CHAPTER 1.1

SYNTACTIC PATTERN RECOGNITION: PARADIGM ISSUES AND OPEN PROBLEMS

Mariusz Flasiński

*IT Systems Department, Jagiellonian University,
ul. prof. St. Łojasiewicza 4, Cracow 30-348, Poland,
Mariusz.Flasinski@uj.edu.pl*

Paradigm issues of syntactic pattern recognition are presented in the chapter. Main open problems are identified and methodological principles are defined with respect to three basic models of this approach. The issues of enhancement of string models' generative power, grammatical inference for tree models, and parsing/induction for graph models are discussed.

1. Introduction

There are two main approaches in the field of pattern recognition: the decision-theoretic approach and the syntactic/structural one. Representing a pattern by a *vector of features* is the main idea of the first approach. Feature vectors representing patterns of a learning set are “placed” in a *feature space* and they are used for partitioning the space into subareas corresponding to classes/categories. Then, various methods (discriminant function-based, probabilistic, artificial neural network-based, etc.) are used for a *classification* of an unknown pattern, i.e. for ascribing it to one of pre-defined classes.

Already at the beginning of the development of the field it turned out that some classes of patterns can be represented better with *structural features* than with feature vectors. This observation ushered in the syntactic approach in the early 1960s. Representing a pattern as a structure in the form of string, tree or graph and a set of structures as a formal language is the main idea of this approach.¹⁻⁵ According to the Chomsky paradigm, such an (infinite) language of structural patterns can be defined with a generator in the form of a *generative grammar*.⁶ Then, a recognition of an unknown pattern represented with a structure can be performed by a *formal automaton*, strictly speaking, by its implementation, namely a *parser*.

The syntactic approach has been applied successfully for such important problems of pattern recognition as, e.g., scene analysis, texture analysis, chemical and biological structure analysis, speech recognition, medical signals analysis, optical character recognition, natural language processing, since its beginning in the early

1960s for nearly three decades. Then, however, its development slowed down, since it has faced certain open problems, such as weak descriptive power of tractable grammars, high computational complexity of parsers, grammatical inference, which are hard to solve.^{5,7}

Indeed, open problems mentioned above have not been solved in a satisfactory way till now. However, in the case of some pattern recognition problems the use of the syntactic approach rather than the decision-theoretic one seems to be the only correct choice from a methodological point of view. Firstly, as we have mentioned above, the syntactic approach prevails over the decision-theoretic one, if patterns cannot be characterized adequately with feature vectors, since they are of the (strong) structural nature. Secondly, sometimes we require that a recognition method not only should classify, in the sense of ascribing an unknown pattern to one of pre-defined classes, but also deliver a description/interpretation of an unknown pattern. (This situation occurs frequently in issues typical for Artificial Intelligence applications and then we talk about *pattern understanding*.⁸) Thirdly, the structural nature of patterns can be complex, in the sense of a structural hierarchy. Then, in the syntactic approach we can represent and analyze a pattern in a hierarchical way. That is, we can decompose a complex pattern into simpler structural subpatterns, these subpatterns can be decomposed into simpler structural subpatterns and so on. At the bottom of such a hierarchy we use pre-defined elementary patterns, called *primitives*.

Paradigm issues which influence further development of syntactic pattern recognition are discussed in the chapter. In the next section three key open problems in the area, namely enhancement of context-free grammars, grammatical inference of tree grammars and constructing efficient graph parsers are identified. These problems are discussed in sections 3, 4, and 5, respectively. Final methodological remarks are included in the last section.

2. Methodological Foundations and Open Problems of Syntactic Pattern Recognition

The theory of formal languages delivers formal tools for syntactic pattern recognition. For its “standard” applications in computer science such as, e.g., programming languages, compilers design formalisms of grammar and automaton are sufficient. This results from the fact that grammar is constructed on the basis of a pre-defined syntax of artefactual language.

However, for pattern languages considered in computer vision their syntax is often not given explicitly. Instead of it a learning set, i.e., a sample set of pattern structures is available. This set, however, is so big that it results in an impossibility of constructing grammar “by hand.” A definition of a grammatical inference (induction) algorithm^a is a solution to the problem. A grammatical induction mod-

^aA grammatical inference algorithm generates a grammar on the basis of a learning set.

Fig. 1. The general scheme of a syntactic pattern recognition system.

ule (cf. Fig. 1) devises a syntactic pattern recognition system with a self-learning mechanism. Since a generation of a grammar is made in the off-line mode, (any) polynomial complexity of this algorithm is sufficient. Let us notice that the experts in the field consider a lack of an induction algorithm a serious limitation of a syntactic pattern recognition model^{5,7,9}. Taking this into account, we can formulate the following principle.

I. *A syntactic pattern recognition model should include the following three components: a generative grammar, a computationally efficient parsing algorithm, and a grammatical induction algorithm of the polynomial time complexity.*

As we have mentioned in the first section, in syntactic pattern recognition an image is decomposed into elementary patterns, called primitives. This decomposition is the result of the image processing phase (cf. Fig. 1). Noise reduction, smoothing, boundary sharpening/accentuation, image restoration, and decomposition/segmentation into primitives are basic operations of this phase. For example, an ECG image (cf. Fig. 2(a)) is filtered (grid lines are to be removed) and smoothed (cf. Fig. 2(b)). Finally, it is decomposed into primitives.

Structural primitives are usually meaningful semantically, in the sense they have a semantic interpretation in a given application area. Therefore, they are very often pre-defined by specialists in the application area. On the other hand, the use of “standard” decomposition/segmentation algorithms that involve (pure) mathe-

Fig. 2. Phases of image processing in a syntactic pattern recognition system (ECG).

mathematical models may result in receiving elementary components which differ, more or less, from pre-defined primitives. If differences are small, one can treat components received as “distortions” of (ideal) primitives. Otherwise, a considerable problem arises. Moreover, a choice of preprocessing operations (smoothing, sharpening, morphological operations, etc.) can influence a correct decomposition strongly. Therefore, we formulate the following principle.

II. *A decomposition/segmentation algorithm to be applied in a syntactic pattern recognition system should be defined with the use of knowledge of an application area which concerns a semantic interpretation of structural primitives (in the case this interpretation is essential). Choosing preprocessing techniques one should also take into account the desired final effect of the image processing stage, i.e., a decomposition of an image into pre-defined primitives.*

In practical terms this means that a lot of experiments with various preprocessing algorithms (and their various parametrizations) should be conducted at the early phases of a project of a syntactic pattern recognition system construction.

