A Semantic Question Answering Framework

Project Supervisor - Prof. U. S. Tiwary

Group Members -

Harsh Shah	IIT2014071
Mohneesh Khaneja	IIT2014041
Yash Jain	IIT2014043
Anupam Jaiswal	IIT2014038
Anujraaj Goyal	IIM2014002

Introduction

- Fetching information specific to some subject of user’s interest, given the vast available data is essential

- This problem holds a significant contribution to the motivation behind the development of Search Engines (for web page content searches) and Question Answering (QA) systems (for document based searches)

- While search engines work in a keyword-based mechanism, QA systems allow user to specify questions in natural language

- Some critical problems that needs to be addressed are internal unambiguous knowledge representation, capturing semantic structure of questions and semantic constraints between the keywords

Motivation

- Despite the Information Retrieval based approaches to QA, there is no easy way to perform a federated search over both structured databases and unstructured text documents, including articles, manuals, reports, emails, blogs, and others.

- With the recent emergence of commercial grade Resource Description Framework (RDF) triple stores, it becomes possible to merge massive amounts of structured and unstructured data by defining a common ontology model for the DBMS schemas and representing the structured content as semantic triples.

Literature Survey

In a recent study, Bouziane et al. divide QA systems into:

1) QA for web of documents and text: that follow three main distinct subtasks: Question Analysis, Document Retrieval, and Answer Extraction to process natural language questions and retrieve precise answers from textual documents.

2) QA for web of data : that apply Named Entity Recognition, Syntactic Parsing, Question Classification, and SPARQL Generation on natural language questions and retrieve precise answers from Linked Data.

Literature Survey

- Development of QA for web of documents and text has been the center of research in the IR and NLP communities for several decades.

- These QA systems rely on shallow, named entity based indexing to retrieve a small set of candidate answer documents from large collections.

- The candidate answer documents undergo deep semantic analysis in a post - processing phase to retrieve the final answers.

- These QA system do not perform well on list and definition questions.

- The lack of semantic knowledge being indexed and queried in the document retrieval phase results in low coverage of answer candidate sentences/documents for further analysis and processing, and thus leads to non optimal performance on certain types of questions.

Literature Survey

- QA for web of data has lately drawn the attention of many researchers and has resulted in the development of several QA systems for Linked Data.

- In such QA systems, unstructured text is first converted semantically in a structured format such as RDF representation.

- Most approaches to QA for web of data use dependency or syntactic parsing to extract and represent a question's semantics as a set of triples and then build a query that is then queried on the structured data.

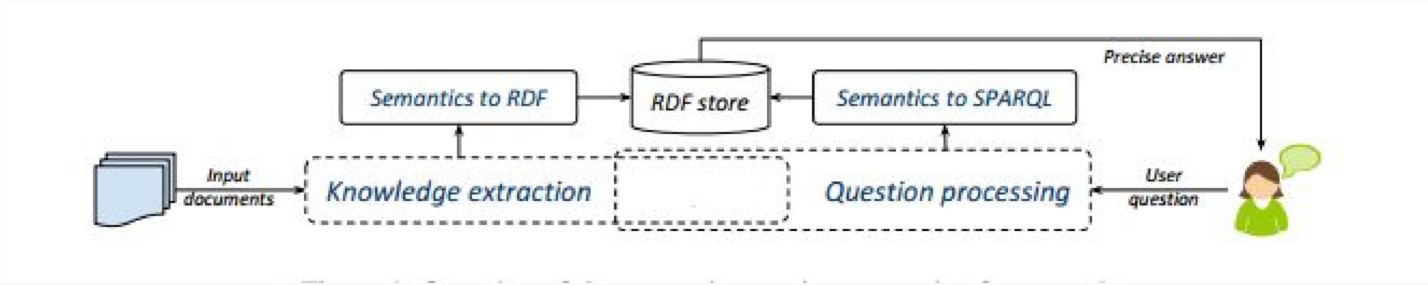
Aim

We aim to develop a robust question answering framework which can perform the following tasks:-

