Thathachar, 1989; McClelland and Rumelhart, 1988) is a five-tuple:

$$\{\Theta, A, B, F(.,.), H(.,.)\}, \tag{5}$$

in which the input and the current state determine the next state as well as the output.

Θ is a set of internal states, A a set of outputs, B a set of input actions, $F(.,.) : \Theta \times B \rightarrow \Theta$ a function that maps the current state and input into the next state, and $H(.,.) : \Theta \times B \rightarrow A$ a function that maps the current state and input into the output.

In Section 2, the basic ideas of the connectionist approach to language modeling are presented. Sections 3 and 4 deal with the two hierarchical levels of the approach, the alphabet level and the production level. In Section 5, the grammatical inference algorithm is applied to an experimental system. Section 6 ends with some conclusions and remarks.

2. The connectionist approach

Alphabets and productions are the key elements of a language. Terminal alphabet symbols build the words, the non-terminal alphabet is used to generate patterns and the set of productions rules the pattern generation process. A language can be processed using a connectionist model, which integrates symbolic processing and automata theory. Automata elements, processing elementary information, serve as building blocks for the rules of the higher order connectionist structure.

In practical control applications, fixed sampling intervals are usually considered. The output is usually a nonlinear function of the input and of the l past values of the output variable:

$$y_{k+1} = h(y_k \ldots y_{k-l}, u_k), \tag{6}$$

l denoting the memory range. This dependence on the past is going to be modeled by *type-l productions*.

An automata structure (shown in Fig. 1) is used to construct the input–output grammar. In the first level (*alphabet level*), the automata identifies the alphabet symbols. In the second order level (*production level*), the productions of the grammar are established. All knowledge about the controlled system being obtained from the input/output information, both levels are inferred from the experimental data.

In the past, several authors have shown that context-dependent languages may be implemented by recurrent neural networks (Boden and Wiles, 2002; Giles et al., 1999; Wiles et al., 2000). Given the particular form of our language formulation for controlled systems, a simpler automata network achieves the same purpose.

3. Alphabet level

The purpose of the first connectionist structure (the "Symbol Identification" box in Fig. 1) is to identify the alphabet symbols (terminal and non-terminal). Finite state automata are used as basic units in this network. These units have activation levels and pass signals along weighted connections. Each unit computes its output by summing the weighted inputs and feeding the result to a nonlinear output function.

Internal states, as well inputs and outputs of the automaton, are two-valued: $\Theta, B, A \in \{-1, +1\}$. Their evolution is determined by Eqs. (7) and (8), B_i denoting an automaton input, w_i an input weight and $\rho(\cdot)$ is the Heaviside function.

Learning consists of changing the connection strengths to reduce the difference between the system alphabet symbols and

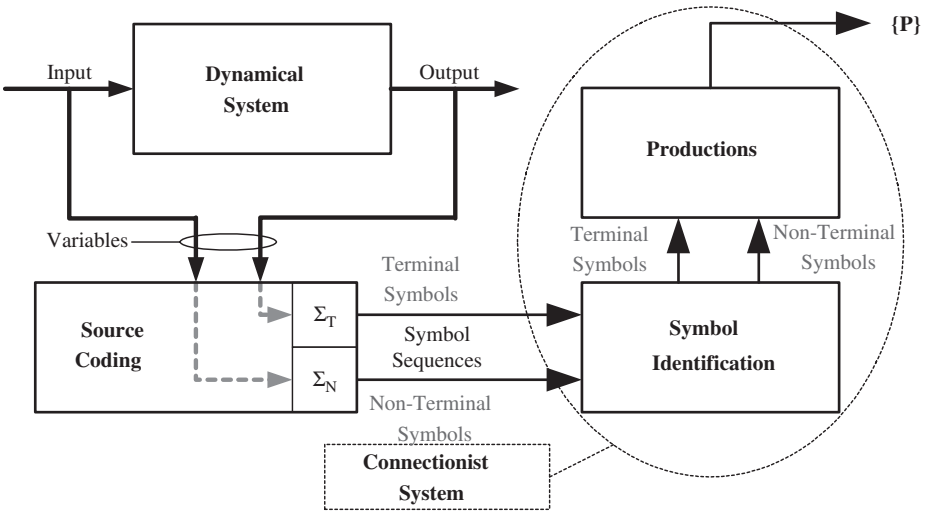

Fig. 1. Hierarchical automata structure for grammar reconstruction.

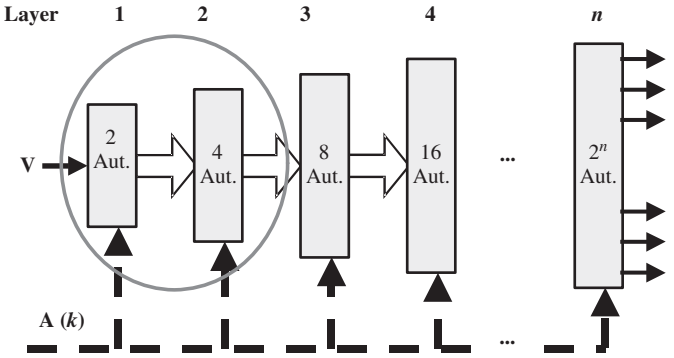

Fig. 2. Automata network for symbol identification.

their representation:

$$\Theta(t+1) = \rho\left(\sum_{i=1}^{2} w_i B_i(t) - 1\right), \tag{7}$$

$$A(t) = \Theta(t). \tag{8}$$

Assuming an n-binary (± 1) coding of each alphabet symbol, the automata are organized as an n-layer feed forward network (Fig. 2), with 2^i automata in layer i, n being the number of bits used to code the alphabet symbols. V, in Fig. 2, is an activation signal with value +1. A maximum of 2^n symbols per alphabet are considered with a total of $\sum_{i=1}^{n} 2^i$ automata in the network. After the learning process, the number of automata can be reduced.