Once an image is decomposed into primitives, they are identified/recognized and the structural representation of the image is generated. For example, after the decomposition of the sub-image shown in Fig. 2(b) primitives are identified as shown in Figs 2(c) and 2(d). From that moment its structural representation $\langle PR \rangle \langle rsR' \rangle \langle ST- \rangle \langle T- \rangle$, where $\langle PR \rangle$, $\langle rsR' \rangle$, $\langle ST- \rangle$, $\langle T- \rangle$ are symbols representing primitives, is treated as the word of the formal language. Then, this word is analyzed by a parser (cf. Fig. 1).

Generally speaking, the main objective of research in syntactic pattern recognition can be defined as:

- constructing classes of grammars of the strong generative power, with corresponding

- parsing algorithms of the low time complexity, and
- grammatical inference algorithms of the polynomial time complexity.

These targets: strong grammar - efficient parser - polynomial inference algorithm concern each of three groups of methods, namely string methods, tree methods and the graph ones. It seems, however, that for each group of these methods one of the three targets can be considered as crucial especially at the current stage of the development of syntactic pattern recognition.

The descriptive power of string grammars is relatively weak as compared to the tree/graph ones. Although a variety of efficient methods based on regular and context-free grammars have been constructed, pattern structures are frequently of the context-sensitive nature. Yet context-sensitive grammars cannot be used, because of non-polynomial time complexities of corresponding automata. Therefore, defining enhancements of context-free grammars so that they can generate some context-sensitive languages seems to be crucial for string methods. We discuss this issue in Section 3. In the theory of tree languages a lot of tree automata have been defined in the 1970s and 1980s. The membership problem in the case of deterministic tree automata used in syntactic pattern recognition is decidable in polynomial time. Therefore, the authorities in the field claim that such automata can be efficiently applied as pattern recognizers.^{2,10} On the other hand, induction of tree grammar seems still to be the key open problem. We present this problem in Section 4. Graph languages are used if pattern structures are complex and cannot be represented in the form of string/tree. Then, graph grammar is used for patterns' generation and graph parser for their analysis. Unfortunately, the problem of parsing of non-trivial graph languages is PSPACE-complete or NP-complete.¹¹⁻¹³ Therefore, the crucial problem of defining graph languages with the polynomial membership problem is discussed in Section 5. We present some methodological remarks concerning graph grammar induction in this section as well.

3. Enhancement of String Models' Generative Power

Let us introduce, firstly, basic notions concerning string languages.

Let V be a set of symbols (alphabet). A set of strings which can be defined by concatenating symbols of V , including the empty string/word (denoted by λ), is denoted by V^* (*Kleene star*). If the empty string is excluded, then the set of strings over V is denoted by V^+ . For example, if A, B, C are sets of symbols, then defining a string s as $s \in A^*BC^+$ means that s is the concatenation of the substring constructed of symbols of A (including the empty substring), one symbol of B , and the substring of symbols of C (excluding the empty substring).

Let G be a generative grammar.⁶ We define basic components of G in a general way, not determining its structure, since structures of string grammars considered in this section are different from the quadruple structure of grammars of the Chomsky

hierarchy. Let $\Sigma \subset V$ be a *set of terminal symbols*, i.e., symbols occurring in words of the language generated by G . $N = V \setminus \Sigma$ is a *set of nonterminal (auxiliary) symbols* which are used for deriving words of the language. P is a *set of productions (rules)* used for the generation of the language. For grammars of the Chomsky hierarchy productions are in the following forms:

- $\alpha \rightarrow \gamma$, $\alpha \in V^*NV^*$, $\gamma \in V^*$ for *unrestricted grammars*, UNR ,
- $\eta_1 A \eta_2 \rightarrow \eta_1 \gamma \eta_2$, $\eta_1, \eta_2 \in V^*$, $A \in N$, $\gamma \in V^+$ for *context-sensitive grammars*, CSG^b ,
- $A \rightarrow \gamma$, $A \in N$, $\gamma \in V^*$ for *context-free grammars*, CFG ,
- $A \rightarrow \gamma$, $A \in N$, $\gamma \in \Sigma \cup \Sigma N \cup \{\lambda\}$ for *regular grammars*, REG^c .

An application of a production to a string $\alpha \in V^*$ consists in replacing the left-hand side of the production (the part before an “arrow”) in α by its right-hand side, which results in obtaining a string $\beta \in V^*$. Such an application is called a *derivation step*, denoted by $\alpha \Rightarrow \beta$. A sequence of derivation steps (including the empty sequence) is called a *derivation*, denoted by $\xRightarrow{*}$.

The last basic element of G is the starting symbol $S \in N$.

A language generated by G is a set $L(G) = \{\alpha \mid S \xRightarrow{*} \alpha, \alpha \in \Sigma^*\}$, i.e., it contains all strings which consist of terminal symbols only which can be derived by productions of G beginning from the starting symbol S .

A generative/descriptive power of a class of grammars is an important notion. Let X denote a class of grammars. A corresponding class X of languages is a set $\mathcal{L}(X) = \{L \mid \exists G \text{ of the class } X : L = L(G)\}$, i.e., it is a set of all the languages L which can be generated by any grammar G of the class X . We say that grammars of a class X are of a stronger generative power than grammars of a class Y if $\mathcal{L}(Y) \subsetneq \mathcal{L}(X)$. For example, for the Chomsky hierarchy the following holds: $\mathcal{L}(REG) \subsetneq \mathcal{L}(CFG) \subsetneq \mathcal{L}(CSG) \subsetneq \mathcal{L}(UNR)$.

In syntactic pattern recognition only the two weaker classes are used, namely regular (REG) and context-free grammars (CFG), since automata for context-sensitive (CSG) and unrestricted (UNR) grammars are inefficient computationally. Regular languages are recognizable by (deterministic) *finite-state automata (FSA)* in the linear time. Although *pushdown automata (PDA)* recognize context-free languages in the exponential time, context-free grammars can be restricted to be the deterministic ones. Then, efficient parsers can be used for analyzing the corresponding languages.

3.1. Basic Types of Enhanced Grammars

As we have mentioned in the previous section, pattern structures are frequently of the context-sensitive nature. We cannot use the class of context-sensitive grammars for a generation of such structures, because corresponding linear bounded automata,

^bAdditionally we assume that a production in the form $A \rightarrow \lambda$ is allowable, if A does not occur in any production of P in its right-hand side.

^cThe definition concerns *right-regular grammars*.

which in this case should be applied for analyzing these structures, are of the non-polynomial time complexity. Therefore, we try to enhance the descriptive power of context-free grammars so that they can generate not only context-free languages, but also certain context-sensitive languages. In this section we present three basic classes of such grammars with various enhancement mechanisms and we introduce other important classes.

The enhancement of context-free grammar consists in adding to its definition a feature/component which allows one to control a derivation. We can make it either by adding control operators to productions or by defining an additional component which controls a derivation process. Let us begin with presenting a class of grammars which belong to the first approach. A class of indexed grammars¹⁴ introduced by Aho in 1968 is defined in the following way.

