Draft. Please no distribute.

Application of ILP Techniques on Hungarian Part-of-speech Tagging

Zoltán Alexin[1], 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: gyimi@inf.u-szeged.hu

Abstract. This paper collects the first results in the study of speech-tagging problem on the Hungarian domain. Orwell's „1984" translations onto different European languages was processed, tagged and disambiguated during a COPERNICUS project. The TELRI corpus was served as training and test data. The Hungarian corpus were cut into training and test pieces and a lexicon of words have been established. The C 4.5 attribute-value learner and the Progol ILP tool were applied and the results are compared. The IMPUT abductive learning tool and RIBL also planned to apply and to add to the comparision. [3]

1 Introduction

The "MULTEXT-East" (MULtilingual TEXT tools and Corpora for Eastern and Central European Languages) Copernicus Project lasted between 1995–1997. The chosen parallel corpus for the English, Bulgarian, Czech, Estonian, Hungarian, Romanian and Slovene languages was: George Orwell's novel „1984". It has been translated onto many languages, and its size on all of the above languages is approximately 100 000 words including punctuation characters. All corpora meet the CES (Corpus Encoding Specification) standard.

The corpus [11] is published on two CD-ROMs. Those contain the learning data for the current paper.

First, in Section 2 the preprocessing phase of the work is proposed together with some statistics obtained. In Section 3 and Section 4 the results given by C 4.5 [10] and Progol [6] are discussed. Section 5 is devoted to the preliminary work on the learned tagger and finally Section 6 contains some preliminary conclusion and further work.

[3] This work was supported by the grant ESPRIT 20237.

2 Preprocessing the Corpus

The Hungarian corpus itself is an SGML file. The file contains precise reference to particular sentences. Each sentence 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 were used as training data, while chapter 3 and 4 were served as test data.

```
<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'>
...
```

Fig. 1. The main structure of the corpus SGML file

On the Figure 1 the main stucture of the corpus file is shown. The whole novel is divided to paragraphs, sentences and tokens. The paragraphs are delimited by `<par >, </par>` tags, the sentences are delimited by `<s >, </s>` tags, and finally the tokens are delimited by `<tok >, </tok>` tags. Each paragraph and sentence is uniquely identified. Each sentence consists of series of tokens. Tokens are not numbered.

2.1 The MULTEXT-East Morpho-Syntactical Description (MSD) Categories

In the MULTEXT project there were established a coding convention that can be used for coding word attributes in wide variety of languages all over Europe. A code string represents 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

Fig. 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 categories of words. The detailed description of the MSD encoding can be found in [8]. The main categories of words can be seen in Figure 2. On the Figure 1 the word *„hideg"(cold)* is tagged to `Afp-sn`. (The trailing hyphens were cut really `Afp-sn-------`). This means the word is an *adjective* that has the following attributes: *qualificative, positive, singular, nominative.*

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

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

2.2 The Corpus Tag (CTAG) encoding for Hungarian language

In order to lessen the number of MSD classes the CTAG encoding was introduced [9]. There are 95 word tags and 13 punctuation tags. The complete list of the of the CTAGs shown on the Figure 4.

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 the following attributes: *plural, nominative, and*

Category	Number of tokens		
	Training	Test	Altogether
adjective	7744	2556	10300
conjunction	5858	2176	8034
determiner	0	0	0
interjection	71	13	84
numeral	915	289	1204
noun	15186	5305	20491
pronoun	4251	1789	6040
particle	0	0	0
adverb	7525	2891	10416
adposition	788	291	1079
article	6902	2335	9237
verb	9710	4002	13712
residual	13	12	25
abbreviation	69	14	83
	59032	21673	80705
punctuation	12487	5234	17721
	71519	26907	98426

Fig. 3. The number of tokens in the Hungarian corpus

some posessive ending. The second letter `P or N` encodes the *number*, namely plural or single, the third letter `N, A, D, O` encodes the *case* namely nominative, accusative, dative and other respectively. The fourth letter stands for some posessive stems not regarding to the *person* and *number* of posessive person.

