Query generation from natural languages

Martin Agfjord

Department of Computer Science and Engineering
University of Gothenburg
Advisors: Per Fredelius (Findwise) & Krasimir Angelov (GU)

Half-time presentation

Expectations

- This project is not about...
  - Building a *strong AI* (human like) application
- Its about
  - Building an application for understanding a subset of questions
  - Retrieving better search results
- Work in progress

Expectations

- This project is not about...
  - Building a *strong AI* (human like) application
- Its about
  - Building an application for understanding a subset of questions
  - Retrieving better search results
- Work in progress

Expectations

- This project is not about...
  - Building a *strong AI* (human like) application
- Its about
  - Building an application for understanding a subset of questions
  - Retrieving better search results
- Work in progress

Expectations

- This project is not about...
  - Building a *strong AI* (human like) application
- Its about
  - Building an application for understanding a subset of questions
  - Retrieving better search results
- Work in progress

Outline

Outline

Outline

Suggestions algorithm

```
suggestions(question) { // natural language question
    // question = “persons who knew java”
    names[] = extractNames(question);
    // “persons who knew java” ===>
    // “persons who knew Skill0”
    question = replaceNamesWithTypes(question, names);

    // suggestions = { ”people who know Skill0”, ... }
    suggestions[] = findQuestions(question);
    for each suggestion in suggestions {
        // “people who know Skill0” ===>
        // “people who know Java”
        suggestion = restoreNames(names, suggestion);
    } return suggestions; }
```

Suggestions algorithm

```
suggestions(question) { // natural language question
    // question = “persons who knew java”
    names[] = extractNames(question);
    // “persons who knew java” ===>
    // “persons who knew Skill0”
    question = replaceNamesWithTypes(question, names);

    // suggestions = { ”people who know Skill0”, ... }
    suggestions[] = findQuestions(question);
    for each suggestion in suggestions {
        // “people who know Skill0” ===>
        // “people who know Java”
        suggestion = restoreNames(names, suggestion);
    } return suggestions; }
```

Suggestions algorithm

```
suggestions(question) { // natural language question
    // question = “persons who knew java”
    names[] = extractNames(question);
    // “persons who knew java” ===>
    // “persons who knew Skill0”
    question = replaceNamesWithTypes(question, names);

    // suggestions = { ”people who know Skill0”, ... }
    suggestions[] = findQuestions(question);
    for each suggestion in suggestions {
        // “people who know Skill0” ===>
        // “people who know Java”
        suggestion = restoreNames(names, suggestion);
    } return suggestions; }
```

Suggestions algorithm

```
suggestions(question) { // natural language question
    // question = “persons who knew java”
    names[] = extractNames(question);
    // “persons who knew java” ===>
    // “persons who knew Skill0”
    question = replaceNamesWithTypes(question, names);

    // suggestions = { ”people who know Skill0”, ... }
    suggestions[] = findQuestions(question);
   for each suggestion in suggestions {
        // “people who know Skill0” ===>
        // “people who know Java”
        suggestion = restoreNames(names, suggestion);
    } return suggestions; }
```

Live example of suggestions

Outline

Understanding Natural Languages

- I use grammars to translate languages
- A grammar is a set of structured rules for strings
- Grammars have been used by compilers for a long time

So how can we build a grammar to translate languages?

We will use Grammatical Framework (GF)

Understanding Natural Languages

- I use grammars to translate languages
- A grammar is a set of structured rules for strings
- Grammars have been used by compilers for a long time

So how can we build a grammar to translate languages?

We will use Grammatical Framework (GF)

Understanding Natural Languages

- I use grammars to translate languages
- A grammar is a set of structured rules for strings
- Grammars have been used by compilers for a long time

So how can we build a grammar to translate languages?

We will use Grammatical Framework (GF)

Understanding Natural Languages

- I use grammars to translate languages
- A grammar is a set of structured rules for strings
- Grammars have been used by compilers for a long time

So how can we build a grammar to translate languages?

We will use Grammatical Framework (GF)

Understanding Natural Languages

- I use grammars to translate languages
- A grammar is a set of structured rules for strings
- Grammars have been used by compilers for a long time

So how can we build a grammar to translate languages?

