Acta Cybernetica **16** (2004) 611–623.

Noun Phrase Recognition with Tree Patterns*

András Hócza[†]

Abstract

This paper offers a method for the noun phrase recognition of Hungarian natural language texts based on machine learning methods. The approach learns noun phrase tree patterns described by regular expressions from an annotated corpus. The tree patterns are completed with probability values using error statistics. The noun phrase recognition parser tries to find the best-fitting trees for a sentence using backtracking technique. The results are used in an information extraction toolchain.

1 Introduction

Noun phrase (NP) recognition is the process of determining whether sequences of words can be grouped together with nouns. NP recognition is an important part of the field of syntactic parsing but, to date, there is no suitable syntactic parser available for the Hungarian language.

Hungarian is an agglutinated language with a rich morphology and relatively free word order, whose properties add difficulties to the full analysis of the Hungarian language compared to Indo-European languages. These difficulties mean that the automatic NP recognition of Hungarian language is too complicated to solve using experts' rules only. An efficient solution for this problem might be the application of machine learning methods, but it requires a large number of training and test examples of annotated NPs. Since the Szeged Corpus[1] [4] became available, new methods have begun to be developed for syntactically parsing Hungarian sentences. The corpus contains texts from five different topic areas and is currently comprised of about 1.2 million word entries, 145 thousand different word forms, and an additional 225 thousand punctuation marks. Initially, corpus words were manually POS tagged and disambiguated by linguistic experts. Later, texts from the Szeged Corpus were parsed, where annotators marked noun phrase structures and clause structures. The extensive and accurate manual annotation of the texts, which required 124 person-months of manual work, is a good feature of the corpus.

*Presented at the 1st Conference on Hungarian Computational Linguistics, December 10–11, 2003, Szeged.

[†]University of Szeged, Department of Informatics, 6720 Szeged, Árpád tér 2. E-mail: hocza@inf.u-szeged.hu, Webpage: http://www.inf.u-szeged.hu

[1]Magyar Távirati Iroda, http://www.mti.hu

After the completion of the annotation work the Szeged Corpus was then used for training and testing machine learning algorithms to retrieve NP recognition rules. This paper introduces an application of the RGLearn [6] algorithm that was used to learn NP tree patterns described by regular expressions. The NP tree patterns are completed with probability values using error statistics. The NP parser uses this grammar to build up the best series of NP trees of a sentence by backtracking. The results look fairly promising after comparing them to related works. This method was developed as a part of a system which extracts information from short business news texts written in the Hungarian language.

This paper is organized as follows. Section 2 describes the difficulties of automatic NP recognition in the Hungarian language. In Section 3 there is a review of related works and a mention of efforts made by Hungarian researchers. Section 4 introduces a large annotated corpus that was used as source of training and test data. Section 5 then presents the method used for learning grammar from an annotated corpus and extending grammar with probability values using test statistics. Section 6 introduces our method of noun phrase recognition. Section 7 presents the test results. Lastly, conclusions and suggestions for future study are given in Section 8.

2 Difficulties of Hungarian Noun Phrase identification

Hungarian is customarily defined as an agglutinative, free word order language with a rich morphology. These properties make its full analysis difficult compared to Indo-European languages. Unambiguous marks for the automatic recognition of NP boundaries do not exist. The right bound of NPs (NP head) could be the nouns, but there is a possibility of replacement of NP heads with its modifiers. Determining the left bound of NPs is harder than the NP head, because it could be a determinant element. However, due to the possibility of a recursive insertion, it is not easy to decide which determinant and NP head belong together. Some of these difficulties can be illustrated in the following short sentences:

These difficulties mean that it is a hard problem to perform automatic NP recognition with experts' rules. In many cases the decision of annotators is based on semantic reasons rather than syntactic or structural ones. Another approach for automatic NP recognition is to use machine learning methods, but this requires a large number of training examples. In the past there was no large corpus for the Hungarian language containing annotated NPs.

3 Related works

Several authors published results of NP recognition parsers especially made for English. Generally the performance is measured with three scores. First, with a percentage of detected noun phrases that is correct (precision). Second, with a percentage of noun phrases in the data that is found by the classifier (recall). And

Free word order:

[Péter] olvas [egy könyvet]. (Peter is reading a book.)
Olvas [Péter] [egy könyvet]. (Peter is reading a book.)
[Egy könyvet] olvas [Péter]. (Peter is reading a book.)

