Application of AGLEARN for Hungarian Part-of-speech Tagging

Zoltán Alexin[1], Szilvia Zvada[2] and Tibor Gyimóthy[2]

1: Department of Applied Informatics, József Attila University
Árpád tér 2, H-6720 Szeged, Hungary
Phone: (36) +(62) 454000/3411 ext., Fax: (36) +(62) 420292
e-mail: alexin@inf.u-szeged.hu
2: Research Group on Artificial Intelligence
Hungarian Academy of Sciences,
Aradi vértanuk tere 1, H-6720 Szeged, Hungary
Phone: (36) +(62) 454139, Fax: (36) +(62) 425508
e-mail: {zvada,gyimi}@inf.u-szeged.hu

Abstract

In this paper we present an application of the AGLEARN method to the part-of-speech (POS) tagging of Hungarian sentences. The task of the AGLEARN is *to infer the semantic functions* associated with the productions. In the learning process the grammar, background semantic functions and examples can be used. We applied the AGLEARN method to infer context rules to choose the correct tags. A corpus with about 100 000 pre-tagged words has been employed for training and testing. By using AGLEARN algorithm learning data sets are generated to the C 4.5 attribute value learner. These generated data contain information about the phrase structure of the sentences. A background attribute grammar has been used to determine the structural information. Our studies showed that using this structural background information the C 4.5 learner was able to infer more precise context rules. [1]

1. Introduction

Attribute grammars can be considered as an extension of context-free grammars, where the attributes are associated with grammar symbols and semantic rules define the values of the attributes. While many different attribute grammar evaluation methods have been defined [1],[3] less attention has been paid to the development of an efficient methodology for preparing attribute grammar specifications. The definition of an attribute grammar usually requires a lot of effort to make a useful tool which infer semantic rules for attribute grammars from examples. In [7] an algorithm (called AGLEARN) was presented for learning semantic functions of attribute grammars, which is a hard problem because semantic functions can also represent relations. The approach was motivated by the fact that there is a close relationship between attribute grammars and logic programs [4], [5], [14]. This relationship is based on the nonterminal-predicate correspondence and a well-known formalism in the logic programming framework which is very similar to the notation of attribute grammars namely *definite clause grammars* [15]. The concept of the AGLEARN method is based on the learning approaches developed in the framework *inductive logic programming* (ILP) [6], [10], [11]. In the ILP environment the language of logic programs is utilised to describe examples, background knowledge

[1]This work was supported by the grants OTKA T25721 and FKFP 1354/97

and concepts. This language is more expressive than propositional languages which are used in many inductive learning systems. The ILP approaches effectively use background knowledge for representing complex objects and relations. The AGLEARN method uses the same concept but has a different representation. Moreover the background knowledge and concepts are represented in the form of attribute grammars. The given context-free grammar and background knowledge together allow one to restrict the space of relations and give a smaller representation of data. This is a new way of using knowledge as a bias for learning. In AGLEARN an example contains a string which can be derived from the target nonterminal and attributes of the target nonterminal computed in this string. In this approach we suppose that the underlying context-free grammar is given. The task of AGLEARN is then *to infer the semantic functions* associated with production. In the learning process the grammar, background semantic functions and examples can be used. In the current approach S-attributed an L-attributed grammars with simple rules can be learned. However, the background knowledge may contain more complex attribute grammars, as in OAG [8].

In this paper we present an application of the AGLEARN method to the part-of-speech (POS) tagging of Hungarian sentences. The disambiguation of possible tags is a non-trivial task because many words may have different meanings e.g the word *múlt*[2] can be annotated as a verb, noun or adjective in different sentences. We applied the AGLEARN method to infer context rules to choose the correct tags. A corpus with about 100 000 pre-tagged words has been employed for training and testing. By using the AGLEARN method learning data sets are generated to the C 4.5 [16] attribute value learner. These generated data sets contain information about the phrase structure of the sentences. A background attribute grammar has been used to determine the structural information. Our studies showed that using this structural background information the C 4.5 learner was able to infer more precise context rules.

In Section 2 a brief introduction to the AGLEARN algorithm is provided, while in Section 3 the POS tagging problem for Hungarian language is discussed. Then the application of the AGLEARN method to the POS tagging problem is described in Section 4. Then, in the final section the conclusions are drawn and suggestions for further research are given.

2. The AGLEARN method

This section briefly introduces the basic attribute grammar terminology that will be used throughout this paper. We also provide an attribute grammar example which will be employed to demonstrate how the learning method works. We follow the formal definition of attribute grammars can be found in [1].

Definition 1 Attribute Grammar. An attribute grammar is a five-tuple $AG = (G, SD, AD, R, C)$, where

(*i*) $G = (N, T, P, S)$ is the underlying context-free grammar.

(*ii*) $SD = (\mathcal{T}, \mathcal{F})$ is the semantic domain, where $\mathcal{T}$ is a finite set of sets (types) and $\mathcal{F}$ is a finite set of functions of type $\tau_1 \times \ldots \times \tau_n \rightarrow \tau_0$, $n \geq 0$ and $\tau_i \in \mathcal{T}$ for $i = 0, \ldots, n$. Functions of arity 0 will be referred as constants.

(*iii*) $AD = (Attr, Inh, Syn, \tau)$ is a description of attributes. Each grammar symbol $X \in N \cup T$ has a set of attributes $Attr(X)$, where $Attr(X)$ can be partitioned into two disjoint subsets denoted by $Inh(X)$ and $Syn(X)$. $Inh(X)$ and $Syn(X)$ denote the inherited and synthesized attributes of X, respectively. The set of attributes will be denoted by $Attr$, i.e. $Attr = \bigcup_{X \in N \cup T} Attr(X)$. Attributes associated with different symbols are considered as different, i.e. if $X \neq Y$ then

[2] past

$Attr(X) \cap Attr(Y) = \emptyset$. We will denote the attribute a of the grammar symbol X by $X.a$, so if a is an attribute, then $\tau(a) \in \mathcal{T}$ is the interpretation domain (type) of a.

