Scientific Bulletin of the „Petru Maior" University of Tîrgu Mureş
Vol. 9 (XXVI) no. 2, 2012, ISSN 2285 – 438X (Online), ISSN–L 1841 – 9267

EFFICIENT ENCODING OF INFLECTION RULES IN NLP SYSTEMS

Péter BARABÁS[1], László KOVÁCS[2]

Department of Information Technology
H-3515 Miskolc-Egyetemváros, Hungary

[1]barabas@iit.uni-miskolc.hu,
[2]kovacs@iit.uni-miskolc.hu

ABSTRACT

The grammatical parsing unit is a core module in natural language processing engines. This unit determines the grammatical roles of the incoming words and it converts the sentences into semantic models. A special grammar rule in agglutinative languages is the inflection rule. The traditional, automata-based parsers are usually not very effective in the parsing of inflection transformations. The paper presents implementation alternatives and compares them from the viewpoint of time efficiency and accuracy. The prototype system was tested with examples from Hungarian.

Keywords: natural language processing, grammar parsing, formal concept analysis

1. Introduction

Natural Language Processing is a main component in intelligent human-computer interfaces. The human oriented interaction in the form of speech dialog provides a high level of usability on a wide range of application areas. A core module in NLP is the grammatical parsing unit. This unit determines the grammatical roles of the incoming words and it converts the sentences into semantic models. The area of Computational Linguistics (CL) is aimed at developing efficient morpheme analyzer engines for parsing the input sentences. In the 1960's, the research in CL was dominated by the symbolic, generic approaches based on the works of Chomsky[9], Harris[12] és Shannon[14]. The main assumption of this approach is that the brain has an inherit knowledge of the human grammar. A child owns a general grammar model which is adapted later to a specific concrete language. Another dominating direction is the cognitive approach represented among others by Browing[8] és Wallace[15]. The cognitive approach uses a model where the brain is initially empty and during the training evolves the structure of a specific grammar. The child owns only a generic ability to build a grammar. In the practical implementations of NLP systems, a full functional grammar parser is required where the main requirements are the high speed (low execution cost), high precision and open interface. To meet these requirements, specific implementations are developed which may differ from the pure theoretical models in many viewpoints.

The main goal of the presented research is to compare the different practical implementation approaches on morpheme analysis for inflectional languages. The traditional approach is the application of a rule-based system with if-then rules like the Porter stemmer [1] works. Sometimes also a dictionary-based architecture can provide a more efficient solution too. In our investigation, also a grammar learning approach was developed to compare the efficiency and accuracy of the two main directions.

For target language, the Hungarian language was selected where the grammatical and semantic roles of a word are usually encoded with suffixes applying different inflection rules. The input word for the engine is an arbitrary word of the sentence, for example:

kutyáitokkal

where the meaning of this word is in English: with your dogs. The output of the engine is the morpheme map of the word like

kutya (stem) + i (plural) + tok (owned by you) + val (with)

As it can be observed, one of the difficulties to cope with is to manage the matching of the attached suffixes, like

tok + val ==> tokkal

The prototype systems were implemented in Java and C# languages, the paper includes the evaluation of the test results.

2. Grammar representations

A classic formalism for representing grammar is the Formal Grammar (FG) mechanism. In FG, Σ denotes the set of characters in the language. The symbol Σ^* is for the finite sequence of the characters. The language L is defined as a subset Σ^*. The language L can be given with a grammar G, where G is a tuple (N, T, P, S) with

- T : set of terminal symbols from Σ;
- N: set of new non-terminal symbols;
- S: the symbol for a sentence;
- P: set of production rules.

