Grammatical Inference as a Tool for Constructing Self-learning Syntactic Pattern Recognition-Based Agents

Janusz Jurek

IT Systems Department, Jagiellonian University
Straszewskiego 27, 31-110 Cracow, Poland
jjurek@wzks.uj.edu.pl
http://www.wzks.uj.edu.pl/ksi/jwj

Abstract. Syntactic pattern recognition-based agents have been proven to be a useful tool for constructing real-time process control intelligent systems. In the paper the problem of self-learning schemes in the agents is discussed. Learning capabilities are very important if practical applications of the agents are considered, since the agents should be able to accumulate knowledge about the environment and flexible react to the changes in the environment. As it is shown in the paper, the learning scheme in the agents can be based on a suitable grammatical inference algorithm.

1 Introduction

Syntactic pattern recognition-based agents have been proven to be a useful tool for constructing real-time process control intelligent multi-agent systems [6]. In [2,3] we describe an example of such system: a multi-agent system implemented for the purpose of the on-line monitoring, diagnosing, and controlling of a very complex industrial-like equipment (a high energy physics detector). The system had to perform four different tasks:

- monitoring and identifying the equipment behaviour,
- analysing and diagnosing the equipment behaviour,
- predicting consequences of the equipment behaviour,
- controlling, i.e. taking proper actions (eg. setting operating modes for equipment components).

The decision about the multi-agent architecture of the system has been caused by the fact that only fully-decentralised, distributed intelligent system consisting of autonomous components acting in a parallel way can meet the hard real-time constraints: the necessity of performing four mentioned above tasks simultaneously and on-line with respect to hundreds of elementary components of the equipment. The system has been built of different types of agents correspondingly to the layers of analysis (i.e. the whole equipment layer, modules (subdetectors) layers, and components layers). We have chosen the syntactic pattern recognition

M. Bubak et al. (Eds.): ICCS 2008, Part III, LNCS 5103, pp. 712–721, 2008.
© Springer-Verlag Berlin Heidelberg 2008

approach as a base for constructing lowest layer agents of reactive type. They are responsible for monitoring and recognition of the behaviour of particular components.

The main part of syntactic pattern recognition agents is a parser. The parser performs syntax analysis of the symbolic description of a component behaviour. The knowledge about the possible behaviour of a component is remembered in the agent in the form of a formal string grammar. The grammar is used to construct a control table for the parser. The advantages of using the syntactic pattern recognition approach consist in very good computational properties (it is possible to implement an advanced parser of the linear, $O(n)$, computational complexity) and the capability to analyse and recognise even exceedingly complex patterns representing a process in time series. These two advantages make the approach better in case of constructing real-time process control intelligent systems than many other computationally inefficient classical artificial intelligence methods [10].

However, after several years of practical experiments with syntactic pattern recognition-based agents we have identified one important problem of the implementation of such agents: the difficulty in providing a self-learning feature. Learning capabilities are very important if we consider practical applications of the agents, since the agents should be able to accumulate knowledge about the environment (i.e. the possible behaviour of a component which is being monitored) and flexible and autonomously react to the changes in the environment. The learning scheme in the agents can be based on a suitable grammatical inference algorithm (i.e. the algorithm of automatical definition of a grammar from the sample of a pattern language). Unfortunately grammatical inference is still one of the main open issues in the syntactic pattern recognition area.

The research into grammatical inference of string grammars started almost forty years ago. The first, basic results were published in the seventies. During next years the methods of grammatical inference were improved in the aspects of the "quality" of generated grammar, and the computational efficiency. Although many different results have been already achieved (especially in case of *regular* grammars) [4,5], there is still lack of models of inferencing *context-sensitive* grammars, that is grammars of big generative/discriminative power. Most methods already developed are based on the identification of structures in words in a sample and the definition of the sequence of the structures. The dependencies between *numbers of symbols* in words are not examined. This results from the fact that the discriminative power of grammars we are able to inferred is too weak to reflect such dependencies. The only one practical method that allows to consider such quantitative information (in the context of syntactic pattern recognition) has been developed by Alquézar and Sanfeliu [1]. They have proposed a new type of grammars (ARE grammars, characterised by a big discriminative power) and a proper grammatical inference algorithm. Although the method is very innovative, it has some disadvantages. In particular, the time complexity of the grammatical inference is exponential, which makes the model unsuitable for the application in a real-time environment.

