

BEKK

# *SEARCHING NEO4J USING NATURAL LANGUAGE*

---

*Oslo Graph Database Meetup  
Ole-Martin Mørk  
27/05/14*



# What can Graph Search do?



Sushi restaurants in Oslo  
liked by my friends

Sushi ▾

 Open Now

1 Alex Sushi

Sushi

CLOSED



2 Nodee Asian Cooking

Sushi

OPEN



3 Rå Sushi Majorstuen

Sushi

OPEN



4 Jonoe

Japanese

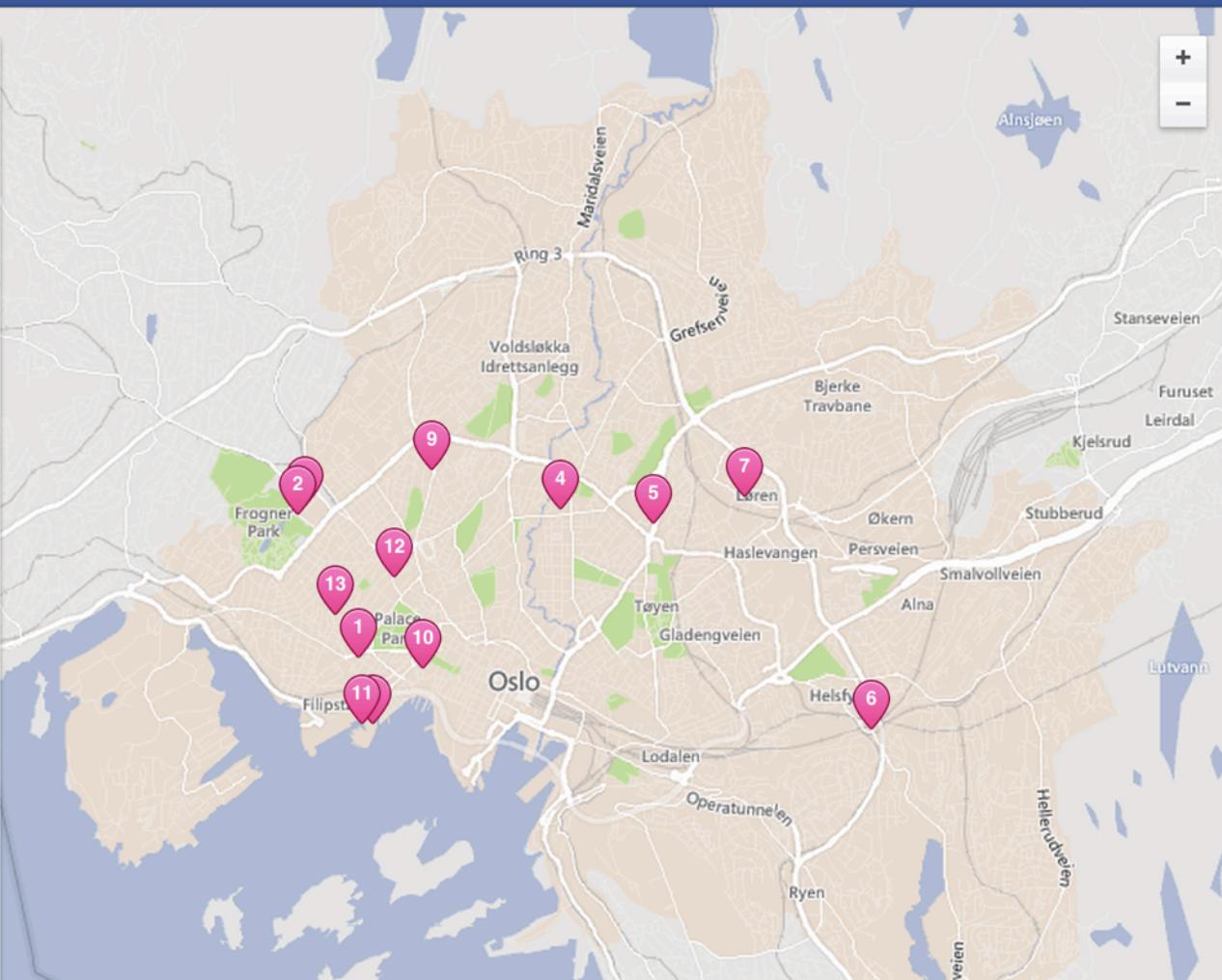
OPEN



5 Grand Sushi

Restaurant

OPEN



*CYPHER* —

---

**MATCH**

```
(me {name:“Ole-Martin”})-[:IS_FRIEND_OF]->(friend),  
(friend)-[:LIKES]->(restaurant),  
(restaurant)-[:LOCATED_IN]->(location {name:“Oslo”}),  
(restaurant)-[:SERVES]->(cuisine {name:“Sushi”})
```