The language $L(G)$ denotes the set of sequences from Σ^* that can be derived from S using P. Based on the complexity of the rule-set, Chomsky [11] has defined four base classes. These four classes are defined as follows: regular grammar with the simplest rules; context-free grammars; context-depending grammar and recursively enumerable grammars. A widely investigated problem is the role of natural languages within the Chomsky classification. Chomsky has argued [12] that NL has context dependent characteristics. Others, like [10] consider NL as a subclass of regular language as every NL is a finite language, thus finite regular automata can be applied to parse the sentences. One practical drawback of the regular approach is that the related regular automaton is too huge for implementation.

The most widely used representation formalisms for formal grammars are the finite deterministic automata (FSA), the stack automata and the TAG architecture. In the case of FSA, every node of the automata corresponds to a non-terminal element of the grammar, while the directed edges are assigned to the terminal symbols [13]. In the case of stack automata, the automaton has a special memory to save the previous states. The TAG (Tree Adjoining Grammar) uses a hierarchy to store the grammar description. The TAG hierarchy is given with a tuple (N,T,I,A) where T denotes the terminal symbols, N is for the non-terminal symbols, I is a finite set of initial trees and A is the set of auxiliary trees. The leaves of the trees are either terminal symbols or non-terminal symbols that can be substituted by another auxiliary tree. The trees can be adjoined via the substitution leaves.

The cognitive linguistics appeared first in 1970's. One of the main pioneers of this approach is Langacker [17]. The cognitive approach assumes that the human language reflects the general cognitive processes of human mind. Several specific grammars were developed usually with some stochastic learning mechanism. The Word Grammar (WG) proposed by Hudson [18] belongs to this category. WG represents the grammar with a knowledge graph including all four levels of the human language, namely the semantic level, the syntax level, the morphology and the phonology.

A different approach is implemented in the Dependency Grammar (DG) proposed by Tesniere [19]. The DG uses dependency description between the words of a sentence. The dependency has a head (verbs) and some dependents. The dependency relation corresponds to grammatical functions. The dependency relationship is described with a dependency tree called stemma. The stemma is well-formed if [20]:

- One and only one element is independent;
- All others depend directly on some element;
- No elements depend directly on more than one;
- If A depends directly on B and some element C intervenes between them (in the linear order of the string), then C depends directly on A or B or some other intervening element.

A similar formalism is used in the Link Grammar (LG). The link grammar uses only binary relationships, i.e. complex relationships within a sentence are represented with relationships between two words of the sentence.

3. Rule-based stemming

In NLP systems the morphology has quite huge role to process input with higher accuracy. There are two main functions of morphology which generally applied in natural language processing: stemming and inflection. In the followings the stemming will be discussed in details.

There are many rule-based stemmer approaches for English language since 1960s. One of the most popular is the Porter-stemmer[1] because of its simplicity and efficiency. Consonants and vowels are distinguished by algorithms where if a letter is not a consonant then it is a vowel. A consonant is denoted by c, a vowel by v. The list of consonants of length greater than 0 can be denoted by C, and the list of vowels of length greater than 0 can be denoted by V. Using previous notations any word can be formalized with one of the following forms:

$$\begin{matrix} CVCV\ldots C \\ CVCV\ldots V \\ VCVC\ldots C \\ VCVC\ldots V \end{matrix} \qquad (1)$$

These formulas can be represented by a single form

$$[C]VCVC\ldots[V] \qquad (2)$$

where [X] denotes arbitrary occurrence of its content. The form can be simplified further with using $(VC)^m$ tag which represents the VC repeated m times. So the final formula can be written as follows:

$$[C](VC)^m[V] \qquad (3)$$

The stemming is performed in 5 steps using rewriting rules. Steps have predefined order and each step contains alternative rules. The rules define suffix replacements belonging to a given condition. A rule is denoted by the following form:

$$(\text{condition})S_1 \rightarrow S_2. \quad (4)$$

A rule can be applied if the ending of word fits to the S1 and after cutting S1 off the condition is fulfilled by the remaining stem.

