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`

Abstract

An article may include an abstract of 100 to 150 words. If you use an abstract, its heading should be centered using font Times, 10 bold. If not, remove the abstract section.

1. Introduction

An alternative approach on building a user friendly interface for a database is natural language translation. This method has been used before 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, 2013) and using *Apache Solr* (Ku, 2011) to give relevant suggestions of valid sentences based on keywords or a partial sentence.

In order to articulate any sentences 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 this allows 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 translations it support. We will therefore use a suggestion engine which purpose is to help the user to find valid sentences based on partial input.

2. Grammatical Framework

GF is an open source development platform for natural languages 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 looks like a string in a language.

An abstract syntax usually consists of unimplemented functions which can be combined to form an abstract syntax tree (AST). A concrete syntax implements the unimplemented functions to represent the semantics. An AST can be *linearized* into a string by executing the functions in a concrete syntax. Conversely, a string can be parsed into an AST by dividing the string into functions.

3. Defining the grammar in GF

We define functions to represent *subjects, predicates* and *objects*. In addition, we also define rules which combines these values into sentences. For example, the sentence `customers who use Solr` is splitted into the AST `MkInstruction Customer (Use (MkModule "Solr"))` by the grammar. This AST will be linearized into `select?q=object_type : Organization AND USES : ( Solr )` by using the concrete syntax for Solr. We make use of arbitrary names in order to let the objects of each sentence take any string.

In addition, we also have functions for boolean operators in the grammar. There are two cases where we allow boolean operators. The first is in between two arbitrary names, e.g. `Java` **`or`** `Haskell`. The second case is in between two combinations of a predicate and an object, e.g. `know Java` **`and`** `work in London`. Boolean operators are easily implemented in Solr syntax since it is natural to use them in a query language.

4. Suggestion Engine

The grammar can translate sentences from English into Solr query language and offers suggestions of words by using incremental parsing (**?**). However, suggestion of words can only be used for the next accepted word of a sentence and can therefore not be used for suggestions on an invalid partial sentence or keywords.

Apache Solr offers functions which approximate how similar a string is to strings stored in the Solr cores. It makes therefore sense to use Solr to match a partial sentence or only keywords into natural language sentences defined by our grammar. We generate all defined sentences and store them in a Solr core.

4.1 Generation of sentences

GF offers a *generator* which can generate all AST's. We generate all AST's up to a certain depth (number of edges from root to leaf) as we have infinitely many AST's because of the recursive boolean functions. As we use arbitrary strings in our grammar, GF will put *Foo* in an AST when it expects a name. We will therefore generate sentences like MkInstruction People (Know (MkSkill "Foo" which linearizes into `people who know Foo`. This linearization would not be very helpful as a suggestion to a user since `Foo` is not a programming language.

5. Apache Solr as storage and suggestion engine

6. Conclusions

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. 2013. Grammatical Framework: Programming with Multilingual Grammars. CSLI Publications, Stanford, 2011. ISBN-10: 1-57586-626-9 (Paper), 1-57586-627-7 (Cloth).

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

B. Megyesi, B. Dahlqvist, E. Pettersson, S. Gustafson-Capková, and J. Nivre. 2008. Supporting research environment for less explored languages: A case study of Swedish and Turkish. In J. Nivre, M. Dahllöf, and B. Megyesi, editors, *Resourceful Language Technology: Festschrift in Honor of Anna Sågvall Hein*, Studia Linguistica Upsaliensia 7, pages 96–110. Acta Universitatis Upsaliensis.

S. Stymne. 2008. German compounds in factored statistical machine translation. In A. Ranta and B. Nordström, editors, *Proceedings of GoTAL, the 6th Interonational Conference on Natural Language Processing*, pages 464–475, Gothenburg, Sweden. Springer.

A. Wierzbicka. 1987. *English Speech Act Verbs: A semantic dictionary*. Academic Press, Sidney.