**RETURN** restaurant

## WE NEED TO DEFINE A GRAMMAR

---

### Oppslagsord      Ordbokartikkell

neo-

**neo-** (fra gr av *neos* 'ny') ny-, i ord som  
*neofascisme, neoimpresjonisme, neorealisme*

neodym

**neodym** n1 (av *didym*) treverdig, gulaktig metallisk grunnstoff (kjem symbol *Nd*)

neoklassisisme

**neoklassisisme** m1 musikkretning på 1900-tallet som tar opp førromantiske stilidealer, jf \**nyklassisme*

neolittisk

**neolittisk** a2 (av gr *lithos* 'stein') i arkeologi: som hører til den yngre steinalder

neologi

**neologi** m1 nydannelse

## *GRAMMAR*

---

A “language” can be formally defined as “any system of formalized symbols, signs, etc. used for communication”

A “grammar” can be defined as a “the set of structural rules that governs sentences, words, etc. in a natural language”

CFG PEG

## *GRAMMAR*

---

Alfred, who loved fishing, bought fish at the store downtown

(Alfred, (who loved (fishing)), (bought  
fish (at (the store (downtown))))))

```
additionExp
  :  multiplyExp
    ( '+' multiplyExp
    | '-' multiplyExp
    )*
;
;
```

```
multiplyExp
  :  atomExp
    ( '*' atomExp
    | '/' atomExp
    )*
;
;
```

```
atomExp
  :  Number
  | '(' additionExp ')'
;
;
```

```
Number
  :  ('0'...'9')+
;
;
```

An additionExp is defined as a multiplyExp  
+ or - a multiplyExp

A multiplyExp is defined as an atomExp \*  
or / an atomExp

An atomExp is defined as a number or a  
parenthesized additionExp

Number is one or more character between  
0-9

```
Rule expression() {
    return sequence(
        term(),
        zeroOrMore(anyOf("+-"), term())
    );
}
```

```
Rule term() {
    return sequence(
        factor(),
        zeroOrMore(anyOf("*/"), factor())
    );
}
```

```
Rule factor() {
    return firstOf(
        number(),
        sequence('(', expression(), ')')
    );
}
```

```
Rule number() {
    return oneOrMore(charRange('0', '9'));
}
```

An expression is a sequence of Term followed by zero or more “+” or “-” followed by a Term

Term is a Factor followed by zero or more sequences of “\*” or “/” followed by a factor

Factor is a number or a parenthesized expression

Number is a one or more characters between 0-9

Name	CFG	<i>Parboiled</i>
Sequence	$a \ b$	Sequence (a, b)
Ordered Choice	$a / b$	FirstOf (a, b)
Zero-or-more	$a^*$	ZeroOrMore (a)
One-or-more	$a^+$	OneOrMore (a)
Optional	$a?$	Optional (a)
And-predicate	$\& \ a$	Test (a)
Not-predicate	$! \ a$	TestNot (a)