A condition generally can be given in terms of m, e.g.

$$(m > 1)\text{ság} \rightarrow \quad (5)$$

The condition part can also contain the followings:

*S: the stem ends with S;
v: the stem contains a vowel;
*d: the stem ends with a double consonant;
*o: the stem ends with cvc, where the second c is not W, X, Y;
expressions: and, or, not.

If more than one rule can be applied then the rule with longest ending will be the winner. After successful appliance of a rule set, the algorithm jumps to the next rule set. If no rule fits in a set the process will continue with next set. After processing 5th rule set, the algorithm will terminate.

A general affix representation language has also been developed by Porter, called Snowball which can handle prefixes besides suffixes. Using Snowball [2] 14 European language has stemmer including Hungarian. Tordai-stemmer [6] is a Hungarian stemmer based on Snowball [2]. There are many alternatives of Porter-stemmer like Lovins-stemmer [3], Paice-Husk-stemmer [4] or Krovetz-stemmer [5].

Since natural languages are usually quite difficult to process, the accuracy of stemmer algorithms cannot be 100%. Even people can make mistakes for some ambiguous words. Generally three kinds of mistakes can be distinguished: under-stemming, over-stemming or misconstruction. The last two error types can be reduced with using of exception dictionary.

4. Architecture of morphology analyzer

The morphology of Hungarian language is very compound with rich inflection, high number of compound words and huge set of derivative form of words. In Hungarian language there are many well-defined grammar rules which can be applied in inflection but there are numerous exceptional forms of inflected words which are not fit to any rules. Inflection transforms a stem with agglutinating suffixes to it. The order of suffixes is not arbitrary; it can be described with a directed graph represented by a finite state automaton.

Regarding previous statements the structure of morphology module of our NLP system can be seen in following figure.

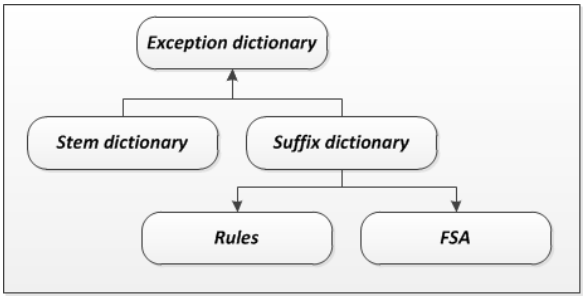

Fig. 1 – Structure of morphology module

The stem dictionary contains the stems of the language which can be defined in two ways. One of them declares that stems contain formative elements, so the stemmer will not cut them. This approach is used e.g. in Tordai-stemmer. The other procedure identifies the formative elements as suffixes also which have to be cut from stem off. The later approach tends to over-stemming more than the first one. In our proposed system, stem dictionary should contain the basic words of Hungarian language with allomorph information. Allomorph is used when there are different forms of the same word e.g. 'kettő', 'két'. In this case the allomorph of 'két' is 'kettő'. The structure of suffix dictionary can be seen in following Table 1.

Table 1. Structure of suffix dictionary

Field name	Description
ID	A number for identifying suffix.
Value	The value of suffix like *'-ban'*.
Vowel_harmony	The vowel harmony of stem besides the given suffix can be applied.
Type	The type of suffix: *'képző'*, *'jel' 'rag'*.
Code	The code of suffix like *cas<acc>*.
Start_class	The POS of word before using suffix.
End_class	The POS of word after agglutinating suffix.
Terminal	It is a logical value which indicates that the suffix is terminal or not.

Dictionary of exceptions contains the modified form (allomorph) of stem and references to the original stem and to a suffix. The suffix should be applied to get the modified allomorph of stem considering that not all inflection of the same stem is exceptional. E.g. in case of accusative of stem 'cső' the result inflected word is 'csövet', where the allomorph is 'csöv', the suffix is '-et' and the original stem is 'cső'. But if we use the '-ban' suffix the result will be 'csőben', which is not exceptional.

