Converting the Corpus Query Language to the Natural Language

Daniela Ryšavá[1], Nikol Volková[1], Adam Rambousek[2]

[1] Faculty of Arts, Masaryk University
Arne Nováka 1, 602 00 Brno, Czech Republic
399364@mail.muni.cz, 399909@mail.muni.cz
[2] Natural Language Processing Centre,
Faculty of Informatics, Masaryk University
Botanická 68a, 602 00 Brno, Czech Republic
rambousek@fi.muni.cz

Abstract. This paper presents the first version of a web application designed to convert queries in the Corpus Query Language to the natural language. The purpose of this application is to help users who want to learn and work with Corpus Query Language, providing tool to find out the explanation of the CQL query. In the future, when the application supports more features of CQL, it may be included as a hint in the web interface of a corpus search engine.

Keywords: Corpus Query Language, CQL, natural language generation, corpus

1 Introduction to Corpus Query Language

Corpus Query Language (CQL) [1,2] is a formal language that allows you to search corpus for grammatically complex patterns. This language was developed at the University of Stuttgart in 1990. Each part of the CQL query consists of a selected attribute and required value, enclosed in square brackets, in the form of `[attribute="value"]`. Apart from the whole word, it is possible to specify regular expressions as values, using special symbols and wildcards to substitute various character composition.

Most common are three types of attributes:

- **lemma**, basic word form, e. g. an infinitive for a verb, a nominative singular for a noun,
- **word**, word in the particular form,
- **tag**, morphological tag, e. g. Czech tags used by the majka tagger [3] *k1* for nouns, *c1* for nominatives, *nS* for singular.

Query may be modified with the exclamation mark to negate the value query, see Example 1. Users may expand the CQL query with various advanced features, some of them are explained in Tables 1 and 2.

Aleš Horák, Pavel Rychlý, Adam Rambousek (Eds.): Proceedings of Recent Advances in Slavonic Natural Language Processing, RASLAN 2015, pp. 43–48, 2015.

Example 1. Query: `[word!="dog"]`
Interpretation: searched expression is not the word *dog*

Table 1. Operators

.	full stop	any character
[]	square brackets	any token
containing	structure containing	structure containing the searched expression
within	within structure	searched expression within structure
<s/>	structure = sentence	structure markup used with operators
<p/>	structure = paragraph	*containing* or *within*
<doc/>	structure = document	

Example 2. Query: `[lemma="ye.."]`
Interpretation: searched expression is a lemma beginning with *ye* followed by two characters: *year, yelk, yeti...*

Example 3. Query: `<s/> containing ([word="dog"][][lemma="cat"])`
Interpretation: searched expression is a sentence containing the word *dog* followed by one arbitrary token and the word *cat*

Example 4. Query: `([lemma="dog"][][lemma="cat"]) within <doc/>`
Interpretation: searched expression is the word *dog* followed by one arbitrary token and the word *cat* within a document

Table 2. Quantifiers

*	asterisk	iteration in range from zero to infinite
+	plus	iteration in range from one to infinite
?	question mark	iteration in range from zero to one
{n}	range in braces	exactly n iterations
{n,}	range in braces	iteration in range from n to infinite
{n,m} or {n, m}	range in braces	iteration in range from n to m

Example 5. Query: `[lemma="moo*"]`
Interpretation: searched expression is the word *mo, moo, mooo...* (the lemma *mo* with arbitrary iteration of the character *o*)

Example 6. Query: `[tag="k5.*"]+`
Interpretation: searched expression is the verb occurring at least one time

Example 7. Query: `[lemma="hoo?d"]`
Interpretation: searched expression is the lemma *hod* and *hood*

Example 8. Query: `[lemma="dog"] []{2} [lemma="cat"]`
Interpretation: searched expression is the lemma *dog* and *cat* with exactly two arbitrary tokens between them

Example 9. Query: `[word="po{2,}r"]`
Interpretation: searched expression is the word *poor* with two or more characters *o*

Example 10. Query: `[tag="k3.*"] []{0,3} [tag="k1.*"]`
Interpretation: searched expression is a pronoun and a noun with no more tokens between them, or up to three arbitrary tokens between them

2 Converting the query

The application is developed in Python 2 programming language. There are two ways to run the application, either as the command line script, or the web application. Both interfaces use the same back-end algorithm to generate the sentence in natural language. As of now, the application produces Czech sentences, but it is possible to add support for other languages.

Input CQL query is split to tokens (enclosed in square brackets), while also detecting any structure operators. In the next step, each token is processed separately, producing parts of the Czech sentence, taking into account both the token query, and any quantifiers. Afterwards, the complete sentence is generated.

The application supports only some features of the CQL and the query has to follow the standard CQL structure. The attribute of the query needs to be specified by typing *word*, *lemma* or *tag* and the value (the searched item) has to be enclosed in double quotation marks, e. g. *"dog"* or *"k1.*nP.*"*. For example, the query may follow the form `[lemma="dog"]` or `[tag="k1.*nP.*"]`.