Missing determiner:

[Péter] olvas [**egy** könyvet]. (Peter is reading a book.)
[Péter] [könyvet] olvas. (Peter is reading a book.)

Missing NP head:

[Péter] [a sárga könyvet] olvassa, [Mari] pedig **[a pirosat]**.
(Peter is reading the yellow book and Mary is reading the red one.)
[a pirosat] = [a piros **könyvet**]

Figure 1: Examples of problems in Hungarian NP identification

third, with the $F_{\beta=1}$ rate which is equal to 2*precision*recall/(precision*recall). The latter rate has been used as the target for optimization.

Abney [1] proposed an approach which starts by finding correlated chunks of words. Ramshaw and Marcus [11] built a chunker by applying transformation-based learning ($F_{\beta=1}$=92.0). They applied their method to two segments of the Penn Treebank [8] and these are still being used as benchmark data sets. Argamon [3] uses memory-based sequence learning ($F_{\beta=1}$=91.6) for recognizing both NP chunks and VP chunks. Tjong Kim Sang and Veenstra [13] introduced cascaded chunking ($F_{\beta=1}$=92.37). The novel approaches attain good accuracies using a system combination. Tjong Kim Sang [14] utilized a combination of five classifiers ($F_{\beta=1}$=93.26).

So far, there is no good-quality NP parser published for Hungarian. The first report on the ongoing work of a Hungarian NP recognition parser [16] is based on the idea of Abney [2] using a cascaded regular grammar. The input to the grammar was a morpho-syntactically annotated text using a scaled-down version of the annotation scheme developed for the Hungarian National Corpus [15]. The grammar was developed by linguistic experts with the help of the CLaRK [12] corpus development system on a text of 928 sentences containing 23991 tokens. These samples were taken from a quality weekly economics journal concentrating on short business news items, which were suitable for an information extraction project. The rules of grammar contain context free patterns of NPs described by regular expressions. The patterns may refer to patterns of morpho-syntactic codes, words or previously recognized NPs. The rules were grouped together by linguistic experts into distinct stages which were applied cyclically by the CLaRK system using a bottom-up tree building technique. A 100-sentence text chunk containing 2537 tokens was produced by manual annotation to serve as the gold standard containing 488 NPs. The system parsing the test set found 611 NPs, of which 323 NPs were correct ($F_{\beta=1}$=58.78).

4 Training data

In order to perform well and learn from the various *Natural Language Processing* (NLP) tasks and achieve a sufficient standard of *Information Extraction* (IE), an adequately large corpus had to be collected that serves as the training database. During the setting-up of the various NLP projects a relatively large corpus of different types of texts was collected, called the Szeged Corpus [4]. For demonstration purposes in the above-mentioned NKFP project, the authors chose a collection of short business news items issued by the Hungarian News Agency[2]. The selected 6453 articles form part of the Szeged Corpus and relate to Hungarian or joint companies' financial and business life. The Szeged Corpus contains 1.2 million text words, 225 thousand punctuation marks, and comes in an XML format using the TEIXLite DTD (Document Type Definition). The first version of the corpus contains texts from five topic areas, roughly 200 thousand words each, meaning a text database of some 1 million words. The second version was extended with a sample of texts of business news totalling another 200 thousand words. The texts are divided into sections, paragraphs, sentences and word entries.

Initially, corpus words were morpho-syntactically analysed with the help of the HuMor[3] automatic preprocessor and then manually POS tagged by linguistic experts. The Hungarian version of the internationally acknowledged MSD (Morpho-Syntactic Description) scheme [5] was used for the encoding of the words. Due to the fact that the MSD encoding scheme is extremely detailed (one label can store morphological information on up to 17 positions), there is a large number of ambiguous cases, i.e. roughly every second word of the corpus is ambiguous. Disambiguation therefore required accurate and detailed work. It required 64 person-months of manual annotation. Currently all possible labels as well as the selected ones are stored in the corpus. About 1800 different MSD labels are used in the annotated corpus. The MSD label corresponds to the part-of-speech determined attribute, and specific characters in each position indicate the value for that attribute.