Definition 1. An indexed grammar is a quintuple $G = (V, \Sigma, I, P, S)$, where: $V, \Sigma \subset V$, $S \in N$ are defined as in the previous section, I is a set of indices, P is a finite set of productions of one of the three forms:

$$(1) A \longrightarrow \alpha \text{ or } (2) A[.] \longrightarrow B[i..] \text{ or } (3) A[i..] \longrightarrow [..]\alpha ,$$

where A and $B \in N$, $i \in I$, $[..]$ represents a stack of indices, a string in I^* , $\alpha \in V^*$.

As we can see, indices are used in productions to define the context during a derivation. The left-most index is treated as the top element of the stack.

Let us define a derivation in indexed grammars as follows.¹⁴ Let $A \in N$, $B \in N$, $X_j \in V$, $\beta \in V^*$, $\gamma \in V^*$, $i \in I$, $\delta_j \in I^*$.

- (1) If $A \longrightarrow X_1...X_k$ is a production of type (1), then $\beta A \delta \gamma \Rightarrow \beta X_1 \delta_1 ... X_k \delta_k \gamma$, where $\delta_j = \delta$ if $X_j \in N$ and $\delta_j = \lambda$ if $X_j \in \Sigma$.
- (2) If $A[.] \longrightarrow B[i..]$ is a production of type (2), then $\beta A \delta \gamma \Rightarrow \beta B i \delta \gamma$.
- (3) If $A[i..] \longrightarrow [..]X_1...X_k$ is a production of type (3), then $\beta A i \delta \gamma \Rightarrow \beta X_1 \delta_1 ... X_k \delta_k \gamma$, where $\delta_j = \delta$ if $X_j \in N$ and $\delta_j = \lambda$ if $X_j \in \Sigma$.

The descriptive power of indexed grammars allows one to generate even (context-sensitive) fractal structures. To show this we define the indexed grammar generating the Sierpiński Carpet (see Fig. 3). Its two-dimensional structure is coded by the string over the set of terminal symbols $\Sigma = \{0, 1, (,)\}^d$ with the help of brackets, according to the scheme depicted in Fig. 4(a) in the following way.

- Fig. 3(b): (111101111).
- Fig. 3(c): ((111101111)(111101111)(1...)(1...)0(1...)(1...)(1...)(111101111)), etc.

The set of productions P is defined as follows.

^d0 denotes a white square and 1 denotes a black square.

Fig. 3. The generation of the Sierpiński Carpet.

1. $S[.] \rightarrow S[i..]$ (the production of type 2)
2. $S \rightarrow (AAAA0AAAA)$ (the production of type 1)
3. $A[i..] \rightarrow [..](AAAA0AAAA)$ (the production of type 3)
4. $A \rightarrow 1$ (the production of type 1)

Now, we can generate the Sierpiński Carpet shown in Fig. 3(c).

$$\begin{aligned}
 S[] &\xrightarrow{(1)} S[i] \xrightarrow{(2)} (A[i]A[i]A[i]A[i]0A[i]A[i]A[i]A[i]) \\
 &\xrightarrow{(3)} ((A[]A[]A[]A[]0A[]A[]A[])A[i]A[i]A[i]0A[i]A[i]A[i]A[i]) \\
 &\xrightarrow{7x(3)} ((A[]A[]A[]A[]0A[]A[]A[])(A[]...)(A[]...)(A[]...)0(A[]...)(A[]...)(A[]...)(A[]...)) \\
 &\xrightarrow{(4)} ((1A[]A[]A[]0A[]A[]A[])(A[]...)(A[]...)(A[]...)0(A[]...)(A[]...)(A[]...)(A[]...)) \\
 &\xrightarrow{7x(4)} (((111101111)(A[]...)(A[]...)(A[]...)0(A[]...)(A[]...)(A[]...)(A[]...)) \\
 &\xrightarrow{56x(4)} (((111101111)(111101111)(1...)(1...)0(1...)(1...)(1...)(1...))
 \end{aligned}$$

The symbol $\xrightarrow{k}$ denotes the application of the k-th production and the symbol $\xrightarrow{n \times (k)}$ denotes the application of the k-th production n times.

Now, we describe the derivation in an intuitive way. We begin with putting the index i on the stack with the help of the first production. The number of indices i corresponds to the number of iterations during the generation of the Sierpiński Carpet. (If we started with the second production, i.e., the number of indices i

Fig. 4. The generation of the Sierpiński Carpet by the indexed grammar.

was equal to zero, then we would obtain the structure shown in Fig. 3(b).) The real generation of the structure is started by applying the second production (cf. Fig. 4(b)). The third production removes the index i and nests iteratively the basic pattern (cf. Fig. 4(c)). The fourth production generates black squares (cf. Fig. 4(d)). (White squares are generated by productions 2 and 3.)

In order to generate more complex structures, one should use the first productions more times. For example, if we applied the first production twice, we would obtained the structure shown in Fig. 3(d). Let us notice that stacks of indices assigned to nonterminal symbols allow us to express contextual dependencies during the derivation according to the following mechanism. If we apply a production to a nonterminal on the stack, then copies of the stack are ascribed to all nonterminals of the right-hand side of the production.

Although indexed grammars are of the strong generated power, neither a polynomial parser nor a polynomial grammatical inference algorithm has been defined for them. Therefore, certain subclasses of indexed languages with better computational properties have been defined. Most of them belong to the class of mildly context-

sensitive languages (MCSLs) ^{15,16} which can be characterized in the following way. They contain context-free languages and some languages with context dependencies ($L_1 = \{a^n b^n c^n \mid n \geq 0\}$, $L_2 = \{a^n b^m c^n d^m \mid n, m \geq 0\}$, $L_3 = \{ww \mid w \in \{a, b\}^*\}$). They are of the constant growth, i.e., if strings of a language are ordered in the sequence according to their length, then two successive lengths do not differ in arbitrary large amounts. The membership problem for MCSLs is solved in deterministic polynomial time. The best known MCSGs include: linear indexed grammars introduced by Gazdar in 1988 ¹⁷, head grammars defined by Pollard in 1984 ¹⁸, combinatory categorial grammars presented by Steedman in 1987 ¹⁹, tree adjoining grammars introduced by Joshi, Levy and Takahashi in 1975 ^{20,21} and two classes defined by Okhotin: conjunctive grammars in 2001 ²² and Boolean grammars in 2004. ²³ Recently, the survey concerning these grammars has been published ²⁴. For all these classes polynomial parsing algorithms have been defined and for none of them a grammatical inference algorithm has been constructed.

Now, we introduce grammars with a separated control mechanism, i.e., the mechanism that is not “hidden” in left- or right-hand sides of productions. Programmed grammars introduced by Rosenkrantz in 1969 ²⁵ are a classical example of such grammars.