In the paper we present the recent results of the research into constructing a self-learning algorithm for syntactic pattern recognition-based agents. In section 2 we describe the general scheme of the agents. Section 3 contains the definitions corresponding to the string grammars which we use as a knowledge base in the agents. The description of the model of the self-learning of the agents (via grammatical inference) is included in section 4. Section 5 contains presentation of some experimental results. Conclusions are included in the final section.

2 Syntactic Pattern Recognition-Based Agents

There are three main elements of syntactic pattern recognition-based agents: a knowledge base (in the form of a formal grammar), a parser (a control table for the parser is constructed on the base of a grammar being the knowledge base), and a self-learning module (based on a grammatical inference algorithm used to update the grammar). The general scheme of a syntactic pattern recognition-based agent is shown in Fig. 1.

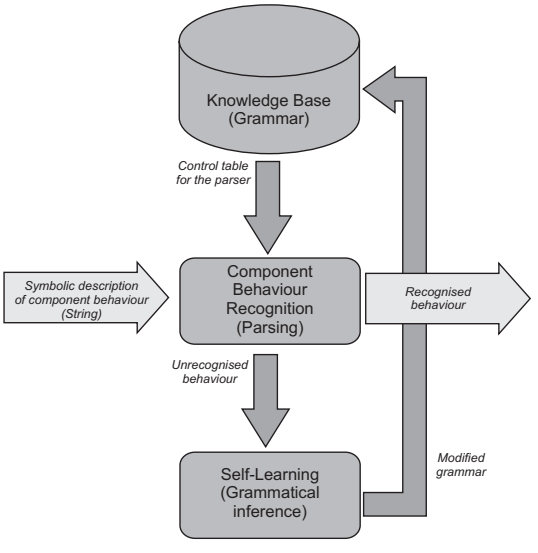

Fig. 1. A general scheme of a syntactic pattern recognition-based agent

A syntactic pattern recognition-based agent is of reactive type: it should be able to respond timely to changes in its environment (changes in the behaviour of a component which is being monitored) and to acquire knowledge about the environment. The agent receives symbolic data (a string of symbols) representing characteristics of a component behaviour. The analysis of the behaviour is made by the parsing of a received string. The results are outputted to the higher layer agents with different architecture which are beyond the scope of the paper [2].

When a received string cannot be recognised by the parser, then the self-learning activity in the agent should take place. The inability to recognise the string pattern means that the current grammar should be replaced by a broaden one which generates the missing pattern. Self-learning can be done via grammatical inference. The result is an updated grammar which becomes a new knowledge base.

The most important decision concerning the implementation of the agent has been the choice of GDPLL(k) grammars [8] as a formal string grammar which is used to store knowledge about a component behaviour and to construct a control table for the parser. The definition of GDPLL(k) grammar and its brief characteristics are included in next section.

3 Quasi Context Sensitive GDPLL(k) Grammars

Let us introduce two basic definitions [3,8].

Definition 1. A *generalised dynamically programmed context-free grammar* is a six-tuple: $G = (V, \Sigma, O, P, S, M)$, where V is a finite, nonempty alphabet; $\Sigma \subset V$ is a finite, nonempty set of terminal symbols (let $N = V \setminus \Sigma$); O is a set of basic operations on the values stored in the memory; $S \in N$ is the starting symbol; M is the memory; P is a finite set of productions of the form: $p_i = (\mu_i, L_i, R_i, A_i)$ in which $\mu_i : M \longrightarrow \{TRUE, FALSE\}$ is the predicate of applicability of the production p_i defined with the use of operations ($\in O$) performed over M; $L_i \in N$ and $R_i \in V^*$ are left- and right-hand sides of p_i respectively; A_i is the sequence of operations ($\in O$) over M, which should be performed if the production is to be applied. □