(iv) R orders a set of evaluation rules (called semantic functions) to each production, as follows: Let $p : X_0 \to X_1 \dots X_{n_p}$ be an arbitrary production of P. Let us denote the attribute occurrence a of X_k by a triple (a, p, k) $(0 \le k \le n_p)$. If there is no confusion $X_k.a$ can be used instead of (a, p, k). An attribute occurrence (a, p, k) is then said to be a defined occurrence if $a \in Syn(X_k)$ and $k = 0$, or $a \in Inh(X_k)$ and $k > 0$. Otherwise an attribute occurrence is called a used occurrence. For each defining attribute occurrence there is exactly one rule in $R(p)$ which determines how one should compute the value of this attribute occurrence. The evaluation rule defining attribute occurrence (a, p, k) takes the form:

$$(a, p, k) = f((a_1, p, k_1), \dots, (a_m, p, k_m))$$

where $f \in \mathcal{F}$ is a function of the form

$$f : \tau(X_{k_1}.a_1) \times \dots \times \tau(X_{k_m}.a_m) \to \tau(X_k.a).$$

An attribute grammar is then said to be in normal form if all defined attribute occurrences depend only on used attribute occurrences in each production.

(v) C orders a finite set of conditions to each production, i.e. $c \in C(p)$ has the form:

$$c((a_1, p, k_1), \dots, (a_m, p, k_m))$$

where $c \in \mathcal{F}$ is a function (relation) of the form

$$c : \tau(X_{k_1}.a_1) \times \dots \times \tau(X_{k_m}.a_m) \to \{true, false\}.$$

A special class of attribute grammars introduced in [9] is the *S-attributed* grammars in which only synthesized attributes are allowed.

As S-attributed grammars are too restrictive in practice a larger class called *L-attributed* grammars being defined as follows:

Definition 2 L-attributed grammar. An attribute grammar is said to be L-attributed if and only if each inherited attribute of X_i in the production $p : X_0 \to X_1 \dots X_{n_p}$ depends only on the set $\bigcup_{1 \le j < i} Attr(X_j) \cup Inh(X_0)$ for $i = 1, \dots, n_p$.

Example 1. We demonstrate the learning method used in an example of *type checking for arithmetic expressions* [18]. We wish to solve the type determination problem for simple arithmetic expressions. Consider the following attribute grammar:

- **nonterminals:** $N = \{Expression, Term, Factor, AddOp, MulOp\}$

- **terminals:** $T = \{Real, Integer, +, -, \times, /, (,)\}$

- **start symbol:** $S = Expression$

- **semantic domain:** $\mathcal{T} = \{T_{mode}, T_{operator}\}$ where

1. $T_{mode} = \{int, real\}$
2. $T_{operator} = \{add, sub, mul, div\}$

$\mathcal{F} = \{add, sub, mul, div, int, real, id, f_1, f_2\}$ where

1. add, sub, mul, div, int and $real$ are constants
2. $id : T_{mode} \rightarrow T_{mode}$ denotes the identity function
3. $f_1 : T_{mode} \times T_{mode} \rightarrow T_{mode}$

   $f_1(op_1, op_2) :=$ **if** $op_1 = real$ **or** $op_2 = real$ **then** real
   else int

4. $f_2 : T_{operator} \times T_{mode} \times T_{mode} \rightarrow T_{mode}$

   $f_2(op_1, op_2, op_3) :=$ **if** $op_1 = mul$ **and**
   $op_2 = int$ **and** $op_3 = int$
   then int
   else $real$

- **attributes**
  $Attr = Syn = \{mode, operator\}$ so that

  1. $Syn(Expression) = Syn(Term) = Syn(Factor) = \{mode\}$
  2. $Syn(AddOp) = Syn(MulOp) = \{operator\}$

- **productions and attribute evaluation rules**

  1. $Expression_0 \rightarrow Expression_1\ AddOp\ Term$
     $R(1)$: $Expression_0.mode := f_1(Expression_1.mode, Term.mode)$

  2. $Expression \rightarrow Term$
     $R(2)$: $Expression.mode := Term.mode$

  3. $Term_0 \rightarrow Term_1\ MulOp\ Factor$
     $R(3)$: $Term_0.mode$
     $:= f_2(MulOp.operator, Term_1.mode, Factor.mode)$

  4. $Term \rightarrow Factor$
     $R(4)$: $Term.mode := Factor.mode$

  5. $Factor \rightarrow Real$
     $R(5)$: $Factor.mode := real$

  6. $Factor \rightarrow Integer$
     $R(6)$: $Factor.mode := int$

  7. $Factor \rightarrow (\ Expression\ )$
     $R(7)$: $Factor.mode := Expression.mode$

8. $AddOp \rightarrow +$
   $R(8)$: $AddOp.operator := add$

9. $AddOp \rightarrow -$
   $R(9)$: $AddOp.operator := sub$

10. $MulOp \rightarrow \times$
    $R(10)$: $MulOp.operator := mul$

11. $MulOp \rightarrow /$
    $R(11)$: $MulOp.operator := div$

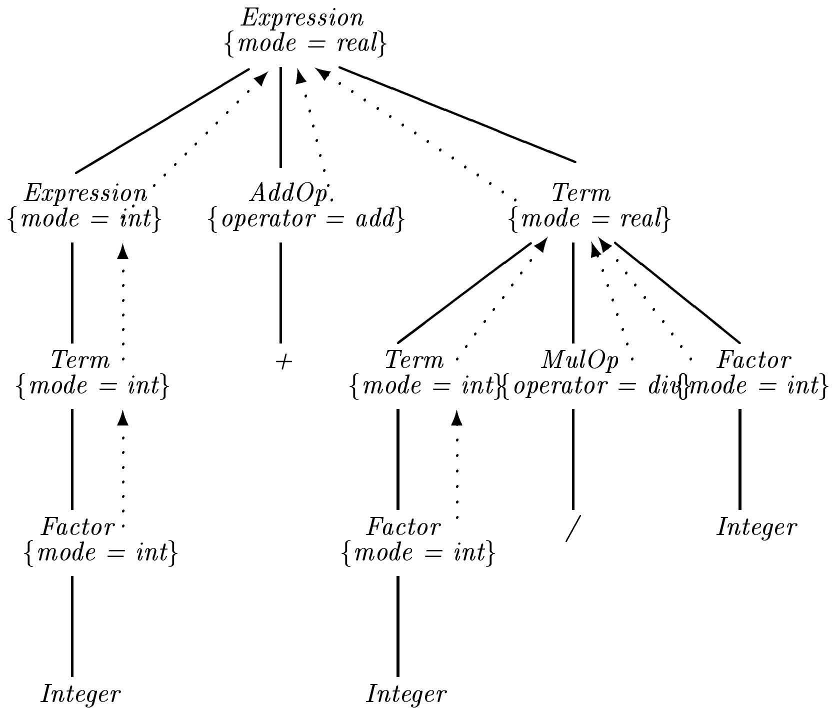

Figure 1: The decorated tree for the expression $Integer + Integer/Integer$

The decorated tree for the input word $Integer + Integer/Integer$ can be seen in Fig. 2.1. The dotted arrows denote the dependencies between attribute instances and the values of the attribute instances in the figure are shown as well. The value of an attribute instance can be computed if all the attribute instances it depends on have already been evaluated. Hence an evaluation strategy must be specified for a given attribute grammar. In the case of S and L attributed grammars the evaluation strategy is very simple and will be defined automatically. The attribute grammar in our example is of type S-attributed because it has only synthesized attributes.

