A Semantic Framework for the Retrieval and Execution of Open Source Code

Mattia Atzeni and Maurizio Atzori

Università degli Studi di Cagliari

Problem Statement

We introduce an unsupervised approach to process questions that cannot be answered by factual QA nor advanced data querying, requiring instead ad-hoc code generation and execution.

*What is the **cube root** of the **max** between 20 and 27?*

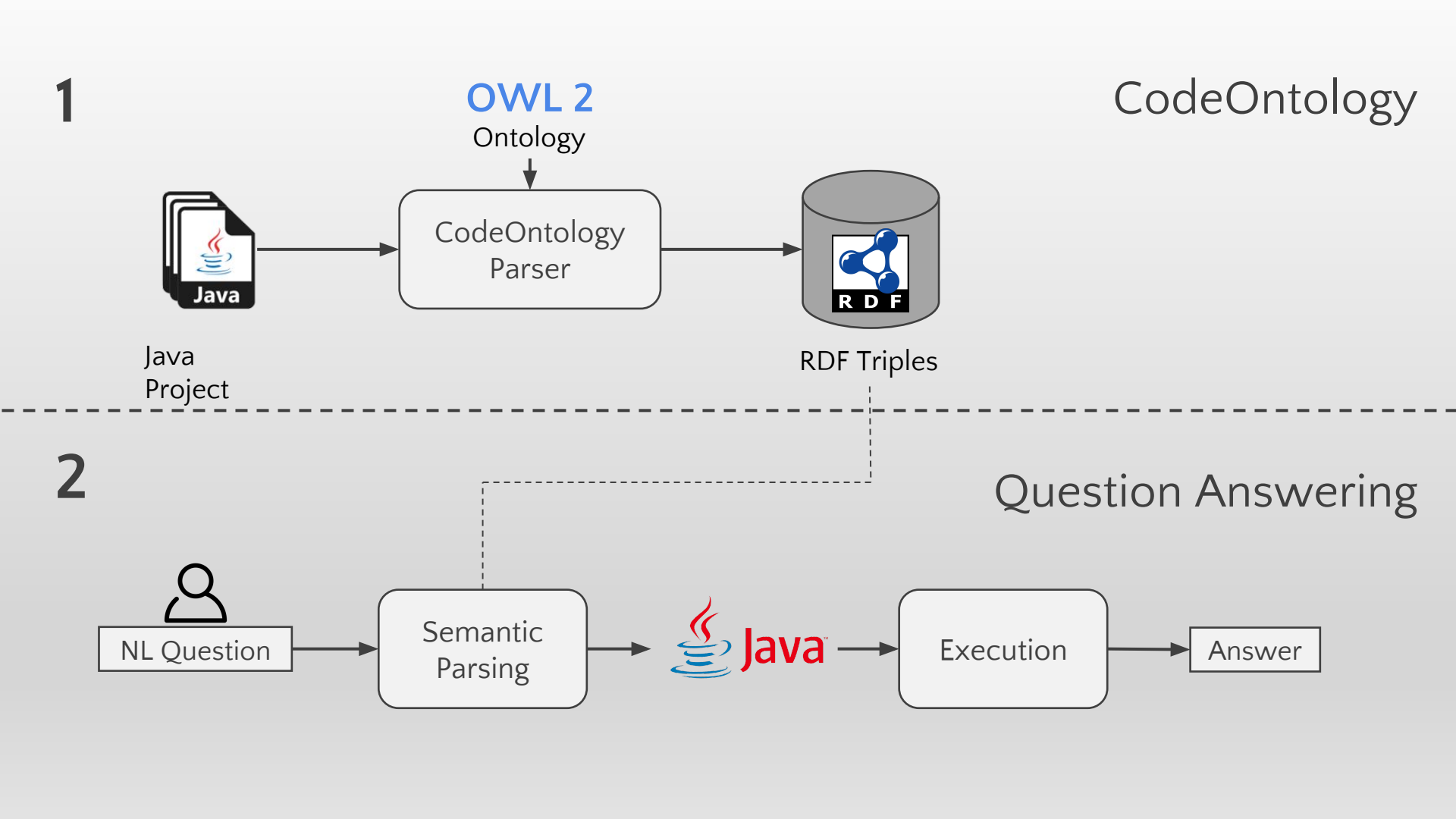
```
java.lang.Math.cbrt(java.lang.Math.max(20, 27))
```

