Grammar-based suggestion engine with keyword search

Martin Agfjord, Krasimir Angelov, Per Fredelius, Svetoslav Marinov

University of Gothenburg, Findwise AB
Gothenburg
martin.agfjord@gmail.com, krasimir@chalmers.se,
{per.fredelius, svetoslav.marinov}@findwise.com

1. Introduction

An alternative approach to building a user friendly interface for a database is natural language translation. This method has been used, for instance, by Facebook (Li, 2013) in their graph search application, which adopts a grammar to translate structured sentences into machine readable syntax. This paper describes how a similar application can be developed by creating a grammar with *Grammatical Framework (GF)* (Ranta, 2011) and by using *Apache Solr* (Kuć, 2011) to give relevant suggestions of valid sentences based on keywords or a partial sentence.

We assume this application will be used by a software development company which has a Solr index with the entities *Customers, People* and *Projects*. An example of a valid sentence in this environment is `people who know Java`, which when executed will instruct the Solr index to retrieve all documents where the field `object_type` has the value `person` and the `KNOWS`-field contains the value `Java`.

While translations like these allow the user to express queries in a natural language, it is not very likely that the user will be able to use the system without any knowledge of the queries that it supports. We will therefore also develop a suggestion engine whose purpose is to help the user to find valid sentences based on partial input.

2. Grammatical Framework

GF is an open source development platform which is specifically designed for creating natural language grammars. GF adopts the use of abstract and concrete syntax much like compilers do. The abstract syntax represents the *semantics* of a sentence and the concrete syntax represents how the semantics is rendered as a string in a language.

An abstract syntax consists of a set of function declarations which can be combined to form an abstract syntax tree (AST). The concrete syntax implements the *linearization* of each function to represent the semantics as a string. An AST can be linearized by executing the functions in the concrete syntax. Since the linearization is reversible, a string can be parsed back into an AST.

3. Defining the grammar in GF

The abstract syntax of the grammar defines functions representing *subjects, predicates* and *objects*. The concrete syntax defines rules which combines those into sentences. For example, the sentence `customers who use Solr` is represented with the tree:

```
MkInstruction Customer (Use (MkObject "Solr"))
```

Here `"Solr"` can be an arbitrary name. This lets the sentence refer to any database object, and the objects do not have to be defined by the grammar. Since it is possible for multiple concrete syntaxes to share the same abstract syntax, the above AST can also be linearized as a Solr query:

```
select?q=object_type : Organization
         AND USES : ( Solr ).
```

The grammar supports a number of different instructions relating customers, people, projects, organizations, locations, program modules and skills.

There are two cases where we also allow boolean operators. The first is in between two arbitrary names, e.g. `Java` **`or`** `Haskell`. The second is in between two combinations of a predicate and an object, e.g. `know Java` **`and`** `work in London`. Boolean operators are easily implementable in Solr syntax since it is natural to use them in a query language. However, they cause ambiguities in the natural languages. In case of ambiguous question, we generate and execute both queries.

4. Suggestion Engine

The grammar translates sentences from English and Swedish into Solr queries. Given a grammar, the GF runtime also offers word-by-word suggestions by using incremental parsing (Angelov, 2009). However, suggestion of words can only be used for the next accepted word of a defined partial sentence and can therefore not be used for suggestions on an invalid partial sentence or on keywords.

Apache Solr offers functions which approximate how similar a string is to strings stored in a Solr index (Kuć, 2011). It makes therefore sense to use Solr for fuzzy matching of a partial sentence or set of keywords into natural language sentences defined by the grammar.

4.1 Generation of sentences

GF offers a *generator* which can do exhaustive generation of all AST's for a given syntactic category. We generate all trees up to a certain depth. The limitation is needed because we have infinitely many trees due to the recursive boolean functions. Since the trees also contain arbitrary names, GF will use `"Foo"` in an AST when it expects a name. We will therefore generate AST's like `MkInstruction People (WorkIn (MkObject "Foo"))` which linearizes into `people who work in Foo`. This linearization will not be very helpful as a suggestion to a user since `Foo` is not a location.