2.1. Learning Semantic Functions of S-attributed Grammars

In this section we will discuss the process of inferring semantic functions and conditions for attribute grammars. The method presented here was motivated by an algorithm for ILP learning described in [6]. Essentially AGLEARN can be summarized in the following three steps:

(i) the learning problem of the semantic functions is transformed into a propositional form

(ii) a propositional learning method is applied to solve the problem in propositional form

(iii) the induced propositional hypothesis is transformed back into semantic functions.

The task of the learning algorithm is to infer semantic functions of the synthesized attributes of the target nonterminal in a given production p. Let p: $X_0 \to X_1X_2\ldots X_{n_p}$ be a production and let a be a synthesized attribute of X_0 to be learned. In addition, let $E_p^+(a)$ and $E_p^-(a)$ denote the set of positive and negative examples for $X_0.a$, respectively, such that if $(w,(a,v)) \in E_p(a)$ then $X_0 \Rightarrow X_1X_2\ldots X_{n_p} \Rightarrow^* w$. Note too that as we are dealing with S-attributed grammars in this section, each example takes the form $(w,(a,v))$ and as we wish to learn the semantic function associated with $X_0.a$ a table $T(a)$ must be constructed, each row of this table corresponding to an example from $E_p(a)$. The table has a set of columns

$$\{class, word, target\} \cup \mathcal{U} \cup \mathcal{F}_{UR} \cup \mathcal{F}_{UCR} \cup \mathcal{F}_{UF} \cup \mathcal{F}_{UCF}$$

where for a given example $e = (w,(a,v))$ the columns are defined as follows:

1. $class(e) = \begin{cases} + & \text{if } e \in E_p^+(a) \\ - & \text{otherwise (i.e. } e \in E_p^-(a)) \end{cases}$

2. $word(e) = w$

3. $target(e)$ is the value of a in e, i.e. $target(e) = v$

4. Let $X_k.b$ be an attribute instance, $0 < k \le n_p$. Then there is a corresponding column $X_k.b$ in $\mathcal{U}$. The value of this column is computed using the semantic functions in the background attribute grammar. With the example $e = (w,(a,v))$ this computation is performed like so:

   (a) An attributed tree is built on the input string w using grammar G and semantic functions R.

   (b) If the subtree derived from symbol X_k contains only nodes corresponding to rule instances belonging to the background rules, then the attributes of this subtree can be evaluated.

   (c) If the subtree derived from symbol X_k contains a node corresponding to a rule instance that has unknown semantic functions the values of the attributes of this node are then asked from the user (*oracle*) for the given derivation.

5. Let $X_{k_1}.a_1,\ldots,X_{k_l}.a_l$ $(0 < k_1,\ldots,k_l \le n_p)$ be applied attribute occurrences and let f: $\tau_1 \times \ldots \times \tau_l \to \{true, false\}$ be a Boolean function ($f \in \mathcal{F}$) such that $\tau_i = \tau(X_{k_i}.a_i)$ $(1 \le i \le l)$. Then there is a column for the relation $f(X_{k_1}.a_1,\ldots,X_{k_l}.a_l)$ in $\mathcal{F}_{UR}$.

6. Let $X_k.b$ be an applied attribute occurrence $(0 < k \le n_p)$, and let $\{c_1,\ldots,c_m\}$ be a set of constant values occurring in the column $X_k.b$ of $\mathcal{U}$. Then for each c_i there is a corresponding column for the relation $X_k.b = c_i$ in $\mathcal{F}_{UCR}$ $(1 \le i \le m)$.

7. Let $X_{k_1}.a_1,\ldots,X_{k_l}.a_l$ $(0 < k_1,\ldots,k_l \le n_p)$ be applied attribute occurrences and let f: $\tau_1 \times \ldots \times \tau_l \to \tau_0$ be a function ($f \in \mathcal{F}_i$) such that $\tau_i = \tau(X_{k_i}.a_i)$ $(1 \le i \le l)$ and $\tau_0 = \tau(a)$. Then there is a column for the relation $a = f(X_{k_1}.a_1,\ldots,X_{k_l}.a_l)$ in $\mathcal{F}_{UF}$.

8. Let $\{c_1,\ldots,c_m\}$ be a set of constant values occurring in $\mathcal{U}$ or in the column of $target(e)$. Then for each c_i there is a corresponding column $a = c_i$ in $\mathcal{F}_{UCF}$ if and only if $\tau(a) = \tau(c_i)$ $(1 \le i \le m)$.

Example 2 continued. We now demonstrate how to construct the corresponding table $T(Term_0.mode)$ for the recursive production:

$$Term_0 \rightarrow Term_1\ MulOp\ Factor.$$

The same method can be applied for the productions 1, 2 and 4. The attribute instances $Term_1.mode$, $MulOp.operator$, and $Factor.mode$ denote the type of expression derived from $Term_1$, the type of the multiplicative operator, and the type of the expression from $Factor$, respectively. As the underlying production is recursive we have to ask the oracle for the value of attribute instance $Term_1.mode$ for a given input. Table 1 shows the learning problem transformed into propositional form. In the column U_1 the sign '*' denotes the values were asked from the oracle. In this particular example we have no item for $\mathcal{F}_{UR}$ as $\mathcal{F}_i$ does not contain any relation.