Problem Statement

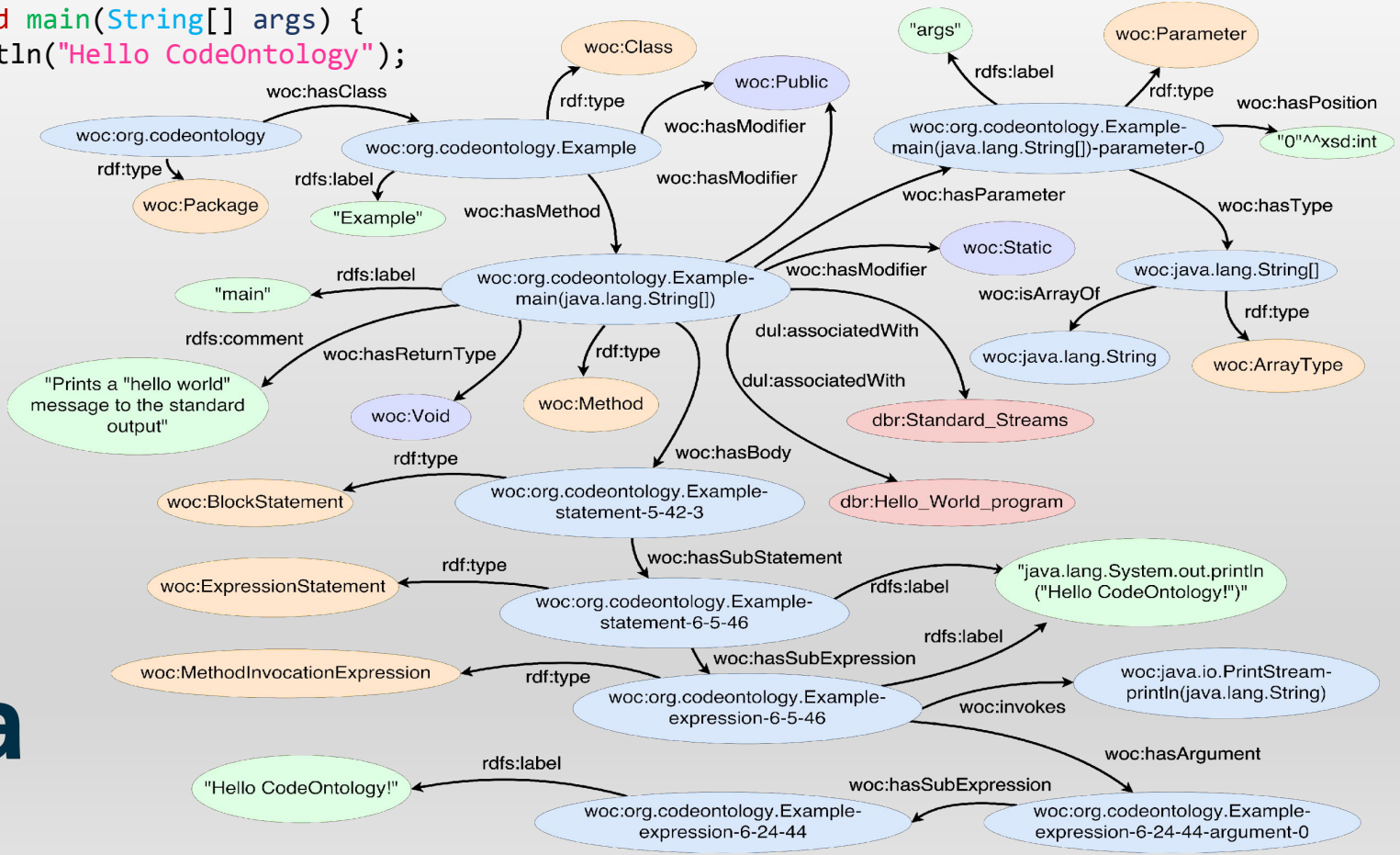
The problem is divided into two subproblems:

1. Providing a **semantic representation** of object-oriented source code

2. Performing **Question Answering** on the resulting Knowledge Base


1
OWL 2
Ontology
CodeOntology
Java
CodeOntology
Parser
R D F
Java
Project
RDF Triples
2
Question Answering
NL Question
Semantic
Parsing
Java
Execution
Answer

```java
package org.codeontology;
public class Example {
  /** Prints a "hello world" message to the standard output */
  public static void main(String[] args) {
    System.out.println("Hello CodeOntology");
  }
}
```



SPARQL Queries

- Select all methods computing the cube root of a real number.

```
SELECT ?method
WHERE {
  ?method a woc:Method ;
      woc:hasParameter/woc:hasType woc:Double ;
      dul:associatedWith dbpedia:Cube_root .
}
```