Definition 2. Let $G = (V, \Sigma, O, P, S, M)$ be a generalised dynamically programmed context-free grammar. The grammar G is called a *Generalised Dynamically Programmed LL(k) grammar*, GDPLL(k) grammar, if: 1) the LL(k) condition of deterministic derivation is fulfilled, and: 2) the number of steps during derivation of any terminal symbol is limited by a constant. □

As we mentioned in Section 2, GDPLL(k) grammars have been chosen to store knowledge about components behaviour in the agents. The choice has resulted from two main features of GDPLL(k) grammars:

1. GDPLL(k) grammars are characterised by very good discriminative properties since their generative power is stronger than context-free grammars and "almost" as big as context-sensitive grammars [8]. It means that the grammars can store knowledge even about highly complicated patterns representing a component behaviour.
2. It is possible to implement a parser for the languages generated by GDPLL(k) grammars which performs syntax analysis in the linear time. This feature is crucial if we consider embedding of the agents in real-time applications.

In case of GDPLL(k) grammars these two contradictory requirements are balanced. However, as we noticed in Section 2, there is one more requirement concerning a class of grammars to be used as a knowledge base in the agents: the ability to construct a grammatical inference algorithm (preferably of polynomial time complexity) needed to provide self-learning feature in the agents. Grammatical inference is a well-known open problem in the syntactic pattern recognition area. It is especially difficult in case of grammars stronger than than context-free ones. We have done the research into the problem of the grammatical inference of GDPLL(k) grammars for a few years [7,9]. The algorithms developed during last years have been mostly dedicated to particular applications. Recently, we have achieved results that allow us to implement a self-learning system which can be used more generally: in many possible practical applications.

4 Grammatical Inference

Firstly, let us present main definitions needed to discuss the grammatical inference algorithm [9].

Definition 3. Let A is a set of all (terminal) symbols which appear in the sample, Z set of integer variables. *Polynomial specification of a language* is of the form: $L_p(A,Z) = S_i^{p_j(n_k)}$ where: p_j is a polynomial of a variable $n_k \in Z$; variable n_k can be assigned only values greater or equal 1; S_i, called *polynomial structure*, is defined in a recursive way:

a) $S_i = (a_{i_1}...a_{i_r})$, where $a_{i_j} \in A$. (S_i is a *basic* polynomial structure), or:
b) $S_i = (S_{i_1}^{p_{i_1}(n_{i_1})}...S_{i_r}^{p_{i_r}(n_{i_r})})$, where S_{i_k} is defined as in a) or b). (S_i is a *complex* polynomial structure).

Extended polynomial specification of a language $L_{ep}(A,Z)$ is defined in the following way:

a) $L_{ep}(A,Z) = L_p(A,Z)$ or:
b) $L_{ep}(A,Z) = L_{ep}^1(A_1,Z_1)\ L_{ep}^2(A_2,Z_2)\ ...\ L_{ep}^z(A_z,Z_z)$, or:
c) $L_{ep}(A,Z) = L_{ep}^1(A_1,Z_1)\ +\ L_{ep}^2(A_2,Z_2)\ +\ ...\ +\ L_{ep}^z(A_z,Z_z)$,

if proper conditions of the unambiguity of the specification are fulfilled in case of all Z_i and A_i (see: [9]). □

Example 1. Let $A = \{a,b,c\}$ be a set of terminal symbols, $Z = \{n,m\}$ be a set of integer variables. Then: $L_p(A,Z) = ((ab)^{2n+1}c^{n^2}(ba)^{2m})^{n+2}(ab)^{m^3}$ is an example of polynomial specification of a language. The polynomial structures in the specification are the following:

