GQA: Grammatical Question Answering for RDF Data

Elizaveta Zimina, Jyrki Nummenmaa, Kalervo Järvelin,
Jaakko Peltonen, Kostas Stefanidis, and Heikki Hyyrö

University of Tampere, Finland
firstname.lastname@uta.fi

Abstract. Nowadays, we observe a rapid increase in the volume of RDF knowledge bases (KBs) and a need for functionalities that will help users access them in natural language without knowing the features of the KBs and structured query languages, such as SPARQL. This paper introduces Grammatical Question Answering (GQA), a system for answering questions in the English language over DBpedia, which involves parsing of questions by means of Grammatical Framework and further analysis of grammar components. We built an abstract conceptual grammar and a concrete English grammar, so that the system can handle complex syntactic constructions that are in the focus of the SQA2018 challenge. The parses are further analysed and transformed into SPARQL queries that can be used to retrieve the answers for the users' questions.

Keywords: Grammatical Framework · DBpedia · question answering · SQA.

1 Introduction

Retrieving information from knowledge bases is a commonplace task, which is becoming further widespread due to digitalisation of society; this has also increased the need for systems that support natural language interaction.

Typically, data in knowledge bases are represented using the RDF model. In RDF, everything we wish to describe is a resource that may be a person, an institution, a thing, a concept, or a relation between other resources. The building block of RDF is a triple of the form (*subject*, *predicate*, *object*). The flexibility of the RDF data model allows representation of both schema and instance information in the form of RDF triples.

The traditional way to retrieve RDF data is through SPARQL [26], the W3C recommendation language for querying RDF datasets. SPARQL queries are built from triple patterns and determine the pattern to seek for; the answer is the part(s) of the set of RDF triples that match(es) this pattern. The correctness and completeness of answers of SPARQL queries are key challenges. SPARQL is a structured query language that allows users to submit queries that precisely identify their information needs, but requires them to be familiar with the syntax, and the complex semantics of the language, as well as with the underlying schema or ontology. Moreover, this interaction mode assumes that users are familiar with the content of the knowledge base and also have a clear understanding of their information needs. As databases become larger and accessible to a more diverse and less technically oriented audience, new forms of data exploration and interaction become increasingly attractive to aid users navigate through the information space and overcome the challenges of information overload [24, 25].

In this work, we present the Grammatical Question Answering (GQA) system for answering questions in the English language over DBpedia 2016-04 [8]. The GQA system's key technology is Grammatical Framework (GF) [23], which provides parsing of questions and extraction of conceptual categories, the main types of which are Entity, Property, Verb Phrase, Class, and Relative Clause. The system "unfolds" the parse layer by layer and matches its components with the KB schema and contents to formulate SPARQL queries corresponding to the English questions. The GQA was tested on the SQA2018 training set [15] and showed high precision in answering both simple and complex questions.

The rest of the paper is structured as follows. Section 2 presents an overview of the GQA system and resources used, while Section 3 describes the GQA grammar. Section 4 provides the details of the parse interpretation module, and Section 5 analyses the system's testing results. The related work is surveyed in Section 6, and conclusions and perspectives for improvement are stated in Section 7.

2 GQA Modules and Resources

2.1 General Architecture of GQA

The keys to successful question answering over knowledge bases are correct interpretation of questions and proper retrieval of information. In the GQA system (Fig. 1), this is realised by means of the two main modules:

• the Conceptual Parsing module, which obtains all possible parses of a question according to the GQA grammar and selects the most suitable one, and

• the Parse Interpretation module, which analyses the question parse, gradually "unfolding" its elements and making requests to the knowledge base in order to control the process of parse interpretation and finally arrive at the most probable answer(s).

