18th European Union Contest for Young Scientists

Stockholm 2006

Gymnázium Jozefa Gregora Tajovského, Tajovského 25,

97411 Banská Bystrica

Syntactic parser of English sentences – proof-of-concept

Information/Computer Science

Author:

Andrej Pančík

Banská Bystrica, 2006

Slovak republic

Contents

Introduction

We started working on this project after we had read the book '*The Language of Mathematics*' (Devlin, 2003).The work of American linguist Noam Chomsky '*Syntactic Structures*' (Chomsky, 1966) is mentioned in the chapter dedicated to mathematical structures . Having read this work, which deals with problem of creation of generative grammar of English language, we decided to create a program application that does not generate English sentences but parses them. The name of this program application is, according to literature, the '*Syntactic parser of English sentences'*. Using this method, we hope to create an expert system that will reasonably react to the users' written sentences. A number of similar systems already exist (we refer to them as '*chatterbots*' - programs designed for pseudo-conversation), but they are mostly based on checking the occurrence of defined words. Therefore these systems can neither react to all the sentence constructions nor all the word forms (both neither defined nor derived). A few of the most famous chatterbots are Eliza, written in 1966, Ramona (Kurzweil, 2006) and ALICE (A.L.I.C.E., 2006). This approach (pseudo-conversation) is far away from our aim - the parsing sentences. After systematic web research, we concluded that this scientific field had not been fully explored and that no reliable program applications that could parse grammatically correct, comprehensible English sentences at least according to word classes had been created. We will now attempt to describe the procedure with which we approached the problem. After the creation of the initial base program, our second goal was to create a group of universal rules that this type of application can be based on. A table of the symbols and abbreviations used can be found in Table 4.

Theoretical analysis of syntactic parser

The problem of simple representation of grammar in a language is well known and quite well covered in literature. The work '*Syntactic structures*' (Chomsky, 1966) deals with the creation of a group of rules that can be used to generate grammatically correct English sentences. It divides the rules into two groups. The first group consists of rules used to create sentence patterns. These rules can then be used to create '*active statements*'. The second group consists of a transformation that changes an '*active statement*' sentence into a questioning, passive, or other type of sentence. An example of the rules in the first group can be found in Table 1. In the book '*Words and Rules*' (Pinker, 2003), there are many morphological rules that can be used to create past tenses, plural forms or compound nouns (i.e. '*mother-in-law*'). These rules seem to make a suitable base for the creation of '*reverse*' grammar (or syntactic parser). But unfortunately, that's not the case. During the creation of sentences, for example, the rule tells us that: In order to describe an attribute of something, simply put an adjective in front of the word that expresses that thing. In shorter terms, this operation can be expressed by NP > A + NP. Then we substitute 'A' by a word that can be an adjective. But if we want to create '*reverse*' grammar (group of rules that can be used to parse sentences), we cannot be sure whether the word we are looking at is an adjective or not (for example, '*battered*' can express an action in past tense or a state). Analogically, we can find homonyms among nouns, verbs and almost all the parts of speech. Nevertheless, it is not a problem of marginal fields of English grammar, but it regards common and most used words, for instance '*can*' - noun (metal container), verb; '*will*' - noun, verb; '*walk*' - noun, verb;

Analysis and concept of algorithmic solution

In this part we will deal with a concept of a program application that is used to demonstrate the functionality of parsing rules. Problem of shared meanings (details are described in the previous chapter) is not easy to solve. Fortunately it is not insoluble, if it were, we probably wouldn't use English, because we would not understand it. However, the solution requires a lot of time and research. We need a little heuristics for practical recognition of parts of speech. Little assumption will help us in the beginning. Assumption, that the article does not have a homonym, so we can unambiguously assign all occurrences of articles. This assumption is not hard to prove because there are only three articles '*a*', '*an*' and '*the*'. If there is a determinant in the sentence, there must be a counterpart - a noun. Hence, if we want to find a noun, we just go through the sentence, seeking a word that could be a noun. For this part there must be a dictionary that is already implemented in order to help us to decide whether the word could or could not be a noun. Having found the first possible noun, we can assume that all the words between the determinant and the possible noun are adjectives. This is the case in which we determined the first couple of meanings sharing (adjective – verb) because all countable nouns in singular form must be determined (attention - it can be determined also by other determiners, for example by pronouns, numerals etc.) and determiner is always before the noun (sometimes can be two or more possible nouns, for example '*car garage*') and adjectives always come after determiner. But this does not cover all the cases. In plural form, nouns do not need to be determined. To solve this partial problem we must firstly assign all unambiguous words. Then we go through the sentence and look for words that can be plural. These words are verbs (after a subject in 3rd form) or they are real plural forms. Having found a word like that we are look for a verb after it. If there is such a verb we just determine that it is plural - noun. If not, we determine