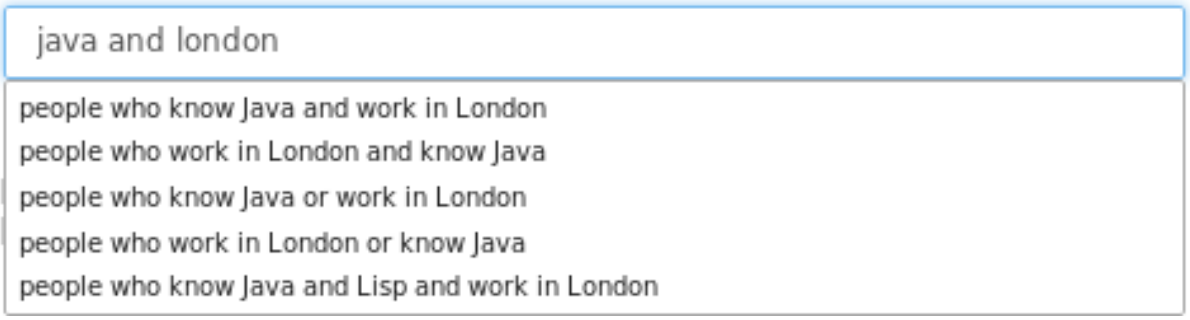

Figure 1: Web interface with suggestions based on the keywords `java and london`

A solution is to replace `Foo` with a relevant name before presenting the suggestion to a user (we assume all names exists in a separate Solr index). However, as we treat all names as objects, it would be possible to suggest many strange sentences to the user, such as `people who work in Haskell`.

In order to make distinctions between different names we introduce *name types*. A `Skill` is a programming language, an `Organization` is an organization, a `Location` is a physical location and a `Module` is an application module. The previous AST now looks like this `MkInstruction People (WorkIn (MkLocation "Foo"))`. The function `MkLocation` reveals that the name in this function is of the type `Location`.

The English linearization of the previous AST will however still not contain anything about the type, since the linearization of the AST `MkSkill "Foo"` is `Foo`. To preserve the type of each name, we post-process each AST and replace `"Foo"` with the actual type and an index, in this example `"Location0"`.

4.2 Suggestion retrieval

We generate sentences as described in the previous section and store them in a Solr index. This section describes how we take an input from the user and present relevant suggestions based on the input. While the method is rather naïve and can definitely be improved, it proves very efficient.

We split the input into words and check if each word is a name by asking the Solr index which contains all names (and their types). If a word is a name, we replace it in the input with its type. For example, the sentence `people java` will be changed into `people Skill0`. We fetch the five most similar sentences from the Solr index by using Solr's query interface. In this example, the first result is the sentence `people who know Skill0`. Finally, we change back Skill0 into `Java` in each sentence.

5. Conclusions

The results from (Agfjord, 2014) show that this method for building a natural language interface makes it possible to translate from natural language into a query language. We do not see any limitation with respect to the latter. Thus any machine readable language could be used instead of the Solr query syntax.

We found out it was easy to develop a translation service by using GF. It is also easy to extend the grammar with more sentences. In addition, by mapping *ungrammatical* sentences into grammatically correct ones we made the suggestion engine more powerful.

The suggestion engine is fast and scalable, as the Solr index only needs to store one sentence per AST for each natural language. This is possible since we only use name types in the sentences and not actual names. A user can find relevant sentences by using only keywords. If a keyword is a name, then the application will recognize the type of the name and it will correctly use it with appropriate sentences.

Further improvements can be done to the suggestion engine, such as heuristics for which names are relevant if no names exists in the input. Another optimization is to remove sentences with the same meaning as another sentence in the index. An example can be seen in Figure 1 where suggestions 1 and 2 have the same meaning but do not share the same AST.

The suggestion engine can easily be used together with speech recognition software. Speech recognition is error-prone when converting spoken input to text string. However, if we use Solr's built-in string approximation functions we can map the input to the most relevant sentence.

By transforming speech to machine readable syntax we open up doors to many new applications such as interfaces for cars and other devices which have limited keyboard interaction.

References

X. Li. 2013. *Under the Hood: The natural language interface of Graph Search*. URL: https://www.facebook.com/notes/facebook-engineering/under-the-hood-the-natural-language-interface-of-graph-search/10151432733048920 [Online; accessed 23-July-2014].

A. Ranta. 2011. *Grammatical Framework: Programming with Multilingual Grammars*. CSLI Publications, Stanford. ISBN-10: 1-57586-626-9 (Paper), 1-57586-627-7 (Cloth).

R. Kuć. 2011. *Apache Solr 3.1 cookbook*. Packt Publishing. ISBN-13: 978-1849512183 (Paper).

K. Angelov. 2009. Incremental parsing with parallel multiple context- free grammars. *European Chapter of the Association for Computational Linguistics*.

M. Agfjord. 2014. *Grammar-based suggestion engine with keyword search*. URL: http://thesis.agfjord.se/thesis.pdf [Online; accessed 28-August-2014].