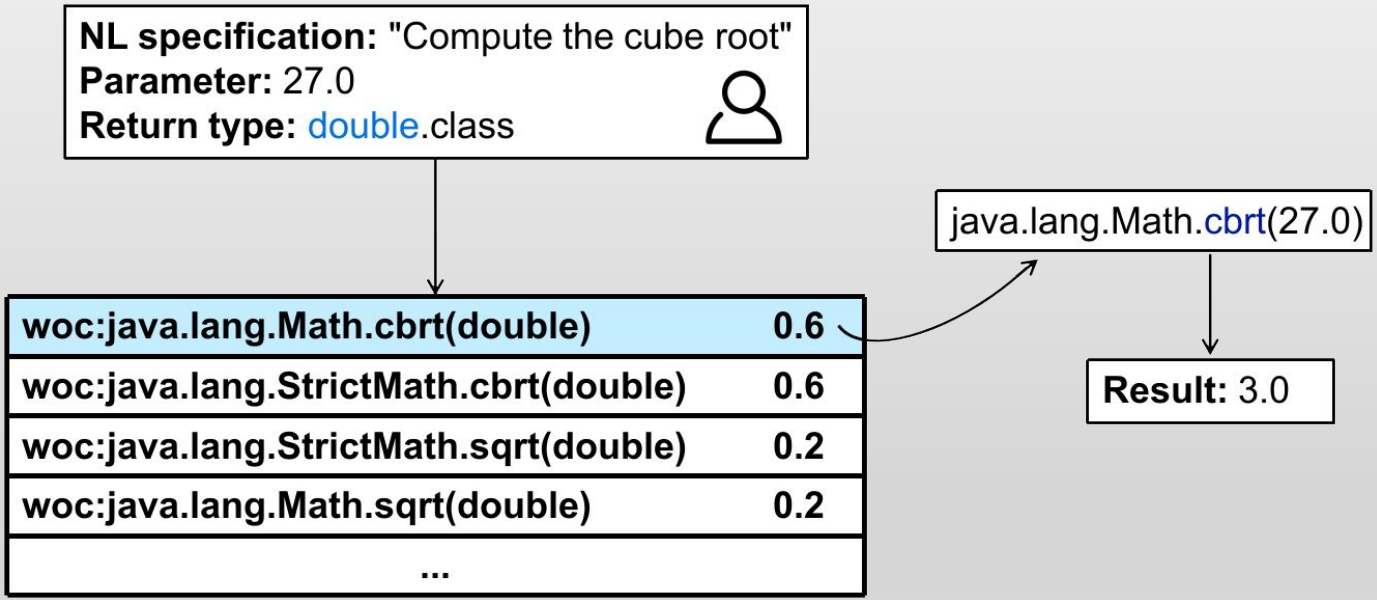
?method
java.lang.Math.cbrt(double)
java.lang.StrictMath.cbrt(double)

Question Answering

We discuss two approaches:

1. a **coarse-grained** approach that only supports natural language sentences corresponding to the execution of a single method;
2. a **fine-grained** approach that supports more complex questions, possibly requiring the execution of multiple methods.

Coarse-Grained Approach



Ranking on CodeOntology

The ranking of Java methods relies on the following attributes:

- the name of the method
- the name of the declaring class
- the documentation comments
- semantic links to DBpedia already provided by CodeOntology

Syntactic Features

LS: normalized Levenshtein similarity between the natural language specification and the name of the method, computed as

$$s_L(s_1, s_2) = 1 - \frac{d_L(s_1, s_2)}{\max\{|s_1|, |s_2|\}}$$

where $d_L(s_1, s_2)$ is the Levenshtein distance between s_1 and s_2, namely the minimum number of single character edits required to turn one string into the other.

Syntactic Features

COM: *n*-gram overlap against the Javadoc comment associated with the method, computed as

$$ngo(S_1, S_2) = 2 \cdot \left(\frac{|S_1|}{|S_1 \cap S_2|} + \frac{|S_2|}{|S_1 \cap S_2|} \right)^{-1}$$

where S_1 and S_2 are set of consecutive *n*-grams from two different sentences.

CN: *n*-gram overlap against the name of the declaring class.

Semantic Features

W2V: cosine similarity between the mean vectors (computed with a pre-trained Word2Vec model) associated with the natural language specification provided by the user and the Javadoc comment.