The rules are the fourth big part of morphology module which declares the substitution of character sequences during inflection. The rules are denoted similar to Porter-stemmer (4); the difference is the condition which denotes the place where the substitution can be made. So if matching rules are searching for a transformation in the center of the word, the terminal rules cannot be processed which can be applied only for the last suffix of word.

The last part of the module is the FSA which describes the possible suffix order for a given part of speech. It means that there are different FSAs for verb, noun, adjective and numeral. Certainly these graphs can be linked. E.g. when a verb will be transformed into noun with derivative suffix, the possible order of suffixes belongs to the noun's FSA.

After declaring the main structural element of morphology module, the tasks of analyzer should also be defined in the following list:

- Determining the stem of the word;
- Analyzing suffix chain which has been cut off from stem;
- Applying FSA to validate suffix chains and excludes invalid results.

The most important task is the stemming since the suffix determination depends on the stem. Stemming can have many ways: one of them is to cut suffixes off from the end of word until the final stem is reached. Other way is to get the first *n* characters of the word and try to match to a stored stem of language. In the proposed solution the second one is used. The algorithm is the following:

1. Let n=1.
2. Get the first n character of word.
3. Check if it has belonging stem.
    a. If stem is found, it adds to result list and go to step 4.
    b. If stem is not found, go to step 4.
4. If length of word is greater than n, increase n with 1 and go to step 2 else go to step 5.
5. Return list of possible stems.

After we get the possible stems, we should determine the suffix chain for each stem. The algorithm is the following:

1. Let p=1, n=1, L=length of longest suffix.
2. Get the first n characters of suffix chain from position p.
3. Check if character sequence is a suffix or not.
    a. If matching suffix is found, add it as a node to the result tree and go to step 4.
    b. It no suffix is found go to step 4.
4. Check if n is less or equal than L. If true, increase n and go to step 2, else go to step 5.
5. Iterate over result suffix list and do first four steps for remaining suffix chains. It should be called recursively for each result list.

The result of suffix chain analysis is a tree with empty root node. All branches should be validated using FSA. The valid branches are the result of the analysis considering a word can be more right analysis.

5. CL-based grammar parser

The inflection transformation has a very complex form. In our approach the rule is given with a set of distinct basic transformation rules. An atomic rule corresponds to an unambiguous simple conversion rule. Here are some examples for atomic rules.

*(#,t) : a character 't' is appended to the end of the word;

ab(i,o)(a,át): the first occurrence of 'abi' is replaced with 'abo' and the ending 'a' is replaced with 'át'.

In natural languages, the transformation rule depends on the base word. Thus, the inflection transformation can be considered as a classification problem where the base word is assigned to the best matching transformation class.

In the literature, inflection is usually controlled by a production rule system where the dominating solution is the application of FST or HMM[13] methods. In our project, a different approach was tested, namely the toolset of Formal Concept Analysis (FCA) [16]. The output of the FSA process is a lattice of formal concepts. This lattice represents the discovered concepts and the generalization and specialization relationships among the concepts. With application of special class label attributes, the concept lattice can be used as classification method. The input for the FCA analysis is a formal concept defined with $K(O_K,A_K,I_K)$ triplet where

- O_K: set of objects;
- A_K: set of attributes;
- I_K: a binary relationship between objects and attributes.

Two mapping functions are defined between objects and attributes with:

$$h_K(X) = \{a | a \in A_K : \forall o \in X : o I_K a\} \quad (6)$$
$$g_K(Y) = \{o | o \in O_K : \forall a \in Y : o I_K a\}$$

where

$$X \subseteq O_K, Y \subseteq O_K \quad (7)$$

A pair of closed object-set and attribute-set is called formal concept:

$$C(X,Y): h_K(X) = Y, g_K(Y) = X \quad (8)$$

Among the sets of formal concepts a partial ordering can be defined with

$$C_1(X_1,Y_1) \geq_K C_2(X_2,Y_2) \Leftrightarrow X_1 \supseteq X_2 \quad (9)$$

A pioneer work on application of concept lattices for classification is the proposal of Zhao and Yao [7]. In their approach, the attribute set of the context is extended with a class label. This label attribute denotes the class membership of the objects. The class label of a node is the aggregation of the class labels in the dominated sub-lattice. A concept in the lattice is consistent if its class label contains only one

class. A concept is most general consistent one if it is consistent but neither of its super concepts is consistent.