- Transform free text or unstructured data to structured format
- Merge with other ontologies and structured data into a consolidated RDF store
- A natural language Question Answering interface

Approach

Our semantic question answering (SQA) framework pinpoints the semantics of both the document collection as well as the user's input question. In the Document Indexing phase, the semantic information derived from document content is represented in an RDF format that facilitates its storage into a RDF triple store. At query time, the user's natural language question is parsed to identify its meaning. This is then automatically converted into a SPARQL query that will be used to retrieve precise answers from the already populated RDF store.



```
Input: Document collection
Output: RDF semantic index

1.   For each input document:
          RDF representation of its
          semantics
2.   Load document RDF triples in RDF
     store
3.   Load custom/domain ontology/WordNet
     in triplestore
4.   Define entailment rules for reasoning
     on the RDF store
5.   Generate new triples using already
     defined entailment rules
```

Algorithm for generating RDF triples

```
Input: Natural language question
Output: Precise answer

1.   question understanding
     a.   NLP of input question
     b.   answer type/answer type term
          detection
2.   SPARQL query formulation
3.   Query the RDF store
4.   If answer(s) found,
     return answers sorted by confidence
```

Algorithm for Natural Language Question to SPARQL Queries

Timeline

	Mid-Sem		End-Sem	
	Phase-I	Phase-II	Phase-III	Phase-IV
	5 Jan- 5 Feb	6Feb- 28Feb	15March- 05April	6 April- 28 April
LITERATURE SURVEY	Done			
PROBLEM IDENTIFICATION	Done			
IDENTIFY AN APPROACH		Done		
RDF GENERATION		Done		
STORING RDF				
SPARQL GENERATION				
CREATING FRAMEWORK				
EVALUATION				

Technologies Used

- Resource Description Framework
- SPARQL
- NLTK
- Python programming language

Technologies Used

- ➔ **Resource Description Framework** - It is a model used for representing relationships among data items. It is based upon the idea of making statements about resources expressions, known as triples. Triples are so named because they follow a subject–predicate–object structure. The subject denotes the resource, and the predicate denotes traits or aspects of the resource, and expresses a relationship between the subject and the object.It is data structured in graph.

  For example, one way to represent the notion "The sky has the color blue" in RDF is as the triple: a subject denoting "the sky", a predicate denoting "has the color", and an object denoting "blue".