$S_1 = ((ab)^{2n+1}c^{n^2}(ba)^{2m})^{n+2}(ab)^{m^3}$	$p_1 \equiv 1$
$S_{1_1} = (ab)^{2n+1}c^{n^2}(ba)^{2m}$	$p_{1_1}(n) = n+2$
$S_{1_2} = ab$	$p_{1_2}(m) = m^3$
$S_{1_{1_1}} = ab$	$p_{1_{1_1}}(n) = 2n+1$
$S_{1_{1_2}} = c$	$p_{1_{1_2}}(n) = n^2$
$S_{1_{1_3}} = ba$	$p_{1_{1_3}}(m) = 2m$

Extended polynomial specification of a language is built as a catenation (case "b" in Definition 3) or alternatives (case "c" in Definition 4) of polynomial specifications. Then: $L_{ep}(A, Z) = ((ab)^{2n+1} + cb^{2m})ab^{n^2}$ can be an example of extended polynomial specification of a language. □

Our approach to the inferencing GDPLL(k) grammars is based on the following method. The input is the sample of a language (containing a new string pattern representing an unrecognised behaviour). We divide the inference process into two phases. The first one is responsible for extraction of the features of the sample and generalisation of the sample. The result is the extended polynomial specification of a language. In the second phase, a GDPLL(k) grammar is generated on the basis of the extended polynomial specification of the language.

Now, let us present the first phase: the method of creating the extended polynomial specification of a language from the sample. Let us notice that our method is canonical. It can be used for the detection of basic features only, and for relatively simple generalisation of the sample. The canonical character of the method may broaden the area of its possible applications.

Let the sample of a language be a set of m words: $Sample = \{w_1, ..., w_m\}$. Let Σ be a common alphabet for all words in the sample. Words belonging to the sample will be written in the form: $w = a_1^{n_1}\ a_2^{n_2}\ ...\ a_k^{n_k}$.

Definition 4. Let us consider two words: $w_1 = a_1^{p_1}\ a_2^{p_2}\ ...\ a_k^{p_k}$ and $w_2 = b_1^{q_1}\ b_2^{q_2}\ ...\ b_l^{q_l}$. We say that words w_1 i w_2 are *sequentially equivalent*, if the following conditions are fulfilled: $k = l$ and $a_i = b_i$ for $i = 1, ..., k$. *The template of an extended polynomial specification* is an expression constructed accordingly to the definition of extended polynomial specification L_{ep}, which is built with sets of sequentially equivalent words instead of polynomial specifications L_p. □

Example 2. Let $A = \{a, b, c\}$ be a set of terminal symbols. Two words: $a^2b^4a^2c^7$ and $a^6b^2a^3c^5$ are sequentially equivalent, since both words are built with the same sequence of different symbols: a, b, a, c. □

The algorithm of the construction of the extended polynomial specification of a language can be divided into three steps. In the first step the template of the extended polynomial specification is built. In the second step each set of sequentially equivalent words (in the template) is generalised in the form of a polynomial specification. In the last step the extended polynomial specification is generated (by applying the results of step 2 to the template).

Algorithm 1. The algorithm of the construction of the extended polynomial specification of the language from $Sample = \{w_1, ..., w_m\}$ consists of three steps:

Step 1. We construct the template of the extended polynomial specification. Let $V\{v_1, ..., v_n\}$ be a set which is a variable in the template of an extended polynomial specification which is being constructed, where $v_1, ..., v_n$ are words or fragments of words belonging to *Sample*. Initially, let the template of an

extended polynomial specification be the variable: $V\{w_1, ..., w_m\}$. While there is such a set V in the template, which is not the set of sequentially equivalent words, we transform the template according to two rules. The first one is: $V\{w_{s_1}a_1w_{r_1}, ..., w_{s_n}a_nw_{r_n}\} := V\{w_{s_1}, ..., w_{s_n}\}\ V\{a_1w_{r_1}, ..., a_nw_{r_n}\}$, if w_{s_i} are sequentially equivalent (for $i = 1, ..., n$) and there are such symbols a_i and a_j different than the last symbol of w_{s_1} that $a_i \neq a_j$. The second one is: $V\{a_1w_1, ..., a_nw_n\} := V\{a_{1_1}w_{1_1}, ..., a_{1_k}w_{1_k}\} + ... + V\{a_{s_1}w_{s_1}, ..., a_{s_k}w_{s_k}\}$, if for any two symbols $a_{x_i}, a_{x_j} : a_{x_i} = a_{x_j}$, and for any two symbols a_{x_i}, a_{y_j} , where $x \neq y : a_i \neq a_j$. The result is the template in which all V are sets of sequentially equivalent words.