The alphabet symbols, denoted $A(k)$, where k labels the code sequence, are fed into the network in a sequential way, to all layers. A word with m symbols is mapped into a $m \times n$ coded sequence. Every time k is a multiple of n (corresponding to an alphabet symbol), one of the network outputs should be +1. The absence of a +1 output means a failure to identify that particular symbol.

Fig. 3 shows an automata network, for a 3-bit coding of the alphabet symbols. Each output of the network corresponds to the recognition of a unique code sequence, denoting an alphabet symbol. The weights are established by a learning process. A weight w_{ij} with even i can only take positive values, and a

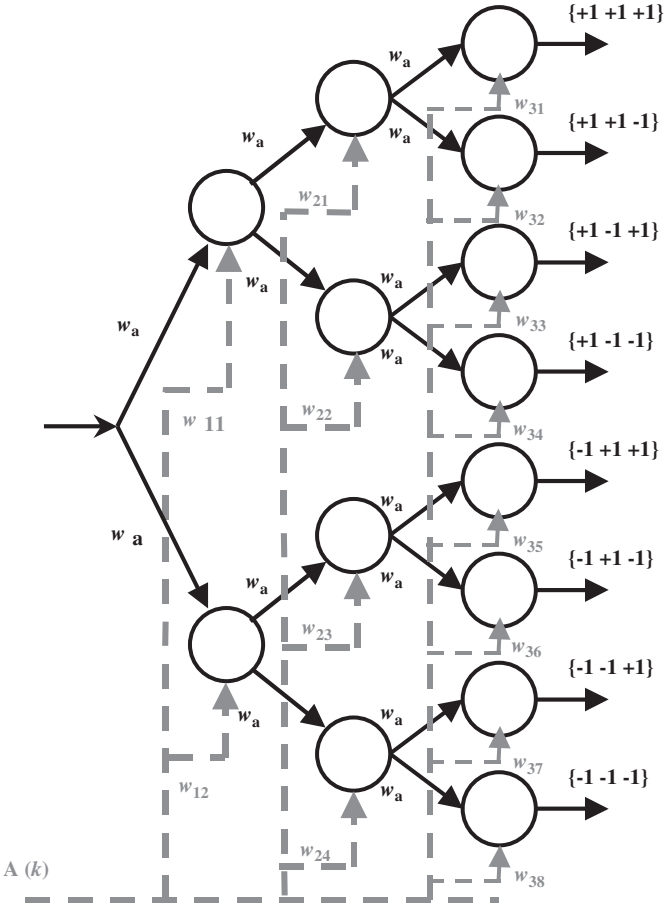

Fig. 3. Automata network for symbol identification—the first three layers.

weight w_{ij} with odd i can only take negative values

$$\begin{cases} w_{ij} \in \left[0, +\frac{1}{2}\right], & \text{for odd } i, \\ w_{ij} \in \left[-\frac{1}{2}, 0\right], & \text{for even } i, \\ w_a = +\frac{1}{2}. \end{cases} \tag{9}$$

Evolution of each automaton output depends on its internal state and on the set of inputs. From the network topology (Figs. 2 and 3) it follows that each unit reacts positively to a code if and only if the previous unit had a positive reaction one-time step before. In this way, as the symbols are fed to the network, the recognition process proceeds as a wave throughout the network.

The full network can identify 2^n alphabet symbols. However, the dynamical system language may not use every symbol in the alphabet. The learning algorithm allows for the reduction of the network size, to represent only the symbols that are actually used by the dynamical system language. During the learning process, the input/output information (terminal and non-terminal symbols) is presented to the network and the automata weights evolve according to the learning rule (10). Each unit learns (learning meaning the evolution of a weight into the desired state: even weights into +1/2 and odd weights into −1/2) if and only if the previous unit has successfully learned in the previous time step.

In the learning rule (10), $A(k)$ denotes the coded symbol and $q(\cdot)$ the function (11). Two distinct learning rates are considered, one for learning—γ_{lrn}—and another for forgetting—γ_{frg}.