We will use Grammatical Framework (GF)

Introducing Grammatical Framework (GF)

- GF is a development platform for natural language grammars
- GF separates the grammar between *abstract* and *concrete* syntax
- An abstract syntax is a tree which captures the meaning of a sentence
- A concrete syntax represent how an abstract syntax looks like a string

The same technique is used by compilers

- Programmer writes source code in concrete syntax
- Compiler translates concrete syntax to abstract syntax
- The rest of the compiler manipulates the trees

Introducing Grammatical Framework (GF)

- GF is a development platform for natural language grammars
- GF separates the grammar between *abstract* and *concrete* syntax
- An abstract syntax is a tree which captures the meaning of a sentence
- A concrete syntax represent how an abstract syntax looks like a string

The same technique is used by compilers

- Programmer writes source code in concrete syntax
- Compiler translates concrete syntax to abstract syntax
- The rest of the compiler manipulates the trees

Introducing Grammatical Framework (GF)

- GF is a development platform for natural language grammars
- GF separates the grammar between *abstract* and *concrete* syntax
- An abstract syntax is a tree which captures the meaning of a sentence
- A concrete syntax represent how an abstract syntax looks like a string

The same technique is used by compilers

- Programmer writes source code in concrete syntax
- Compiler translates concrete syntax to abstract syntax
- The rest of the compiler manipulates the trees

Introducing Grammatical Framework (GF)

- GF is a development platform for natural language grammars
- GF separates the grammar between *abstract* and *concrete* syntax
- An abstract syntax is a tree which captures the meaning of a sentence
- A concrete syntax represent how an abstract syntax looks like a string

The same technique is used by compilers

- Programmer writes source code in concrete syntax
- Compiler translates concrete syntax to abstract syntax
- The rest of the compiler manipulates the trees

Introducing Grammatical Framework (GF)

- GF is a development platform for natural language grammars
- GF separates the grammar between *abstract* and *concrete* syntax
- An abstract syntax is a tree which captures the meaning of a sentence
- A concrete syntax represent how an abstract syntax looks like a string

The same technique is used by compilers

- Programmer writes source code in concrete syntax
- Compiler translates concrete syntax to abstract syntax
- The rest of the compiler manipulates the trees

Introducing Grammatical Framework (GF)

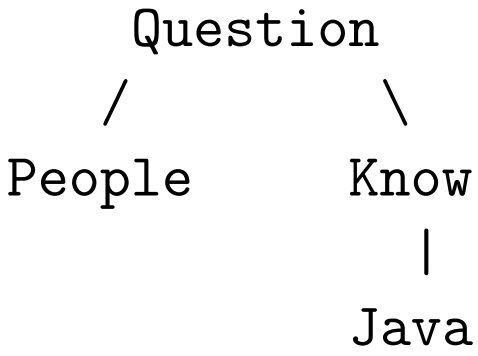
- GF is a development platform for natural language grammars
- GF separates the grammar between *abstract* and *concrete* syntax
- An abstract syntax is a tree which captures the meaning of a sentence
- A concrete syntax represent how an abstract syntax looks like a string

The same technique is used by compilers

- Programmer writes source code in concrete syntax
- Compiler translates concrete syntax to abstract syntax
- The rest of the compiler manipulates the trees

Example of a grammar

Abstract syntax



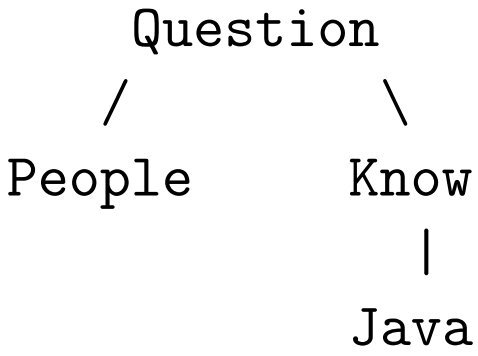
Concrete syntaxes

```
people who know Java                                  -- English
personer som kan Java                                 -- Swedish
q=object_type : Person AND expertise : Java -- Solr
MATCH (n:Person) - [KNOWS] ->                         -- Cypher
    (:Expertise { name : 'Java' }) RETURN n;
```