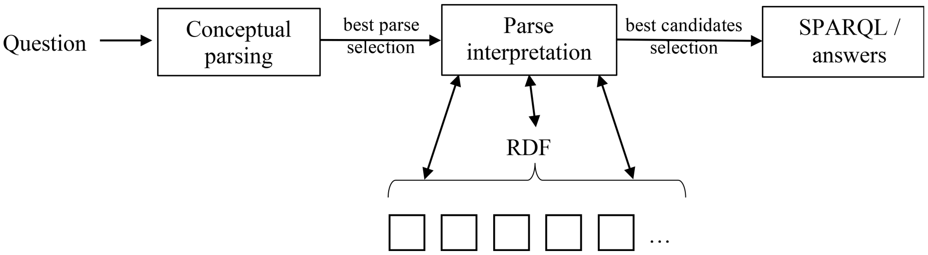

Fig. 1. General Architecture of GQA

The parsing module exploits the technology of Grammatical Framework, and the GQA grammar is built on top of the existing GF Resource Grammar Library. The parse analysis is performed in Python, and the KB is indexed by means of Apache Lucene.

2.2 Grammatical Framework

Grammatical Framework (GF) [22, 23] is a functional programming language and a platform for multilingual grammar-based applications. A GF grammar consists of an abstract syntax, which declares the grammar's meanings and their correlation in the form of abstract syntax trees, and one or more concrete syntaxes, which determine the mapping of abstract syntax trees onto strings and vice versa.

The analysis of natural language strings is realised through parsing, that is, deriving an abstract syntax tree (or several) from a string. Conversely, strings can be generated by means of linearisation of parse trees.

Natural language grammars implemented in GF are collected in the GF Resource Grammar Library (RGL) [5]. The concrete grammars for all (currently 40) languages in the RGL comply with the common abstract syntax, which aims to reflect the universal linguistic categories and relations.

Thus, in most cases the RGL categories correspond to parts of speech (noun, verb, etc.) and textual units (sentence, question, text, etc.). These categories can act as arguments to functions forming other categories. For example, we can define (in a simplified way) an abstract syntax rule that takes an pronoun (*PN*) and a verb (*V*) and forms a sentence (*S*) as follows:

```
mkS : PN -> V -> S.
```

Let us write the linearisation definition of this rule for an English concrete syntax, taking only the present simple tense. To make the correct agreement possible, we first need to specify the *Person* and *Number* parameters:

```
Person = Fisrt | Second | Third ;
Number = Sg | Pl ;
```

and then can define the linearisation types for *NP*, *V* and *S*:

```
PN = {s : Str ; p : Person ; n : Number} ;
V  = {s : Person => Number => Str} ;
S  = {s : Str}.
```

Any *PN* has a permanent string value *s* as well as inherent values *p* and *n*, standing for *Person* and *Number*. The string value *s* of *V* is represented by an inflectional table, so that it can change depending on the *Person* and *Number* values. *S* has only a simple string value *s*.

Finally, the linearisation definition for the rule *mkS* can be written as:

```
mkS pron verb = {s = pron.s ++ verb.s ! pron.p ! pron.n}.
```

The string value *s* of *S* is obtained by joining the generated string values of *PN* and *V* so that the verb agrees with the pronoun in *Person* and *Number*.

We can now add some vocabulary rules to the abstract syntax:

```
we_PN, you_PN, she_PN : PN ;
sleep_V : V ;
```

and their linearisation definitions to the concrete syntax:

```
we_PN   = {s = "we"  ; p = First  ; n = Pl} ;
you_PN  = {s = "you" ; p = Second ; n = Pl} ;
she_PN  = {s = "she" ; p = Third  ; n = Sg} ;
sleep_V = {s = table {
          First  => table {Sg => "sleep"  ; Pl => "sleep"} ;
          Second => table {Sg => "sleep"  ; Pl => "sleep"} ;
          Third  => table {Sg => "sleeps" ; Pl => "sleep"}}} ;
```

So the parse trees of the strings *we sleep, you sleep* and *she sleeps* will look like:

```
mkS we_PN sleep_V ;
mkS you_PN sleep_V ;
mkS she_PN sleep_V.
```