After the learning process, the network is pruned, eliminating all units for which the corresponding upstream unit has not identified any code (units with weights in the range—$0.4<w_{ij}<+0.4$):

$$\begin{cases} w_{ij}(k+1)=\begin{cases} q(w_{ij}(k)+\gamma_{\mathrm{lrn}}A(k)) & \text{if } w_{ij}(k)\mathrm{A}(k)>0 \\ q(w_{ij}(k)-\gamma_{\mathrm{frg}}A(k)) & \text{if } w_{ij}(k)\mathrm{A}(k)<0 \end{cases} & \text{only if } B_{i-1}(k)>0, \\ w_{1j}(k+1)=\begin{cases} q(w_{1j}(k)+\gamma_{\mathrm{lrn}}A(k)) & \text{if } w_{1j}(k)\mathrm{A}(k)>0 \\ q(w_{1j}(k)-\gamma_{\mathrm{frg}}A(k)) & \text{if } w_{1j}(k)\mathrm{A}(k)<0 \end{cases} & \text{only if } V>0, \end{cases} \tag{10}$$

Table 1
Coding of a terminal alphabet

Symbol	Coding		
a	−1	−1	−1
b	−1	−1	+1
c	−1	+1	−1
d	−1	+1	+1
e	+1	−1	−1
f	+1	−1	+1
g	+1	+1	−1
h	+1	+1	+1

$$q(\upsilon)=\begin{cases} -\frac{1}{2}, & \upsilon\leqslant -\frac{1}{2}, \\ \upsilon, & -\frac{1}{2}<\upsilon<\frac{1}{2}, \\ \frac{1}{2}, & \upsilon\geqslant\frac{1}{2}. \end{cases} \tag{11}$$

To illustrate the learning process, consider the 8-symbol terminal alphabet coded by three binary codes, shown in Table 1. This leads to an automata network of 14 units.

A sequence of 1666 terminal symbols was considered (obtained from a dynamical system) with the first 333 being completely random, and the remaining 1333 having the following distribution: 25% are random, 50% represent the symbol 'f' and 25% the symbol 'e'. Fig. 4 shows the evolution of the weights in the learning process according to the learning rule (10). One sees that the $(i+1)$th layer weights evolve only after the ith layers have learned. In the end of the learning process, only the weights w_{11} (first layer), w_{22} (second layer) and w_{33} and w_{34} (third layer) have reached $\pm 1/2$. All remaining units are pruned, leading to the network shown in Fig. 5.

Once the learning process is complete, the resulting network can be used as a symbol recognizer. Having information about the symbols (terminal and non-terminal), the symbol identification networks are used as basic units for the higher order connectionist structure (production level).

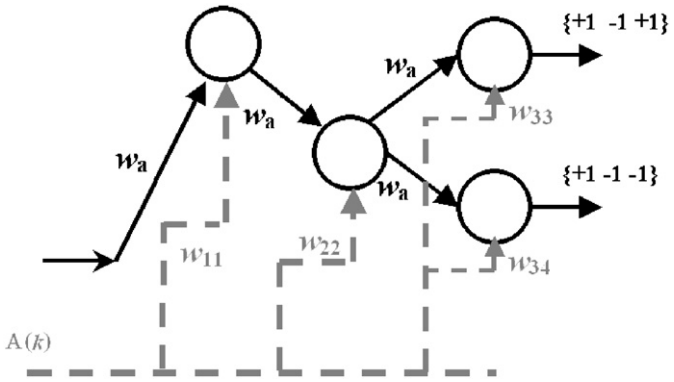

Fig. 5. Automata network for symbol identification, after learning and pruning.

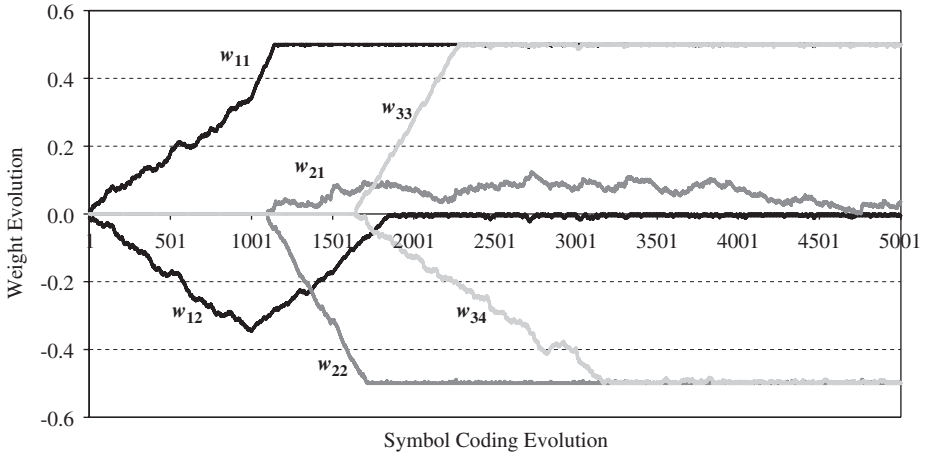

Fig. 4. Weight evolution in an automata network for symbol identification: an example.

4. Production level

This second level of the connectionist structure generates the grammar productions in the controlled dynamical system. It is also built automatically by a learning-by-examples process.

The lower order automata network defined before is used as a two-state unit that recognizes alphabet symbols. Each symbol has its recognition automaton, as shown in Eq. (12):

$$\begin{aligned} &A(t) \in \Sigma_T, \\ &B(t) \in \{0,1\}, \\ &\Theta(t+1) = \begin{cases} 1 & \text{if the symbol is identified,} \\ 0 & \text{if the symbol is not identified,} \end{cases} \\ &B(t) = \Theta(t). \end{aligned} \tag{12}$$

A grammar production may be of several distinct types depending on the behavior and memory range of the dynamical system. *Type-p productions* are substitution rules of the form (13) coding for the evolution of the output variable, as a function of its past p values and the value of the control variable U (Martins et al., 2002):

$$y_1 \ldots y_p U_k \rightarrow y_1 \ldots y_p y_{p+1} \delta. \tag{13}$$

The high order network (the "Productions" box in Fig. 1) uses several recognition automata to infer the language productions from the behavior of the dynamical system. Each automaton is a (p_{max}+2)-layers feedforward network, p_{max} denoting the maximum order allowed for the productions. A four-layer network can establish *type*-0, *type*-1 and *type*-2 *productions* using the first three layers, the fourth layer being the output layer. Inputs are sequences of terminal and non-terminal symbols (words generated by the controlled system evolution), and the output is the next terminal symbol in the word.