The word CTAGs: AA, AAX, AN, ANX, ANY, AO, AOX, AOY, AP, APX, AW, CW, I, MA, MAX, MAY, MN, MNX, MNY, MO, MOX, MOY, NPA, NPAX, NPAY, NPD, NPDX, NPN, NPNX, NPNY, NPO, NPOX, NPOY, NSA, NSAX, NSAY, NSD, NSDX, NSDY, NSN, NSNX, NSNY, NSO, NSOX, NSOY, PPA, PPD, PPN, PPNY, PPO, PPOY, PSA, PSAX, PSAY, PSD, PSDX, PSN, PSNX, PSNY, PSO, PSOX, PSOY, RG, RO, RP, ST, T, VA, VMCP1P, VMCP1S, VMCP2P, VMCP2S, VMCP3P, VMCP3S, VMIP1P, VMIP1S, VMIP2P, VMIP2S, VMIP3P, VMIP3S, VMIS1P, VMIS1S, VMIS2P, VMIS2S, VMIS3P, VMIS3S, VMMP1P, VMMP1S, VMMP2P, VMMP2S, VMMP3P, VMMP3S, VMN, X, Y.

The punctuation CTAGs: CDBLQ, COLON, COMMA, DASH, EXCL, HELLIP, HYPHEN, LPAR, ODBLQ, PERIOD, QUEST, RPAR, SCOLON.

Fig. 4. The list of Hungarian CTAGs

2.3 The initial data set

The original Hungarian corpus was converted to prolog. First a lexicon of words were established. The lexicon contained all words in the same form as it was appeared in the corpus with all of its stems and its CTAG. If a word has more than one CTAGging then it was added multiple times to the lexicon. In the second step the whole SGML file was converted to prolog using the lexicon. The conversion went sentence by sentence. Sentences were recognized upon `<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', [(an, [an, vmis3s]), comma, an, an, nsn,
   (vmis3s, [an, nsn, vmis3s]), comma, (t, [psn, t]), npn,
   rg, ma, vmis3p, period]).
```

The first argument of the `s` predicate is the sentence identifier, the second is the list of CTAGs of words. If the CTAGging of a particular word is ambiguous then CTAG is represented by a pair of the right CTAG and the list of possible other CTAGging according to the lexicon. This way both the training and the testing part of the corpus was converted to this simpler form. There were 4549 sentences in the training set and 2183 sentences in the test data.

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

The number of CTAGs			
CTAG	Training	Test	Altogether
AA	62	16	78
AAX	4	3	7
AN	6469	2107	8576
ANX	80	18	98
ANY	7	3	10
AO	374	114	488
AOX	12	4	16
AOY	7	2	9
AP	61	34	95
APX	5	1	6
AW	663	254	917
all	7744	2556	10300

Fig. 5. The distribution of adjective 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 was written in SICStus Prolog 3.7.1.

The number of CTAGs			
CTAG	Training	Test	Altogether
NPA	306	106	412
NPAX	99	29	128
NPAY	1	0	1
NPD	50	9	59
NPDX	37	6	43
NPN	832	277	1109
NPNX	183	69	252
NPNY	3	1	4
NPO	457	122	579
NPOX	114	36	150
NPOY	1	3	4
NSA	1295	476	1771
NSAX	527	184	711
NSAY	8	6	14
NSD	357	89	446
NSDX	98	29	127
NSDY	9	1	10
NSN	5560	2051	7611
NSNX	1345	463	1808
NSNY	14	9	23
NSO	2778	953	3731
NSOX	1106	384	1490
NSOY	6	2	8
all	15186	5305	20491

Fig. 6. The distribution of noun CTAGs in the corpus

2.4 A Simple Probabilistic Tagger

A simple tagger can be constructed upon the empirical probabilities of different CTAGs of a particular word. The empirical probability distribution is set upon the training data. Each time an ambiguity is occured, the CTAG having the largest probability is returned as the result of the tagging. Although the applicability of such a tagger is questionable, the results are rather interesting. In the Figure 7 and Figure 8 the results are shown. The results shown in Figure 8 is obtained by using the whole novel to establish a lexicon i.e. a good complete lexicon exists, while in the first case the lexicon is obtained from the training data.

In the Figure 3 the number of tokens in the test data is shown. The tokens returned in by the tagger was tested against the corpus. The CTAGs of the words were either correct or incorrect. Some words might not have been recognized because the tagger was trained only on the first part of the novel. Those CTAGs get the label *skipped*. The number of elements in these three classes 21487, 963 and 4457. The total number of the tokens in the text is 26907. The accuracy is computed by the

$$a = \frac{N(agrees)}{N(agrees) + N(differs)} * 100.0\%$$

formula. The number of punctuation tokens were 5234. When this number is deducted from the number of tokens whose CTAGs agreed, then a little smaller accuracy was obtained.

$$\bar{a} = \frac{N(agrees) - N(punctuation)}{N(agrees) - N(punctuation) + N(differs)} * 100.0\%$$