- They are all string representations of the same abstract syntax

Example of a grammar

Abstract syntax



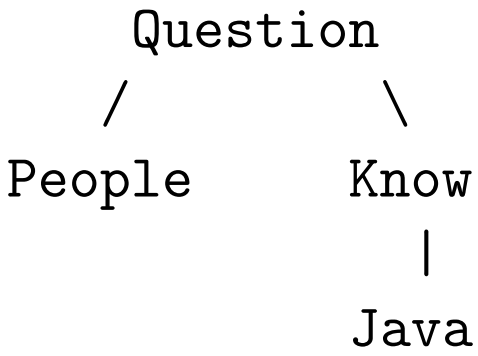
Concrete syntaxes

```
people who know Java                                  -- English
personer som kan Java                                 -- Swedish
q=object_type : Person AND expertise : Java           -- Solr
MATCH (n:Person) - [KNOWS] ->                         -- Cypher
    (:Expertise { name : 'Java' }) RETURN n;
```

- They are all string representations of the same abstract syntax

Example of a grammar

Abstract syntax



Concrete syntaxes

```
people who know Java                                  -- English
personer som kan Java                                 -- Swedish
q=object_type : Person AND expertise : Java -- Solr
MATCH (n:Person) - [KNOWS] ->                         -- Cypher
    (:Expertise { name : 'Java' }) RETURN n;
```

- They are all string representations of the same abstract syntax

Live demo of a grammar

Grammatical framework (GF)

- Given an abstract syntax and concrete syntaxes
  - GF can derive a *generator*, a *parser* and *linearizers*
- GF has a resource library which supports 36 natural languages
  - You leave it to the linguistic to design the language
  - You design your domain

Grammatical framework (GF)

- Given an abstract syntax and concrete syntaxes
  - GF can derive a *generator*, a *parser* and *linearizers*
- GF has a resource library which supports 36 natural languages
  - You leave it to the linguistic to design the language
  - You design your domain

Outline

Neo4j integration

- Create a concrete language for Cypher, Neo4j's query language

Example: 'people who know Java and Python'

```
MATCH (n:Person) - [:KNOWS] ->
          (:Expertise { name : 'Java' }),
        (n) - [:KNOWS] ->
          (:Expertise { name : 'Python' })
RETURN n;
```

Neo4j integration

- Create a concrete language for Cypher, Neo4j's query language

Example: 'people who know Java and Python'

```
MATCH (n:Person) - [:KNOWS] ->
          (:Expertise { name : 'Java' }),
      (n) - [:KNOWS] ->
          (:Expertise { name : 'Python' })
RETURN n;
```

Neo4j integration cont'd

How to express: 'people who know (Java and Python) or Haskell' ?

```
MATCH (n:Person) - [:KNOWS] ->
        (:Expertise { name : 'Java' }),
      (n) - [:KNOWS] ->
        (:Expertise { name : 'Python' })
RETURN n
UNION
MATCH (n:Person) - [:KNOWS] ->
        (:Expertise { name : 'Haskell' })
RETURN n;
```

Neo4j integration cont'd

Express: 'people who know C and ((Java and Python) or Haskell)'?
One possible solution:

```
MATCH (n:Person) - [:KNOWS] ->
        (:Expertise { name : 'Java' }),
      (n) - [:KNOWS] ->
        (:Expertise { name : 'Python'}),
      (n) - [:KNOWS] - >
        (:Expertise { name : 'C' })
RETURN n
UNION
MATCH (n:Person) - [:KNOWS] ->
        (:Expertise { name : 'Haskell' }),
      (n) - [:KNOWS] ->
        (:Expertise { name : 'C' })
RETURN n;
```

Outline

Additional features

- More questions
  - “people”, “customers”
  - “people who know java, sorted by name”
- Filtering on name suggestion
  - Only suggest names that matches existing documents
- Speech to text