Definition 2. A programmed grammar is a quintuple $G = (V, \Sigma, J, P, S)$, where: $V, \Sigma \subset V$, $S \in N$ are defined as in the previous section, J is a set of production labels, P is a finite set of productions of the form :

$$(r) \quad \alpha \longrightarrow \beta \quad S(U) \quad F(W), \text{ in which}$$

$\alpha \longrightarrow \beta, \alpha \in V^* N V^*, \beta \in V^*$, is called the core, (r) is the production label, $r \in J$, $U \subset J$ is the success field and $W \subset J$ is the failure field.

We define a derivation in programmed grammars in an informal way as follows. Firstly, the production labeled with (1) is applied. If a production (r) can be applied, then after its application the next production is chosen from its success field U . Otherwise, the next production is chosen from the failure field W .

Let us define the programmed grammar generating the Sierpiński Carpet (see Fig. 3). The set of productions P is defined as follows.

1. $S \rightarrow (A_n A_n A_n A_n 0 A_n A_n A_n A_n) \quad S(\{2, 4\}) \quad F(\emptyset)$
2. $A_n \rightarrow (A_0) \quad S(\{2\}) \quad F(\{3\})$
3. $A_0 \rightarrow A_n A_n A_n A_n 0 A_n A_n A_n A_n \quad S(\{3\}) \quad F(\{2, 4\})$
4. $A_n \rightarrow 1 \quad S(\{4\}) \quad F(\emptyset)$

Let us generate the Sierpiński Carpet shown in Fig. 3(c). (The graphical interpretation of main phases of the derivation is shown in Fig. 5).

$$\begin{aligned}
 S &\xrightarrow{(1)} (A_n A_n A_n A_n 0 A_n A_n A_n A_n) \xrightarrow{(2)} ((A_0) A_n A_n A_n 0 A_n A_n A_n A_n) \\
 &\xrightarrow{7 \times (2)} ((A_0)(A_0)(A_0)(A_0)0(A_0)(A_0)(A_0)(A_0)) \\
 &\xrightarrow{(3)} ((A_n A_n A_n A_n 0 A_n A_n A_n A_n)(A_0)(A_0)(A_0)0(A_0)(A_0)(A_0)(A_0)) \\
 &\xrightarrow{7 \times (3)} ((A_n A_n A_n A_n 0 A_n A_n A_n A_n)(A_n \dots)(A_n \dots)(A_n \dots)0(A_n \dots)(A_n \dots)(A_n \dots)(A_n \dots))
 \end{aligned}$$

Fig. 5. The generation of the Sierpiński Carpet by the programmed grammar.

$$\xRightarrow{64 \times (4)} ((111101111)(111101111)(1...)(1...)0(1...)(1...)(1...)(1...))$$

For programmed grammars the non-polynomial parsing algorithm has been defined.

The static control formalism of programmed grammars (success and failure fields contain fixed production indices) has been extended in GDPLL(k) grammars (Generalized Dynamically Programmed LL(k) grammars).^{24,26} We will not introduce them formally in the paper, but we will characterize them in an intuitive way. (The survey²⁴ includes a good introduction to these grammars.) In GDPLL(k) grammars, instead of success and failure fields, each production is devised with the control tape. The head of the tape can write/read indices of productions and can move. A derivation is performed according to the content of the tape. The derivation is deterministic due to the constraint analogous to that introduced by Rosenkrantz and Stearns for LL(k) grammars.²⁷ The additional restriction which limits the number of recursive derivation steps enables good computational properties. In spite of these constraints DPLL(k) grammars generate the large subclass of context-

sensitive languages, e.g., $L_1 = \{a^n b^n c^n \mid n \geq 0\}$, $L_2 = \{a^n b^m c^n d^m \mid n, m \geq 0\}$, $L_4 = \{a^{2^n} \mid n \geq 0\}$.²⁸ The parsing algorithm for GDPLL(k) grammars is of the $O(n)$ time complexity, and the grammatical inference algorithm is of the $O(m^3 \cdot n^3)$ time complexity, where m is the sample size, n is the maximum length of a string in the sample.²⁹

Apart from enhanced context-free grammars based on the Chomskyan paradigm, two models going beyond this paradigm are worth mentioning due to polynomial complexities of parsing algorithms. The hybrid syntactic-structural model based on *augmented regular expressions*, called *AREs*, was introduced in 1997 by Alquezar and Sanfeliu³⁰ and *Contextual Grammars*, *CGs*, were defined by Marcus in 1969.³¹ For *ARE* the (non-polynomial) grammatical inference algorithm was defined and for Contextual Grammars the polynomial grammatical inference algorithm was constructed. (The survey²⁴ contains brief characteristics of these models.)

3.2. Methodological Remarks about String Models

Polynomial parsing algorithms have been defined for enhanced context-free grammars with the derivation control operators “hidden” in productions (linear indexed grammars, head grammars, combinatory grammars, conjunctive/Boolean grammars) as well as for grammars with the control mechanism separated from left- and right-hand sides of productions. However, the (polynomial) algorithm of grammatical inference has been defined only for the latter (DPLL(k) grammars²⁹). The general idea of grammar induction methods consists in finding similarities among sample strings. The alphabet of the inferred grammar includes only terminal symbols which occur in the language sample and nonterminals which are added as classes of abstractions for certain terminal substrings. The only operator used is the “invisible” catenation operator. However, if grammar productions include operators which appear and disappear during a derivation and do not occur in a derived word (such as brackets [] and indices for indexed grammars), the grammatical induction problem becomes difficult. Thus, we can summarize our considerations on syntactic pattern recognition models based on enhanced context-free grammars with the following methodological principle.

III. *A development of an efficient grammatical inference module for a syntactic pattern recognition system is easier for grammars with separated derivation control mechanisms than in the case of using additional control operators in left- or right-hand sides of productions. The use of models going beyond the Chomskyan paradigm in the field of syntactic pattern recognition is worth studying.*

4. Tree-based Syntactic Pattern Recognition

Tree languages and automata were introduced in the mid 1960s as formalisms for solving decidability problems in the second order logic.^{32,33} The theory is still being developed intensively in relation with logic. This delivers new models of tree

automata and corresponding tree grammars.³⁴

Tree grammars are generalizations of string grammars, in the sense that their left- and right-hand sides are trees. For more general *context-free tree grammar*³⁵ the left-hand side is a two-level tree^e or a one-level tree consisting of the root labeled by nonterminal only, whereas for *regular tree grammar*³⁶ the left-hand side is a nonterminal (root). For regular tree grammars *finite tree automata* have been defined. Generally, such automata walk along tree paths in parallel. This means that they can be, somehow, in “more than one state at a moment” and a tree is analyzed with the help of more than one reading head. Two basic classes of finite tree automata are distinguished. *Frontier-to-root automata* (bottom-up tree automata)^{32,33} analyze a tree from leaves upward to the root, whereas *root-to-frontier automata* (top-down tree automata)³⁷ perform an analysis in the opposite direction^f. For context-free tree grammars, top-down pushdown tree automata (PDTA's)³⁸ which are analogous to string LL-parsers and bottom-up tree pushdown automata (TPDA's)³⁹ which are analogous to string LR-parsers have been defined.

