NLP-Reduce: A "naïve" but Domain-independent Natural Language Interface for Querying Ontologies

Esther Kaufmann, Abraham Bernstein, and Lorenz Fischer
University of Zurich
Department of Informatics
Binzmuehlestrasse 14, CH-8050 Zurich, Switzerland
{kaufmann, bernstein}@ifi.unizh.ch

Abstract

Casual users are typically overwhelmed by the formal logic of the Semantic Web. The question is how to help casual users to query a web based on logic that they do not seem to understand. An often proposed solution is the use of natural language interfaces. Such tools, however, suffer from the problem that entries have to be grammatical. Furthermore, the systems are hardly adaptable to new domains. We address these issues by presenting *NLP-Reduce*, a "naïve," domain-independent natural language interface for the Semantic Web. The simple approach deliberately avoids any complex linguistic and semantic technology while still achieving good retrieval performance as shown by the preliminary evaluation.

1 Introduction

The logic-based underpinnings of the Semantic Web provide a stable scaffolding for machine-based processing. Casual or occasional users, however, are typically overwhelmed with formal logic. The result is a gap between the Semantic Web and the average user who is mostly unable to command formal logic. The gap manifests itself in a disconnection between the user's information needs and the query language with which the user tries to find the required information in ontology-based data [Chakrabarti, 2004; Spink *et al.*, 2001; Spoerri, 1993]. Nevertheless, querying is a major interaction mode with the Semantic Web; bridging it is, therefore, central for the success of the Semantic Web for end users.

This paper proposes to address the gap by presenting *NLP-Reduce*, a "naïve" but domain-independent natural language interface for querying Semantic Web knowledge bases (KBs). The approach is simple and does not use any complex natural language processing (NLP) technologies. Compared to a full NLP engine it employs only a reduced set of NLP operators, such as synonym expansion and stemming (hence its name *NLP-Reduce*), which makes the interface robust to deficient input and completely portable. NLP-Reduce does not claim to be "intelligent" by interpreting and understanding the input queries; it "only" tries to link the words of a query and their synonyms to the expressions used in a KB.

2 NLP-Reduce

The system consists of five main parts: a user interface, a lexicon, an input query processor, a SPARQL query generator, and an ontology access layer.

The *user interface* allows the user to enter full natural language queries, sentence fragments, or just keywords. After executing a query, it displays the generated SPARQL query, the results, and some execution statistics to the user.

When a KB is loaded into NLP-Reduce, the *lexicon* is automatically built by extracting all explicit and inferred subject-property-object triples that exist in the KB. For each triple the synonyms of the labels are obtained from WordNet providing a larger vocabulary that can be deployed when querying. To improve recall each word in the lexicon is additionally stemmed using the Porter Stemmer [Porter, 1980].

The *input query processor* first reduces a query by removing stop words and punctuation marks. It then passes the stemmed words to the query generator.

The *query generator* is the core component of NLP-Reduce. It basically tries to match the query words to the synonym-enhanced triples stored in the lexicon and generates SPARQL queries for the matches. Consider the example query "Where is a restaurant in San Francisco that serves good French food?", for which the following steps are performed by the query generator:

1. The query generator first searches for triples in the lexicon in which at least one of the query words occurs within the label of an object property. For the example query the triples containing the properties <`isIn`>, <`isInCity`>, <`isInCounty`>, and <`isInRegion`> are returned, as they all contain the words "is" and "in." The query generator ranks the found triples according to a rating system that favors triples that cover more words as well as triples whose word stems show a better agreement with the query words over others. For example, the query words "is" and "in" get a higher score with <`isIn`> than with <`isInCity`>.
2. The generator then searches for properties in the lexicon that can be joined with the triples found in step 1 by the remaining query words. In our example it searches for the remaining query words "where," "restaurant," "San Francisco," "serve," "good," "French," and "food" in the lexicon's triples to combine them with the triple set

identified by step 1 taking domain and range information into consideration. Since "where" can be related to "location" through their synonyms, the triples containing the property <location> are found. The words "serve" and "food" both retrieve triples with the property <foodType>. If a query word produces triples based on several different properties, the ones with the highest rating score are favored. The triple sets from step 2 are then combined with the result of step 1, where the joining element is the class "restaurant."