Step 2. We generalise each set of sequentially equivalent words (in the template) in the form of a polynomial specification. Let $P = \{w_1, ..., w_q\}$ be a set of sequentially equivalent words. Let the word $w_i \in P$ be of the form: $w_i = a_1^{n_{i_1}}\ a_2^{n_{i_2}}\ ...\ a_k^{n_{i_k}}$. The canonical algorithm of the construction of polynomial specification $L_p(A, Z)$ for P is the following. For $L_p(A, Z)$ we define: $A = \{a_1, ..., a_k\}$, and $Z = \{x_1, ..., x_k\}$. We assume that $L_p(A, Z) = a_1^{p_1(x_1)}\ ...\ a_k^{p_k(x_k)}$. For each $j = 1, ..., k$ we search for the linear function of variable x_j, which describes dependencies between numbers n_{i_j} (where: $i = 1, ..., q$). Let the function be of the form: $p_j(x_j) = d_jx_j + r_j$. Then we look for linear dependencies between $x_j = x_1, ..., x_k$ allowing to reduce Z set. For each pair x_g and x_h (where $g = 1, ..., k$, $h = 1, ..., k$) we check whether $x_h = x_g + s$ for each words in P and a constant value s. If the result of the test is positive, we store a new function for the index h, and we delete variable x_h from Z set. As the outcome of this step we get polynomial specification $L_p(A, Z)$ such that all words from P belong to $L_p(A, Z)$, as well as all other words which are considered to be "similar".

Step 3. We define the extended polynomial specification by inserting polynomial specifications (generated in step 2) to the template of the extended polynomial specification (generated in step 1). □

The algorithm above is of polynomial complexity. The extended polynomial specification being the result of the algorithm is the input for the second phase.

Now, let us present the algorithm of automatic generation of a GDPLL(k) grammar from polynomial specification of a language (as the main algorithm of the second phase). The algorithm can be divided into two steps. In the first step we define a separate grammar for each polynomial specification which is included in the *extended* polynomial specification. In the second step we construct a target GDPLL(k) grammar for extended polynomial specification on the basis of the grammars constructed in the first step.

Algorithm 2. The algorithm of automatic generation of a GDPLL(k) grammar from polynomial specification of a language consists of two steps:

Step 1. We define a separate grammar for each polynomial specification which is included in the extended polynomial specification in the following way. Let $L_p(A, Z)$ be a polynomial specification of a language. We will construct

a GDPLL(k) grammar $G = (V, \Sigma, O, P, S, M)$ generating the language. Let $\Sigma := A$. For each variable $n \in Z$ in $L_p(A, Z)$ we define two variables: v_n and d_n in the grammar memory M. For each polynomial structure S in $L_p(A, Z)$ we define: a nonterminal symbol $X_S \in V$, and two memory variables: c_S and e_S in M. For each $S^{p(n)}$ structure in $L_p(A, Z)$ we define a set of productions in P in the following way. We use memory variables to implement "loops" during derivation. Current value of n (in exponent expression) is stored in v_n. Variable c_S is a counter of repetitions of S structure. Variable e_S contains the current evaluation of exponent expression for S structure. Boolean variable d_n stores information whether n is fixed or not (i.e. if the value of n has been determined before). Operations on the memory defined for each production are responsible for "programming" proper number of repetitions during derivation. (Formal definition of the set of production is described in [7]).