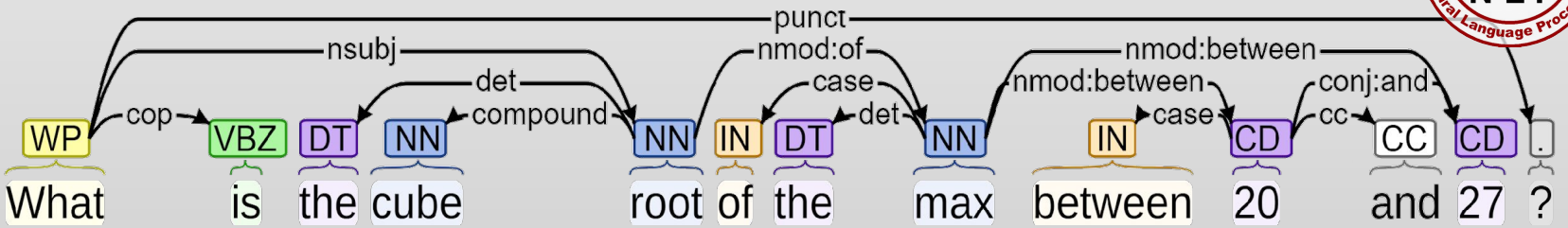
NED: ratio of DBpedia links shared by the method and the natural language specification.

Experiments

	Features	MAP@1	MAP
Syntactic Features	LS	0,70	0,78
	LS + CN	0,71	0,79
	LS + COM	0,86	0,89
	LS + CN +COM	0,87	0,90
Semantic Features	NED	0,61	0,71
	W2V	0,74	0,82
	W2V + NED	0,75	0,82
Syntactic + Semantic Features	LS + W2V + NED	0,80	0,86
	LS + CN + COM + W2V + NED	**0,90**	**0,92**

Fine-Grained Approach

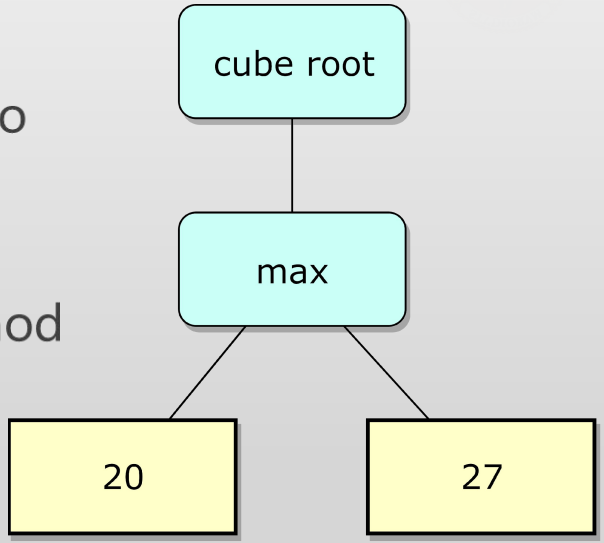
Given a natural language question, the fine-grained approach starts by performing Dependency Parsing.