2.3 DBpedia

The GQA system exploits the English DBpedia version 2016-04 [8] that involves: entity labels (6.0M), redirect pages having alternative labels of entities and leading to pages with "canonical" labels (7.3M), disambiguation pages containing entities with multiple meanings (260K), infobox statements (triples where properties are extracted from infoboxes of Wikipedia articles, often noisy and unsystematic; 30.0M), mapping-based statements (triples taken from the hand-generated DBpedia ontology, cleaner and better structured than the infobox dataset; 37.5M), and instance type statements attributing an entity to one or several DBpedia ontology types, e.g. *person, country, book*, etc. (36.7M).

To be able to access this large volume of information quickly, we built an index of all necessary data components by means of Apache Lucene [2]. For infobox and mapping-based statements, we also make inverse indexes, so that the search can be done either by subjects or by values of RDF triples.

3 The GQA Grammar

3.1 Overview

The GQA grammar complies with the morphological principles underlying the RGL and at the same time operates with new conceptual categories that make it possible to classify questions and to reveal their components and relations between them. The main conceptual categories in the GQA grammar include:

- *Entity*: usually a proper name, e.g. *In which country does the **Nile**[1] start?*
- *Property*: a noun phrase that coincides with the label of some property in DBpedia, e.g. *What is the **area code** of Berlin?*

[1] Words belonging to the category/rule under consideration are given in bold.

- *VPChunk*: corresponds to all possible morphological forms of *VP* (verb phrase) and usually represents a reformulation of some property (e.g. the verb *dissolve* in the *when*-question *When did the Ming dynasty* ***dissolve****?* indicates the *dissolution date* property),
- *Class*: a noun phrase that corresponds to an ontology class in DBpedia, e.g. *List all the* ***musicals*** *with music by Elton John.*
- *RelCl*: a relative clause, e.g. *Who owns the company* ***that made the Edsel Villager****?*
- *Q*: the starting category comprising all kinds of questions learnt through the analysis of the training sets. For example, the question *When did Operation Overlord commence?* will be recognised by the function *WhenDidXVP* with the definition *Entity* → *VPChunk* → *Q*.

3.2 Simple Entity

So-called simple entities in the GQA grammar are the only elements that are not actually coded, but are rather regarded as symbols, that is, lists of words separated by whitespaces in natural language questions. Thus, simple entities are formed by means of functions:

oneWordEnt : *String* → *Entity ;*

twoWordEnt : *String* → *String* → *Entity ;*

up to *tenWordEnt*.

The English DBpedia 2016-04 comprises 13.3M labels, i.e. names of entities and their redirects (names of pages redirecting to the main pages, e.g. *Houses of Parliament* → *Palace of Westminster*). Indexing the inverse properties (described in Section 3.3) reveals even more entities without separate pages in DBpedia. It was impossible and in fact not necessary to code such a number of entities in the grammar, since it would dramatically slow the parsing down. Instead, we built an index of all DBpedia labels and use it for finding entity links. In many cases the input phrase needs some modification before looking for it in the index. These modifications include:

- removal of the article in the beginning of the phrase (*the Isar* → *http://dbpedia.org/resource/Isar*),
- capitalisation of all words, excluding auxiliary parts of speech (*maritime museum of San Diego* → *http://dbpedia.org/resource/Maritime_Museum_of_San_Diego*),
- rearrangement of components separated by the *in* preposition (*lighthouse in Colombo* → *http://dbpedia.org/resource/Colombo_Lighthouse*).

Quite often the label found in the index leads us to the disambiguation page containing links of several possible entities. For example, the entity *Sherlock* in the question *What company involved in the development of Sherlock did Arthur Levinson work for?* is highly polysemous (20 readings in DBpedia), and we should check all of them to find the one that matches all properties expressed in the question (*http://dbpedia.org/resource/Sherlock_(software)*).

