# Interactive System for Creation of Notes

Martin NEMČEK*

*Slovak University of Technology in Bratislava*
*Faculty of Informatics and Information Technologies*
*Ilkovičova 2, 842 16 Bratislava, Slovakia*
`xnemcekm@stuba.sk`

**Abstract.** We are overwhelmed by information from various topics. The challenge in education is to create notes which covers important subset of information. There are known methods to extract information from text. In this article we propose a system to extract the notes from text which are important for educational purpose, so it should create personalized notes for students. We use mainly syntactic text analysis. Notes are created by help of part-of-speech tags and dependencies between words in sentences. The outcome will be an interactive application for creating notes based on learned rules from user.

## 1  Introduction

Computers are not able to understand information in natural language. There are some methods how to process texts and extract information from it.

In our proposed system the notes are created from sentences by extracting relevant information from them. We use syntactic analysis of sentences and extract relations and dependencies between words from these sentences. The final result of our proposed method are personalized notes. The user will be able to modify the automatically created notes. The system will then learn new rules from these changes and takes them into account for the next time.

## 2  Our proposed system

A rule consists mainly from two parts - *list of data of original sentence* and *list of data of note*. Each entry in *list of data of original sentence* and *list of data of note* contains these parts: relation name and list of grouped dependencies with the same relation name. Every dependency contains a governor token, a dependent token and its position considering all dependencies. The governor and dependent token consists of Part-Of-Speech (POS) tag and index of word in sentence to which is token connected. Index of the token is bounded with a position of its word in sentence.

Dependencies from the second list are applied to sentence to create a new note. The rule may order to create a compound note from a sentence. The compound note is composited of some simple sentences. The positions in sentence on which the note should be split into smaller sentences are

---

kept within the rule.

When processing a sentence an applicable rule has to be looked up in database before creation of the note. Dependencies of rule and dependencies of the sentence being processed has to correspond to each other. Evaluation is based upon two conditions. The sentence that is being processed has to have the exact amount of entries in list of data of original sentence while these entries contain exactly the same relation names as the rule's relation names.

The applicable rule is found if these two conditions are met. However, the conditions can cause a situation that more than one rule is found. In this case we have to calculate the match probability of this sentence and the original sentence obtained from the rule. The rule with the highest probability of the match is applied.

Calculating the match consists of several steps. First, the POS tags match of governor and dependent tokens is calculated separately. Indices of governor and dependent tokens are calculated also separately. These first steps determine if the sentence contains arbitrary dependency with same value of POS tag or index. In followed step is determined a half-match of dependencies. Half-match of dependency is match of POS tag and index at the same time at governor or dependent token of dependency. We calculate matches of POS tags and index of governor or dependent token for every dependency. Finally, in the last step we calculate the number of absolute-matched dependencies. Absolute-match dependencies is the total match of POS tags and indices in governor and dependent tokens. Every step has assigned a rating. If a condition in the step is evaluated as true, the rating of the step is added to the final result. The final result is a percentage value of the match. The rating is based on importance of the step in calculating a precise match, while depending on the number of steps and dependencies, so the final result cannot exceed a limit of 100%. A pseudo code for an algorithm calculating the match is shown in Algorithm 1 and specific example is section 2.1.

---

**Algorithm 1** Calculating match

---

1:  **procedure** CALCULATEMATCH($sentence, originalDependencies$)
2:      $oneCompareTypeRating \leftarrow$ calculate percentage rating of one comparison
3:      **for all** $originalDependencies$ **do**
4:          **if** count($sentence, dependency$) = count($originalDependencies, dependency$) **then**
5:              $match \leftarrow match + oneCompareTypeRating$
6:          $counter \leftarrow counter +$ count($originalDependencies,dependency$)
7:      $oneCompareTypeRating \leftarrow oneCompareType/counter$
8:      **for all** $originalDependency$ **do**
9:          **for all** $dependency$ **do**
10:             **for all** $comparison$ **do**
11:                 **if** applyComparison($sentence, comparison, dependency$) **then**
12:                     $match \leftarrow match + oneCompareTypeRating$
            **return** $match$

---

If rule look up does not find any applicable rule, it means that the system have not processed the same or similar sentence yet. A manual rules of parser are used in this case. The output of the parser is a note. A new rule is created based on the note. Dependencies of original sentences are taken and used to create a *list of data of original sentence*. This list is then assigned to the rule. Dependencies of note are used to create a *list of data of note* which is then also assigned to the rule. The sentence ends are determined depending on how many sentences the note contains. POS tags and indices of tokens are stated by the corresponding words of the original sentence and the newly created note.

By the principle of rule look up, the sentence being processed has to contain dependencies from the *list of data of original sentence* and also dependencies from the *list of data of note*.

The process of applying a rule has several steps. For each dependency in the list of data of original sentence, the respective dependency is looked up in sentence that is being processed. The

word corresponding with dependent token from the looked up dependency is taken and added to the note on its index position. In case of dependency relation *nominal subject* the word corresponding with governor is also added. After processing all dependencies the last minor changes are done such as capitalization of the first letter of the note, splitting note into more sentences if rule defined so. Algorithm 2 shows pseudo code of the process of applying rule on sentence.

---

**Algorithm 2** Applying rule

---

1:  **procedure** APPLYRULE($sentence, rule$)
2:      $note \leftarrow$ new Note
3:      **for all** $ruleDependencies$ **do**
4:          $dependency \leftarrow$ findDependency($sentence, ruleDependency$)
5:          **if** isFound($dependency$) **then**
6:              add($note$, getDependent($dependency$))
7:              **if** isNominalSubject(relation($dependency$)) **then**
8:                  add($note$, getGovernor($dependency$))
9:      splitToSentences($note$, sentencesEnds($rule$))
          **return** $note$
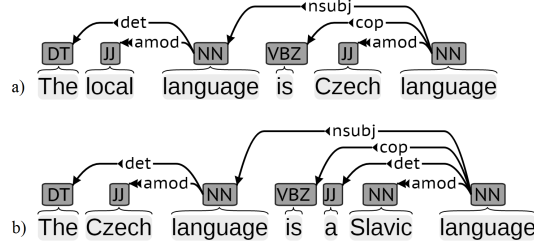
---

## 2.1   Example



*Figure 1: Example sentences*

Consider situation when we have rules for sentences "The local language is Czech language." (a) and "The Czech language us a Slavic language."(b) and we are processing the first one. Figure 1 shows structures of the mentioned sentences. There occurs a situation where at least two rules from these two sentences are applicable for the first sentence. Assume that we are processing sentence (a) and calculating match with sentence (b). In this process we iterate over all dependencies of processed sentence (a). In this case, the first dependency is det with governor token's POS tag NN and index 3 and dependent token's POS tag DT and index 1. First, we find out, if destination sentence (b) contains any dependency, which has any token with POS tag of NN or DT and any dependency with any token having index equal to 3 or one. This is the separate calculation of POS tags and indices. In the second step, we try to find in destination sentence (b) any dependency, which has dependent or governor token with POS tag of NN and index of 3 or POS tag of DT and index equal 1. This is the half-match step. The last step consists of checking, if destination sentence (b) contains dependency, where governor token's POS tag is NN and index is 3 and dependent's token POS tag is DT and index is 1. If any of these step were true, the rating of that particular step is added to the final result and iteration continues with following dependency calculating at the same principle until all dependencies were iterated over.