class	*word*	*target*	$\mathcal{U}$			$\mathcal{F}_{UCR}$						$\mathcal{F}_{UF}$		$\mathcal{F}_{UCF}$	
		$T_0.m$	U_1	U_2	U_3	R_1	R_2	R_3	R_4	R_5	R_6	F_1	F_2	C_1	C_2
+	3×2.5	*real*	int_*	*mul*	*real*	*T*	*F*	*T*	*F*	*F*	*T*	*F*	**T**	*F*	*T*
+	5×3	*int*	int_*	*mul*	*int*	**T**	*F*	**T**	*F*	**T**	*F*	**T**	*F*	*T*	*T*
+	1.5×4	*real*	$real_*$	*mul*	*int*	*F*	*T*	*T*	*F*	*T*	*F*	*F*	**T**	*T*	*F*
+	$2.5/3$	*real*	$real_*$	*div*	*int*	*F*	*T*	*F*	*T*	*T*	*F*	*F*	**T**	*T*	*F*
+	$2/3$	*real*	int_*	*div*	*int*	*T*	*F*	*F*	*T*	*T*	*F*	*F*	**T**	*F*	*F*
+	$6/3.4$	*real*	int_*	*div*	*real*	*T*	*F*	*F*	*T*	*F*	*T*	*F*	**T**	*F*	*T*
−	2×3.2	*int*	int_*	*mul*	*real*	*T*	*F*	*T*	*F*	*F*	*T*	*T*	**F**	*T*	*F*
−	$4.3/2$	*int*	$real_*$	*div*	*int*	*F*	*T*	*F*	*T*	*T*	*F*	*T*	**F**	*F*	*T*
−	$8/3$	*int*	int_*	*div*	*int*	*T*	*F*	*F*	*T*	*T*	*F*	*T*	**F**	*T*	*T*

Table 1: **The generated propositional table**

- U_1: $Term_1.mode$
- U_2: $MulOp.Operator$
- U_3: $Factor.mode$
- R_1: $Term_1.mode = int$
- R_2: $Term_1.mode = real$
- R_3: $MulOp.Operators = mul$
- R_4: $MulOp.Operators = div$
- R_5: $Factor.mode = int$
- R_6: $Factor.mode = real$
- F_1: $Term_0.mode = int$
- F_2: $Term_0.mode = real$
- C_1: $Term_0.mode = Term_1.mode$
- C_2: $Term_0.mode = Factor.mode$
- T: $true$
- F: $false$

Example 3 continued. We suppose an attribute learner to be able to find the solution

$\oplus_1$: $R_1 = true$ & $R_3 = true$ & $R_5 = true$ & $F_1 = true$
$\oplus_2$: $F_2 = true$

The final step of the learning procedure is to transform these rules into semantic functions. The transformed semantic function $R(3)$ has the form of

if $Term_1.mode = int$ & $MulOp.Operators = mul$ & $Factor.mode = int$
then
  $Term_0.mode = int$ // from $\oplus_1$
else
  $Term_0.mode = real$ // from $\oplus_2$

which is a correct solution.

In this paper we discussed only the learning of the semantic functions for S-attributed grammars because this approach was used to the POS tagging problem. The learning of semantic functions for L-attributed grammars was investigated in [7].

```
<par from='Ohu.1.2.1'>
<s from='Ohu.1.2.1.1'>
  <tok type=WORD>
    <orth>Der&uuml;lt</orth>
    <disamb><base>der&uuml;lt</base><msd>Afp-sn</msd><ctag>AN</ctag>
    </disamb>
    <lex><base>der&uuml;lt</base><msd>Afp-sn</msd></lex>
    <lex><base>der&uuml;l</base><msd>Vmis3s---n</msd></lex>
  </tok>
  <tok type=PUNCT>
    <orth>,</orth>
    <ctag>COMMA</ctag>
  </tok>
  <tok type=WORD>
    <orth>hideg</orth>
    <disamb><base>hideg</base><msd>Afp-sn</msd><ctag>AN</ctag></disamb>
    <lex><base>hideg</base><msd>Afp-sn</msd></lex>
  </tok>
...
</s>
<s from='Ohu.1.2.1.2'>
...
</par>
<par from='Ohu.1.2.2'>
...
```

Figure 2: The main structure of the corpus SGML file

3. The Hungarian POS tagging problem

The part-of-speech tagging is an important step in natural language processing. When a sentence is read each word is labeled by its morpho-syntanctic description (e.g. *csináltam*[3] is a verb: past tense, singular, 1st person). This process is called tagging and has key-role in the parsing of the sentences. Since in each language there are words that may have several (2–4) different taggings a working tagger must contain a disambiguation module besides the morphological analyzer.

[3] I did

There are two main approaches which exist for the disambiguation module: the probabilistic (HMMs — hidden Markov models) and the rule based ones. This paper focuses on a rule-based disambiguation module of a tagger for which the rules are learned by machine learning algorithms. Two learning algorithms, the C 4.5[16] and the AGLEARN[7] were tested and the results compared. The following section is now devoted to the TELRI Hungarian corpus that has been used as training and test data for learning algorithms.

3.1. Preprocessing the Corpus

The "MULTEXT-East" (MULtilingual TEXT tools and Corpora for Eastern and Central European Languages) Copernicus Project lasted from 1995–1997. The aim of this project was to establish a Corpus Encoding Specification (CES) and to propose a tag system for most European languages, the results of this project having been published on two CD-ROMs[17]. The material contains George Orwell's novel "1984" translated into many East European languages: Bulgarian, Czech, Estonian, Hungarian, Romanian and Slovene. Each translation is approximately 100 000 words including punctuation characters. All corpora meet the CES (Corpus Encoding Specification) standard and were tagged with the proposed MSD tagging system.

The Hungarian corpus itself is an SGML file. The file contains a precise reference to particular sentences, each sentence being denoted by a label like 'Ohu1.1.2.1' (e.g. Orwell, Hungarian, 1st chapter, 1st section, 2nd paragraph, 1. sentence). The novel consists of four chapters, Chapter 1 and 2 having been used as training data while chapter 3 and 4 served as test data.

In Figure 2 the main structure of the corpus file is shown, the whole novel having been divided into paragraphs, sentences and tokens. The paragraphs are delimited by `<par >, </par>` tags, sentences delimited by `<s >, </s>` tags, and the tokens are delimited by `<tok >, </tok>` tags. Each paragraph and sentence is uniquely identified. Each sentence consists of a series of tokens. Tokens are not numbered.

3.2. The MULTEXT-East Morpho-Syntactical Description (MSD) Categories

In the "MULTEXT-East" project a coding convention was introduced that could be used for coding word attributes for a wide variety of languages. A code string represented a word and all of its syntactic attributes (type, gender, number, case, definiteness, etc).

