IB047

Corpus Query Language

Pavel Rychlý

pary@fi.muni.cz

March 22, 2024

Corpus Query Language

Test it at `http://ske.fi.muni.cz/`
Access to more corpora at
`https://app.sketchengine.eu/`
Use: Concordance – Advanced – Query type=CQL
Corpus: BNC (tagged by CLAWS)

Corpus Query Language

Test it at `http://ske.fi.muni.cz/`
Access to more corpora at
`https://app.sketchengine.eu/`
Use: Concordance – Advanced – Query type=CQL
Corpus: BNC (tagged by CLAWS)

- Query – pattern matching a set of single tokens or token sequences

Corpus Query Language

Test it at `http://ske.fi.muni.cz/`
Access to more corpora at
`https://app.sketchengine.eu/`
Use: Concordance – Advanced – Query type=CQL
Corpus: BNC (tagged by CLAWS)

- Query – pattern matching a set of single tokens or token sequences
- Each token consists of attributes (depending on corpus configuration):
  word, lemma, tag, lempos, lc
- Use *[attribute=”value”]* for each token sub-pattern.

Very simple queries

```
[word="dream"]
[word="Dream"]
[lc="dream"]
[lemma="dream"]
[lempos="dream-n"]
[word="The"] [word="dream"]
[word="the"] [lemma="dream"]
[tag="AJ0"] [lempos="dream-n"]
```

Regular Expression in Attributes

Value is a regular expression in a *[attribute="value"]* expression.

```
[word="dream.*"]
[word="[dD]ream"]
[word="[0-9]*"] [lc="dreams"]
[tag="NN."] [lempos="dream-v"]
[word="[0-9]{5,}"] [word="\."]
[word="\("] [word="0[0-9]{3}"] [word="\)"]
[word==")"] [word=="."]
```

```
[word="[A-Z][0-9A-Z]{2,3}"] [word="[0-9][0-9A-Z]{2}"]
```

Regular Expressions

PCRE library used for evaluation of REs
Several useful special sequences

- `\d` – any decimal digit
- `\D` – any character that is not a decimal digit
- `\w` – any ”word” character
- `\W` – any ”non-word” character
- `(?i)` – ignore case

```
[word="\d\d\W"]
```

Logical combinations of attributes

Boolean combinations (*AND*, *OR* and *NOT*) of *[attribute="value"]* expressions.
Use: &, |, !=, ()

```
[word="dream" & tag="NN1"]
[lemma="dream" & tag="VV."]
[word="dream" | word="Dream"]
```

```
[word="the" | tag="DPS"][lempos="dream-n" & tag="NN2"]
[word="the" | (tag="DPS" & lemma!="my")][lemma="dream"]
```

Regular expressions of tokens

Regular expressions on token level:

?	optional token
*	any number of repetition
+	at least one
{N}	exact number of repetitions
{M,N}	from M to N repetitions
[ ]	any token

```
[tag="DPS"] [] [lemma="dream"]
[tag="DPS"] [tag="AJ0"]? [lemma="dream"]
[tag="AJ0"]{2} [lemma="dream"]
[word="the"] []{0,3} [lempos="dream-n"]
```

Within

within keyword at the end of a query

- `within <s/>` restricts result to one sentence
- `within <bncdoc id="A01"/>` restricts result to a subcorpus

```
[lemma="dream"] within <bncdoc id="A01"/>
[word="the"] []{3,5} [lemma="dream"]
[word="the"] []{3,5} [lemma="dream"] within <s/>
```

Numbers instead of values

- `within <bncdoc #1>` restricts result to second document
- can be used also for tokens

```
[lemma="dream"] within <bncdoc #1/>
[#400]
[#123-125]
```

Within

More *within* combinations: Boolean combinations of regular expressions

```
[lemma="dream"] within <bncdoc author=".*Smith.*"/>

[lemma="dream"] within <bncdoc wriaud="Teenager"
                                & wriase="Female"/>

[word="the"] []{3,5} [lemma="dream"]
                within <s/> within <bncdoc id="A0."/>

[word="the"] []{3,5} [lemma="dream"] within <phr/>
```

Within

within could be inverted

```
[word="THE"] within <head/>
```

```
[word="THE"] within !<head/>
```

Structure boundaries

Structure boundaries: start/end of a structure, whole structure

```
<s> [lemma="dream"]
```

```
[word=="?"] </bncdoc>
```

```
<head /> within <bncdoc alltyp="Written-to-be-spoken"/>
```

Global conditions

Global condition

- numeric labels of tokens
- testing agreement or disagreement of attribute values

```
[tag="NN."] [word="and"] [tag="NN."]
```

Global conditions

Global condition

- numeric labels of tokens
- testing agreement or disagreement of attribute values

```
[tag="NN."] [word="and"] [tag="NN."]


1:[tag!="NN."] [word="and"] 2:[tag!="NN."]
                        & 1.tag = 2.tag

1:[] [word="and"] 2:[] & 1.k=2.k & 1.c=2.c
```