Step 2. We construct a target GDPLL(k) grammar for extended polynomial specification in the following way. Let $L_{ep}(A, Z)$ be an extended polynomial specification of language L. Let us assume that $L_{ep}(A, Z)$ is built by m polynomial specifications $L_p^i(A^i, Z^i)$ (we will use L_p^i to denote $L_p^i(A^i, Z^i)$). Let us assume that for each L_p^i a grammar $G^i = (V^i, \Sigma^i, O, P^i, S^i, M^i)$ has been constructed (in step 1) in such a way that: $N^i \cap N^j = \emptyset$ and $M^i \cap M^j = \emptyset$ for $i \neq j$. Now we define: $V := \bigcup V^i$, $\Sigma := \bigcup \Sigma^i$, $P := \bigcup P^i$ and $M := \bigcup M^i$, for $i := 1, ..., m$. Then for each extended polynomial specification L_{ep}^j (where $j \geq m$) included in the construction of the specification $L_{ep}(A, Z)$ we define: if $L_{ep}^j \neq L_p^j$, then a new symbol E^j is added to N; if $L_{ep}^j = L_{ep}^{j_1} L_{ep}^{j_2} ... L_{ep}^{j_z}$, then a new production p: $E^j \longrightarrow E^{j_1} ... E^{j_z}$ is added to P, where symbols E^j represent adequate extended polynomial specifications of the language; if $L_{ep} = L_{ep}^1 + L_{ep}^2 + ... + L_{ep}^z$, then z new productions: $E^j \longrightarrow E^{j_1}$, ..., $E^j \longrightarrow E^{j_z}$ are added to P. □

The result of Algorithm 2 is a new (updated) definition of a GDPLL(k) grammar being a new knowledge base for the agent. Both algorithms presented above are of polynomial computational complexity, so the whole grammatical inference process is also computationally efficient (the time complexity of the whole grammatical inference is $O(m^3 * n^3)$, where m is the number of words in a sample, n is the maximum length of a word in a sample).

5 Implementation and Experimental Results

All grammatical inference algorithms presented in previous section (being the realization of the learning self-learning scheme in the syntactic pattern recognition-based agents) have been implemented and tested. The experimental implementation has been prepared in C++ language with the use of the object-oriented approach.

Two aspects of the grammatical inference module functioning have been verified: the correctness of the results of the inference, and the time efficiency.

Table 1. Inference of the grammar generating language: $\{a^n b^n a^n : n = 1, 2, \ldots\}$

Sample	Number of words	Maximum word length	Time (s)
1	3	9	27×10^{-5}
2	7	30	28×10^{-5}
3	10	90	29×10^{-5}
4	50	900	33×10^{-5}
5	100	900	39×10^{-5}
6	1000	900	246×10^{-5}

In order to test the module we have used mainly such samples that describe context-sensitive languages (intuitively: samples in which context dependencies concerning the number of given symbols in a word can be observed).

A typical example of a strong context-sensitive language (a language which cannot be generated by a context-free grammar) is the language: $\{a^n b^n a^n : n = 1, 2, \ldots\}$. The context property of the language can be easily proven on the basis of the well-known pumping lemma. We have prepared a rich set of test data for this language. The set has included samples having from 3 to 1000 words. In all cases grammatical inference module has recognised proper dependencies inside words (the same number of a, b, and c symbols in a word) and dependencies between words (all words are of the same pattern $a^n b^n a^n$). Then a proper GDPLL(k) grammar has been automatically generated.

The results of time tests of the grammatical inference module are shown in Table 1. The tests have been performed on relatively old PC (processor: AMD Athlon XP 2600+ 2.1 GHz, Microsoft Windows XP Professional system).

All the experiments and tests confirm that the algorithms of grammatical inference are of good (polynomial) computational complexity. Usually, execution time is much better than predicted on the base of (pessimistic) estimation $O(m^3 * n^3)$, where m is the number of words in a sample, n is the maximum length of a word in a sample.

6 Concluding Remarks

In the paper we have presented the recent results of the research into construction of syntactic pattern recognition-based agents. Such agents have been designed as a tool for the implementation of real-time process control intelligent multi-agent systems. The process diagnostic and control is a natural application for agents, since process controllers are themselves autonomous reactive systems. On the other hand, the real-time requirements make the construction of the agents particularly difficult [11,12,13].