that it is a verb. This is a second case of vocal and written similarity of words. The third case is about modal verbs like '*can*' or '*will*'. If we obeyed the described process, in all but one case, we would rightly assume that '*can*' and '*will*' are nouns (we are talking now only about these cases). That case is when the nouns '*can*' and '*will*' are determined by possessive pronouns. Hence we can go through the sentence and look for a possible noun. Then, because of reasons described above, we can say that the word we have found is a noun. That is how we determined the cases when '*can*' and '*will*' are nouns. So we can now assign all other words that are similar to modal verbs (because all nouns '*can*' and '*will*' are already assigned). In this phase we know which word represents which part of speech. That is enough information to use reverse grammar. To be sure that our rules can parse all sentences we must include all transformation described in work '*Syntactic structures*' (Chomsky, 1966). The most effective way is storing the internal structure in a graph. Strictly speaking, in a binary tree - data structure in which each component has up to two successors - children. Tree like this can look like Figure 1. Everytime the rule is applied; two children in the tree join together into their superior component. Final result is tree structure in which there is a component '*Sentence*' on the top. Next to the top component we can find other complementary components. In simple English sentence there is almost always a subject in the first place, verb in the second and object (if any) in the third place. After them there are adverbials of manner, place and time. Therefore if we check left child of the component '*Sentence*', we will know the subject part of the sentence. Analogically, we can take right part of the sentence for the verb part of the sentence.

Description of program application

Implementation procedure of the algorithm, described in the previous part, into different programming languages varies. Complexity of a program lies in structures chosen to store components. We have decided to use the programming language Pascal with IDE (integrated development environment) Turbo Pascal 7 from Borland. From the beginning the program was treated as an experimental application dedicated to demonstrate our results, not as a final product. First phase consists of parts of speech recognition. We follow the procedure described in the previous part. (Table 2). After this phase the generation of grammar tree follows. For representation of this tree we use an array of components. Each component points to its superior and inferior components. It is generated using the rules based on the work '*Syntactic structures*' (Chomsky, 1966). List of these rules is in Table 3. They were completely reconstructed and enriched for our needs. We store suffixes of words before the word itself for better manipulation (you should consider it before using of our rules). Possible suffixes are '*S*' (3rd person singular), '*Ing*' (Ing-form), '*En*' (our representation of past tense and past participle). Second very important thing is that the rules must be executed in the listed order.

Discussion

The described syntactic parser has many fields of applications. In combination with the products of Microsoft – Speech (Microsoft, 2006), it is possible to implement the speech recognition directly for transforming the spoken sentences into written sentences. These sentences can be then transformed into grammar tree by the syntactic parser. In this way we can construct for example an elevator that moves like you want or a fridge that understands your orders. Utilization of this technology in health services can make the controlling of beds for immobile patients by voice possible. By using the syntactic parser we can improve machine translation of English into other languages. It can also be used to extend the abilities of '*antispam*' filters. As an extension it is possible to create a semantic analyzer. With this semantic analyzer we can improve existing '*chatterbots*' (this term is illustrated in the introduction) and we can make an illusion of the computer answering like a human being. Generation of sentences (with slightly adjusted rules) from database of information can help create information system for doctors (after submitting symptoms, computer specifies diagnose and lists the full medical procedure).

Conclusion

Program application that we have created by the analysis of '*reverse*' grammar partly fulfils our thesis from the beginning. Unfortunately it is far away from what we need for practical use. There are several reasons for that. For example, the parsing of sentences with multiple word classes created by conjunctions (e.g. and, or) is not implemented. Analyzed sentence must be simple and must constitute a statement. Sentence must not contain '*not*', shortened forms (e.g. 're, 's), '*like*', proper nouns, adverbs and numerals. Intelligent detection of error input (input sentence must be grammatically correct) is not implemented yet. All verbs must be in one word (so the phrasal verbs are excluded). In our work we mainly wanted to proof the concept of reverse grammar implementation ('*proof-of-concept*' - a short and/or incomplete realization of a certain method or idea(s) to demonstrate its feasibility). Therefore we can forget about the imperfections in our program application. We are still working on this project and it is very probable that the described problems will be solved in the horizon of few months.

Bibliography

A.L.I.C.E. AI Foundation (2006). http://www.alicebot.org/.

Belán, J. (2004). Zmaturuj z anglického jazyka 1. Brno: Didaktis.

Devlin, K. (2003). Jazyk matematiky. Prague: Argo & Dokorán.

Chomsky, N. (1966). Syntaktické štruktúry. Prague: Akademie věd.

Kurzweil, R. (2006). http://www.kurzweilai.net/ramona/bottom.php.

Microsoft. (2006). http://www.microsoft.com/speech/download/sdk51/.

Pinker, S. (2003). Slová a pravidlá. Bratislava: Kalligram.