4.1. Tree Automata and Tree Grammar Inference for Pattern Analysis

The predominance of tree automata over the string ones in the case of multidimensional patterns was noticed in syntactic pattern recognition just after introducing the first such automata, i.e., in the early 1970s.^{10,40} The advantage of tree models of the string ones results from the possibility of expressing the catenation relation between primitives/subpatterns only in the second case. Constructing *efficient* automata for tree grammars of the sufficient descriptive power is more difficult than for string grammars, since in case of the former we do not have the linear ordering of a structure, but only the partial ordering.

Nevertheless, because of the intensive development of tree automata theory, mentioned above, quite a lot of successful extensions of such automata have been made for pattern recognition applications. Recognition of noisy and distorted patterns have been the main goal of defining enhanced tree models. In general, two basic mechanisms have been used. Firstly, grammar and its corresponding automaton can be extended to the stochastic ones by randomizing both grammar productions and automaton transitions. Then, a stochastic automaton gives recognition results together with computed probabilities of belonging to considered classes. The final recognition is made with the help of the *maximum-likelihood* classification rule. The first stochastic tree automaton for pattern recognition was defined by Fu in 1980⁴¹ and Abe and Mamitsuka applied this approach for predicting protein structures in 1997.⁴² The second approach consists in the use of a distance measure between an unknown structure and an undistorted (prototype) structure defined

^eWe present tree grammars and automata in an intuitive way here. So, we do not introduce such notions as ranked alphabets, terms, distinct variables, etc.

^fThere are also finite tree automata which use other strategies of walking, e.g., two-way automata, tree-walking automata.

as the minimum number of error transformations required to obtain one structure from another. Lu and Fu introduced two basic types of (minimum-distance) *error-correcting tree automata*: the structure-preserved ones (SPECTA's) ⁴³ which do not encompass modifications of tree structure and the generalized ones (GECTA's) ⁴⁴ allowing structural modifications. In 1982 Lee defined fuzzy tree automata, ⁴⁵ whereas in 2000 López, Sempere and Garcia introduced the more efficient version of error-correcting tree automata. ⁴⁶

Although a variety of inference algorithms for string grammars and automata have been introduced, ^{47,48} only a few algorithms in the case of tree structures have been defined for forty years. The first algorithms were proposed in the late 1970s by Gonzales, Edwards and Thomason ⁴⁹ and Brayer and Fu. ⁵⁰ Then, inference algorithms were defined by Levine ⁵¹ in 1982, Garcia, Vidal and Casacuberta in 1987, ⁵² Sakakibara in 1990, ⁵³ Knuutila and Steinby in 1994, ⁵⁴ Fernau in 2002, ⁵⁵ and Sempere and López in 2003. ⁵⁶ In 1991 Barrero introduced the expansive tree grammar inference scheme using negative samples. ⁵⁷ The stochastic tree language inference method was proposed by Carrasco, Oncina and Calera-Rubio in 2001 ⁵⁸ and the error-correcting tree language inference model was defined by López and España in 2002. ⁵⁹

4.2. Methodological Remarks about Tree Models

This seems that the problem of grammatical inference is still the key open problem in the area of tree models of syntactic pattern recognition. Studying the papers on tree grammar/automaton inference mentioned above, a certain analogy to the induction for string languages can be noticed. The inference scheme for expansive regular tree grammars based on the two-stage approach, i.e., firstly defining a canonical grammar, then constructing a derived grammar, proposed by Barrero, ⁵⁷ is an excellent example of the use of such an analogy-based methodology^g. Let us analyze this analogy, which is depicted in Fig. 6. The inference of a canonical regular string grammar consists in defining a *unique* set of productions for every string of a sample as shown in Fig. 6(a). As we also see in Fig. 6(a), we can define, by analogy, a *unique* set of tree grammar productions for every tree of a sample. Since for big samples the number of nonterminals of canonical grammar is excessive, in the second stage derived grammar is constructed by partitioning these nonterminals into equivalence classes as shown in Fig. 6(b). Let us notice that by analogy to string regular grammars, we can make such partitioning for expansive tree grammars (cf. Fig. 6(b)).

The method proposed by Sakakibara ⁵³ is another excellent example of making use of inferencing (context-free) string languages for induction of tree grammars.

Let us summarize our considerations with the following methodological recommendation.

^gAs we have mentioned above, in this case the standard scheme is considerably enhanced by using negative samples. ⁵⁷

string $a_1 a_2 \dots a_5 \in S^+ \implies X_1 \rightarrow a_1 X_2, X_2 \rightarrow a_2 X_3, \dots, X_5 \rightarrow a_5 \in P_C$

canonical grammar

string $\begin{matrix} \vdots \\ A_1 \rightarrow a \bar{A}_1 \\ \vdots \\ A_n \rightarrow a \bar{A}_n \\ \vdots \end{matrix}$

and

$A = \{A_1, \dots, A_n\}$
 $\bar{A} = \{\bar{A}_1, \dots, \bar{A}_n\}$

$\implies$

derived grammar

$\begin{matrix} \vdots \\ A \rightarrow a \bar{A} \\ \vdots \end{matrix}$

and

$A = \{A_1, \dots, A_n\}$
 $A^1 = \{A_1^1, \dots, A_n^1\}$
 $\vdots$
 $A^m = \{A_1^m, \dots, A_n^m\}$

$\implies$

(b) Defining derived grammar on the basis of canonical grammar.

Fig. 6. Tree grammar inference vs string grammar inference - analogies.

IV. Schemes used for inferring string grammars/automata can be useful for constructing (analogous) tree grammars/automata inference algorithms.

Fig. 7. Examples of graph representations for syntactic pattern recognition.

5. Graph-based Syntactic Pattern Recognition

Graph grammars are the strongest descriptive formalisms used for generating multidimensional patterns, since one can define any relations between pattern components, whereas in trees cycles are forbidden. For example, graphs generated by parsable ETPL(k) graph grammars^{60–62} used for the CAD solid analysis⁶³ and for the (Polish) sign language recognition⁶⁴ are shown in Figs. 7(a) and 7(b), respectively. On the other hand, only a few graph-grammar-based syntactic pattern recognition models have been presented for the last fifty years. This results from the hard, i.e., PSPACE-complete or NP-complete, membership problem for graph grammars of the strong descriptive power.^{11–13} We analyze this problem in the next section.