As an example, the network shown in Fig. 6 represents the *type*-2 *production* abA→abbδ. The first three nodes are recognition automata and the last one is an output automaton. The first recognition automaton recognizes the non-terminal symbol 'A' and other two the terminal symbols 'b' and 'c'. If the sequence of symbols (coming from the dynamical system) is the correct one, the output automaton produces the terminal symbol 'b'. In this feedforward network, all weights have unit values and the terminal word is sequentially fed to every network layer, following the temporal evolution. A valid *type-p production* eliminates $\sum_{i=1}^{p_{max}-p}(\text{num}(\Sigma_T))^i$ *type-p*+1 *productions*, thus reducing the number of connections between layers. p_{max} denotes the maximum type of allowed productions.

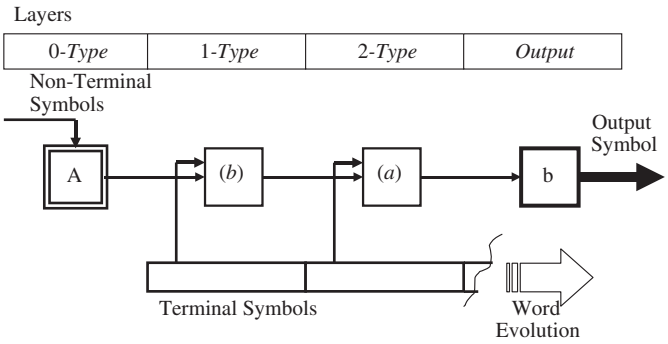

Fig. 6. Automata network representing the *type*-2 *production* abA→abbδ.

The learning process uses *reinforcement learning* laws (14), which evolve the weights towards unity every time the example word corresponds to the production represented by those weights (Sutton and Barto, 1998). γ denotes the learning rate. Furthermore, a *type-n*+1 *production* is learned only when there is no possibility to learn *type-n productions*. The learning algorithm initiates the learning from the *lower-type productions* towards *higher-type productions*:

$$\begin{cases} w_{pij}(t+1) = \gamma + (1-\gamma) w_{pij}(t) & \text{if terminal symbol} = \text{automata}_j, \\ w_{pij}(t+1) = (1-\gamma) w_{pij}(t) & \text{if terminal symbol} \neq \text{automata}_j. \end{cases} \tag{14}$$

For each type of production, the learning algorithm considers the following steps:

1. All connections between each layer and the output layer are established.
2. The weights are randomly initialized.
3. The weights evolve according to the reinforcement learning rule (14).
4. Only the weights that reach unit value are considered as valid. All other connections are pruned.
5. After the weight evolution stops, the learning proceeds to the next layer (next type of production).

As a simple illustrative example consider a controlled dynamical system generating a language with the following alphabets: Σ_N = {A,B} and Σ_T = {a, b, c}. Let the input/output evolution of the system be the following:

$$\left\{ \begin{matrix} \text{B} & \text{B} & \text{B} & \text{B} & \text{B} & \text{B} & \text{B} & \text{B} & \text{B} & \text{B} & \text{B} & \text{B} & \text{A} & \text{A} & \text{A} & \text{A} & \text{B} & \text{B} & \text{B} & \text{B} & \text{B} & \text{B} & \text{B} & \text{B} & \text{A} & \text{A} & \ldots \\ \text{a} & \text{b} & \text{b} & \text{c} & \text{a} & \text{b} & \text{b} & \text{c} & \text{a} & \text{b} & \text{b} & \text{c} & \text{a} & \text{a} & \text{a} & \text{a} & \text{b} & \text{b} & \text{c} & \text{a} & \text{b} & \text{b} & \text{c} & \text{a} & \text{a} & \text{a} & \ldots \end{matrix} \right\}.$$

5. Grammatical inference

Grammatical inference refers to the learning of a grammar based on a set of sample words generated by a linguistic source (Fu and Booth, 1975).

A grammatical inference procedure must be established in order to construct the production level networks from the words generated by the dynamical system. The only a priori information is the terminal and non-terminal alphabets (learned by the lower order symbol network) and the maximum allowed type of productions.

The maximum production type is either guessed from qualitative knowledge of the memory range of the system or adjusted at a later step, in view of the actual performance of the control system.

The first line is a sequence of commands and the bottom line the resulting sequence of outputs. Assuming a maximum of *type*-2 *productions*, Fig. 7 shows the evolution of the learning process, for each network layer (from *type*-0 *productions* to *type*-2 *productions*), as the input/output information arrives from the dynamical system. The evolution of the production weights is depicted as a function of the learning iterations.