## *PEG VS CFG*

---

PEGs firstOf operator vs CFG's | operator

PEG does not have a separate tokenizing step

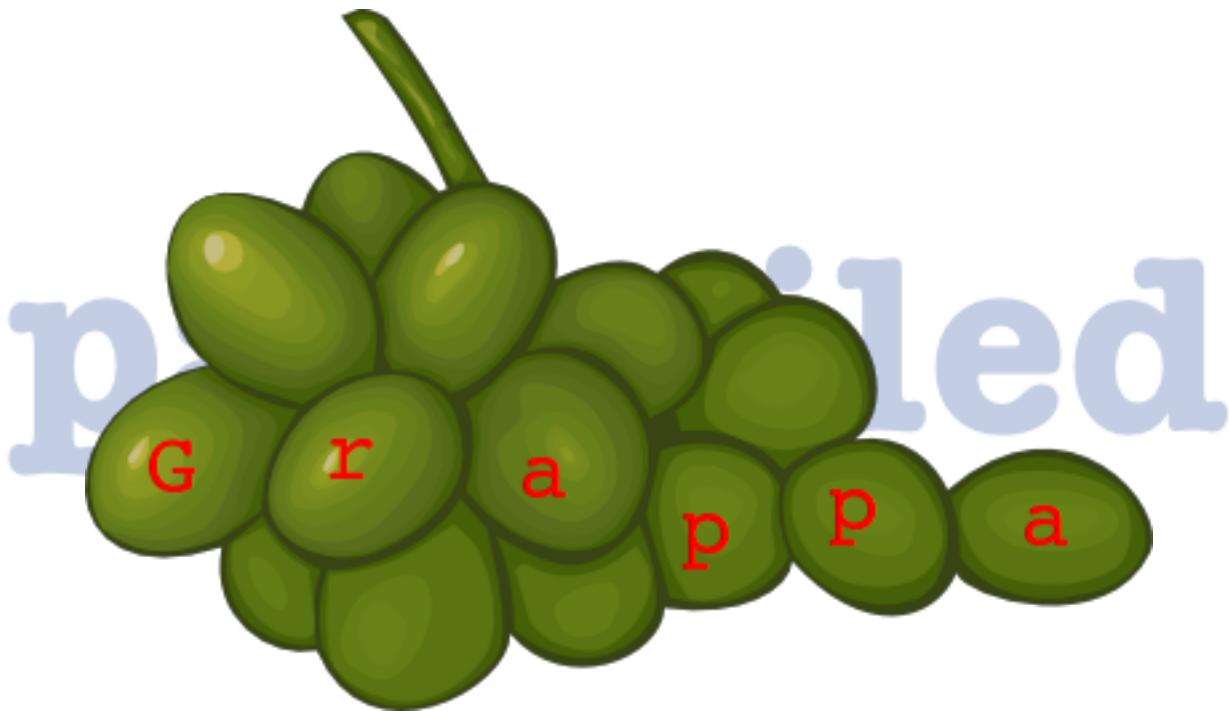
CFG might come across as more powerful, but also more difficult to master

PEG does not allow ambiguity in the grammar

PEG is easier to write, CFG is easier to analyze..?

*PARBOILED*

---



Parsing expression grammars parser

Fork of Parboiled

Lightweight

Easy to use

Implementation in Java

Rules are written in Java



# Examples

I went for a walk downtown

sequence( “I”, “went”, “for”, “a”, “walk”, “downtown”)

I      went     for a walk   downtown  
wend                                       to the city

```
sequence(  
  string("I"),  
  firstOf("went", "wend"),  
  sequence("for", "a", "walk"),  
  firstOf("downtown", "to the city"));
```

I    went    downtown  
walked    for a walk    *today*  
              to the city

sequence(

...,

optional(string("today")));

*today* I <sup>went</sup>  
walked for a walk <sup>downtown</sup>  
<sup>to the city</sup> *today*

sequence(

today(),

...,

today());

Rule today() { return optional(String("today")); }

```
public Rule anyOf(String characters)
public Rule noneOf(String characters)
public Rule string(String string)
public Rule ignoreCase(String string)
public Rule firstOf(Object... rules)
public Rule oneOrMore(Object... rules)
public Rule optional(Object rule)
public Rule sequence(Object... rules)
public Rule zeroOrMore(Object rule)
public Rule nTimes(int repetitions, Object rule)

public boolean push(V value)
public V pop()
```

*today* I **went** downtown *today*  
walked for a walk to the city

sequence(

...,

firstOf(

sequence(push("downtown"), "downtown",

sequence(push("city"), "to the city")));