5.1. Constructing Graph Languages with the Polynomial Membership Problem

There are two main reasons of difficulties with constructing syntax analysis algorithms for graph languages. First of all, let us notice that in each step of syntax analysis, a *handle*, i.e., a piece of a structure (substring, subgraph) to be matched with predefined structures^h stored in the parser memory, should be identified. However, this means looking for a subgraph which is isomorphic to the given one, which is the NP-complete problem.

Secondly, for string/tree grammars one knows where to glue the right hand-side of the production during a derivational stepⁱ. However, in the case of graph grammars one has to specify how to embed the right-hand side graph in the derived graph with the *embedding transformation*. For graph grammars of the strong descriptive

^hThese structures are predefined on the basis of right-hand sides of grammar productions.

ⁱThis results from the rigid structure of strings/trees.

power this transformation allows us to modify a derived graph largely. On the other hand, the embedding transformation acts at the border between the left- (right-) hand sides of productions and their context, i.e., its behavior is context-sensitive-like even for graph grammars, which are regular-like or context-free-like.

A few graph-parsing-based methods have been defined for pattern representation/recognition applications till now^j, because of two problems mentioned above. Web automata were defined by Rosenfeld and Milgram in 1972.⁶⁵ The efficient parser for expansive graph grammars was proposed by Fu and Shi in 1983.⁶⁶ In 1990 two parsing algorithms for plex grammars were proposed by Bunke and Haller⁶⁷ and Peng, Yamamoto, and Aoki.⁶⁸ In the early 1990's two parsing methods for relational grammars were defined independently by Wittenburg⁶⁹ and Ferruci, Tortora, Tucci, and Vitiello.⁷⁰ The efficient, $O(n^2)$, parser for the ETPL(k) subclass of the well-known edNLC⁷¹ graph grammars was constructed in 1993.^{61,62} The parsing method for reserved graph grammars was defined by Zhang, Zhang, and Cao in 2001.⁷²

Let us summarize our analysis with the following recommendation.

V. *If a certain class of graph grammars is to be used for constructing a syntactic pattern recognition model, the polynomial time complexity of the membership problem for languages generated by this class has to be obtained.*

5.2. Methodological Remarks about Graph Models

There are three main types of restrictions which are imposed on definitions of graph grammars in order to reduce the computational complexity of corresponding graph automata/parsers. First two types are specific for graph languages. First of all, in order to cope with the NP-completeness of extracting graph handles, the partial or linear ordering can be defined for graph structure. Such ordering can be introduced arbitrarily on the basis of characteristics of images/patterns. However, the ordered graph representation should be defined in a unique and an unambiguous way with respect to the represented image/pattern. (That is, only one graph representation is defined for a given pattern and a given graph structure represents only one pattern.) Efficient graph models have been obtained due to imposing the graph ordering, e.g., for Ω -graphs generated by expansive graph grammars⁶⁶ and for *IE*-graphs generated by ETPL(k) graph grammars.⁶¹

The second type of restrictions which is specific for graph languages concerns the embedding transformation. We should restrain its abundant capabilities in order to make its behavior more predictable during syntax analysis. There are two general kinds of such constraints^k. Firstly, limitations can be imposed on the form of the right-hand side graph (indirect limitations of the embedding transformation power).

^jThe classes of graph grammars used for syntactic pattern recognition applications have to be of the strong descriptive power.

^kAlthough the terminology of the taxonomy of constraints is introduced on the basis of the theory of NLC-type graph grammars, it can be used, by analogy, for other classes of graph grammars.

Fig. 8. Graph grammar parsing vs string grammar parsing - analogies.

For example, for *boundary grammars* no two nonterminal nodes are connected in the right-hand side graphs (and the starting graph). For *linear grammars* there is at most one nonterminal node in the derived graph. Only two nodes (one terminal and one nonterminal) or one terminal node can occur in the right-hand side graphs

in *regular grammars*. Secondly, limitations can be imposed on the form of the embedding transformation directly. For example, for *apex grammars* only terminal nodes are allowed in formulas of the embedding transformation. For *analogous grammars* a subset of the right-hand side graph nodes having the same embedding behavior as the replaced node is determined and the remaining right-hand side nodes are not used in the embedding. For *node identification grammars* the embedding rule determines only one right-hand side node which is identified with the node replaced.

Limitations of the third type are imposed in order to obtain deterministic parsing and they are not specific for graph languages. Therefore, they can be defined by analogy to string languages. For example, the deterministic derivation during top-down parsing of ETPL(k) graph languages has been obtained by analogy to the string deterministic LL(k) grammars.²⁷ In Fig. 8(a) the property of the unambiguous choice of LL(k) grammar production with respect to the *k-length prefix handle* in the (leftmost) derivation is shown. The analogous property for ETPL(k) graph grammars has been defined on the basis of the notion of the *k-successors graph handle*. Such a handle is a two-level graph which in Fig. 8(b) consists of the node labeled by *a* and having the index (i) determining the place of the production application and its *k* successors $a_1, \dots, a_k$. Now, the property for ETPL(k) grammars can be formulated as follows. For any (leftmost) derivational step one should be able to choose the production in an unambiguous way on the basis of an analysis of the *k-successors graph handle* (cf. Fig. 8(b)).

The problem of grammatical inference for graph languages is even more complex than graph syntactic analysis. In the area of syntactic pattern recognition only a few such algorithms have been defined for parsable graph languages.^{66,73,74} The polynomial algorithm for the parsable ETPL(k) grammars⁶⁰⁻⁶² has been defined with the use of analogies to the well-known scheme of the *formal derivatives method* for string grammars.² At the end of this section let us formulate the following methodological recommendation.

VI. *Imposing the arbitrary partial/linear ordering on graph structure and limiting power capabilities of the embedding transformation are main techniques used to reduce the computational complexity of graph parsers. In order to achieve deterministic parsing one should look for analogical constructs and mechanisms in theory of string languages.*

6. Final Methodological Remarks

After considering methodological issues which are specific for three main groups of syntactic pattern recognition methods, we can formulate general concluding principles for the field. Firstly, as we have discussed, the stronger generative power of a grammar class is, the higher computational complexity of the corresponding syntax analysis algorithm. Of course, the same concerns grammatical inference algorithms. Therefore we propose the following methodological recommendation.