Category	Code	Category	Code
adjective	A	particle	Q
conjunction	C	adverb	R
determiner	D	adposition	S
interjection	I	article	T
numeral	M	verb	V
noun	N	residual	X
pronoun	P	abbreviation	Y

Table 2: The main categories of words in MSD

For example the Hungarian word *asztalnak* will get the `Nc-sg------` code, which means: noun, common, single, genitive. Those attributes that are not present or not applicable are denoted by hyphens. The first position is reserved for the main word categories. (See [12] for a detailed description of the MSD encoding method.) The main word categories can be seen in Table 2. In Figure 2 the word *"hideg"(cold)* is tagged to `Afp-sn`. (The trailing hyphens were cut `Afp-sn-------`). This means the word is an *adjective* that has the following attributes: *qualificative, positive, singular, nominative.*

Category	Number of tokens		
	Training	Test	Altogether
adjective	7157	2382	9539
conjunction	5408	1994	7402
determiner	0	0	0
interjection	72	17	89
numeral	1046	323	1369
noun	14881	5170	20051
pronoun	4552	1923	6475
particle	0	0	0
adverb	7869	2969	10838
adposition	827	280	1107
article	6856	2336	9192
verb	10288	4254	14542
residual	13	12	25
abbreviation	66	13	79
	59035	21673	80708
punctuation	12484	5235	17719
	71519	26908	98427

Table 3: The number of tokens in the Hungarian corpus

Words in the Hungarian language may have many attributes. For example a noun might have 1324 different MSD codes according to its stems, adjectives 2772 and pronouns approximately 3000 different codes. The number of so many subclasses of words makes the learning task very hard along with the understanding and evaluation of the results.

In Table 3 the distribution of main word categories in the corpus is presented. The training and testing part is shown separately.

3.3. The Corpus Tag (CTAG) encoding for Hungarian language

In order to reduce the number of MSD classes the CTAG encoding scheme was introduced [13]. There are 120 word tags, 4 punctuation tags and 1 tag for *unknown* words. A complete list of the CTAGs shown in Figure 3.

The first letter of a CTAG denotes the main category of the word like in MSD, while the remaining long suffix is cut and replaced by new suffixes. For example NPNX means *noun* that has the following attributes: *plural, nominative, and some possessive ending.* The second letter `P or N` encodes the *number*, namely plural or singular, while the third letter `N, A, D, O` encodes the *case*, namely nominative, accusative, dative and other respectively. The fourth letter stands for some possessive stems not referring to the *person* and *number* of possessive person.

3.4. The initial data set

The original Hungarian corpus was converted into prolog. First a lexicon of words was established. The lexicon contained all words in the same form as it had appeared in the corpus along with all of its stems and its CTAG. If a word could be annotated by more than one CTAG it was then added several times to the lexicon. In the second pass the whole SGML file was converted to prolog using the lexicon. The conversion was performed sentence by sentence, sentences being recognized upon the `<s >` and `</s>` tags. For example the Hungarian sentence *"Derült, hideg áprilisi nap volt, az órák éppen tizenhármat ütöttek."* [4] is converted to the following prolog fact.

```
s('Ohu.1.2.1.1', [(asn, [asn, vmis3s]), wpunct, asn, asn, nsn,
   vmis3s, wpunct, (t, [psn, t]), npn, rg, ms, vmis3p, spunct]).
```

The first argument of the `s` predicate is the sentence identifier, the second is the list of CTAGs of words. If the tagging of a particular word is ambiguous then the CTAG is represented by a pair of the right CTAG and the list of other possible tagging according to the lexicon. In this way both the training and the testing parts of the corpus can be converted to this simpler form. In our case there were 4583 sentences in the training set and 2185 sentences in the test data set, i.e. 6768 sentences altogether.

The learning examples for C 4.5 and AGLEARN were generated by prolog programs from the above data. [5]

The number of noun CTAGs			
CTAG	Training	Test	Altogether
NPA	304	106	410
NPAX	106	30	136
NPAY	0	0	0
NPD	50	9	59
NPDX	37	7	44
NPDY	0	0	0
NPN	832	279	1111
NPNX	193	70	263
NPNY	4	1	5
NPO	470	128	598
NPOX	119	36	155
NPOY	1	3	4
NSA	1310	475	1785
NSAX	555	206	761
NSAY	4	0	4
NSD	359	90	449
NSDX	131	35	166
NSDY	0	0	0
NSN	5286	1928	7214
NSNX	1319	445	1764
NSNY	13	10	23
NSO	2686	932	3618
NSOX	1098	380	1478
NSOY	4	0	4
all	14881	5170	20051

Table 4: The distribution of noun CTAGs in the corpus

[4] It was a bright cold day in April, and the clocks were striking thirteen.
[5] All programs mentioned in this paper were written in SICStus Prolog 3.7.1.

The word CTAGs: APA, APAX, APAY, APD, APDX, APDY, APN, APNX, APNY, APO, APOX, APOY, ASA, ASAX, ASAY, ASD, ASDX, ASDY, ASN, ASNX, ASNY, ASO, ASOX, ASOY, CP, I, MD, MP, MPX, MPY, MS, MSX, MSY, NPA, NPAX, NPAY, NPD, NPDX, NPDY, NPN, NPNX, NPNY, NPO, NPOX, NPOY, NSA, NSAX, NSAY, NSD, NSDX, NSDY, NSN, NSNX, NSNY, NSO, NSOX, NSOY, PPA, PPAX, PPAY, PPD, PPDX, PPDY, PPN, PPNX, PPNY, PPO, PPOX, PPOY, PSA, PSAX, PSAY, PSD, PSDX, PSDY, PSN, PSNX, PSNY, PSO, PSOX, PSOY, RG, RO, RP, RQ, RV, ST, T, VA, VMCP1P, VMCP1S, VMCP2, VMCP2P, VMCP2S, VMCP3P, VMCP3S, VMIP1P, VMIP1S, VMIP2, VMIP2P, VMIP2S, VMIP3P, VMIP3S, VMIS1P, VMIS1S, VMIS2, VMIS2P, VMIS2S, VMIS3P, VMIS3S, VMMP1P, VMMP1S, VMMP2, VMMP2P, VMMP2S, VMMP3P, VMMP3S, VMN, X, Y.

The punctuation CTAGs: OPUNCT, CPUNCT, WPUNCT, SPUNCT.

The unknown CTAG: UNKNOWN.

Figure 3: The list of Hungarian CTAGs

4. The application of the AGLEARN method to the POS tagging problem

4.1. Learning by C 4.5

In POS tagging a Hungarian sentence, a so-called `removable` predicate is learned. Such an approach having been first presented in James Cussens' paper [2]. What happens is that whenever a word is tagged to more than one CTAG, some of them (hopefully all but one) can be removed. Such a predicate can be learned by the C 4.5, the training data having been taken from the training part of the Hungarian corpus. The learned predicate will say that e.g. PSN can be removed from the ambiguities if some conditions hold, these conditions coming from the neighbouring CTAGs. A window of 7 CTAGs have been set, 7 CTAGs before and 7 CTAGs after the current ambiguity. Narrowing the window to a smaller size can be done later inside the C 4.5 by setting some columns in the data file to `ignore`.