In the inflection rule concept lattice, the attributes of the context correspond to the labeled character sequences of the words. A label contains positional data on the sequences. The intension of a concept is given by a set of labeled substrings called generalized word. The generalized word at a new concept is constructed with intersection of the corresponding generalized words. A default class value is also defined here as the class with highest support within the dominated concept nodes. According to [7], a concept lattice can be converted into a decision tree for determining the class attribute from the content attributes. The generated decision tree is a binary rooted tree, where each node is assigned to a generalized word. The classification process at a given node works in the following steps.

- If the node consistent, the search terminates and the current transformation rule is applied.
- If the node is inconsistent, the child nodes are tested. If no child node exists, the default rule is applied; otherwise all child nodes are tested. The test determines a match similarity value for every child. The child with maximum similarity is selected as next target node.

The construction of the classification lattice is based on a corresponding training set. The training set contains samples on inflection rules, like (labda, labdát). In this example, the first word is the base word 'ball' and the second is the word in accusative.

The parser module contains beside the inflection engine another unit to manage the different suffixes. In the language, there is a relative rigid rule on combination of the different suffixes. As the order of the components can be described with a regular grammar, a Finite State Automata (FSA) was implemented to control the ordering of the morphemes. The FSA contains a finite set of states where every state here corresponds to a morpheme. There is an edge from morpheme A to morpheme B if AB is grammatical sequence of suffixes.

The third unit in the grammar module is the stem dictionary. The language has a set of valid stems which can be inflected with different suffixes. As this set is a list of static words it can be implemented with a trie (or prefix) structure. The trie structure is a special tree to store words. The words with the same prefix part share the same tree section started at the root. This structure is very suitable for efficient storage and search operations as the same prefix part is stored only once for several words.

6. Implementation and test results

The speed of grammar parsing depends among others on used structures, search algorithms, database operations and indexing. The test system was developed in Java language using MySQL database. The following optimizations are used for acceleration:

1. Connecting to database and querying data as less as possible.
2. Do not use IN clause in select statements for large datasets.
3. Cache as much database records as objects into memory as possible.
4. Make index for fields which are queried frequently. E.g. stem values, suffix values, etc…
5. Use ArrayList instead of Vector since first one is not synchronized and faster.
6. Use Hashmap instead of Hashtable since first one is not synchronized and faster.
7. Be aware of slow string operations.

We should optimize two factors: speed and memory costs. If we want to make as fast system as can than we will have the higher memory costs. If we want the least memory costs, the speed of analyzer will decrease.

Finally the following table summarizes the size of morphology database:

Table 2. Size of morphology database

Structure	Number of records
Stems	~94000
Exceptional words	~23000
Rules	~150
Suffixes	~540

Words form Szószablya [7] project has been used for training and testing. There are about 2.3 million of words in word database. The analysis of a single word takes approx. 2 ms, so 500 words can be analyzed per second on the average.

The other measure of analysis is the accuracy. From 2.3 million words the portion of incorrectly parsed words takes approx. 10-12%.

The grammar engine using concept lattice was implemented in Java application. The input for the application is a sentence in Hungarian. The output of the module is the morpheme structure of the words. For every word a morpheme analysis is performed. On the Figure 1, the input sentence is: Mit olvas Péter (what is Peter reading). The question word 'Mit' is parsed as

Mi | FNNM | ACC

where the first component denotes the stem, the second denotes the grammatical role of the stem, the third one is for the accusative case.

