A Constraint Based System for Lexical Knowledge Retrieval

Stoyan Mihov

Linguistic Modelling Laboratory
Laboratory for Parallel and Distributed Processing
Bulgarian Academy of Sciences
stoyan@bgcict.acad.bg

September, 1996

Abstract

This paper is concerned with the lexical knowledge retrieval system created at the Linguistic Modelling Laboratory. The main goal of the system is to provide a powerful and comfortable interface for lexical knowledge retrieval from large morphological dictionary. To achieve this a constraint based approach is applied that leads to a very effective algorithm. The algorithm for query building, which is also used for retrieving of general grammatical knowledge, is presented in details. In our opinion this method is very suitable for knowledge retrieval in domains with complex and irregular classifications.

1 Introduction

Recently, many systems containing large amount of lexical knowledge have been built. They make use of different approaches for processing and knowledge representation. Detailed study of the problem is presented in [Pu95] and [Br92].

The Linguistic Modelling Laboratory is working on a Large Morphological Dictionary that will cover most of the wordforms in modern Bulgarian (see [Pa93]). Now the system contains grammatical information for more then 500'000 wordforms and is systematically upgraded. The grammatical information is structured in Features Structures (refer to [Sh86] for a good introduction). It is known that Feature Structures (FSs) are de facto standard for representing linguistic information. That is because they allow comfortable description of knowledge with complex classifications and many irregularities.

The program which interfaces the lexical knowledge in our system is called kernel program. We can think the knowledge as a set containing all 500'000 feature structures corresponding to the information of wordforms. The kernel input is a Feature Structure – the

Input FS (Query)	Output FSs
$\begin{bmatrix}\text{wordform} & \text{`BOEHHA'}\end{bmatrix}$	$\begin{bmatrix}\text{wordform} & \text{`BOEHHA'} \\ \text{stem} & \text{`BOEHEH'} \\ \text{lexeme info} & \begin{bmatrix}\text{part of speech} & \text{adjective} \\ \text{gradability} & \text{ungradable}\end{bmatrix} \\ \text{grammeme info} & \begin{bmatrix}\text{degree} & \text{positive} \\ \text{gender} & \text{feminine} \\ \text{number} & \text{singular} \\ \text{definiteness} & \text{indefinite}\end{bmatrix} \\ \text{ending} & \text{`A'} \\ \text{alternation} & \left\langle \begin{bmatrix}\text{from} & \text{`E'} \\ \text{to} & \text{`'} \\ \text{pos} & 2\end{bmatrix} \right\rangle\end{bmatrix}$
$\begin{bmatrix}\text{stem} & \text{`CTABA'} \\ \text{grammeme info} & \begin{bmatrix}\text{tense} & \text{present} \\ \text{person} & 3\end{bmatrix}\end{bmatrix}$	$\begin{bmatrix}\text{wordform} & \text{`CTABA'} \\ \text{stem} & \text{`CTABA'} \\ \text{lexeme info} & \begin{bmatrix}\text{part of speech} & \text{verb} \\ \dots & \end{bmatrix} \\ \text{grammeme info} & \begin{bmatrix}\text{tense} & \text{present} \\ \text{person} & 3 \\ \text{number} & \text{singular} \\ \dots & \end{bmatrix} \\ \dots & \end{bmatrix}$ $\begin{bmatrix}\text{wordform} & \text{`CTABAT'} \\ \text{stem} & \text{`CTABA'} \\ \text{lexeme info} & \begin{bmatrix}\text{part of speech} & \text{verb} \\ \dots & \end{bmatrix} \\ \text{grammeme info} & \begin{bmatrix}\text{tense} & \text{present} \\ \text{person} & 3 \\ \text{number} & \text{plural} \\ \dots & \end{bmatrix} \\ \dots & \end{bmatrix}$

Figure 1: Examples of the kernel functioning.

query constraint that should be satisfied. The output contains all Feature Structures included in our knowledge base, unifiable with the input. See Figure 1 for some examples of the kernel functioning. It is clear that it is very inefficient to keep all 500'000 FSs in memory and to check every one for the query constraint satisfaction. The kernel program uses a synthesizing algorithm which is explained in details in [Mi96]. Actually the grammatical information of the output Feature Structures is built by the unification of certain *basic constraints*. Each basic constraint is a Feature Structure which corresponds to a feature-value pair of the grammatical information. In Figure 2 there are some examples of Feature Structures[1] which present basic constraints corresponding to feature-values.

This paper presents in details our approach for query building. It is shown how general grammatical knowledge is retrieved by our procedure. In the second section we describe the problem for query building. Then in Section 3 the algorithm is presented and explained. In Section 4 are some details about the implementation. Some comments on possibilities for generalization are given in the conclusion.