Fig. 7 illustrates the following learning stages:

1. Establishment of all possible *type*-0 *productions*;
2. Learning of the *type*-0 *production* weights;
3. *type*-0 productions learned;
4. Establishment of all possible *type*-1 *productions*, maintaining the previously learned *type*-0 *productions* (the established

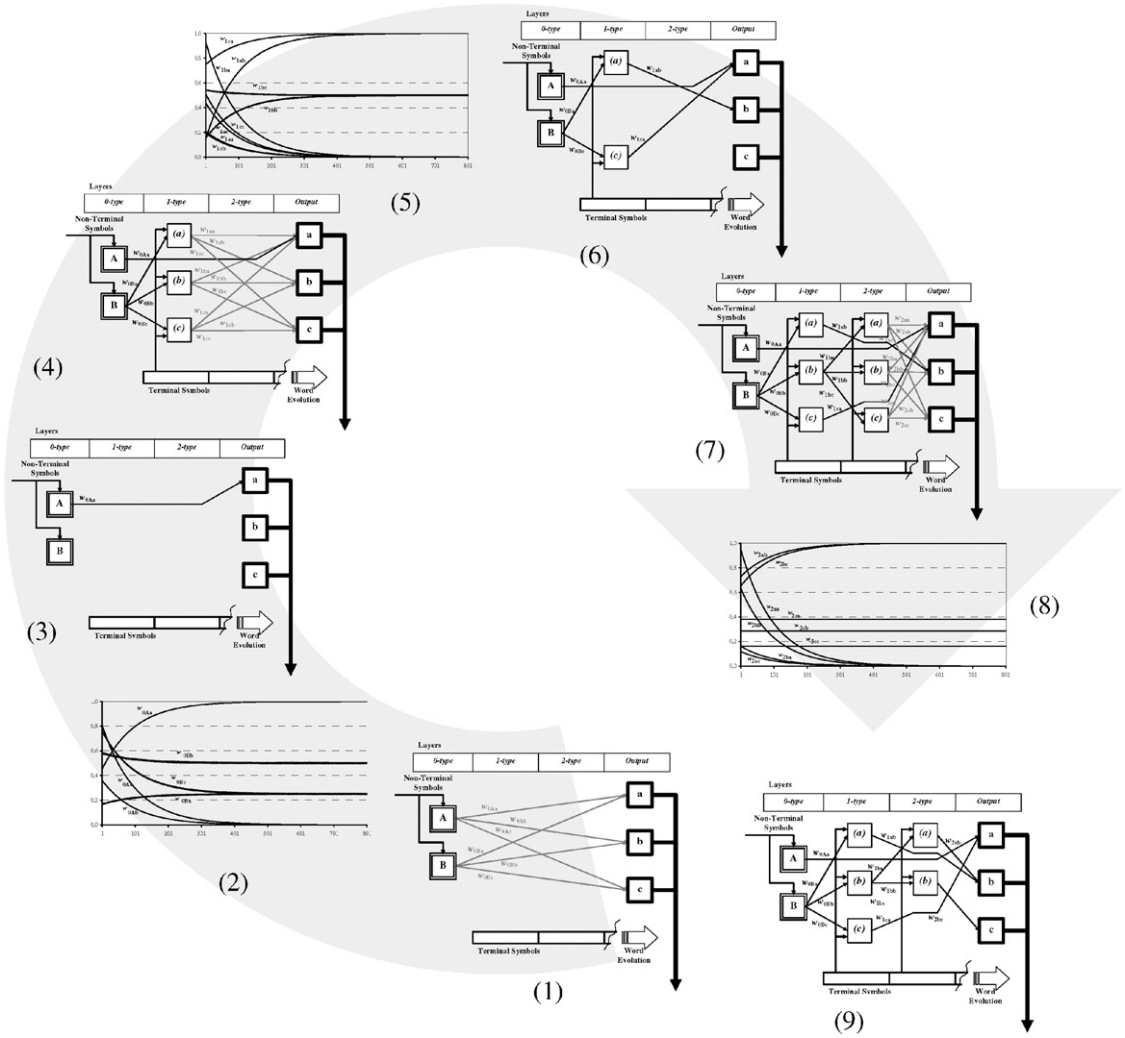

Fig. 7. Learning evolution in a higher order automata network.

type-0 *productions* inhibit nine of the eighteen possible *type*-1 *productions*);

5. Learning of the *type*-1 *production* weights;
6. *type*-1 productions learned;
7. Establishment of all possible *type*-2 *productions*, maintaining the previously learned *type*-0 and *type*-1 *productions* (the obtained *type*-0 and *type*-1 *productions* inhibit eighteen of the twenty-seven possible *type*-2 *productions*);
8. Learning of the *type*-2 *productions* weights;
9. *type*-2 productions learned.

The high order network that is obtained represents the following set of productions:

$$P = \{A \to a\delta, aB \to ab\delta, cB \to ca\delta, abB \to abb\delta, bbB \to bbc\delta\}.$$

Given any controlled system input/state evolution, the above methodology allows for the automatic build up of an automata network representing the set of *type-p productions* that define the dynamical evolution of the system.

The complexity of the learning process is a function of the size of the alphabets and of the learning rate. The larger the alphabet, that is, the more detail is considered in the modeling process, the larger is the number of automata, thus increasing the complexity of learning. On the other hand, the learning process may be sped up by considering larger learning rates. However, if the learning rate is too large, the evolution of the weights may display oscillations, which is not desirable. Fig. 8 shows the learning speed (number of iterations) for different learning rates, in the previous example.