3.3 Property

Infobox and Ontology Properties. We collected the labels of all properties in the 2016-04 version of DBpedia. This yielded almost 17,000 grammar rules, most of which are due to DBpedia's noisiness (e.g. properties *cj*, *ci*, or *ch*) and are of no use for question answering. This set still contains "valid" properties (e.g. *height* or *spouse*) and is not large enough to slow down parsing significantly.

Search is performed among property links that start either with *http://dbpedia.org/property/* (Wikipedia infobox properties) or *http://dbpedia.org/ontology/* (better structured ontology based on the most commonly used infoboxes). In the GQA grammar property name endings correspond to the function names with suffixes *_O* (e.g. *owner_O* in the above example) or *_P* showing whether *ontology* or *property* is used in the full link. As a result, properties often duplicate each other, and the correct answer can be obtained with any of them (e.g. *http://dbpedia.org/ontology/birthPlace* or *http://dbpedia.org/property/birthPlace*). Thus, the best parse of the question *What is the birthplace of Louis Couperin?* will contain the function *birthPlace_O*, since *_O* functions are given higher priority in selecting the best parse, but the *http://dbpedia.org/property/birthPlace* property will be also checked if no answer is found with the ontology property.

At the same time, some infobox properties do not have corresponding ontology properties (e.g. *http://dbpedia.org/property/placeOfBirth*), but are identical in meaning with other properties and in some cases can be the only key to the answer. Analysing the training datasets, we develop a separate database *Functions* with groups of identical properties, each group stored under the name of the rule that would be most likely selected in the best parse.

Inverse Properties. The question type *What is the [property] of [entity]?* assumes that after the resolution of the entity we need to look for the value of the corresponding property that should be present in this page. A number of syntactic and semantic constructions require so-called inverse search, when we look for a subject, not a value in a triple. For example, the SPARQL query for the question *Whose deputy is Neil Brown?* looks like

```
SELECT DISTINCT ?uri WHERE {?uri <http://dbpedia.org/property/
deputy> <http://dbpedia.org/resource/Neil_Brown_(Australian_
politician)>}.
```

Since the genitive construction is used, the target variable *?uri* is in the position of the triple subject.

We built a separate "inverse" index, which makes this kind of search possible. In our *Functions* database we assign a Boolean value to each property, depending on the function that calls it and showing whether the direct (*False*) or inverse (*True*) search is needed. The value *True* is mostly assigned to properties assumed in participle clauses with past participles constructions and some verb phrases, described in Section 3.5.

3.4 Class

A DBpedia ontology class can be considered as a generic term that an entity belongs to (e.g. *written work, beverage, holiday*, etc). As well as ontology property URIs, class URIs have the prefix *http://dbpedia.org/ontology/*. Classes in questions can have various syntactic roles and forms, e.g. in English they can be inflected for case and number. For example, the parses of the questions *Which magazine's editor is married to Crystal Harris?* and *Give me all magazines whose editors live in Chicago* both should contain the function *Magazine_Class*, so that the parse analysis module will "know" that the search should be done among the entities belonging to the class with the URI *http://dbpedia.org/ontology/Magazine*.

This morphological flexibility of classes is obtained by means of the RGL paradigms. We collected 764 classes currently existing in the DBpedia ontology and parsed them as common noun chunks, which can take any morphological form. For example, in the GQA grammar the *Magazine_Class* function is linearised as

```
Magazine_Class = UseN magazine_N.
```

The linearisation of *magazine_N* is taken from the RGL English wide-coverage dictionary. This noun type complies with the inflection table

```
Sg => table {Nom => magazine ; Gen => magazine + "’s"} ;
Pl => table {Nom => magazine + "s" ; Gen => magazine + "s’"},
```

building the singular and plural noun forms in the nominative and genitive cases.