The ambiguity-classes and their occurences in the corpus are presented in Table 5. The most frequent ambiguity-classes cover 6667 from the 7229 ambiguity occurences (92.22%) in the training set, and cover 2443 from the 2677 cases (91.60%) in the test set. The learned tagger is generally tries to remove as many ambiguities as it can, and the remaining ones can be eliminated upon the orthographies of the words perhaps in a probabilistic way.

Most ambiguities are caused by few (1–10) words. There are only 3–4 major classes that contain more words. The `[PSN,T]` and `[MS,T]` classes contain only one word each.

Learning `removable` rules for an ambiguity-class

For generating learning examples for the C 4.5 the prolog form of the corpus was used. Learning rules for removing the `T` [6] CTAG from the `[PSN,T]` ambiguity-class is presented in the following to illustrate how the whole process works.

```
s('Ohu.1.2.6.3', [(ms, [ms, t]), asn, nso, nsnx, wpunct,
(t, [psn, t]), nsn, vmis3s, asn, npox, aso, t, asn, nsn, st, spunct]).
```

[6] The `T` CTAG in the Hungarian language is equal to the `Tf----` MSD class that contains the articles in the Hungarian language. Only 3 words belong to this class: *a(the), az(the), egy(a, an)*. In the `[PSN,T]` ambiguity-class there is only one word: *az* that means *that* and *the*.

The number of ambiguity classes					
	Occurence			Occurence	
Class	Training	Test	Class	Training	Test
[ASN,MS]	55	12	[MS,RG]	70	19
[ASN,NSN]	96	27	[MS,T]	751	222
[ASN,NSN,PSN]	52	22	[NSN,PSN]	111	52
[ASN,NSN,VMIS3S]	50	30	[NSN,RG]	59	18
[ASN,PSN]	68	33	[NSN,RG,RP]	112	46
[ASN,RG]	92	23	[NSN,VMIP3S]	52	41
[ASN,VMIS3S]	490	182	[PSN,PSO]	69	38
[ASO,RG]	74	32	[PSN,RP]	143	57
[CP,PSO]	50	30	[PSN,T]	1867	620
[CP,PSO,RG]	87	44	[PSO,RG]	217	85
[CP,RG]	880	294	[PSO,RG,RP]	93	45
[CP,RG,VMIP3S]	247	125	[RG,RP]	150	59
[CP,RP]	334	149	[RG,ST]	285	100
[CP,VMIS3S]	113	38			
			Altogether	6667	2443

Table 5: The most frequent ambiguity classes (which occured more than 50 times in the training data)

Each time a `[PSN,T]` ambiguity occurred in a sentence a learning example was generated. In the previous sentence the tagging of the sixth word was ambiguous, as it could have been either `PSN` or `T`. From this sentence the following example was generated for the C 4.5:

```
...
Ohu.1.2.5.9, xxx, xxx, xxx, xxx, xxx, xxx, xxx,
     nsd, pso, t, nso, vmis3s, vmn, wpunct, c_t
Ohu.1.2.6.3, wpunct, nsnx, nso, asn, ms, xxx, xxx,
     nsn, vmis3s, asn, npox, aso, t, asn, c_t
Ohu.1.2.6.4, wpunct, nsnx, nsn, asn, t, wpunct, nsn,
     asn, nsny, wpunct, psn, nsn, nsox, t, c_t
...
```

The first column in the data file is the sentence identifier, the last being the target class (here `c_t`) because the right tagging was `T`. One training example is generated for each ambiguity. The data file contains 16 attributes. The neighbouring tokens' window could be found between the first and the last column: $before_1, \ldots before_7$ and $after_1 \ldots after_7$. A new `xxx` token was introduced to denote missing tokens. In Table 6 the results are presented. The learned rules are shown in *if* statement form below:

```
if ((token1_after == 'cp') ||
   ((token1_before == 'wpunct') && (token1_after == 'rg')) ||
   ((token1_before == 'xxx') && (token1_after == 'rg')) ||
    (token1_after == 'spunct') ||
   ((token1_before == 'wpunct') && (token1_after == 'st')) ||
    (token1_after == 't') ||
    (token1_after == 'vmcp3s') ||
   ((token1_before == 'wpunct') && (token1_after == 'vmip3s')) ||
    (token1_after == 'vmis3s') ||
    (token1_after == 'vpunct')) then Class = c_psn else Class = c_t
```

Window size	Number of rules	Accuracy			
		Training examps. (1867)		Test examps. (620)	
		%	#errors	%	#errors
1	25	98.50%	17	97.30%	28
2	21	98.90%	15	97.60%	35
3	20	98.90%	16	97.40%	34
4	18	99.00%	16	97.40%	35
5	18	99.00%	16	97.40%	33
6	20	98.90%	17	97.30%	32
7	20	98.90%	17	97.30%	32

Table 6: The results of C 4.5 on learning `removable_t__psn_t`

4.2. Learning by AGLEARN method

In the method presented in the previous section the training examples contained CTAGs. The C 4.5 system was used to infer decision rules to solve the `[PSN,T]` ambiguity problem. The rules learned by the C 4.5 system only having contained tests for the values of CTAGs.

When applying of the AGLEARN method to this problem an attribute grammar was constructed which had the following starting (target) production (a larger part of this attribute grammar being listed in Appendix A):

$Sentences \rightarrow Sentence\_id$ "." $BeforeCtags$ "," $AfterCtags\ Sentences$
$Sentences \rightarrow \lambda$

The nonterminal *Sentences* has an attribute *psn_or_t*. This attribute may have two values (psn,t) so we learned rules using AGLEARN to determine the correct tag for the sentences where the `[PSN,T]` ambiguity problem appears. The input sets for this attribute grammar were the training examples described in the previous section. The nonterminal *BeforeCtags* processes those CTAGs that precede the `[PSN,T]` ambiguity position in the sentence. In our case we only investigated the value of the nearest preceding CTAG ($Before1Ctag$) the other CTAGs having been skipped. The CTAGs were partitioned into groups according to their role in the sentences, these groups being listed in the Appendix B. Then by using the background attribute grammar the group value for the $Before1Ctag$ was computed. Similarly, the nonterminal *AfterCtags* was for the processing of the CTAGs following the ambiguity position. The group value for the $After1Ctag$ was also computed. In addition in the subsequent CTAGs we tried to identify a characteristic phrase structure called *syntagma* using the background attribute grammar. A list of the possible syntagmas is described in Appendix B. Table 7 below contains a description of the attribute instances that can be used at the initial production.