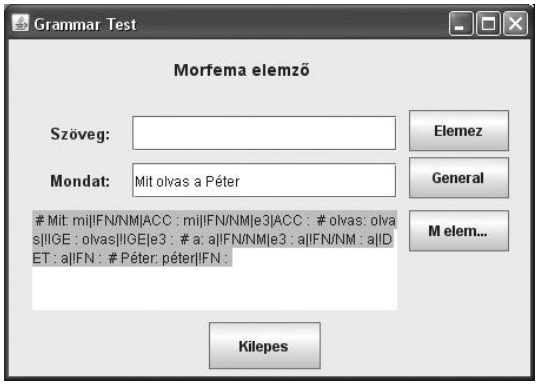

Figure 2, Morpheme parser application

Based on the test results, the following experiences can be emphasized:

- the speed of the parsing should be improved, it is significantly slower than the direct parsing methods;
- the generalization logic of the CL-based solution is similar to the human's logic;
- the CL-based solution requires a relative large memory storage, with a lot of non-used nodes.

Based on the experiences, the CL-based solution has some unique good property, but some further improvements are needed to apply it for large scale problem domains.

5. Conclusions

The grammar syntax parsing is a key module in natural language interfaces. A special grammar rule in agglutinative languages is the inflection rule. The traditional, automata-based parsers are usually not very effective in the parsing of inflection transformations. The paper presented some significant inflection rule implementation methods and compared them regarding speed and accuracy. The dictionary-based rule parsers provide the best efficiency regarding the speed factor. The paper presents also an alternative way using concept lattice –based classification method which shows a very good generalization capability.

References

[1] Porter MF (1980) *An algorithm for suffix stripping*. Program, 14: 130-137.

[2] Porter, Martin. *Snowball stemmers and resources page*. On line http://www.snowball.tartarus.org. [Visited 13/07/2005]

[3] Lovins J. B. (1968). *Development of a stemming algorithm. Mechanical Translation and Computational Linguistics*, II, 22-3 1.

[4] Paice, C. D. (1994). *An evaluation method for stemming algoritms*. In Proceedings of ACM-SIGIR94, pages 42–50.

[5] Krovetz, R. (1993). *Viewing morphology as an inference process*. In Proceedings of ACM-SIGIR93, pages 191–203.

[6] Tordai, A., de Rijke, M.: *Hungarian monolingual retrieval at clef* (2005)

[7] Halácsy P., Kornai A., Németh L., Rung A., Szakadát I., Trón V.: *A szószablya projekt* – www.szoszablya.hu. MSZNY 2003, 298–299, Szeged, Magyarország (2003)

[8] Browing, M.: *Null Operator Constructions*, Ph.D. thesis, MIT, 1987.

[9] Chomsky, N.: *Aspects of the Theory of Syntax*. Cambridge, MIT Press, 1965.

[10] Chomsky, N.: *Formal properties of grammar*, MIT Press, 1963.

[11] Gildea, D., Jurafsky, D.: *Automatic Induction of Finite State Transducer for Simple Phonological Rules*, Meeting of ACL, 1995.

[12] Harris, Z.: *Methods in Structural Linguistics*, University of Chicago Press, 1951.

[13] Manning, C., Schütze, H.: *Foundations of Statistical Natural language Processing*, MIT Press, 1999.

[14] Shannon, C.: Prediction and entropy of printed English, *Bell System Technical Journal*, 1951.

[15] Wallace,C.: Seneca Morphology, *International Journal of American Linguistic*, 1960.

[16] Jurafsky D., Martin J. H.: *Speech and Language Processing*, Prentice Hall, 2000.

[17] Krenn, B., Samuelsson C.: *The Linguistic's Guide to Statistics*, 1997.

[18] Hudson, R.: *Language Networks: The new Word Grammar*, Oxford University Press, 2007

[19] Tesniere,L.: *Elements de syntaxe structurale*, Paris, Klincksieck, 1959

[20] Robinson, J. *Dependency structures and transformation rules*, Language, 46. 259-285, 1970