```
Tokens that agrees:  21487
Tokens that differs: 963
Tokens that skipped: 4457
Accuracy:  95.71%
Accuracy (without punctuations):   94.41%
```

Fig. 7. The accuracy results obtained by the simple tagger with the lexicon of training text

```
Tokens that agrees:  26055
Tokens that differs: 852
Tokens that skipped: 0
Accuracy:  96.83%
Accuracy (without punctuations):   96.07%
```

Fig. 8. The accuracy results obtained by the simple tagger with the complete lexicon

3 Learning Disambiguation Rules by C 4.5

For POS tagging a Hungarian sentence a so-called `removable` predicate is learned. Such approach was first presented in James Cussens' paper [4]. This means that whenever a word is tagged to more than one CTAGs, some of them (all but one – hopefully) can be removed. The learned predicate will say that e.g. AA can be removed from the ambiguities if some conditions hold. These conditions come from the neighbouring CTAGs. A window of 7 CTAGs have been set, 7 CTAGs before the current ambiguity and 7 CTAGs after it. Narrowing the window to a

The number of ambiguity classes					
Class	Occurance		Class	Occurance	
	Training	Test		Training	Test
[AN,AP]	70	23	[NSA,NSN]	169	49
[AN,CW,VMIS3S]	113	38	[NSA,RG]	55	38
[AN,MN]	55	12	[NSAX,NSN]	70	29
[AN,NSN]	480	128	[NSN,PSN]	158	70
[AN,NSN,PSN]	52	22	[NSN,RG]	956	414
[AN,NSN,VMIS3S]	936	278	[NSN,RP]	119	46
[AN,PSN]	95	42	[NSN,VMIP3S]	62	47
[AN,RG]	117	41	[NSN,VMIS3S]	121	67
[AN,VMIP3S,VMIS3S]	84	30	[NSO,PSO]	56	10
[AN,VMIS3P]	147	41	[NSO,RG]	118	50
[AN,VMIS3S]	1570	549	[PSN,PSO]	288	153
[AO,RG]	68	39	[PSN,RP]	143	57
[AP,AW]	128	38	[PSN,T]	1867	620
[AP,RG]	110	24	[PSO,RG]	372	138
[AW,RG]	106	31	[PSO,RG,RP]	93	45
[CW,I]	297	115	[PSO,RP]	85	34
[CW,PSO]	50	30	[RG,RP]	115	36
[CW,RG]	934	336	[RG,ST]	176	62
[CW,RP]	334	149	[VMCP3S,VMIP3S]	243	70
[CW,VMIP3S]	247	125	[VMIP3P,VMIP3S]	64	53
[CW,Y]	827	269	[VMIP3S,VMMP3S]	64	40
[MN,T]	751	222			

Fig. 9. The most frequent ambiguity classes (occured more than 50 times among the training data)

smaller size can be done later inside C 4.5 by setting some columns in the data file to `ignore`.

Two main directions were tried. The first learns universal `removable` predicate for all ambiguities, while the second learns `removable` predicate for a specific abiguity class. In the following subsections these attempts are discussed. The ambiguity-classes and their occurances in the corpus is presented in the Figure 9. The occurance of CTAGs in the ambiguities only(!) can be seen in Figure 10.

The learned tagger is likely to remove as much ambiguities as it can, and the left ones can be eliminated upon the written form of the words, in a probabilistic way. Such a framework were already presented in section 2.4.

3.1 Learning `removable` rules for a particular CTAG

For generating learning examples for C 4.5 the prolog form of the corpus was used. Learning rules for removing the `T` CTAG is presented in the following as an

The number of CTAGs in the ambiguities			
CTAG	Training	Test	Altogether
AN	3841	1259	5100
AO	185	86	271
AP	336	100	436
AW	304	97	401
CW	2802	1062	3864
I	307	120	427
MN	888	262	1150
NSA	295	116	411
NSAX	100	47	147
NSN	3364	1251	4615
NSNX	124	50	174
NSO	401	145	546
NSOX	136	21	157
PSD	64	28	92
PSN	2646	977	3623
PSO	1035	455	1490
RG	3748	1445	5193
RP	1091	461	1552
ST	457	159	616
T	2618	842	3460
VMCP3S	246	70	316
VMIP3P	116	79	195
VMIP3S	886	417	1303
VMIS3P	170	48	218
VMIS3S	2877	980	3857
VMMP3S	65	41	106
Y	851	273	1124
all	29953	10891	40844

Fig. 10. The most frequent CTAGs among the ambiguities (occured more then 50 times among the training data)

example of the whole process. The `T` CTAG in the Hungarian language is equals to the `Tf----` MSD class that contain the articles in the Hungarian language. Only 3 words belongs to this class: *a(the), az(the), egy(a, an)*.

Each time when `T` occures in a sentence in an ambiguity list a learning example is generated. In the next sentence the tagging of the first word is ambiguous, it can be `MN` or `T`. From the sentence a positive example is generated, because the right CTAG here is `MN` i.e. `T` can be removed from the ambiguity list.