VII. *A syntactic pattern recognition model should be defined for the specific problem of the strictly-defined application area according to the Ockham Razor principle with respect to the grammar generative power. It means that a class of grammars used should be of the weakest descriptive power yet sufficient to generate all patterns considered.*

Secondly, although the issue of recognition of noisy and distorted patterns has been discussed in the section about tree methods, the problem concerns, of course, all the three groups of methods. Similarly as tree automata, string and graph parsers can be extended to handle noisy/distorted patterns. Additionally, if distortions are more “subtle” than differences between symbolic (category-based) primitives, *attributed grammars* can applied.⁷⁵ Attributes characterize features of primitives in detail, e.g., numeric features, and productions contain the (standard) syntactic part and the semantic part in the form of *semantic rules*. The rules are used to evaluate attributes of primitive symbols appearing in the production in terms of attributes of other symbols. This way, during parsing the distance between an unknown pattern and ideal patterns defined by the grammar can be computed on the basis of structural distortions as well as of corresponding vectors of attributes. So, let us introduce the following principle.

VIII. *If a syntactic pattern recognition model is to be used for analyzing real-world objects of the fuzzy/vague nature, it should be enhanced by such formalisms as error-correcting parsers, stochastic parsers, fuzzy parsers or attributed grammars.*

In the chapter we have focused primarily on classical issues of syntactic pattern recognition, that is, on grammars generating patterns and corresponding parsing/inference algorithms used for syntax analysis. However, it should be noted once more that issues related to image preprocessing and generation of structural representations of patterns are also of great importance for constructing effective syntactic pattern recognition systems.

References

1. H. O. Bunke and A. Sanfeliu, Eds., *Syntactic and Structural Pattern Recognition - Theory and Applications*. (World Scientific, Singapore, 1990).
2. K. S. Fu, *Syntactic Pattern Recognition and Applications*. (Prentice Hall, Englewood Cliffs, 1982).
3. R. C. Gonzales and M. G. Thomason, *Syntactic Pattern Recognition: An Introduction*. (Addison-Wesley, Reading, 1978).
4. T. Pavlidis, *Structural Pattern Recognition*. (Springer, New York, 1977).
5. E. Tanaka, Theoretical aspects of syntactic pattern recognition, *Pattern Recognition*. **28**, 1053–1061, (1995).
6. N. Chomsky, Three models for the description of language, *IRE Trans. Information Theory*. **2**, 113–124, (1956).
7. L. Goldfarb. Pattern representation and the future of pattern recognition. In *Proc. ICPR-2004 Workshop*, Cambridge, UK (Aug., 2004).
8. R. Tadeusiewicz and M. R. Ogiela, *Medical Image Understanding Technology*. (Springer, Berlin-Heidelberg-New York, 2004).

9. A. K. Jain, R. P. W. Duin, and J. Mao, Statistical pattern recognition: a review, *IEEE Trans. Pattern Analysis and Machine Intelligence*. **22**, 4–37, (2000).
10. K. S. Fu. Tree languages and syntactic pattern recognition. In ed. C. H. Chen, *Pattern Recognition and Artificial Intelligence*. Academic Press, New York, (1976).
11. F. J. Brandenburg. On the complexity of the membership problem of graph grammars. In *Proc. Int. Workshop on Graphtheoretic Concepts in Computer Sci. (WG'83)*, pp. 40–49, Osnabrück, Germany (June, 1983). Trauner Verlag.
12. A. O. Slisenko, Context-free grammars as a tool for describing polynomial-time subclasses of hard problems, *Information Processing Letters*. **14**, 52–56, (1982).
13. G. Turan. On the complexity of graph grammars. Rep. automata theory research group, Szeged, Hungary, (1982).
14. A. V. Aho, Indexed grammars - an extension of context-free grammars, *Journal of the Association for Computing Machinery*. **15**, 647–671, (1968).
15. A. K. Joshi, K. Vijay Shanker, and D. Weir. The convergence of mildly context-sensitive grammar formalisms. Technical report, University of Pennsylvania, Department of Computer and Information Science (January, 1990).
16. K. Vijay Shanker and D. J. Weir, The equivalence of four extensions of context-free grammars, *Mathematical Systems Theory*. **27**, 511–546, (1994).
17. G. Gazdar. Applicability of indexed grammars to natural languages. In eds. U. Reyle and C. Rohrer, *Natural Language Parsing and Linguistic Theories*. D. Reidel Publ. Comp., Englewood Cliffs, (1988).
18. C. Pollard. *Generalized Phrase Structure Grammars, Head Grammars, and Natural Language*. PhD thesis, Stanford University, CA, USA, (1984).
19. M. Steedman, Combinatory grammars and parasitic gaps, *Natural Language and Linguistic Theory*. **5**, 403–439, (1987).
20. A. K. Joshi, L. S. Levy, and M. Takahashi, Tree adjunct grammars, *Journal of Computer and System Sciences*. **10**, 136–163, (1975).
21. A. K. Joshi. How much context-sensitivity is necessary for characterizing structural descriptions – Tree Adjoining Grammars. In ed. D. D. et al., *Natural Language Processing – Theoretical, Computational and Psychological Perspective*. Cambridge University Press, New York, NY, (1985).
22. A. Okhotin, Conjunctive grammars, *Journal of Automata, Languages and Combinatorics*. **6**, 519–535, (2001).
23. A. Okhotin, Boolean grammars, *Information and Computation*. **194**, 19–48, (2004).
24. M. Flasiński and J. Jurek, Fundamental methodological issues of syntactic pattern recognition, *Pattern Analysis and Applications*. **17**, 465–480, (2014).
25. D. J. Rosenkrantz, Programmed grammars and classes of formal languages, *Journal of the Association for Computing Machinery*. **16**, 107–131, (1969).
26. M. Flasiński and J. Jurek, 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).
27. D. J. Rosenkrantz and R. E. Stearns, Properties of deterministic top-down grammars, *Information and Control*. **17**, 226–256, (1970).
28. J. Jurek, Recent developments of the syntactic pattern recognition model based on quasi-context sensitive languages, *Pattern Recognition Letters*. **26**, 1011–1018, (2005).
29. J. Jurek, Towards grammatical inferencing of GDPLL(k) grammars for applications in syntactic pattern recognition-based expert systems, *Lecture Notes in Computer Science*. **3070**, 604–609, (2004).
30. R. Alquezar and A. Sanfeliu, Recognition and learning of a class of context-sensitive languages described by augmented regular expressions, *Pattern Recognition*. **30**, 163–