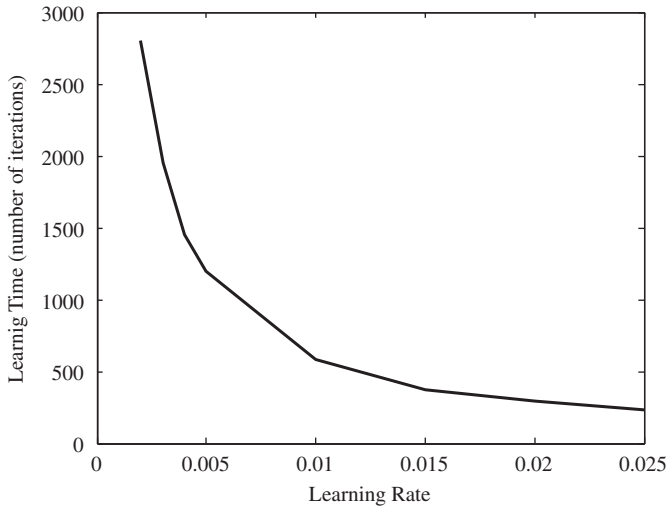

Fig. 8. Learning time (number of iterations).

Because too smaller a learning rate hinders the time performance of the system, the learning rate should be chosen only sufficiently slow to insure a smooth evolution of the learned weights.

6. An experimental application

As a practical application of the method, we have considered an electromechanical drive system composed of a V/Hz-based speed command, a power inverter and an induction motor with a variable mechanical load (Fig. 9), the rotor speed being the output variable.

The electrical drive is a nonlinear system which, neglecting the power inverter internal dynamics, may be described by the Eq. (15), in a dq frame, oriented with the stator voltage vector and assuming $u_{sd} = u_s$ and $u_{sq} = 0$. u_{sd} and u_{sq} denote the applied voltage to the motor, ψ_{ds} and ψ_{qs} denote the motor stator fluxes, i_{ds} and i_{qs} the motor stator currents, all in the dq frame. ω denotes the rotor speed, T the load torque, f the stator frequency, α the V/Hz coefficient, R_s the stator resistance, τ_r the rotor time constant, σ the magnetic dispersion coefficient, L_s the stator self-inductance coefficient, J the inertia coefficient and B the friction coefficient. The stator current frequency (Hz) is the system's input (imposed by the power inverter), and the rotor speed (rad/s) the system's output:

$$\begin{cases} \frac{d\psi_{ds}}{dt} = \alpha f - R_s i_{ds} + f\psi_{qs}, \\ \frac{d\psi_{qs}}{dt} = -R_s i_{qs} - f\psi_{ds}, \\ \frac{di_{ds}}{dt} = \frac{1}{\sigma L_s}\alpha f - \frac{1}{\tau} i_{ds} + (f-\omega) i_{qs} + \frac{1}{\sigma\tau_r L_s}\psi_{ds} + \frac{1}{\sigma L_s}\omega\psi_{qs}, \\ \frac{di_{qs}}{dt} = \frac{1}{\tau} i_{qs} - (f-\omega) i_{ds} + \frac{1}{\sigma\tau_r L_s}\psi_{qs} + \frac{1}{\sigma L_s}\omega\psi_{ds}, \\ \frac{d\omega}{dt} = -\frac{1}{J}(i_{qs}\psi_{ds} - i_{ds}\psi_{qs}) - \frac{B}{J}\omega - \frac{1}{J}T. \end{cases} \quad (15)$$

In our experiment we have used a squirrel cage induction motor with the following nominal characteristics: $P_N = 2200\,W$ (electrical power), $U_N = 380\,V$ (input voltage), $I_N = 5.2\,A$ (input current), $\cos\varphi = 0.84$ (power factor), $n_N = 1410\,rpm/min$ (mechanical speed) and the following parameters: $\alpha = 5.4\,V/Hz$, $R_s = 3.2\,\Omega$, $\tau_r = 0.1303\,s$, $\sigma = 0.0641$, $L_s = 0.43\,H$, $J = 0.012\,kg\,m^2$, $B = 0.0014\,N\,ms$.

Fig. 10 shows the formal language modeling results considering two distinct alphabets. The first has a quantification interval of 0.05 'per unit' (pu) of the input and output nominal values (50 Hz for the input and 147.6 rad/s for the output), yielding 6-symbol

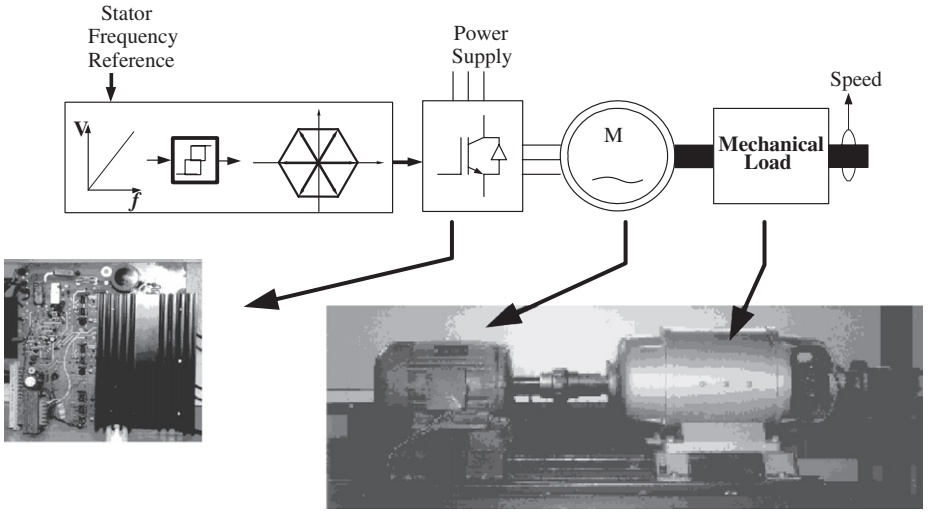

Fig. 9. Induction motor electromechanical drive.