```
s('Ohu.1.2.6.3', [(mn, [mn, t]), an, nso, nsnx, comma,
(t, [psn, t]),nsn, vmis3s, an, npox, aw, t, an, nsn, st, period]).
```

If the right tagging would have been `T` then a negative example had been

```
Ohu.1.2.6.3,  xxx, xxx, xxx, xxx, xxx, xxx, xxx,
              an, nso, nsnx, comma, t, nsn, vmis3s, +
Ohu.1.2.11.4, nsn, an, vmip3s, nso, t, cw, comma,
              nsn, an, an, nso, st, comma, psa, +
Ohu.1.2.12.4, vmis3s, xxx, xxx, xxx, xxx, xxx, xxx,
              nsa, t, nsn, an, an, nso, comma, +
...
Ohu.1.2.1.1,  comma, vmis3s, nsn, an, an, comma, an,
              npn, rg, ma, vmis3p, period, xxx, xxx, -
Ohu.1.2.1.3,  rg, pso, rp, vmmp3s, rg, cw, comma,
              an, nsn, period, xxx, xxx, xxx, xxx, -
Ohu.1.2.2.1,  xxx, xxx, xxx, xxx, xxx, xxx, xxx,
              nso, an, nsn, an, nsn, cw, an, -
...
Ohu.1.2.6.3,  comma, nsnx, nso, an, mn, xxx, xxx,
              nsn, vmis3s, an, npox, aw, t, an, -
...
```

Fig. 11. The C 4.5 learning data for `removable_t`

generated. This situation is demonstrated with the same sentence. There is another ambiguity, a `[PSN, T]` where the correct tagging is `T`. Therefore a negative example for removing `T` is also generated from the same sentence. In the Figure 11 some lines of the learning datafile for C 4.5 is presented. The data files contain 16 attributes, the first is the sentence identifier, the last is the class. The neighbouring tokens' window can be found between them: $before_1, \ldots before_7$ and $after_1 \ldots after_7$. A new `xxx` token was introduced to denote missing tokens.

The results of C 4.5 is shown in the Figure 12. One obvious finding is that the results are sligthly depending on the window size. Almost the same result is obtained in each case. If minor differences are taken into account then the best results are obtained when the window size was 1. It means that the previous and the following tokens are the most important in the decision on disambiguation.

3.2 Learning `removable` rules for an ambiguity-class

Considering the previous example the rules for removing `T` CTAG results in a sophisticated decision-tree because the `T` CTAG may occure in different ambiguity classes. In this special case `T` may occure in the `[PSN, T]` and in the `[MN, T]` classes. The C 4.5 result must be a decision-tree that really tries to unite these two trees.

It comes naturally that if the learning is restricted to a certain ambiguity class then the learned rules must be better (simpler and easier to understand). This means a small change in the generator program and in the structure of the data file. An example is generated if and only if the ambiguity list in the sentence is equal to the ambiguity class considered. The examples are not positive

Window size	Number of rules	Accuracy			