For example, the MSD label Nc-pa can be understood as

POS: **N**oun,
Type: **c**ommon,
Gender: **-** (not applicable to Hungarian),
Number: **p**lural,
Case: **a**ccusative.

In general, the NP building process of a sentence produces detailed NP trees much like Figure 2. These general NP trees must be simplified because, of course, simpler trees more readily support information extraction. This simplification was done by linguistic experts in the manual annotation phase. Hence the training corpus contains simplified NP trees. However the morpho-syntactic labels of corpus for POS tagging are more informal using the MSD encoding. The sentences of the corpus are stored like in Figure 3.

[2]MTI, Magyar Távirati Iroda (http://www.mti.hu), "Eco" service.
[3]The HuMor morpho-syntactic analyser is a product of MorphoLogic (Budapest) Ltd..

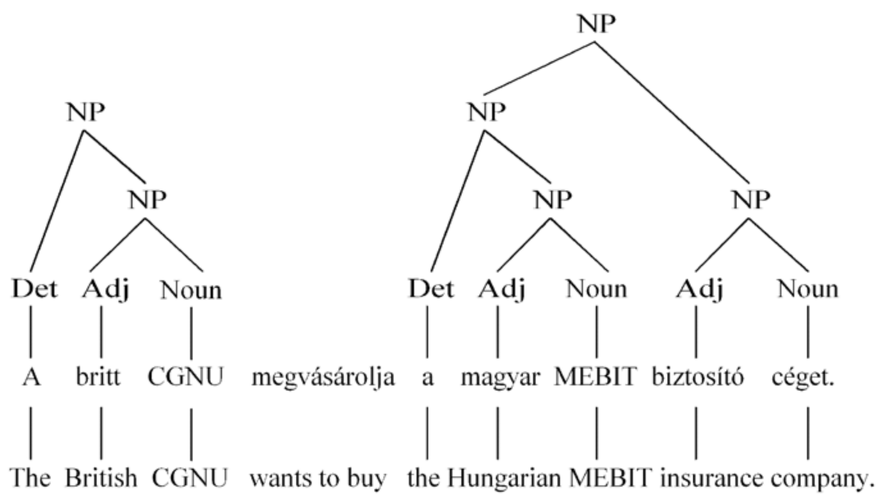


Figure 2: NP trees of a Hungarian sentence (with its English equivalent) from a short business news item.

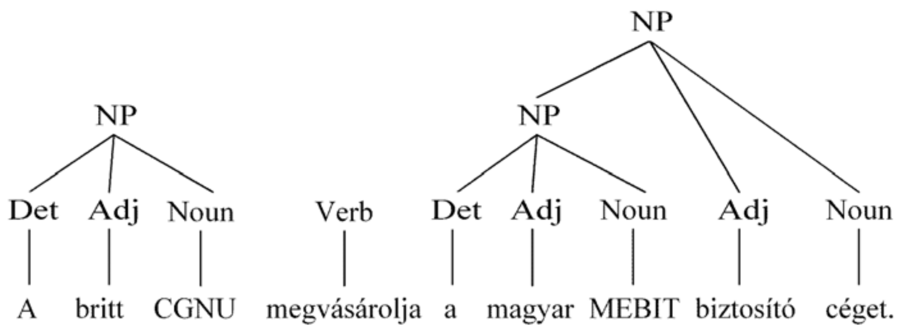


Figure 3: Simplified NP trees with MSD encoding from the sentence in Figure 2.

5 Learning tree patterns

In this section the learning task of noun phrases will be described which contains the preprocessing of training data, unification of tree patterns, searching for repetition of POS tag labels in tree patterns, handling of equivalent sub-trees and, finally, the complete unification algorithm. An application of RGLearn [6] is used as an NP tree learner. RGLearn is a rule generalization method based on a previous algorithm called RAPIER [9]. The RAPIER (Robust Automated Production of Information Extraction Rules) system learns rules by using a combined bottom-up and top-down learning algorithm. RAPIER is a modified version of the GOLEM [10] machine

[Tf Afp-sn Np-sn]
[[Tf Afp-sn Np-sn] Afp-sn Nc-pa]

Figure 4: Patterns from the sentence in Figure 3.

learning method. GOLEM uses the LGG (Least General Generalization) method for compression, i.e. it always compresses two examples, whereas RAPIER always generalizes two randomly selected rules.