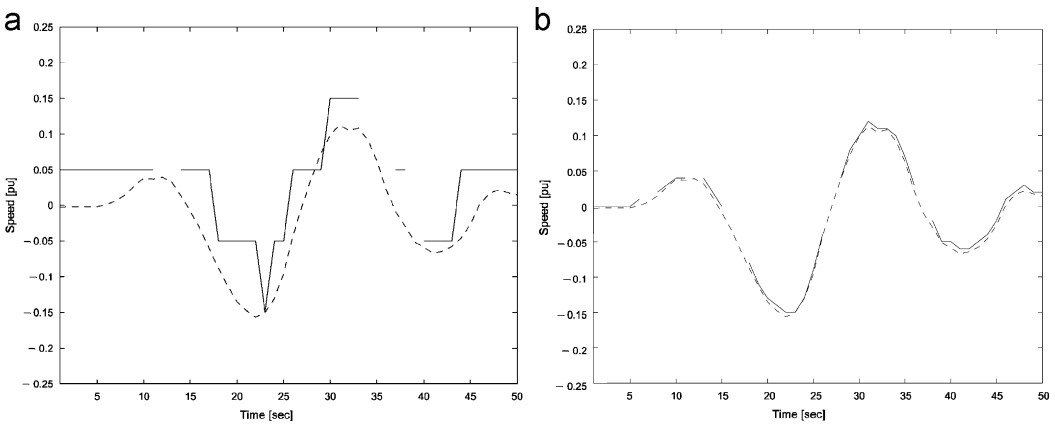

Fig. 10. Electromechanical drive modeling results: (a) 6-symbol alphabet and (b) 60-symbol alphabet.

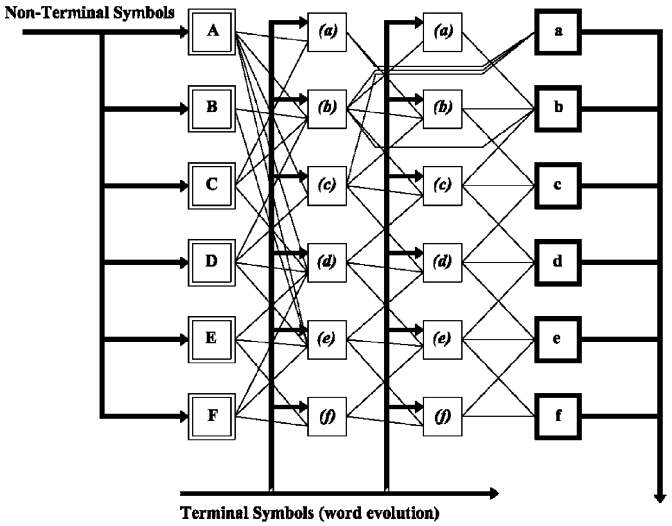

Fig. 11. Layer structure of the 6-symbol automata network.

terminal and non-terminal alphabets. In the second alphabet, the quantification interval is five times higher, that is, 0.01 'per unit'. This yield a 60-symbol terminal and non-terminal alphabets.

For the first alphabet, the inferred grammar has 36 productions (0 type-0 productions, 4 type-1 productions and 32 type-2 productions) and a higher order automata network with 18 recognition automata. Fig. 11 presents the layer structure of this automata network. With the second alphabet, the inferred grammar has 183 productions (5 type-0 productions, 156 type-1 productions and 22 type-2 productions) and a higher order automata network with 180 recognition automata.

Apart from the quantification error, the recognition results may be considered satisfactory, since the evolution of the drive speed and grammar results are similar. The quantitative modeling performance improves for the second grammar, since the difference between the measured drive speed and the grammar results is smaller. However, considering the qualitative recognition of the drive speed as the main modeling feature, the results are satisfactory in both cases. The limited alphabet of the first grammar is not a drawback in a qualitative recognition process.

However, for some control purposes, the qualitative modeling may not be sufficient. In this case, the larger alphabet should be considered, increasing the number of symbols and productions. The increased complexity of the grammar increases the processing time. This is the main reason to use the parallel-processing capabilities of an automata network.

Fig. 12 shows the experimental dynamical performance, when a sinusoidal reference is considered. The drive speed reference is the continuous black line and the actual speed is the continuous gray line. It shows how an accurate language modeling leads to a good control performance of this drive.