		Training examps. (2618)		Test examps. (842)	
		%	#errors	%	#errors
1	34	97.20%	74	96.70%	28
2	39	97.30%	70	96.60%	29
3	41	97.40%	68	96.60%	29
4	38	97.40%	67	96.10%	33
5	39	97.40%	69	96.60%	29
6	44	97.60%	64	96.40%	30
7	43	97.60%	64	96.10%	33

Fig. 12. The results of C 4.5 on learning `removable_t`

and negative any longer. The learned classes can be labelled by CTAGs in the ambiguity list so that each example generated is labelled by the right (*should-be*) tagging of that token. In the corpus the ambiguity classes had 2, 3 and 4 elements. Many of them caused by hundreds of words (structural ambiguity), but there are some like `[PSN, T]` and `[MN, T]` that are caused by single words (similar words with different meaning). In the `[PSN, T]` there is only one word: *az* that means *that* and *the*. In the `[MN, T]` class there is also one word: *egy* that means *one* and *a, an*.

Window size	Number of rules	Accuracy			
		Training examps. (751)		Test examps. (222)	
		%	#errors	%	#errors
1	6	98.00%	15	99.10%	2
2	6	98.00%	15	99.10%	2
3	6	98.00%	15	99.10%	2
4	6	98.00%	15	99.10%	2
5	6	98.00%	15	99.10%	2
6	6	98.00%	15	99.10%	2
7	6	98.00%	15	99.10%	2

Fig. 13. The results of C 4.5 on learning `removable_t__mn_t`

In the Figure 14 the 6 learned rules for `removable_t__mn_t` is presented.

For the comparision the next learning task is to discover removable rules for `T` within two ambiguity classes — in the `[PSN, T]` and in the `[MN, T]`. In the Figure 13 and Figure 15 the results are presented. It obviously shows that in the Section 3.1 the union of these two decision trees were learned. There might be some interference between the rules, which cause the less stability when only the `removable_t` is learned.

```
Rule 2: token1_before = aw
        ->  class mn  [31.4%]
Rule 3: token1_after = an
        ->  class t  [98.4%]
Rule 4: token1_after = cw
        ->  class mn  [63.0%]
Rule 5: token1_before = cw
        ->  class t  [98.9%]
Rule 11: token1_before = t
        ->  class mn  [50.0%]
Rule 15: token1_before = xxx
        ->  class t  [97.1%]
```

Fig. 14. The learned rules for `removable_t__mn_t`

Window size	Number of rules	Accuracy: Training examps. (1867) %	Training examps. (1867) #errors	Test examps. (620) %	Test examps. (620) #errors
1	32	98.10%	36	96.80%	20
2	33	98.30%	31	96.60%	21
3	33	98.30%	31	96.60%	21
4	33	98.30%	31	96.60%	21
5	32	98.10%	36	96.80%	20
6	32	98.10%	36	96.80%	20
7	33	98.10%	35	96.10%	24

Fig. 15. The results of C 4.5 on learning `removable_t__psn_t`

4 Learning Disambiguation Rules by Progol

In the preliminary work, the same task was given to Progol as described in Section 3.2. The representation of the learning examples was sligthly changed. The neighbouring tokens are given by two lists that may have any number of elements:

```
removable_t__mn_t( 'Ohu.1.2.1.3',
     [rg,pso,rp,vmmp3s,rg,cw,comma,vmn,aw,psn,vmis3s,rg,cw],
     [an,nsn,period]).
removable_t__mn_t( 'Ohu.1.2.2.2',
     [nsox,psn],
     [dash,nso,an,nsn,nsox,aw,cw,an,dash,nsn,vmis3s,t,nso,ro,period]).
removable_t__mn_t( 'Ohu.1.2.2.3',
     [colon,nsn,an,cw,nso,comma,pso,an,vmis3s,nsn,an,t,rg],
     [mn,nsn,an,comma,an,an,an,comma,an,an,nsn,nsnx,period]).
...
```

The input file for Progol is also generated by a prolog program. The main structure can be outlined as follows:

```
%%%%%%%%%%%%%%%%%%%%
% Grammar learning problem. Learns removable_t__mn_t/3,
%       predicate for Hungarian language.

:- modeh(1,removable_t__mn_t(+number,+ctaglist,+ctaglist))?
:- modeb(4,aa(+ctag))?
:- modeb(4,aax(+ctag))?
... <introduce a predicate for recognizing each CTAG>
...
%%%%%%%%%%%%%%%%%%%%%%%
% Background knowledge
aa(aa).
aax(aax).
an(an).
... <the backround knowledge is simply the set of known CTAGs>
empty([]).
tk1(A, [A|_]).
tk2(A, [_, A|_]).
tk3(A, [_, _, A|_]).
... <the tk-i predicates can project select the first, second, ...
     elements of lists>

removable_t__mn_t('Ohu.1.2.1.3',
     [rg,pso,rp,vmmp3s,rg,cw,comma,vmn,aw,psn,vmis3s,rg,cw],
     [an,nsn,period]).
removable_t__mn_t('Ohu.1.2.2.2',
     [nsox,psn],
     [dash,nso,an,nsn,nsox,aw,cw,an,dash,nsn,vmis3s,t,nso,ro,period]).
... <positive examples>
:-removable_t__mn_t('Ohu.1.2.6.3',
     [],
     [an,nso,nsnx,comma,t,nsn,vmis3s,an,npox,aw,t,an,nsn,st,period]).
:-removable_t__mn_t('Ohu.1.2.11.4',
     [nsn,an,vmip3s,nso,t,cw,comma,vmis3s,cw,comma,nsax,
      an,vmis3s,comma,nso,t,rp,vmis3s,nsox,t,pso,nsd,t,rg],
     [nsn,an,an,nso,st,comma,psa,t,an,ao,vmis3s,rp,period]).
... <negative examples>
generalise(removable_t__mn_t/3)?
test('ltagp-t--mn-t.test')?
quit?
```

Progol is used to infer `removable` rules for a CTAG that occure in a certain ambiguity class e.g. `removable_t__mn_t` or `removable_t__psn_t`. Only a very restricted background was added to the initial theory. Progol then expected to return somehow similar results that were obtained by C 4.5.

The learned rules can be separated to two main parts: some sporadic clauses that cannot be generalized and some rules found. The results are shown below:

```
removable_t__mn_t('Ohu.1.2.6.8',
   [comma,nsox,pso,rp,vmis3s,rg,nsn,colon,vmn,vmis3s,rg,comma,rg,cw],
   [nsn,an,an,nsn,nsox,comma,ppn,cw,nsn,st,cw,rg,aw,vmis3p,rp,st,period]).
removable_t__mn_t('Ohu.1.2.11.7',
   [rg,nsn],
   [aa,vmis3s,psd,comma,vmis3s,psn,nsax,comma,cw,vmis3s,comma,
    cw,t,nsn,nsa,period]).
...
removable_t__mn_t(A,B,C) :- tk1(D,C), tk2(E,C), an(D), nsn(E).
removable_t__mn_t(A,B,C) :- tk1(D,B), nsox(D).
removable_t__mn_t(A,B,C) :- tk1(D,B), cw(D).
removable_t__mn_t(A,B,C) :- tk1(D,B), tk2(E,B), vmis3s(D).
removable_t__mn_t(A,B,C) :- tk2(D,C), rg(D).
removable_t__mn_t(A,B,C) :- tk2(D,B), t(D).
removable_t__mn_t(A,B,C) :- tk2(D,C), nsa(D).
removable_t__mn_t(A,B,C) :- tk2(D,B), rg(D).
removable_t__mn_t(A,B,C) :- tk1(D,B), vmn(D).
removable_t__mn_t(A,B,C) :- tk1(D,C), psn(D).
removable_t__mn_t(A,B,C) :- empty(B), tk1(D,C), nso(D).
removable_t__mn_t(A,B,C) :- tk1(D,C), tk2(D,B).
removable_t__mn_t(A,B,C) :- tk1(D,B), psa(D).
removable_t__mn_t(A,B,C) :- tk1(D,C), nsox(D).
removable_t__mn_t(A,B,C) :- tk1(D,B), vmis3p(D).
removable_t__mn_t(A,B,C) :- tk1(D,B), psn(D).
removable_t__mn_t(A,B,C) :- tk2(D,B), vmip3s(D).
removable_t__mn_t(A,B,C) :- tk1(D,C), pso(D).
removable_t__mn_t(A,B,C) :- tk2(D,B), scolon(D).
removable_t__mn_t(A,B,C) :- tk1(D,C), ao(D).
removable_t__mn_t(A,B,C) :- tk1(D,C), rg(D).
removable_t__mn_t(A,B,C) :- tk1(D,B), tk2(E,B), cw(E), rg(D).
removable_t__mn_t(A,B,C) :- tk1(D,B), tk1(E,C), nsn(D), nso(E).
removable_t__mn_t(A,B,C) :- tk1(D,B), vmip3s(D).
removable_t__mn_t(A,B,C) :- tk2(D,C), nsox(D).
removable_t__mn_t(A,B,C) :- tk1(D,C), tk2(E,B), nsa(D).
removable_t__mn_t(A,B,C) :- tk2(D,C), st(D).
removable_t__mn_t(A,B,C) :- empty(B), tk1(D,C), nsn(D).
removable_t__mn_t(A,B,C) :- tk1(D,C), tk1(E,B), nsn(D), rg(E).
removable_t__mn_t(A,B,C) :- tk1(D,C), nsnx(D).
removable_t__mn_t(A,B,C) :- tk1(D,C), tk1(E,B), an(D), rg(E).
removable_t__mn_t(A,B,C) :- tk1(D,C), tk1(E,B), nso(D), rg(E).
...
```

Four examples were run for learning `removable_t__mn_t`, `removable_mn__mn_t`, `removable_t__psn_t` and `removable_psn__psn_t`. The accuracy of the learned rules are shown in the Figure 16. The learning task for `removable_t__mn_t` and `removable_mn__mn_t` were complementers of one another i.e. the sets of positive and negative examples were exchanged. Only one rule found by Progol appeared in the result of C 4.5 as well c.f. Figure 14. The rule is:

```
removable_t__mn_t(A,B,C) :- tk1(D,B), cw(D).
```

task	#inferred rules	Test examples		
		size (#examples)	%	#errors
`removable_mn__mn_t`	19	222	98.65%	3
`removable_t__mn_t`	76	222	88.75%	25
`removable_t__psn_t`	110	620	96.13%	24
`removable_psn__psn_t`	135	620	88.39%	72

Fig. 16. The accuracy results obtained by Progol

It can be said, that Progol learned more specific rules than C 4.5. On the other hand the size of the inferred theories were larger in the latter case.

5 The Learned Tagger

The rules returned by the C 4.5 has been built into a working tagger prototype. The tagger works with the Prolog versions of the learned rules.