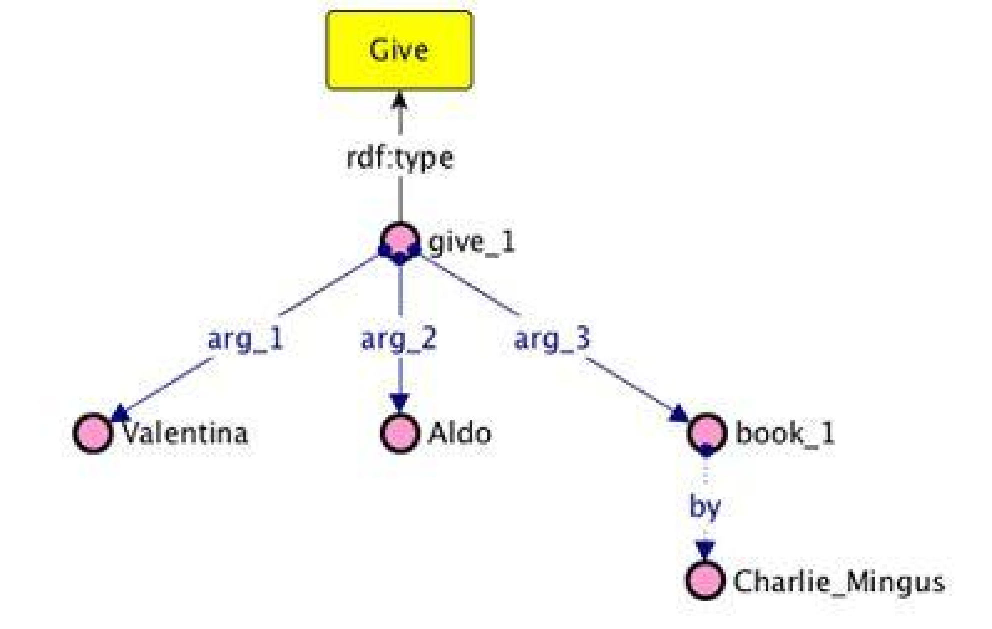
Technologies Used

➔ **SPARQL** (Sparql Protocol and RDF Query Language) : SPARQL is a semantic query language for databases, able to retrieve and manipulate data stored in Resource Description Framework (RDF) format. It allows for a query to consist of triple patterns, conjunctions, disjunctions, and optional patterns.

➔ **NLTK (Natural Language Toolkit) :** NLTK is a suite of libraries and programs for symbolic and statistical natural language processing (NLP) for English written in the Python programming language.

Example

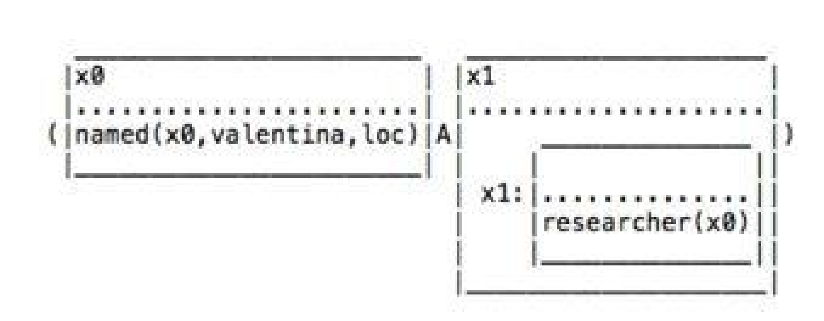
Sentence : Valentina gave Aldo a book by Charlie Mingus



Example

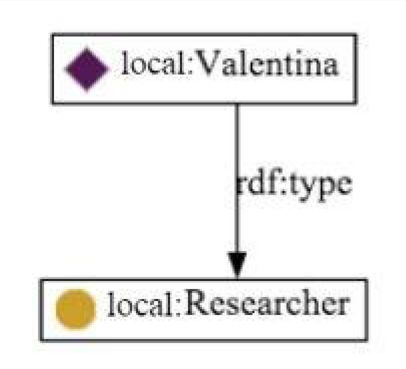
Figure(1) is an example of intermediate representation of a sentence known as Discourse Representation Structures(DRS) informally called as “boxes” in the process of converting text to RDF