5.1 Preprocessing of training data

The initial step for generating training data is to collect every complete NP tree from an annotated training corpus.

Complete NP tree means: the NP tree can contain other NPs but cannot be contained in another NP.

The collected training examples are complete context-free NP trees without words. For example, the sentence in Figure 3 has three NPs and two complete NP trees. The description of NP trees contains only lexical codes and NP begin and end marks ([,]), like the examples in Figure 4.

5.2 Unification of tree patterns

The collected NP tree patterns provide useful rules for NP recognition. With the help of these rules the NP parser is able to reconstruct the NP structures of training sentences. But, in order to perform the NP recognition of an unknown text to a fair accuracy, the collected NP tree patterns must be generalized. Generalization means the compacting of the rule set and expanding its coverage to include similar cases that do not occur in training examples. Various methods are used in the generalization learning phase. One of these methods is the most general unification of the rule set, which is the unification process of every possible similar rule one step at a time until similar rules exist in the rule set.

The meaning of similarity between rules is given by the following definition: rules R_i and R_j are similar if

- R_i and R_j contain the same number of tags,
- the first letters of the tags are the same in each rule position, which means NP start and end tags ([,]) and POS codes of words are the same,
- $\left(\sum_{t=1}^{T} Dif\left(R_{it}, R_{jt}\right)\right) \leq threshold$

where R_{it} is the t'th tag of rule R_i, T is the number of tags in the rules, Dif is the number of different letters of two tags, and finally *threshold* is a predefined value for the maximum difference. In general the *threshold* value is 1 or 2. The unification of similar rules means its replacement with a new, more general rule. Different parts in similar rules are changed to macro characters ('?', '*') in new rules, like the example in Figure 5. The symbol '?' in a pattern covers any letter.

Similar patterns:

[[Tf Afp-sn Nc-**sn**] Afp-sn Nc-sn]
[[Tf Afp-sn Nc-**p**n] Afp-sn Nc-sn]

Unified pattern:

[[Tf Afp-sn Nc-**?**n] Afp-sn Nc-sn]

Figure 5: The unification of similar patterns. The POS codes of Tf, Afp-sn and Nc-sn correspond to Det, Adj and Noun.

5.3 Repetition of POS tag labels in tree patterns

The next possible step in compacting the rule set is to reduce repetitions using the following generalization assumption for these cases: if some part of the pattern is repeated twice, it may be repeated infinite times in similar patterns of other examples taken from the training or test data. A simple repetition is a group of similar POS tags. The compacted pattern for this group is a label - the most general unification of similar POS tag labels. The regular operator '+' indicates that this tag can be repeated an indefinite number of times. Complex repetitions are when more than one similar POS tag label can be placed into the part of the tree pattern with free word order. To detect complex repetitions, examples of similar tree patterns are needed with a different POS tag label. In unified patterns the operator '|' (logical or) indicates that the set of POS tags can be used at that position of the tree pattern. Simple and complex repetitions can also be combined which are based on similar tree patterns as well. Examples of simple, complex and combined repetitions and their compacted versions are shown in Figure 6.

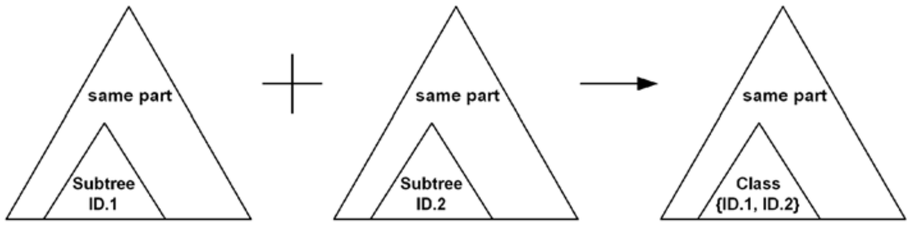
Kind	**Repetition in patterns**	**Unified patterns**
Simple	the greatest Hungarian insurance company [Tf **Afc-sn Afp-sn Afp-sn** Nc-sn]	[Tf **(Af?-sn)+** Nc-sn]
Complex	the great and famous company [Tf **Afp-sn Ccsw Afp-sn** Nc-sn] [Tf **Afc-sn Afp-sn Afp-sn** Nc-sn]	[Tf Af**?**-sn **(Afp-sn|Ccsw)** Afp-sn Nc-sn]
Combined	[Tf **(Af?-sn)+** Nc-sn] [Tf **Af?-sn (Afp-sn|Ccsw) Afp-sn** Nc-sn]	[Tf **(Af?-sn|Ccsw)+** Nc-sn]