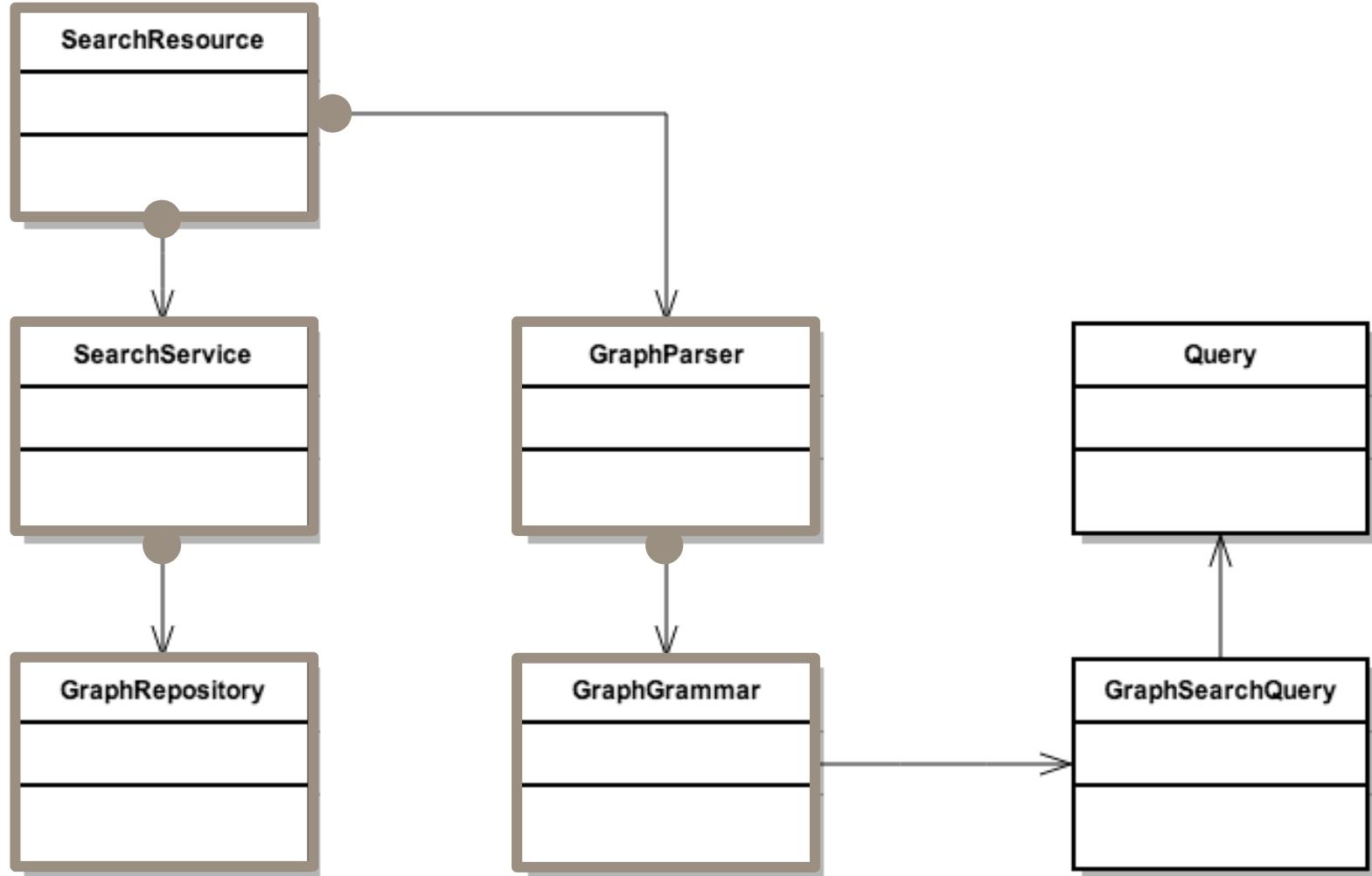
# Implementation

# BEKK CV-db



```
public Rule expression() {
    return sequence(
        start(),
        firstOf(
            people(), projects(), technologies(ROOT)
        ),
        oneOrMore(
            firstOf(
                and(),
                sequence(know(SEARCH), subjects()),
                sequence(workedAt(SEARCH), customers()),
                sequence(know(SEARCH), customers()),
                sequence(know(SEARCH), technologies(MIDDLE)),
                sequence(workedWith(), consultants()),
                know(PARAM),
                workedAt(PARAM) )) );
}
```

```
[expression] 'Finn prosjekter med Ole-Martin Mørk'
[firstOf] 'prosjekter '
[projects] 'prosjekter '
[projectSequence] 'prosjekter '
["prosjekter"] 'prosjekter'
[whiteSpace] ' '
[oneOrMore] 'med Ole-Martin Mørk'
[firstOf] 'med Ole-Martin Mørk'
[sequence] 'med Ole-Martin Mørk'
["med"] 'med'
[whiteSpace] ' '
[consultants] 'Ole-Martin Mørk'
["Ole-Martin Mørk"] 'Ole-Martin Mørk'
[whiteSpace]
```



# Demo

BΕKK

*THANK YOU!*

---

---

*github.com/olemartin*

---

*twitter.com/olemartin*

---

*Ole-Martin Mørk*

---