Name	Type	Description
BeforeCtags.before1_ctag	CTAG	CTAG value of the Before1ctag
BeforeCtags.group	GROUP	group value of the Before1ctag
AfterCtags.after1_ctag	CTAG	CTAG value of the After1ctag
AfterCtags.group	GROUP	group value of the After1ctag
AfterCtags.syntagma	SYNTAGMA	a recognized syntagma in the subsequent CTAGs

Table 7: The main attributes used by AGLEARN

The learning table generated by the AGLEARN method contains these attribute columns and two further columns for the relations:

$BeforeCtags.before1\_ctag = AfterCtags.after1\_ctag$ and
$BeforeCtags.group = AfterCtags.group$.

For each sentence in the training examples a row was generated for this table. This table was used by the C 4.5 system to help generate decision rules to the [PSN,T] problem. From the generated rules the following semantic function could be prepared for the attribute *psn_or_t*. A part of the generated C 4.5 data is shown below:

```
Ohu.1.2.5.7,  rg, Oth, cp, Oth, AttSynt, false, true, c_psn
Ohu.1.2.14.1, cp, Oth, t, Oth, SubjSynt, false, true, c_psn
Ohu.1.2.14.2, wpunct, Oth, t, Oth, noneSynt, false, true, c_psn
Ohu.1.2.15.1, wpunct, Oth, vmis3s, Pred, AccSynt, false, false, c_psn
...
Ohu.1.2.1.1,  wpunct, Oth, npn, Subj, SubjSynt, false, false, c_t
Ohu.1.2.2.1,  xxx, none, nso, AdvOth, noneSynt, false, false, c_t
Ohu.1.2.2.6,  cp, Oth, nsn, Subj, SubjSynt, false, false, c_t
Ohu.1.2.2.9,  nsax, Acc, asn, Att, AttSynt, false, false, c_t
Ohu.1.2.2.10, vmip3s, Pred, nsa, Acc, AccSynt, false, false, c_t
...
```

The learned semantic rule can be seen below:

```
if ((after1_ctag == 't') ||
    (after1_ctag == 'wpunct') ||
    (after1_ctag == 'spunct') ||
    (after1_ctag == 'cp') ||
   ((before1_ctag == 'rg') && (after1_group == Oth) &&
   (after1_syntagma == AdvOthSynt)) ||
   ((before1_ctag == 'wpunct') && (after1_group == Att) &&
   (after1_syntagma == noneSynt)) ||
    (after1_group == Pred))
  psn_or_t = c_psn ;
else
  psn_or_t = c_t ;
```

5. Conclusion and Future Work

In Table 8 the results of the C 4.5 and the AGLEARN algorithms on the largest ambiguity classes (more than 100 elements) are displayed. The sign $-$ in the column MARK denotes those classes where the two methods have the same results. We denote by $+$ if the use of AGLEARN led to certain minor improvements, while $++$ was used to denote more significant improvements. We can conclude that by using AGLEARN the accuracy of the inferred rules can be increased. The main reason of this is that AGLEARN makes intensive use of the background knowledge incorporated in the background attribute grammar rules. Note that the attribute grammar described in the previous section can also be applied to learn decision rules for any ambiguity class where only the name of target attribute and the training examples have to be modified. Furthermore this attribute grammar can be used as a final tagger tool. In this case the all decision rules (semantic functions) inferred by AGLEARN have to be integrated into the target production and the learning table generation functions must be removed.

Ambiguity class	Results by C 4.5				Results by AGLEARN				Mark
	Training		Test examples		Training		Test examples		
	#err	%	#err	%	#err	%	#err	%	
[ASN,VMIS3S]	40	8.2%	18	9.9%	40	8.2%	18	9.9%	-
[CP,RG,VMIP3S]	14	5.7%	31	24.8%	14	5.7%	31	24.8%	-
[CP,RG]	142	16.8%	74	25.2%	141	16.0%	70	23.8%	+
[CP,RP]	41	12.3%	16	10.7%	19	5.7%	8	5.4%	++
[CP,VMIS3S]	2	1.8%	0	0.0%	2	1.8%	0	0.0%	-
[NSN,PSN]	24	21.6%	16	30.8%	2	1.8%	5	9.6%	++
[PSN,RP]	9	6.3%	3	5.3%	9	6.3%	3	5.3%	-
[PSN,T]	25	1.5%	17	2.7%	21	1.1%	16	2.6%	+
[PSO,RG]	73	33.6%	34	40.0%	35	16.1%	19	22.4%	++
[RG,RP]	57	38.0%	15	25.4%	57	38.0%	15	25.4%	-
[RG,ST]	104	36.5%	44	44.0%	84	29.5%	40	40.0%	++

Table 8: Comparison of the results obtained by C 4.5 and AGLEARN

We are going to built a complete tagger for Hungarian language which integrates a morphologic analyser with the disambiguation rules generated by AGLEARN. With the help of linguists these rules should be carefully investigated, and we also plan to develop more relevant background rules for the syntactical structures of the Hungarian language.

The authors would like to thank Csaba Oravecz and Tamás Váradi at the Research Institute for Linguistics of the Hungarian Academy of Sciences for their kind assistance and discussion.

Bibliography

[1] Alblas, H. 1991. Introduction to Attribute Grammars. LNCS 545, Springer Verlag, 1–16.

[2] Cussens, J.: Part-of-Speech Tagging Using Progol in Proc. of the Seventh International Workshop on Inductive Logic Programming (ILP97) Prague, Czech Republic, in the LNAI series Vol **1297** 37–44 Springer Verlag (1997)

[3] Deransart, P.,Jourdan, M.,Lorho, B. 1988. Attribute Grammars - Definitions, Systems and Bibliography. LNCS 323, Springer Verlag.

[4] Deransart, P.,Maluszyński, J. 1985. Relating Logic Programs and Attribute Grammars. Journal of Logic Programming 2, 119–156.

[5] Deransart, P.,Maluszyński, J. 1993. A Grammatical View of Logic Programming. The MIT Press.