The function *UseN* creates a common noun out of a noun (*UseN* : $N \rightarrow$ *CN*), which has the same type definition as *Class*. Finally, any grammatical form of *Class* is detected as a class chunk :

```
Class_Chunk : Class -> ClassChunk
Class_Chunk cl = {s = variants {cl.s ! Sg ! Nom; cl.s ! Pl ! Nom;
                                cl.s ! Sg ! Gen; cl.s ! Pl ! Gen}},
```

so that *magazine*, *magazine's*, *magazines* and *magazines'* in a question parse will be represented by the branch `Class_Chunk Magazine_Class`. In this way, the RGL makes it simpler to focus on the semantic structure of a question, without a great effort in grammar analysis.

Studying the training sets, we revealed a number of reformulations of class names. For example, the *television show* class can be meant if a question contains words like *program, series, show, sitcom, television program, television series*, etc. At the same time, the word *program* can also refer to another class – *radio program*. As with the *Functions* database for ambiguous properties, we built the *Classes* database containing groups of related classes and their textual expressions. The largest groups are gathered under the words *league* (*sports league, American football league, Australian football league*, etc. – 27 options) and *player* (*soccer player, basketball player, ice hockey player*, etc. – 26 options). Currently, the system checks all the classes of the group and selects the largest set of answers belonging to the same class. Those class linearisations that are highly ambiguous significantly decrease the speed of finding the answer.

3.5 Verb Phrase and Participle

Questions can refer to a property in several ways, one of which is question predicates expressed by verbs and verb phrases. For example, in the question *Who owns Torrey Pines Gliderport?* the verb *owns* should be attributed to the property *owner*. To enable our system to do this, the GQA grammar should contain the verb *own* and the parse interpretation module should "know" how to map certain verbs with corresponding properties.

Analysing the training set, we extracted verbs and verb phrases from questions and correlated them with corresponding properties in SPARQL queries. These correlations are recorded in the *Functions* database, which is used at the parse interpretation step.

In the GQA grammar, verbs and verb phrases often act as verb phrase chunks (*VPChunk*), so that they can be used in any morphological form, in a similar way as class chunks described in Section 3.4. Thus, *own, owns, owned*, etc. will be recognised by the grammar under the rule *own_ V2*[2], due to the verb paradigms inherited from the RGL grammar.

In many cases one and the same verb can refer to different properties. For example, judging by the SPARQL query for the question *Which company owns Evraz?*, we should attribute the verb *own* also to the property *owningCompany*. All alternative readings are stored in the *Functions* database under the corresponding rule, e.g.

```
"own_V2": [("http://dbpedia.org/ontology/owner", False),
           ("http://dbpedia.org/ontology/owningOrganisation", False),
           ("http://dbpedia.org/ontology/owningCompany", False),
           ("http://dbpedia.org/property/owners", False),
           ("http://dbpedia.org/ontology/parentCompany", False)...]
```

The *False* value means that the corresponding property does not involve inverse search (see Section 3.3). At the parse interpretation step the system checks all properties that the function refers to.

Examples of more complex verb phrases include: *originate in, work for, lead to the demise of, do PhD under*, etc. In the GQA grammar they are referred as *idiomatic VPSlashChunk*s (*IVPS*).

Similarly to verb phrases, participial constructions (e.g. *located at, followed by, starring in*, etc.), are also learnt from the training sets and stored in the *Functions* database. An exception is the most common model *past participle + by*, when the

[2] Following the logic of the RGL grammar, in order to form a *VPChunk*, a *V2* (two-place verb) is turned into the intermediate category *VPSlashChunk* (*V2_ to_ VPSlash* : *V2* → *VPSlashChunk*) and then is complemented by a noun phrase, in our case – *Entity* (*VPSlash_ to_ VPChunk* : *VPSlashChunk* → *Entity* → *VPChunk*). A *VPSlashChunk* can be also formed by a participial construction (e.g. *BeDoneBy_ VPslash* : *Participle2* → *VPSlashChunk*).

verb that the participle is derived from already exists in *Functions*, e.g. *written by* in the question *What are the movies written by Nick Castle?* The properties that *written by* refers to are the same as those of the verb *write*, except that we need to carry out the inverse search, since now the author is known and the objects in question are subjects of RDF triples, not values, as with the verb.