```
tk1(A, [A|_]).

% Rule 2:
%        token1_before = aw
%        ->  class mn  [31.4%]
removable_t__mn_t(A, _) :- tk1(aw, A).

% Rule 3:
%        token1_after = an
%        ->  class t  [98.4%]
removable_mn__mn_t(_, A) :- tk1(an, A).
```

In the Figure 14 above, the C 4.5 rules were presented. Here, the prolog equivalents of the rules are shown. C 4.5 always uses a default class, which can be easily encoded in Prolog. The first tagger disambiguates ambiguities belonging to the `[MN, T]` and `[PSN, T]` class. The results can be seen below:

Sample	Rules				Tagging			
rules set	applied	ok	failed	uncov.	resolved	good	bad	accuracy
`mn_t`	232	130	1	101	131	130	1	99.237%
`mn_t & default class t`	223	218	2	3	220	218	2	99.091%
`psn_t`	645	457	10	178	467	457	10	97.859%
`psn_t & default class t`	646	501	23	122	524	501	23	95.611%
`mn_t, psn_t & default class t`	871	719	25	127	744	719	25	96.640%
`progol mn_t`	227	190	1	36	191	190	1	99.476%
`progol psn_t`	636	518	9	109	527	518	9	98.292%
`progol mn_t, psn_t`	868	708	10	150	718	708	10	98.607%

Fig. 17. The accuracy results obtained by the tagger

6 Conclusion and Future Work

In the forthcoming time more systematic work is needed: the already prepared learning tasks must be completed. By the help of linguists the results should be well understood. Finally, the rules are to be built together in a working tagger. Experiencing with that tagger is also an interesting continuation. Consulting with linguists can help to add backround knowledge to Progol. A relevant backround theory for syntactic structure for the Hungarian natural language is one of our main interest.

The current study is planned to extend by application of IMPUT [1, 2] and RIBL [3]. IMPUT is planned to assist in refinement of an overly-general tagging program that is constructed by consulting with linguists. It can be done from scratch or the results of C 4.5 and Progol can be used as a starting point.

Researchers in the GMD expressed their interest experiencing with RIBL on the Hungarian CTAGging domain.

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

References

1. Alexin, Z., Gyimóthy, T., Boström, H.: IMPUT: An Interactive Learning Tool based on Program Specialization Intelligent Data Analysis Vol **1** 4 Elsevier Holland, (1997)
2. Alexin, Z., Gyimóthy, T., Boström, H.: Integrating Algorithmic Debugging and Unfolding Transformation in an Interactive Learner in Proceedings of the 12th European Conference on Artificial Intelligence ECAI-96 ed. Wolfgang Wahlster, Budapest, Hungary (1996) 403–407 John Wiley & Son's Ltd. (1996)
3. Bohnebeck, U., Horváth, T., Wrobel, S.: Term Comparisions in First-Order Similarity Measures in Proc. of the Eighth International Workshop on Inductive Logic Programming (ILP98) Madison, Wisconsin, USA, in the LNAI series Vol **1446** 65–79 Springer Verlag (1998)

4. 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)
5. Cussens, J.: Notes on inductive logic programming methods in natural language processing (European work) `http://www.cs.york.ac.uk/ jc/pub.html`
6. Muggleton, S.: Inverse entailment and Progol, New Generation Computing Vol 13. (special issue on Inductive Logic Programming) Ohmsha, (1995) 245–286
7. Muggleton, S., De Raedt, L.: Inductive Logic Programming: Theory and methods, Journal of Logic Programming 19 (20) (1994) 629–679
8. 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
9. Oravecz, Cs.: Part-of-Speech Tagging in the Hungarian National Corpus — a Case Study
10. Quinlan, J. R.: C4.5 Program for Machine Learning San Mateo, CA Morgan Kaufmann (1993)
11. East meets West — A Compendium of Multilingual Resources
Eds: Tomaž Erjavec, Ann Lawson and Laurent Romary
`http://www.ids-mannheim.de/telri/cdrom.html`