2 Problem description

The main goal of the system is to provide an appropriate way for knowledge retrieval. There are very different necessities concerning system interaction. For example in syntactic analysing systems, the grammar information should be extracted after the input of wordform, as it is shown in the first example in Figure 1. In Natural Language Generation the system should find the wordform by processing input of a stem and some grammatical information. The professional linguist should be able to extract all wordforms (or stems, endings...) which

[1] For a clearer presentation, in the example we are noting the unbound anonymous variables explicitly with '_'.

Feature-value	Corresponding basic constraint
part of speech – noun	$\begin{bmatrix} \text{wordform} & \_ \\ \text{stem} & \_ \\ \text{ending} & \_ \\ \text{lexeme info} & \begin{bmatrix} \text{part of speech} & \text{noun} \\ \text{noun type} & \_ \\ \text{animateness} & \_ \\ \text{humanness} & \_ \\ \text{gender} & \_ \\ \text{verb type} & \text{not defined} \\ \text{gradable} & \text{not defined} \\ \text{transitivity} & \text{not defined} \\ \text{numeral type} & \text{not defined} \\ \ldots & \end{bmatrix} \\ \text{grammeme info} & \begin{bmatrix} \text{number} & \\ \text{definiteness} & \_ \\ \text{article form} & \_ \\ \text{tense} & \text{not defined} \\ \text{person} & \text{not defined} \\ \text{case} & \text{not defined} \\ \ldots & \end{bmatrix} \\ \ldots & \end{bmatrix}$
non-finite form – participle	$\begin{bmatrix} \text{wordform} & \_ \\ \text{stem} & \_ \\ \text{ending} & \_ \\ \text{lexeme info} & \begin{bmatrix} \text{part of speech} & \text{verb} \\ \text{verb type} & \_ \\ \text{noun type} & \text{not defined} \\ \text{gradable} & \text{not defined} \\ \text{transitivity} & \text{not defined} \\ \text{numeral type} & \text{not defined} \\ \ldots & \end{bmatrix} \\ \text{grammeme info} & \begin{bmatrix} \text{finiteness} & \text{non-finite} \\ \text{non-finite form} & \text{participle} \\ \text{number} & \_ \\ \text{tense} & \_ \\ \text{definiteness} & \text{not defined} \\ \text{case} & \text{not defined} \\ \ldots & \end{bmatrix} \\ \ldots & \end{bmatrix}$

Figure 2: Example of feature-value pairs and the corresponding *basic constraints*.

satisfy a certain constraint. On the one hand, there should be no limitations on the contents of a query constraint. But on the other hand, many of the queries are inconsistent with the grammatical knowledge. For example there are no items in the knowledge base, which satisfy following FS: $\begin{bmatrix} \text{lexeme info} & \begin{bmatrix} \text{gradability} & \text{gradable} \end{bmatrix} \\ \text{grammeme info} & \begin{bmatrix} \text{tense} & \text{imperfect} \end{bmatrix} \end{bmatrix}$. That is because only adjectives and adverbs are gradable and they don't allow a tense characterization. It is meaningless to process a query which we can a priori consider as inconsistent. That is why we have to implement a more sofisticated query building algorithm.

We think that the best way is to build the query incrementally. That means that the user should be able to specify the query step by step, selecting a feature-value pair. The purpose is to receive information which feature-value pairs are acceptable (do not lead to an inconsistent query) after every step. If we switch those feature values as unacceptable, the user won't be able to build an inconsistent query. Moreover the user will receive general grammatical information about the consistency of some constraint combinations.

Another advantage of the system would be if the algorithm automatically selects some feature-value pairs which are derivable from the existing query specification. For example if the user selects 'part of speech – verb' and 'gender – masculine' then the 'finiteness' has to be 'non-finite'.

To achieve the above mentioned requirements for the query building procedure, a deduction procedure has to be implemented. This procedure should check on every step the consistence of every feature-value pair with the query and should select the feature-values which are deducible from the specification of the query.

3 The Query Building Algorithm

The simplest way to fulfit the above algorithm specification is to generate a list of all possible combination of feature-values. Unfortunately there are thousands of possible combination of grammatical feature values in the system. That is why an algorithm based on this information would be very inefficient. Our algorithm is based on the *basic constraints* corresponding to feature-values which are only about 150 in the system. Those constraints are already defined in the system, because the kernel program is producing the result via their unification.

The FSs used in our application are of the classic type. That means that they are not sorted and we don't allow negation and disjunction inside the FSs. The generalization of the algorithm in order to use disjunctive FSs is not a serious problem, but if we want to use negation inside the FSs, the algorithm should be generally revised. Using negation we loose the nice classic semantic about FS's – the interpretation that a FS represents partial knowledge. For a comprehensive study of FS semantics see [Ki89].

Some notion preliminaries: when we write a feature-value pair in fact we mean a pair of feature path and value, where feature path is a list of features. In our application we are interested only in features carrying grammatical information (other features like 'stem', 'wordform' etc. could not be classificated). That is why we note only the last feature in the path and the value and call this a feature-value pair. In the application there are about 60 features (feature paths) carrying grammatical information. All feature-value pairs are about

150. This is a rather small number, hence the algorithm based on this information will be comparable effective. Bellow we present our algorithm which satisfies all requirements mentioned above.