Figure 6: The unification of repetition using similar patterns.

5.4 Equivalent sub-trees, sub-tree classes

Repetitions like those described previously do not contain NP bound marks ('[', ']') because preserving the NP tree structure is very important. NP bound marks are not elements of patterns because their function is the separation (and recognition) of trees and sub-trees. The usage of sub-trees for NP tree building requires that NP identification distinguish one NP from another. NP tree building without NP identification often gives rise to overgeneration, i.e. the generation of impossible tree structures for a given natural language. Hence it cannot be assumed that each NP is equivalent. But there are equivalent sub-trees when the context of two sub-trees is the same in their bigger trees (see Figure 7). Using the above notation the unification for sub-trees is given by the following process:

- Selecting rules that contain equivalent sub-trees, as in Figure 7.
- Collecting equivalent sub-trees to equivalence classes.
- The replacement of equivalent sub-trees with its equivalence class identification code. This means that bigger trees will be practically identical, so the unnecessary rules can be deleted from the rule set.



Example:

Similar patterns	**Unified patterns**
[**[Tf Np-sn**] Afp-sn Nc-pa] [**[Tf Afp-sn Np-sn**] Afp-sn Nc-pa]	[**CLASS.1** Afp-sn Nc-pa] CLASS.1: ID.1 - [Tf Np-sn] ID.2 - [Tf Afp-sn Np-sn]

Figure 7: The unification of similar patterns that contain different sub-trees and an example of unification.

5.5 The most general unification of the rule set

Bringing together the unification methods discussed in previous sections, the algorithm of the most general unification is quite simple: repeating unification methods one after another until something changes in the rule set. Expressed in formal terms, the procedure is the following:

```
repeat
        unification of similar tree patterns
        unification of repetitions
        unification of equivalent sub-trees
until (something changes in the rule set)
```

5.6 Analysis of the grammar

The most general unification of the rule set allows the parser to be able to recognize every possible NP in an unknown text as well. But the cost of increasing generalization is decreasing accuracy. The accuracy for various rules is quite different: there are a lot of rules with very good (>95%)and very poor (<5%) accuracies. It seems obvious that bad rules ought to be dropped from the rule set (e.g. a rule with an accuracy of below 50% produces more errors than good recognition). But the strategy of an NP tagger is to find the best possible NP tree series for a sentence, so the rule with a poor accuracy (and low frequency) does not make so many errors because it is generally not chosen. For optimization the accuracy of NP recognition can be found experimentally. Which is the best threshold value for dropping rules because of its poor accuracy? According to the results shown in Figure 8, we should choose a rule accuracy threshold for dropping rules of about 5%-10%.

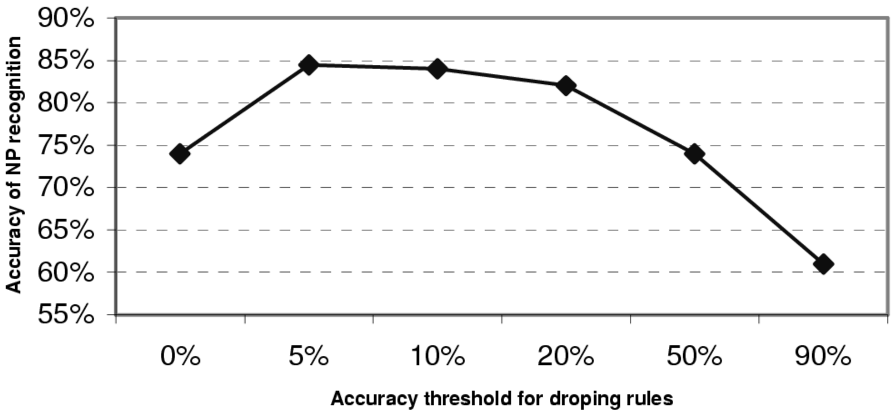


Figure 8: Where is the better threshold for dropping rules?