3.6 Relative Clause

Below are examples of common rules building relative clauses in the GQA grammar. Words belonging to elements of relative clauses are in square brackets:

- *who/that/which + verb phrase*
  `WhoVP_relcl : VPChunk -> RelCl`
  Who are the people ***who [influenced the writers of Evenor]****?*
- *whose + property + is/are + entity*
  `WhosePropIsX_relcl : Property -> Entity -> RelCl`
  What is the city ***whose [mayor] is [Giorgos Kaminis]****?*
- *whose + property + verb phrase*
  `WhosePropVP_relcl : Property -> VPChunk -> RelCl`
  List the movies ***whose [editors] [are born in London]****.*
- *where + entity + verb phrase*
  `WhereXVP_relcl : Entity -> VPChunk -> RelCl`
  Where was the battle fought ***where [2nd Foreign Infantry Regiment] [participated]****?*

3.7 Complex Entity

Some complex constructions can perform the grammatical functions of entities. The most important of them in the GQA grammar are:

- homogeneous simple entities connected with the conjunction *and*. They are built through the rule:
  `EntityAndEntity : Entity -> Entity -> Entity ;`
  so that the corresponding branch of the tree for the question *In which team did* ***Dave Bing and Ron Reed*** *started their basketball career?* looks like:
  `EntityAndEntity (twoWordEnt "Dave" "Bing") (twoWordEnt "Ron" "Reed"),`
- a property of an entity:
  `PropOfEnt_to_Entity : Property -> Entity -> Entity`
  What is the alma mater of ***the successor of F. A. Little, Jr.****?*
  `PropOfEnt_to_Entity successor_O (fourWordEnt "F." "A." "Little," "Jr.")`
- a simple entity followed by some property:
  `EntProp_to_Entity : Entity -> Property -> Entity`
  Name ***Ivanpah Solar power facility owner****.*
  `EntProp_to_Entity (fourWordEnt "Ivanpah" "Solar" "power" "facility") owner_O.`
- a class with a relative clause:
  `Class_to_Ent : ClassChunk -> RelCl -> Entity`

What are the notable works of ***the person who produced Queer as Folk****?*

```
Class_to_Ent (Class_Chunk Person_Class) (WhoVP_relcl
(VPSlash_to_VPChunk (V2_to_VPSlash produce_V2) (threeWordEnt
"Queer" "as" "Folk")))
```

3.8 Question

A question's type is determined by the topmost rule in its parse, constructing the category *Q*. Currently the GQA grammar has 53 question-building rules, such as:

```
WhoVP : VPChunk -> Q ;
```
Who [developed Google Videos]?

```
WhereIsXDone : Entity -> V2 -> Q ;
```
Where was [WiZeefa] [founded]?

```
WhatPropOfEntIsPropOfEnt : Property -> Entity -> Property ->
Entity -> Q ;
```
Which [home town] of [Pavel Moroz] is the [death location] of [Yakov Estrin]?

```
WhatIsClassRelCl : ClassChunk -> RelCl -> Q ;
```
What are the [schools] [whose city is Reading, Berkshire]?

```
WhatClassVPAndVP : ClassChunk -> VPChunk -> VPChunk -> Q ;
```
Which [royalty] [was married to ptolemy XIII Theos Philopator] and [had mother named Cleopatra V]?