Sentence : Valentina is a researcher



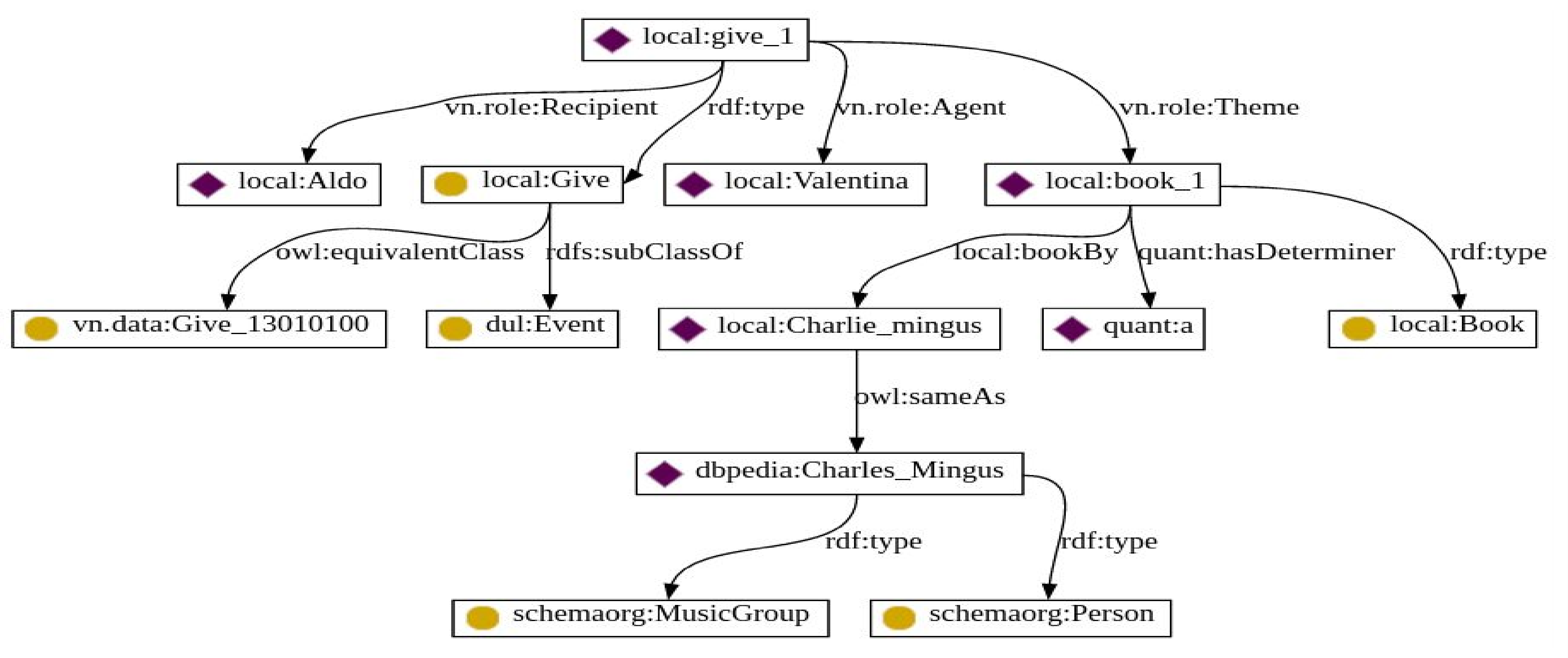
```
  _________________________   _____________________
 |x0                      | |x1                   |
 |........................| |.....................|
(|named(x0,valentina,loc) |A|                     |)
 |________________________| |   ______________    |
                            |  |              |   ||
                            |x1:|..............|  ||
                            |  |researcher(x0)|   ||
                            |  |______________|   ||
                            |_____________________|
```

Figure (1)



Example: Text to RDF

Sentence: Valentina give Aldo a book by Charlie Mingus



Sample RDF Triple

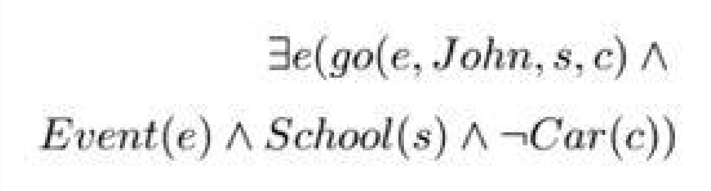
```
<rdf:Description rdf:about="http://dbpedia.org/resource/Charles_Mingus">
   <rdf:type rdf:resource="http://schema.org/MusicGroup"/>
   <rdf:type rdf:resource="http://schema.org/Person"/>
 </rdf:Description>
```

Inherent difficulties in the english language

A sentence with ambiguous interpretations :-

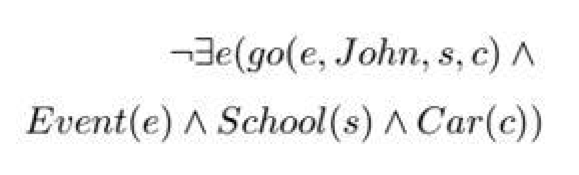
Sentence : John did not go to school by car

1) There is an event in which John went to school, but not by car



$$\exists e(go(e, John, s, c) \land$$
$$Event(e) \land School(s) \land \neg Car(c))$$

2) There is no event in which John went to school by car



$$\neg\exists e(go(e, John, s, c) \land$$
$$Event(e) \land School(s) \land Car(c))$$

Thank You