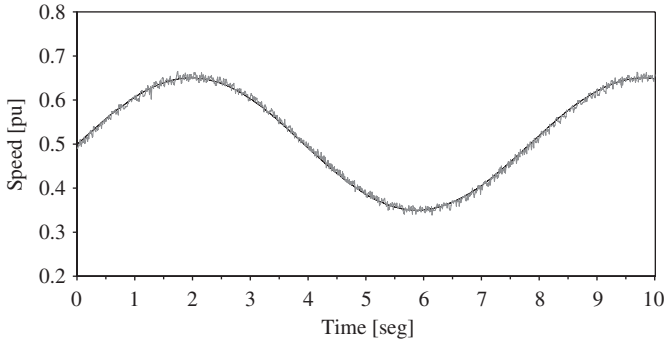

Fig. 12. Formal language electromechanical drive control.

7. Conclusions

1. In this paper a connectionist approach was developed to construct context-dependent grammars representing controlled dynamical systems. The automata network is used to obtain combinatorial optimization and fast temporal processing. The structure is organized in two levels: lower order symbol level and higher order production level. Both levels are established by a learning-by-examples algorithm. The first level identifies the alphabet symbols. The second level identifies the grammar productions. After the learning process, the connectionist structure itself is a model of the controlled dynamical system.
2. The reinforcement-learning concept that is used provides robustness, both for symbol identification and for production inference. The practical feasibility of the method has been shown by an application to a nonlinear system of industrial relevance.
3. When a reliable analytical model for the controlled system is available, other control algorithm might perform as accurately or faster than the one discussed in this paper. The main advantage of a language inference approach is the fact that the algorithm itself constructs a model for the system with very few a priori assumptions. Combining learning automata with a formal language methodology, the present approach may be more efficient than, for example, neural networks when different regions in state space have different dynamical nature. This is because regions of different dynamical complexity would be represented by productions of different types.
4. The grammatical inference approach seems also appropriate for fault detection and system diagnosis of nonlinear complex dynamical systems (Martins et al., 2001) (Inagaki et al., 2007). The only requirement is to have access to the input–output data strings, no explicit model of the system structure being required. Control of the performance and computational effort is easily achieved, since the user can select the quantization level and the maximum level of *type-p productions*.

References

Barnden, J.A., Pollack, J.B., 1991. Advances in Connectionist and Neural Computation Theory. Vol. 1. High Level Connectionist Models. Ablex, Norwood.
Berbee, H., 1987. Chains with complete connections: uniqueness and Markov representation. Probability Theory and Related Fields 76, 243–253.
Boden, M., Wiles, J., 2002. On learning context free and context sensitive languages. IEEE Transactions on Neural Networks 13 (2), 491–493.
Bühlmann, P., Wyner, A., 1999. Variable length Markov chains. The Annals of Statistics 27, 480–513.
Chomsky, N., 1965. Aspects of the Theory of Syntax. MIT Press, Cambridge, MA.
Fu, K.S., Booth, T.L., 1975. Grammatical inference: introduction and survey—Part I. IEEE Transactions on SMC 5 1, 95–111.
Gallant, S.I., 1993. Neural Network Learning and Expert Systems. MIT Press, Massachusetts.
Giles, C.L., Omlin, C.W., Thornber, K.K., 1999. Equivalence in knowledge representation: automata, recurrent neural networks, and dynamical fuzzy systems. Proceedings of the IEEE 87 (9), 1623–1640.
Honavar, V., Uhr, L., 1994. Artificial Intelligence and Neural Networks: Steps Towards Principled Integration. Academic Press, San Diego.
Inagaki, S., Hayashi, K., Suzuki, T., 2007. Fault detection and diagnosis of manipulator based on probabilistic production rule. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences 2007 E90-A (11): 2488–2495.
Iosifescu, M., 1990. Dependence with Complete Connections and its Applications. Cambridge University Press, Cambridge.

Martins, J.F., Vilela Mendes, R., 2001. Neural networks and logical reasoning systems: a translation table. International Journal of Neural Systems 11, 179.
Martins, J.F., Pires, A.J., Vilela Mendes, R., Dente, J.A., 2001. Language identification of controlled systems: modeling, control and anomaly detection. IEEE Transactions on Systems, Man and Cybernetics (Part C) 31 (2), 234–242.
Martins, J.F., Pires, A.J., Vilela Mendes, R., Dente J. A., 2002. Formal language based learning algorithm for electrical drives control. In: Conference on Simulated Evolution and Learning-SEAL'02. Singapore.
McClelland, J.L., Rumelhart, D.E., 1988. Explorations in Parallel Distributed Processing. Bradford Books/MIT Press, Cambridge. MA.
Narendra, K.S., Thathachar, M., 1989. Learning Automata: An Introduction. Prentice-Hall International.
Pinker, S., Prince, A., 1988. On language and connectionism: analysis of a parallel distributed processing model of language acquisition. Cognition 28.
Saloma, A., 1985. Computation and Automata. University Press, Cambridge.
Sutton, R.S., Barto, A.G., 1998. Reinforcement Learning: An Introduction. MIT Press.
Thomason, M.G., 1990. Introduction and overview. In: Bunke, H., Sanfeliu, A. (Eds.), Syntactic and Structural Pattern Recognition: Theory and Applications. World Scientific, London, pp. 119–144.
Tsetlin, M., 1962. On the behaviour of finite automata in random media. Automatic Remote Control 22, 210–1219.
Wiles, J., Blair, A.D., Boden, M., 2000. Representation beyond finite states: alternatives to push-down automata. In: Kolen, J.F., Kremer, S.C. (Eds.), A Field Guide to Dynamical Recurrent Networks. IEEE Press, New York, pp. 129–142.