Note that structural words (*who, that, is, the, an,* etc.) are included only in the linearisation definitions of the rules and are not "visible" in parse trees, which is one of the main distinctions of our conceptual grammar from the "linguistic" RGL grammar. Certain words are made optional, for example, *different* in *Count the different types of Flatbread.*

One could also notice that the GQA grammar involves chunks, i.e. grammatically independent elements, which leave a possibility for ungrammatical constructions, e.g. *Which are the* ***television show*** *which have been created by Donald Wilson?* This example would have the same parse tree as a question *Which is an television shows which was created by Donald Wilson?*

4 Parse Analysis and Answer Extraction

4.1 Finding the Best Parse

The GQA grammar is ambiguous, i.e. a question can have several parses. One reason is that the grammar can consider a row of up to 10 tokens as an entity. For example, the phrase *the designer of REP Parasol* can be interpreted as

`PropOfEnt_to_Entity designer_O (twoWordEnt "REP" "Parasol")`,

meaning that the system detected the entity *REP Parasol* and its property *designer*. At the same time, the system can output an alternative reading

```
fiveWordEnt "the" "designer" "of" "REP" "Parasol",
```

which would not allow us to resolve the reference correctly. Thus, in choosing the parse that is further passed for interpretation we prioritise the one that has the least quoted strings in it. If this selection still gives us two or more parses with an equal number of "unresolved" strings (that is, components of entity names), we apply other prioritisation rules, e.g. preference of idiomatic constructions, class names over property names (which can sometimes coincide), and ontology property names over infobox property names (ontology properties are used in the *Functions* database as keys).

4.2 Unfolding the Parse and Querying the Index

The analysis of a selected parse starts from detecting the top function, determining the question type, and its arguments. Each argument is analysed separately, going from the outer function to the innermost one. For example, the best parse of the question *What are the awards won by the producer of Elizabeth: The Golden Age?* looks like:

```
WhatIsX (Class_to_Ent (Class_Chunk Award_Class) (WhoVP_relcl
(VPSlash_to_VPChunk (BeDoneBy_VPslash (DoneBy_PP win_V2)
 (PropOfEnt_to_Entity producer_O (fourWordEnt "Elizabeth:" "The"
"Golden" "Age")))))).
```

The innermost components of the parse are the property *producer* and entity *Elizabeth: The Golden Age*. The system looks for the entity's link, then checks that it contains the *producer* property. If not, the system tries to check if the entity is ambiguous and look through other readings. If the property is found, its value (another entity link(s)) is passed to the next level, where we look for the property *award* or *awards*. Thus, we "unfold" the parse layer by layer, starting from the inner components and making queries to DBpedia, so that we can be sure that the search is conducted in the right direction.

4.3 Spell Checking

GQA employs a string matching algorithm to tackle spelling errors in input questions. If the system cannot find a DBpedia page for a phrase that it considers as an entity name, it uses closest match by an approximate string matching method[3] to find the most likely entity page. In future we are planning to improve entity resolution by employing entity linking systems, such as DBpediaSpotlight [7] and FOX [14] (for a survey on entity linking and resolution, see [20]).

If the answer is still not found, the system tries to match question words against the GQA grammar's vocabulary (about 2500 words). For question words not found in the vocabulary, the system takes the one having the closest match to a vocabulary word by the string matching method, and corrects it to that match.

[3] *get_ close_ matches* in Python's difflib finds matches, *ratio* computes distances.

5 Testing

The GQA system was tested on the Scalable Question Answering Challenge (SQA) over Linked Data training set [15]. Participants of the SQA challenge are provided with an RDF dataset (DBpedia 2016-04) and a large training set (5000 questions) of questions with corresponding correct answers and SPARQL queries retrieving those answers. The evaluation of performance is based both on the number of correct answers and time needed.

The GQA system's performance results are presented in Table 1.