The main task of the analysis phase is to determine the probabilistic worth of rules and drop weak rules from the rule set using previous optimizations. The probabilistic value of a tree pattern is the accuracy normalized by the frequency. Summarizing the results of the learning and analysis phase is a context-free probabilistic grammar described with regular expressions for the recognition of NP trees.

6 Building of NP trees

The main task of the NP tagger is to find the best series of NP trees for input sen-tences. Input data contains disambiguated POS tag labels and it may come from the Szeged corpus at the testing phase of the method or it may be provided by a POS tagger tool [7] as a practical application of method. The following natural language processing (NLP) modules are used to determine linguistic features for NP recognition, as shown in Figure 9. They consist of tokenization, sentence segmentation, morpho-syntactic analysis and part-of-speech (POS) tagging.

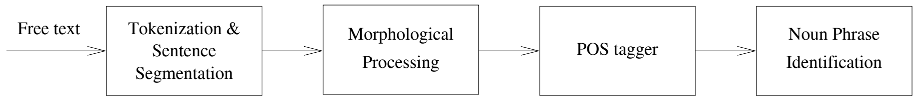

Free text
Tokenization & Sentence Segmentation
Morphological Processing
POS tagger
Noun Phrase Identification

Figure 9: The system architecture of natural language processing pipeline

The NP processing is performed sentence by sentence, but sentences can be divided into smaller parts between verbs and the border of clauses. The usage of this division makes NP processing faster. The initial phase of tagging a sentence is to label all possible trees in every word position of a sentence. Naturally there is some overlap-ping among the labeled NP trees and a word position can be the beginning of many NP trees. The preferred attributes for evaluating NP trees are the probability, phrase length (number of word positions in NP) and depth of the NP tree. There is a back-tracking algorithm that searches for the best (or best of n) NP tree series. After finding the best series of NP tree patterns, the actual NP tree structure of a given sentence can be reconstructed (built up) from tree patterns. The NP recognition algorithm of a sentence involves the following:

```
BEST = nil
Labeling of each word position with possible matching tree patterns
repeat
    CURRENT = next combination of tree patterns
    if (probability of CURRENT > probability of BEST)
        then BEST = CURRENT
until (not computed each possible combination of tree patterns)
```

Text category	**Precision**	**Recall**	$F_{\beta=1}$
Corpus version 1.0	75.72%	81.69%	78.59%
Business news extension	79.86%	86.63%	83.11%
Business news extension	79.86%	86.63%	83.11%

Figure 10: Test results on the two corpus domains.

7 Results

The method presented for learning and recognizing NP trees was applied on two cor-pus domains: the first version of the corpus containing texts from five different topic areas and the second domain from the business news extension. The sentences of domains were randomly divided into train (90%) and test (10%) sets. The preprocess-ing phase collected from the first domain had 297,077 complete NP trees and the unification learner produced 7,476 NP tree patterns. The second domain contains 51,112 complete NP trees and 1,856 acquired NP tree patterns. The training phase associated probability values with the patterns. The evaluating of the grammar was performed on 10% of the test set. A summary of test results is shown in Figure 10.

So far, the results seem encouraging. Based on experiments, this NP tagger as an element of a natural language processing pipeline provided enough information for information extraction. There are some differences in the results of the two corpus domains because of their various characteristics. The business news part contains relatively homogeneous texts with often repeated idioms. The first domain is more heterogeneous in its five topics. After analyzing the reasons for errors it was found that, in many cases, extra semantic information was needed to select the proper NP tree structure. Overall, the recognizing of large (long and deep) NP trees seems to be quite a hard problem.

8 Summary and future work

In the current paper, the author presented a general learning method for NP recogni-tion that has been applied to learning and recognizing NP trees. The NP tree learning method was applied to a previous machine learning method RGLearn. The NP recog-nizing method, as an element of a natural language processing pipeline, provides suf-ficiently rich information to support information extraction.