Although we have proven practical usefulness of the syntactic pattern recognition-based agents by the application in a real-time process control system [2,6], we have observed that there is a significant problem concerning the definition of the universal self-learning method in the agents. Let us notice, that learning capabilities are very important if we consider practical applications of the agents, since they should be able to gain knowledge about the changes in

the environment. In the paper we have shown that the learning scheme in the agents can be based on a grammatical inference algorithm. If a new (unrecognised) pattern of the behaviour in the monitored process appears, an updated grammar (being the knowledge base in the agents) will be generated. We have developed the algorithm that can automatically produce the definition of a very strong (in the sense of discriminative power) quasi-context sensitive grammar. The algorithm is of polynomial computational complexity.

The method presented in the paper still needs a thorough practical evaluation. We are going to test it in the environment of several different applications where the need of a self-learning feature of the agents is particularly noticeable. The discussion of the results will be a subject of further publications.

References

1. Alquézar, R., Sanfeliu, A.: Recognition and learning of a class of context-sensitive languages described by augmented regular expressions. Pattern Recognition 30, 163–182 (1997)
2. Flasiński, M.: Automata-Based Multi-agent Model as a Tool for Constructing Real-Time Intelligent Control Systems. In: Dunin-Keplicz, B., Nawarecki, E. (eds.) CEEMAS 2001. LNCS (LNAI), vol. 2296, pp. 103–110. Springer, Heidelberg (2002)
3. Flasiński, M., Jurek, J.: Dynamically Programmed Automata for Quasi Context Sensitive Languages as a Tool for Inference Support in Pattern Recognition-Based Real-Time Control Expert Systems. Pattern Recognition 32, 671–690 (1999)
4. De La Higuera, C.: Current Trends in Grammatical Inference. In: Amin, A., Pudil, P., Ferri, F.J., Iñesta, J.M. (eds.) SPR 2000 and SSPR 2000. LNCS, vol. 1876, pp. 28–31. Springer, Heidelberg (2000)
5. De La Higuera, C.: A bibliographical study of grammatical inference. Pattern Recognition 38, 1332–1348 (2005)
6. Jurek, J.: Syntactic Pattern Recognition-Based Agents for Real-Time Expert Systems. In: Dunin-Keplicz, B., Nawarecki, E. (eds.) CEEMAS 2001. LNCS (LNAI), vol. 2296, pp. 161–168. Springer, Heidelberg (2002)
7. Jurek, J.: Towards grammatical inferencing of GDPLL(k) grammars for applications in syntactic pattern recognition-based expert systems. In: Rutkowski, L., Siekmann, J.H., Tadeusiewicz, R., Zadeh, L.A. (eds.) ICAISC 2004. LNCS (LNAI), vol. 3070, pp. 604–609. Springer, Heidelberg (2004)
8. Jurek, J.: Recent developments of the syntactic pattern recognition model based on quasi-context sensitive languages. Pattern Recognition Letters 26, 1011–1018 (2005)
9. Jurek, J.: Generalisation of a Language Sample for Grammatical Inference of GDPLL(k) Grammars. In: Computer Recognition Systems 2. Advances in Soft Computing series, pp. 282–288. Springer, Heidelberg (2007)
10. Negnevitsky, M.: Artificial Intelligence. A Guide to Intelligent Systems. Addison-Wesley, Reading (2002)
11. Niederberger, C., Gross, M.: Hierarchical and Heterogenous Reactive Agents for Real-Time Applications. Computer Graphics Forum 22, 323–331 (2003)
12. Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach, 2nd edn. Prentice-Hall, Englewood Cliffs (2002)
13. Soto, I., Garijo, M., Iglesias, C.A., Ramos, M.: An agent architecture to fulfill real-time requirements. In: Proceedings of the Fourth International Conference on Autonomous Agents, Barcelona, Spain, June 03–07, 2000, pp. 475–482 (2000)