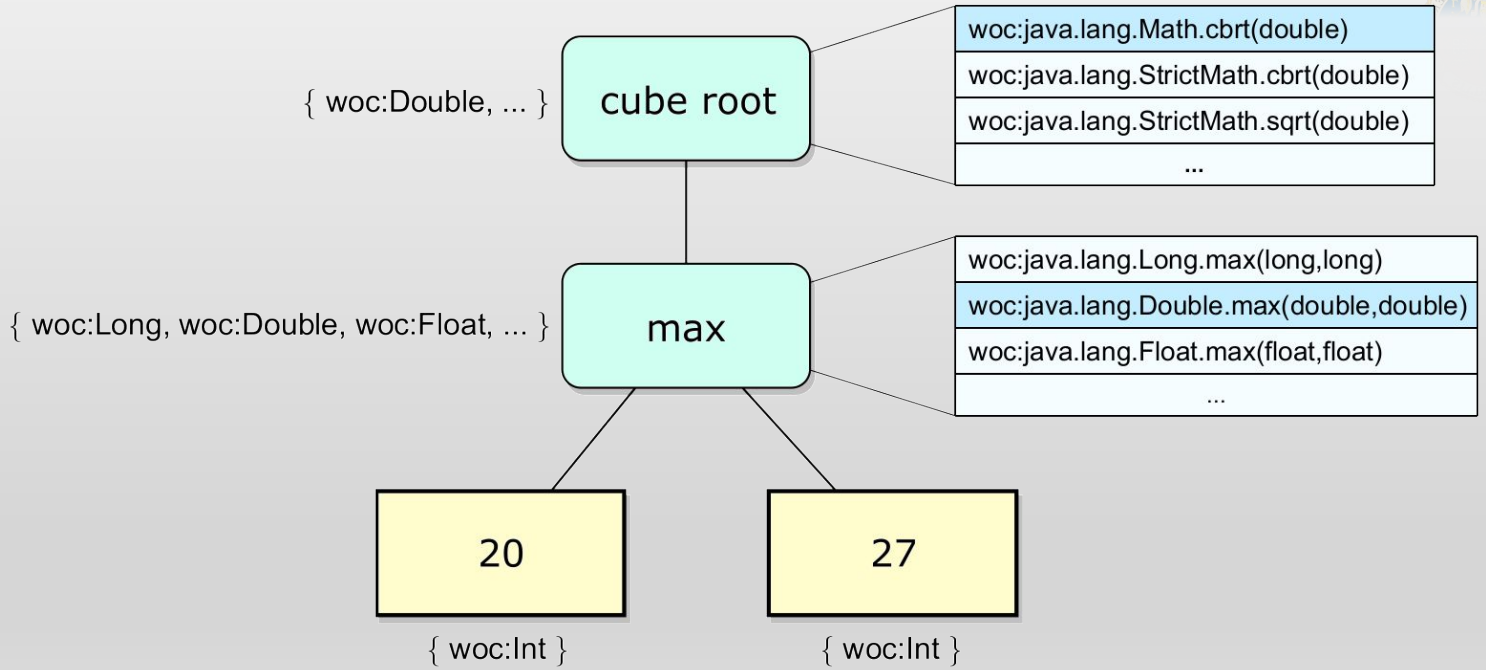
Dependency Graph Unfolding

The dependency graph is unfolded into a tree, such that:

- the set of nodes N can be partitioned into two subsets, $\mathcal{L}$ and $\mathcal{M}$
- $\mathcal{L}$ is the set of literal nodes
- $\mathcal{M}$ is the set of nodes corresponding to method invocations
- Each node in $\mathcal{L}$ is a leaf
- $N = \mathcal{L} \cup \mathcal{M}$ and $\mathcal{L} \cap \mathcal{M} = \emptyset$



Mapping to a Feasible Tree



Mapping to a Feasible Tree

We need to select a method for each ranking while maximizing the total score:

$$\text{Maximize} \sum_{i \in \mathcal{M}} \sum_{(m_{ij}, s_{ij}) \in \mathcal{R}_i} x_{ij} \cdot s_{ij}$$

subject to:

- $x_{ij} \in \{0,1\}$
- $\sum_{j=1}^{|\mathcal{R}_i|} x_{ij} = 1$, for all $i \in \mathcal{M}$
- the combination of selected methods can be compiled.

Greedy Search

We want to improve the algorithm so that it is robust to two kinds of situations:

- the tree resulting from the unfolding of the dependency graph has more nodes than needed;
- the dependency graph produced by Stanford CoreNLP contains some errors.

Hence, we perform a **greedy search** on the tree structure.

Objective Function

Starting from an initial tree $\mathcal{T}_o$, the algorithm performs a greedy search with a *Best-Improvement* strategy in order to maximize, under the same contraints introduced above, the following objective function:

$$z(\mathcal{T}_k) = \frac{1}{|\mathcal{M}_k|} \cdot \sum_{i \in \mathcal{M}_k} \sum_{(m_{ij}, s_{ij}) \in \mathcal{R}_i^k} x_{ij} \cdot s_{ij} \; - \lambda \cdot NTED(\mathcal{T}_k, \mathcal{T}_o)$$

where $\lambda \in [0,1]$ is a constant and $NTED(\mathcal{T}_k, \mathcal{T}_0)$ is the normalized Tree Edit Distance between $\mathcal{T}_k$ and $\mathcal{T}_0$, defined as

$$NTED(\mathcal{T}, \mathcal{T}') = \frac{TED(\mathcal{T}, \mathcal{T}')}{\max\{|\mathcal{T}|, |\mathcal{T}'|\}}$$

Experiments

The proposed approach has been compared with the WolframAlpha computational knowledge engine, on a dataset containing 120 questions on math and string manipulation.

	QA over CodeOntology	Wolfram Alpha
Number of Questions	120	120
Processed Questions	116	108
Correct Answers	109	98
Precision (global)	0,91	0,82
Precision (processed questions)	0,94	0,91

Thank You

Mattia Atzeni and Maurizio Atzori