Algorithm 1 *The Query Building Algorithm.*

```
Step 1. Set the initial query FS to the empty FS.
        Set all Feature-value pairs to ‘acceptable’.
Step 2. Wait the user to selects a (‘acceptable’) feature-value pair.
Step 3. Unify the query FS with the basic constraint corresponding to
        the selected feature-value pair and mark it as ‘sellected’.
Step 4. For every ‘acceptable’ basic constraint
        check the unifiability with the query
           if the basic constraint don’t satisfy the query
           then set it to ‘unacceptable’
Step 5. For every feature check
           if exactly one value for this feature is acceptable then
              select this feature-value and go to Step 3.
Step 6. if there are no more acceptable pairs or the user finished
           then go to Step 7 else go to Step 2.
Step 7. Call the kernel program with the query as input.
```

This algorithm is almost self-explaining. In Step 1 the initialization is made. Step 2 and Step 3 build the query by unification of the basic constraint corresponding to the selected feature-value pair. Step 4 checks all other basic constraints for consistence with the query and switches all feature-values which are inconsistent with the query to an ‘unacceptable’ state. Step 5 is responsible for the automatic deduction of feature-values. Step 6 and Step 7 close the loop and invoke the kernel program respectively. We omit a detailed proof about the correctness of this algorithm which in our point of view is rather obvious. Maybe the only non trivial problem is the termination. The following lema is concerned about that.

Lema 1 *The Algorithm 1 is always terminating.*

Proof The two loops are always passing through Step 3 where a feature-value pair is set to ‘selected’. There are only a finite number of pairs. Hence after a finite number of iterations there will be no more ‘acceptable’ pairs, which guaranties the termination of the algorithm. Please note also that in the Algorithm is essentially used the “Closed World Assumption”. In Step 5 we assume that there are no other values possible for the feature. In our application, where the general grammatical knowledge is fixed, this assumption is generally true. In fact, exactly this Step is responsible for the automathic feature-value selection, when the value of a feature can be deduced from the information already specified by the query. Without the ‘Closed World Assumption’ we couldn’t deduce anything new in our system.

4 Implementation

We have created a prototype version in Sicstus Prolog with Tcl/Tk on Windows environment, which is able to extract about 20 FS per second. The system is supplied with a very friendly user interface. The query is created using the Windows Graphical User Interface. By clicking on a feature in the list box, another list box is displayed, where the acceptable values for this feature are listed. By selecting a value, the unification with the corresponding constraint is invoked. If some feature or value is disabled then the corresponding entry in the list box will be switched "gray" (unacceptable). There are several options for the output format. The user can choose between the output of the whole FSs or only the values of some features (e.g. wordform, stem etc.).

A more faster version will be created using C/C++ language soon. This implementation will provide a retrieval speed of about 200 FS per second. There will be no other differences between the C/C++ and the Prolog version. We hope that this system will be widely used for Bulgarian language education and research purposes.

Also there is a World Wide Web version planned. The idea is to specify the query using the form options in HTML. Then the query will be passed to the knowledge retrieval system using a CGI-script. In this way the resources will be accessible through INTERNET.

5 Conclusion

The most interesting part of the algorithm in our opinion of view is the untraditional deduction procedure. It is clear that the application is very simple. That is why in fact the loop Step 3 – Step 5 will not make new changes to the acceptability of feature-value pairs. We think that this deduction procedure could be classified as new approach to certain problems. It leads to a very elegant and effective algorithm for deduction in domains with complex classifications. In current version only classic feature structure interpretation and unification in empty theory are applied. At the moment we are working on the generalization of this approach to allow more powerful constraint based technics.

At the end I want to thank to all member of the team of Linguistic Modelling Laboratory for the support and fruitful atmosphere, without which this work could not exist.

References

[Br92] Ted Briscoe, Ann Copestake, Valeria de Paiva (eds.): *Default Inheritance in Unification Based Approaches to the Lexicon.*, ESSLLI'92 readings. 1992.

[Ki89] Paul J. King: *A logical formalism for head-driven phrase structure grammar.* Doctoral dissertation. Manchester University, Manchester, England. 1989.

[Mi96] Stoyan Mihov: *A Constrained Based Approach for the Large Bulgarian Morphological Dictionary.* To appear.

[Pa93] Elena Paskaleva, Kiril Simov, Mariana Damova, Milena Slavcheva: *The long journey from the core to the real size of a large LDB.* In: Acquisition of Lexical Knowledge from Text (eds. Boguraev, Pustejowski), Columbus, Ohio, 1993, 161 - 169

[Pu95] James Pustejovsky: *The Generative Lexicon.* MIT Press. 1995.

[Sh86] Stuart Shieber: *An Introduction to Unification-Based Approaches to Grammar.* CSLI Lecture Notes #4, 1986.