[6] Džeroski, S.,Lavrač, N. 1993. Inductive Learning in Deductive Databases. IEEE Transactions on Knowledge and Data Engineering, Vol. 5. No. 6.

[7] Gyimóthy, T.,Horváth, T. 1997. Learning Semantic Functions of Attribute Grammars. Nordic Journal of Computing 4(1997), 287-302.

[8] Kastens, U. 1980. Ordered Attribute Grammars. Acta Informatica 13 (1980), 229–256.

[9] Knuth, D.E. 1968. Semantics of Context-Free Languages. Mathematical Systems Theory 2, 2, 127–145. Correction: Mathematical Systems Theory 5, 1, 1971, 95–96.

[10] MUGGLETON, S. 1992. Inductive Logic Programming. Academic Press, London.

[11] MUGGLETON, S., DE RAEDT, L. 1994. Inductive Logic Programming: Theory and Methods. Journal of Logic Programming, 12.

[12] Specification and Notation for Lexicon Encoding Copernicus Project 106 "MULTEXT-EAST" Work Package WP1 – Task 1.1 Deliverable D1.1 F Task leaders: Tomaž Erjavec, Monica Monachini

[13] Oravecz, Cs.: Part-of-Speech Tagging in the Hungarian National Corpus — a Case Study

[14] PAAKKI, J. 1990. A Logic-Based Modification of Attribute Grammars for Practical Compiler Writing. In Proc. of the Seventh Int. Conference on Logic Programming (D.H.D. Warren, P.Szeredi, eds.), Jerusalem, 1990. The MIT Press, 203–217.

[15] PEREIRA, F.C.N.,WARREN, D.H.D. 1980. Definite Clause Grammars for Language Analysis - A Survey of the Formalism and a Comparison with Augmented Transition Networks. Artificial Intelligence 13, 231–278.

[16] QUINLAN, J.R. 1993. C 4.5: Programs for Machine Learning. Morgan Kaufmann Publisher.

[17] East meets West — A Compendium of Multilingual Resources
Eds: Tomaž Erjavec, Ann Lawson and Laurent Romary
`http://www.ids-mannheim.de/telri/cdrom.html`

[18] WILHELM, R. 1971. Attribuirte Grammatiken. Informatik Spektrum 2, 123-130.

Appendix

Appendix A: A part of the background attribute grammar

```
Sentences = Sentence_ID ","  BeforeCtags "," AfterCtags Sentences ;
do
end

Sentences = ;
do
end
...

AfterCtags = Acc_Group "," Synt_Acc;
do
    syntagma = Synt_Acc.syntagma;
    ctag= Acc_Group.ctag;
    group = Acc;
end

AfterCtags = AdvDat_Group "," Synt_AdvDat;
do
    syntagma = Synt_AdvDat.syntagma;
    ctag= AdvDat_Group.ctag;
    group =  AdvDat;
end

AfterCtags = AdvOth_Group "," Synt_AdvOth;
```

```
do
    syntagma = Synt_AdvOth.syntagma;
    ctag= AdvOth_Group.ctag;
    group =  AdvOth;
end
...

Synt_Acc = Pred_Group "," Ctags;
    do
      syntagma= AccSynt;
    end
Synt_Acc = NonPred_Group "," Synt_Acc;
    do
      syntagma= Synt_Acc.syntagma;
    end
Synt_Acc = Class_ID;
    do
      syntagma= noneSynt;
    end

Synt_AdvDat = Pred_Group "," Ctags;
    do
      syntagma= AdvDatSynt;
    end
Synt_AdvDat = NonPred_Group "," Synt_AdvDat;
    do
      syntagma= Synt_AdvDat.syntagma;
    end
Synt_AdvDat = Class_ID;
    do
      syntagma= noneSynt;
    end

Synt_AdvOth = Pred_Group "," Ctags;
    do
      syntagma= AdvOthSynt;
    end
Synt_AdvOth = NonPred_Group "," Synt_AdvOth;
    do
      syntagma= Synt_AdvOth.syntagma;
    end
Synt_AdvOth = Class_ID;
    do
      syntagma= noneSynt;
    end

Synt_Subj = Pred_Group "," Ctags;
    do
      syntagma= SubjSynt;
    end
Synt_Subj = NonPred_Group "," Synt_Subj;
    do
      syntagma= Synt_Subj.syntagma;
    end
Synt_Subj = Class_ID;
    do
```

```
      syntagma= noneSynt;
    end
...
```

Appendix B: The list of CTAG groups

```
% group of ctags which might be the subject of the sentence
Subj_Group =
npn ,npnx,npny,nsn,nsnx,nsny,
ppn,ppnx,ppny,psn,psnx,psny;

% group of ctags which might be the predicate of the sentence
Pred_Group =
vmcp1s,vmcp1p,vmcp2,vmcp2s,vmcp2p,vmcp3s,vmcp3p,
vmip1s,vmip1p,vmip2,vmip2p,vmip2s,vmip3s,vmip3p,
vmis1s,vmis1p,vmis2,vmis2p,vmis2s,vmis3s,vmis3p,
vmmp1s,vmmp1p,vmmp2,vmmp2p,vmmp2s,vmmp3s,vmmp3p,
va;

% group of ctags which might be the accusative of the sentence
Acc_Group =
vmn,apa,apax,apay,asa,asax,asay,
npa,npax,npay,nsa,nsax,nsay,
ppa,ppax,ppay,psa,psax,psay;

% group of ctags which might be the dative adverb of the sentence
AdvDat_Group =
apd,apdx,apdy,asd,asdx,asdy,
npd,npdx,npdy,nsd,nsdx,nsdy,
ppd,ppdx,ppdy,psd,psdx,psdy;

% group of ctags which might be the other adverb of the sentence
AdvOth_Group =
apo,apox,apoy,aso,asox,asoy,
npo,npox,npoy,nso,nsox,nsoy,
ppo,ppox,ppoy,pso,psox,psoy;

% group of ctags which might be the attribute of the sentence
Att_Group =
apn,apnx,apny,asn,asnx,asny,
mp,mpx,mpy,ms,msx,msy;

% group of other ctags in the sentence
Oth_Group
md,i,cp,cpunct,spunct,wpunct,
rg,ro,rp,rq,rv,st, t,
unknown,x,y,xxx;

Groups =  Subj, Pred, Acc, AdvDat, AdvOth, Att, Oth, none ;

Syntagmas = SubjSynt, PredSynt, AccSynt, AdvDatSynt,
 AdvOthSynt, AttSynt, OthSynt, noneSynt ;
```