- 182, (1997).
31. S. Marcus, Contextual grammars, *Rev. Roum. Math. Pures Appl.* **14**, 1525–1534, (1969).
 32. J. E. Doner, Decidability of the weak second-order theory of two successors, *Notices American Mathematical Society.* **12**, 365–468, (1965).
 33. J. W. Thatcher and J. B. Wright, Generalized finite automata theory with an application to a decision problem of second-order logic, *Mathematical Systems Theory.* **2**, 57–81, (1968).
 34. H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree Automata Techniques and Applications, (2007). <http://www.grappa.univ-lille3.fr/tata>.
 35. W. C. Rounds, Mappings and grammars on trees, *Mathematical Systems Theory.* **4**, 257–287, (1970).
 36. W. S. Brainerd, Tree generating regular systems, *Information and Control.* **14**, 217–239, (1969).
 37. M. O. Rabin, Decidability of second-order theories and automata on infinite trees, *Trans. American Mathematical Society.* **141**, 1–35, (1969).
 38. I. Guessarian, Pushdown tree automata, *Mathematical Systems Theory.* **16**, 237–263, (1983).
 39. K. M. Schimpf and J. H. Gallier, Tree pushdown automata, *Journal of Computer and System Sciences.* **30**, 25–40, (1985).
 40. K. S. Fu and B. L. Bhargava, Tree systems for syntactic pattern recognition, *IEEE Trans. Computers.* **4**, 83–100, (1971).
 41. K. S. Fu. Stochastic tree languages and their application to picture processing. In ed. P. R. Krishnaiah, *Multivariate Analysis V*. North-Holland, Amsterdam, (1980).
 42. N. Abe and H. Mamitsuka, Predicting protein secondary structure using stochastic tree grammars, *Machine Learning.* **29**, 275–301, (1997).
 43. S. Lu and K. S. Fu. Structure-preserved error-correcting tree automata for syntactic pattern recognition. In *Proc. IEEE Conf. Decis. Control*, pp. 413–419, (1976).
 44. S. Y. Lu and K. S. Fu, Error-correcting tree automata for syntactic pattern recognition, *IEEE Trans. Computers.* **27**, 1040–1053, (1978).
 45. E. T. Lee, Fuzzy tree automata and syntactic pattern recognition, *IEEE Trans. Pattern Analysis and Machine Intelligence.* **4**, 445–449, (1982).
 46. D. López, J. M. Sempere, and P. Garcia, Error correcting analysis for tree languages, *International Journal of Pattern Recognition and Artificial Intelligence.* **14**, 357–368, (2000).
 47. Y. Sakakibara, Recent advances of grammatical inference, *Theoretical Computer Science.* **185**, 15–45, (1997).
 48. C. Higuera de la, *Grammatical Inference*. (Cambridge University Press, Cambridge, 2010).
 49. R. C. Gonzales, J. J. Edwards, and M. G. Thomason, An algorithm for the inference of tree grammars, *International Journal of Computer and Information Science.* **5**, 145–164, (1976).
 50. J. M. Brayer and K. S. Fu, A note on k -tail method of tree grammar inference, *IEEE Trans. Systems, Man and Cybernetics.* **7**, 293–299, (1977).
 51. B. Levine, The use of tree derivatives and a sample support parameter for inferring tree systems, *IEEE Trans. Pattern Analysis and Machine Intelligence.* **4**, 25–34, (1982).
 52. P. Garcia, E. Vidal, and F. Casacuberta, Local languages, the successor method and a step towards a general methodology for the inference of regular grammars, *IEEE Trans. Pattern Analysis and Machine Intelligence.* **6**, 841–845, (1987).

53. Y. Sakakibara, Learning context-free grammars from structural data in polynomial time, *Theoretical Computer Science*. **76**, 223–242, (1990).
54. T. Knuutila and M. Steinby, Inference of tree languages from a finite sample: an algebraic approach, *Theoretical Computer Science*. **129**, 337–367, (1994).
55. H. Fernau, Learning tree languages from text, *Lecture Notes in Artificial Intelligence*. **2375**, 153–168, (2002).
56. J. Sempere and D. López, Learning decision trees and tree automata for a syntactic pattern recognition task, *Lecture Notes in Computer Science*. **2652**, 943–950, (2003).
57. A. Barrero, Inference of tree grammars using negative samples, *Pattern Recognition*. **24**, 1–8, (1991).
58. R. C. Carrasco, J. Oncina, and J. Calera-Rubio, Stochastic inference of regular tree languages, *Machine Learning*. **44**, 185–197, (2001).
59. D. López and S. España, Error-correcting tree language inference, *Pattern Recognition Letters*. **23**, 1–12, (2002).
60. M. Flasiński, Parsing of edNLC-graph grammars for scene analysis, *Pattern Recognition*. **21**, 623–629, (1988).
61. M. Flasiński, On the parsing of deterministic graph languages for syntactic pattern recognition, *Pattern Recognition*. **26**, 1–16, (1993).
62. M. Flasiński, Power properties of NLC graph grammars with a polynomial membership problem, *Theoretical Computer Science*. **201**, 189–231, (1998).
63. M. Flasiński, Use of graph grammars for the description of mechanical parts, *Computer-Aided Design*. **27**, 403–433, (1995).
64. M. Flasiński and S. Myśliński, On the use of graph parsing for recognition of isolated hand postures of Polish Sign Language, *Pattern Recognition*. **43**, 2249–2264, (2010).
65. A. Rosenfeld and D. L. Milgram, Web automata and web grammars, *Machine Intelligence*. **7**, 307–324, (1972).
66. Q. Y. Shi and K. S. Fu, Parsing and translation of attributed expansive graph languages for scene analysis, *IEEE Trans. Pattern Analysis and Machine Intelligence*. **5**, 472–485, (1983).
67. H. O. Bunke and B. Haller, A parser for context free plex grammars, *Lecture Notes in Computer Science*. **411**, 136–150, (1990).
68. K. J. Peng, T. Yamamoto, and Y. Aoki, A new parsing scheme for plex grammars, *Pattern Recognition*. **23**, 393–402, (1990).
69. K. Wittenburg, Earley-style parsing for relational grammars. In *Proc. IEEE Symp. Visual Lang. (VL'92)*, pp. 192–199, (1992).
70. F. Ferruci, G. Tortora, M. Tucci, and G. Vitiello, A predictive parser for visual languages specified by relational grammars. In *Proc. IEEE Symp. Visual Lang. (VL'94)*, pp. 245–252, (1994).
71. D. Janssens and G. Rozenberg, On the structure of node-label-controlled graph languages, *Information Sciences*. **20**, 191–216, (1980).
72. D. Q. Zhang, K. Zhang, and J. Cao, A context-sensitive graph grammar formalism for the specification of visual languages, *The Computer Journal*. **44**, 186–200, (2001).
73. D. López and J. M. Sempere, Handwritten digit recognition through inferring graph grammars. a first approach, *Lecture Notes in Computer Science*. **1451**, 483–491, (1998).
74. M. Flasiński, Inference of parsable graph grammars for syntactic pattern recognition, *Fundamenta Informatica*. **80**, 379–413, (2007).
75. D. Knuth, Semantics of context-free languages, *Mathematical Systems Theory*. **2**, 127–145, (1971).