2.1 Searching for words and lemmas

If users want to search for certain word or lemma, they use the default attribute-values queries, as seen in Figure 1.

```
Dotaz v CQL: [lemma="chytat"][word!="lelky"]
Hledaný výraz je slovo, které má základní tvar "chytat", následuje slovo
jakékoliv kromě "lelky".
Dotaz v CQL: [lemma!="ztratit"][word="hlavu"]
Hledaný výraz je slovo, které má základní tvar jakýkoliv kromě "ztratit",
následuje slovo "hlavu".
```

Fig. 1. Searching for words and lemmas.

2.2 Searching for Part of Speech

Users may also search for particular morphological tags. In current version, the application supports the attributive tagset, used in the ajka/majka morphological analyzer[3]. If users want to specify combination of morphological tags, they have to enter the tags in the same order that is produced by the morphological analyzer. See Figure 2 for an example.

```
Dotaz v CQL: [tag="k7c3"][tag!="k1.*"]
Hledaný výraz je slovo definované značkou jako prepozice, dativ, následuje
slovo nedefinované značkou jako substantivum.
```

Fig. 2. Searching for POS.

2.3 Specifying number of tokens

It is possible to set the number of repeats for any token in the query, e. g. which parts of the query are required or optional. Number of tokens can be specified by quantifiers, i. e. *, ?, +, *{n}*, *{n,}*, *{n,m}* (see Table 2 for the list of quantifiers, and Figure 3 for an example).

```
Dotaz v CQL: [tag="k1.*"]* [word!="a"]{2,5} [lemma="konec"]+
Hledaný výraz je slovo definované značkou jako substantivum opakující se
libovolněkrát, následuje slovo jakékoliv kromě "a" opakující se 2-5krát,
následuje slovo, které má základní tvar "konec" vyskytující se minimálně
jednou.
```

Fig. 3. Searching number of tokens.

2.4 Searching in structures

If the corpus has sentence, paragraph or document markup, it is possible to take these into consideration when writing CQL queries. Operators *s*, *p* and *doc* are valid structures and users are able to match tokens in such structures. The first type of the structure is specified with operator *containing*, and matches all structures of your choice containing e. g. an adjective followed by noun. The other type is specified with operator *within*, and matches a number of tokens within a structure of your choice. See Figure 4 for examples.

[3] For a complete list of tags, see `http://nlp.fi.muni.cz/projekty/ajka/tags.pdf`

```
Dotaz v CQL: <s/> containing [tag="k2.*"] [tag="k1.*"]
Hledaný výraz je věta, v níž se nachází slovo definované značkou jako
adjektivum, následuje slovo definované značkou jako substantivum.
Dotaz v CQL: [word="pes"] [lemma="kousat"] within <p/>
Hledaný výraz je slovo "pes", následuje slovo, které má základní tvar
"kousat", to celé v rámci odstavce.
```

Fig. 4. Searching in structures.

2.5 Searching for complex patterns

If the value of a query includes any characters, other than letters or numbers, the query is evaluated as a regular expression. However, the current version of the application does not evaluate the value of a regular expression, because it is a complex tasks by itself. On the other hand, some frequent query types are detected. The application is able to detect the pattern .* in the CQL value, and from the position of this pattern in the query string decides whether the users are searching for words/lemmas starting with, ending with, or containing particular characters.

```
Dotaz v CQL: [lemma=".*at"][lemma!="bez"][word="pov.*nu.*"]
Hledaný výraz je slovo, které má základní tvar končící na "at", následuje slovo,
které má základní tvar jakýkoliv kromě "bez", následuje slovo začínající na "pov"
a obsahující "nu".
```

Fig. 5. Searching for complex patterns.

3 Conclusion and future work

The application is able to convert several frequent types of CQL queries to an explanation in Czech, even more advanced features like quantifiers or structure queries. Web application is available for users at `http://nlp.fi.muni.cz/projekty/cql2cz/`.

However, the application does not support all of the advanced CQL features, and current version has some issues with complex queries. We plan to address the issues and extend the CQL support, in future versions. In the next version, following areas will be covered.

3.1 The sequence of morphological categories

When users specify morphological tags with regular expressions, like *"k1.*c4"*, only the tags (*k1* and *c4*) are detected, and the rest of the query is stripped. However, the full regular expression may hold important information, thus need to be examined by the application more closely.

3.2 Regular expressions inside value

Currently, only the pattern .* is detected in the regular expressions. In future versions, the application will be able to detect other types of regular expressions correctly.

3.3 Complex patterns

The application should be able to detect and process more complex query structure, for example nested queries like `[word="haunt" and tag="k1.*"]`.

Acknowledgement This work has been partly supported by the Masaryk University within the project *Čeština v jednotě synchronie a diachronie – 2015* (MUNI/A/1165/2014) and by the Ministry of Education of CR within the LINDAT-Clarin project LM2010013.

References

1. Evert, S.: The CQP query language tutorial. (2005)
2. Sketch Engine, Corpus Querying: `https://www.sketchengine.co.uk/xdocumentation/wiki/SkE/CorpusQuerying`
3. Jakubíček, M., Kovář, V., and Šmerk, P.: Czech Morphological Tagset Revisited. In *Proceedings of Recent Advances in Slavonic Natural Language Processing* (2011) 29–42