Parallel corpora

Parallel corpora – separate corpus for each language, 1-to-1 alignment using `<align>` tag.
Query can limit the search to segments with aligned parts containing a subquery hits.

```
[lemma="hrad"] within kacen: [word="castle"]

[lemma="hrad"] within ! kacen: [word="castle"]
```

Meet/Union queries

- combining and nesting simple (one-token) queries
- not a sequence of tokens
- meet and union operators

Union

Union operator:

- union Q1 Q2

```
(union [word="dream"] [word="dreams"])
[word="dream" | word="dreams"]
```

Meet

Meet operator:

- meet Q1 Q2 W-BEG W-END
- find Q1 with Q2 in window from W-BEG to W-END
- W-BEG, W-END defaults to 1

```
(meet [word="my"] [word="dream"])
[word="my"] [word="dream"]
(meet [word="my"] [word="dream"] 1 3)
[word="my"] []{0,2} [word="dream"]
(meet [word="black"] [word="white"] -3 3)
```

Meet/union combination

use a meet/union operator in place of a simple query

```
(meet [word="and"] (meet [word="black"]
  [word="white"] -3 3) -2 2)
```

Within keyword

within works with any subquery not only a structure

```
[lemma="dream"] within ([word="my"] [lemma="dream"])

(meet [lemma="dream"] [word="my"] -1 -1)

[word="the"] [] {0,3} [lemma="dream"]
          within ([tag="AT."][tag="AJ."]{0,4}[tag="NN."])
```

containing keyword

containing keyword

- inverts within keyword
- matches results of the first subquery which contains matches of the second subquery

```
<phr/> containing [lemma="dream"]

(meet [lemma="dream"] [word="my"] -1 -1)

[word="the"] []{1,3} [lemma="dream"]
        containing [lemma="wild"]
```

Combinations of containing/within

Both keyword forms a query which can be used as subquery, they can be nested.

```
[lemma="break"] within (<s/> containing [lemma="rule"])

[lemma="student"] within
      (<s/> containing [lemma="break"]
             containing [lemma="rule"])

[lemma="break"] within ([]{5} containing [lemma="rule"])
```

Query Evaluation Concept

- Stream processing
- Streams of token numbers or ranges
- SQL: tables + relational algebra operators
- Unix text tools: text files + filters
- CQL: token streams + simple stream operators

Query Streams

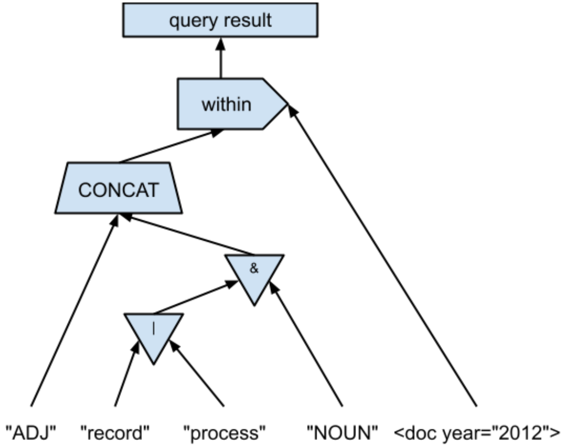
- only one item each time
- *sorted* token numbers
- FastStream
- RangeStream – sorted pairs

FastStream: C++ implementation

```
class FastStream {
public:
    virtual Position peek() = 0;
    virtual Position next() = 0;
    virtual Position find (Position pos) = 0;
    virtual Position final() = 0;
};
```

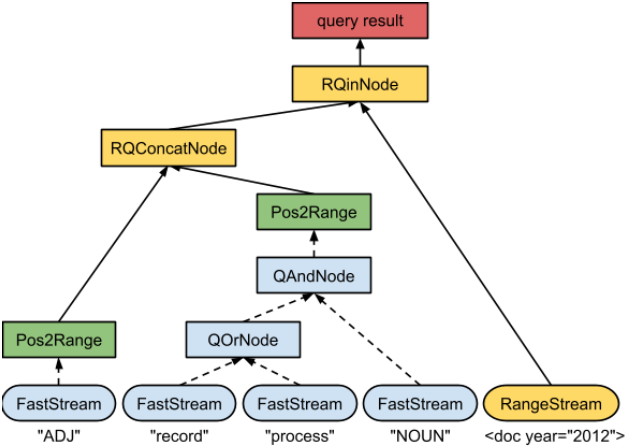
Query Evaluation

```
[tag="ADJ"]+ [(word="record" | word="process") & tag="NOUN"] within <doc year="2012"/>
```



Query Evaluation

```
[tag="ADJ"]+ [(word="record" | word="process") & tag="NOUN"] within <doc year="2012"/>
```



Sketch Engine API

- web API using same addresses as web interface
- result as json
- documentation at `https://www.sketchengine.eu/apidoc/`