3. The generator now searches for all datatype property values that match the remaining words of the query. For the last query words "San Francisco," "good," and "French," the triples [<Restaurant> <isIn> 'sanFrancisco'], [<Restaurant> <rating> 'good'], and [<Restaurant> <foodType> 'french'], are retrieved. The rating system again ranks the matches. The triples have to be combined with the beforehand identified and joined triples conforming to domain and range information. The class "City" with the label 'sanFrancisco' can be combined with the property <isIn>, which has the range "City".

4. As there are no more query words left, the query generator now generates the corresponding SPARQL query for the join of the retrieved triples that achieved the highest scores in steps 1 to 3. Additionally, it removes semantically equivalent duplicates and passes the SPARQL query to the ontology access layer.

The resulting SPARQL query that the query generator assembled for the example query is (note that we simplified the query by removing the URIs):

```
SELECT distinct * WHERE {
?Restaurant <#location> ?Location .
?Restaurant <#rating> 'good' .
?Restaurant <#foodType> 'french' .
?Restaurant <#isIn> ?City .
?City <#label> 'sanFrancisco' .
?Restaurant <#type> <#Restaurant> .
?City <#type> <#City> . }
```

To execute the generated SPARQL query, NLP-Reduce uses Jena as *ontology access layer* and the Pellet reasoner.

3 Preliminary Evaluation

To evaluate the performance of NLP-Reduce, we implemented a protoype in Java. We translated two Mooney natural language KBs [Tang and Mooney, 2001] into OWL and ran the provided 251 queries on the restaurant KB and the 879 queries on the geography KB. As NLP-Reduce abandons any complex linguistic analysis, it obviously cannot answer queries which require a dependency structure of the sentence elements (e.g., "Which restaurants are closer to...?"). Therefore, some queries provided by the Mooney data sets could not be answered.

NLP-Reduce successfully answered 94.6% (192 queries) of the restaurant queries, thereby achieving 69.6% average recall and 67.7% average precision. The system could also provide an answer for 66.0% (308 queries) of the geography queries with an average recall of 76.4% and an average precision of 70.7%. Note that we calculated recall and precision in a very strict manner, i.e., we assigned 0% recall as well as 0% precision to queries such as "How many Chinese restaurants are there in the Bay Area?" if NLP-Reduce found 1016 restaurants instead of the correct number 1047 to the query.

Though the results are not striking, we believe that the approach is promising. NLP-Reduce processes queries as bag of words not exploiting sophisticated linguistic or semantic techniques (except the use of WordNet and the Porter stemmer) as typical NLP systems do. Dependencies between words or phrases in the queries are only identified by the relationships that exist between the elements in the queried KB. The approach, therefore, highly depends on the quality and choice of vocabulary of the KBs. *This weakness is also its major strength, as it does not need any adaptation for new KBs, i.e., it is completely portable.* Under this perspective, the retrieval performance is surprisingly good.

4 Conclusions

In order to be usable by casual users, the logic-based scaffolding of the Semantic Web needs to be made accessible for querying. We think that natural language interfaces show a potential for end-user access to the Semantic Web but suffer from their inapplicability to new domains and their dependency on correct user input. To that end, we introduced NLP-Reduce, which is completely portable and robust to ungrammatical input. Preliminary evaluation results have shown that our simple, domain-independent approach provides a chance to offer the Semantic Web's capabilities to the general public.

References

[Chakrabarti, 2004] Soumen Chakrabarti. Breaking through the syntax barrier: Searching with entities and relations. In *15th European Conference on Machine Learning (ECML 2004)*, pages 9–16, Pisa, Italy, September 2004.

[Porter, 1980] Martin F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[Spink *et al.*, 2001] Amanda Spink, Wolfram Dietmar, Jansen Major B.J., and Saracevic Tefko. Searching the web: The public and their queries. *Journal of the American Society for Information Science and Technology*, 52(3):226–134, 2001.

[Spoerri, 1993] Anselm Spoerri. InfoCrystal: A visual tool for information retrieval management. In *Second Intl. Conference on Information and Knowledge Management*, pages 11–20, Washington, D.C., 1993. ACM Press.

[Tang and Mooney, 2001] Lappoon R. Tang and Raymond J. Mooney. Using multiple clause constructors in inductive logic programming for semantic parsing. In *12th European Conference on Machine Learning (ECML-2001)*, pages 466–477, Freiburg, Germany, 2001.