The NP tree learner method id based on rule generalization that includes unification algorithms. The generated grammar is a set of NP tree pattern that is described with regular expressions. After the unification phase probabilistic values are added to the grammar using error statistic analysis of the training examples. The NP tree tagger uses pattern matching and backtrack algorithm to search for the best-fitting NP tree sets. The NP tree structure of given sentence is reconstructed from tree patterns.

In the future, running parallel to the development of the Szeged corpus, the author intends to develop learning and recognizing method for constructing the syntax tree of sentences. The usage of ontological information that can be the extension of a mor-pho-syntactic description is also planned. The system will be primarily applied to the business news domain. In the 6th Framework Programme, an international research consortium is planning to develop a multilingual IE system to be applied to above-mentioned two domains.

References

[1] Abney S. (1991) Parsing by chunks, in Principle-Based Parsing. Kluwer Academic Publishers.

[2] Abney S. (1996) Partial Parsing via Finite-State Cascades, in Proceedings of ESSLLI'96 Robust Parsing Workshop, pp. 1-8.

[3] Argamon, S., Dagan, I., and Krymolowski, Y. (1998) A memory-based approach to learning shallow natural language patterns, in Proceedings of 36th Annual Meeting of the Associa-tion for Computational Linguistics (ACL), Montreal, pp. 67-73.

[4] Alexin, Z., Csirik, J., Gyimóthy, T., Bibok, K., Hatvani, Cs., Prószéky, G., Tihanyi, L. (2003) Manually Annotated Hungarian Corpus, in Proceedings of the Research Note Sessions of the 10th Conference of the European Chapter of the Association for Computational Linguis-tics EACL03, Budapest, Hungary, pp. 53-56.

[5] Erjavec, T. and Monachini, M., ed. (1997) Specification and Notation for Lexicon Encoding, Copernicus project 106 "MULTEXT-EAST", Work Package WP1 - Task 1.1 Deliverable D1.1F.

[6] Hócza, A., Alexin, Z., Csendes, D., Csirik, J., Gyimóthy, T. (2003) Application of ILP methods in different natural language processing phases for information extraction from Hungarian texts in Proceedings of the Kalmár Workshop on Logic and Computer Science, Szeged, Hungary, 1-2 October, pp. 107-116

[7] Kuba, A., Bakota, T., Hócza, A., Oravecz, Cs. (2003) Comparing different POS-tagging tech-niques for Hungarian in Proceedings of the MSZNY 2003, Szeged, Hungary, pp. 16-23

[8] Marcus, M. P., Santorini, B., and Marcinkiewicz, M. A. (1993) Building a large annotated corpus of English: the Penn Treebank, Association for Computational Linguistics.

[9] Muggleton, S. and Feng, C. (1992) Efficient Induction of Logic Programs, in Inductive Logic Programming (ed.: S. Muggleton), Academic Press, New York, pp. 281-297.

[10] Plotkin, G.D (1970) A note on inductive generalization, Machine Intelligence (eds: B. Meltzer and D. Michie), Vol 5.

[11] Ramshaw, L. A., and Marcus, M. P. (1995) Text Chunking Using Transformational-Based Learning, in Proceedings of the Third ACL Workshop on Very Large Corpora, Association for Computational Linguistics.

[12] Simov K. (2001) CLaRK - an XML-based System for Corpora Development, in Proceedings of the Corpus Linguistics 2001 Conference, Lancaster, pp. 553-560.

[13] Tjong Kim Sang, E. F., and Veenstra, J. (1999) Representing text chunks, in Proceedings of EACL '99, Association for Computational Linguistics.

[14] Tjong Kim Sang, E. F. (2000) Noun Phrase Recognition by System Combination, in Proceed-ings of the first conference on North American chapter of the Association for Computational Linguistics, Seattle, pp. 50-55.

[15] Váradi, T. (2002) The Hungarian National Corpus, in Proceedings of the Second International Conference on Language Resources and Evaluation LREC2002, Las Palmas de Gran Ca-naria, pp. 385-389.

[16] Váradi T. (2003) Shallow Parsing of Hungarian Business News, in Proceedings of the Corpus Linguistics 2003 Conference, Lancaster, pp. 845-851.

Received May, 2004