Table 1. Performance of GQA over the SQA training test set

Number of Questions	**Questions Answered**	**Questions Answered Correctly**	**Precision**
5000	1384 (27.68%)	1327 (26.54%)	95.88%

Table 2. The correlation of simple and complex questions answered by GQA

Simple Questions Answered	**Complex Questions Answered**	**Simple Questions Answered Correctly**	**Complex Questions Answered Correctly**	**Simple Questions Precision**	**Complex Questions Precision**
520	864	484	843	93.08%	97.57%

The majority of questions in the SQA task (about 68%) can be considered as complex questions, i.e. those questions whose SPARQL answers involve more than one RDF triple. The GQA system managed to correctly answer 484 simple questions (36.47% of the number of correctly answered questions) and 843 complex questions (63.53% of the number of correctly answered questions) (Table 2). Complex questions sometimes can seem confusing even for a human:

- *How many other architect are there of the historic places whose architect is also Stanford White?*
- *How many TV shows were made by someone who was associated with Lewis Hamilton?*
- *Which siler medalist of the Tennis at the 2012 Summer Olympics Men's singles was also the flagbearer of the Switzerland at the 2008 Summer Olympics?* (*sic*)

The reason for the relatively modest score for simple question answering is the system's current inability to analyse previously unseen formulations. However, if the system managed to output some answer, it was almost always correct, thus gaining the precision of 95.88%. The rare mistakes in the output queries were mostly related to incorrect entity resolution.

Further analysis of the output revealed that the GQA system often has a stricter approach to answer selection than the one used in the ground truth answer SPARQL queries. This is mostly related to class checking: whenever the GQA system detects some class in the parse, it outputs only those entities that belong to this class, whereas it is not always true in the SQA training set. For example, the question *Which politician is currently ruling over Rishkiesh?* (*sic*) has the answer *http://dbpedia.org/resource/Indian_ National_ Congress*, which does not belong to the *Politician* class. Another example is the question *What are the beverages whose origin is England?* having the SPARQL query

```
SELECT DISTINCT ?uri WHERE {?uri <http://dbpedia.org/ontology/
origin> <http://dbpedia.org/resource/England>},
```

which outputs everything that originated in England, not only beverages. We did not check all 5000 questions of the training set, but at least found 3 similar mistakes in the first 100 questions.

Correctly answered questions took 0.2–1 seconds in general. Answering questions with highly ambiguous elements could take up to 45 seconds. About 10 seconds was needed each time when the system failed to produce an answer.

On the whole, testing proved our assumption that the main shortcoming of the GQA system is its focus on the controlled language, which could be improved by some methods that tolerate broader variation of question wording, e.g. lexical matching of DBpedia properties and question tokens excluding entity names, which was implemented in [21]. At the same time, the grammatical approach can be quite reliable in defining the structure of a question, especially if it is syntactically complex.

6 Related Work

Numerous attempts have been made to develop question-answering systems involving various KBs and techniques: learning subgraph embeddings over Freebase, WikiAnswers, ClueWeb and WebQuestions [4], viewing a KB as a collection of topic graphs [29], learning question paraphrases over ReVerb [12], using domain-specific lexicon and grammar [27], automated utterance-query template generation [1], using multi-column convolutional neural networks [10], etc. A number of efforts were made to develop semantic parsing-based QA systems [3, 11].

Our idea of employing GF in question answering was inspired by the systems squall2sparql [13] and CANaLI [18], which got the highest scores in answering English language questions in the multilingual tasks of QALD-3 and QALD-6 [19] respectively. These systems accept questions formulated in a controlled natural language, which reduces ambiguity and increases the accuracy of answers. However, this approach requires manual reformulation of questions into the format acceptable for the systems. We develop a more extensive controlled language, which tolerates various question formulations, making the question answering process totally automatic.

GF was successfully applied in the QALD-4 question answering challenge for Biomedical interlinked data [17]. Van Grondelle and Unger [28] exploited GF in their paradigm for conceptually scoped language technology, reusing technology components and lexical resources on the web and facilitating their low impact adoption. GF was also used to convert natural language questions to SPARQL in the English [6] and Romanian cultural heritage domain [16].

7 Conclusion

GQA is an attempt to use the grammatical approach in resolving mostly complex, syntactically rich questions and converting them into conceptual parses, which then can be mapped on to DBpedia components. The approach involving controlled natural language, on the one hand, increases the probability of understanding questions accurately and retrieving correct answers. On the other hand, one would never foresee all possible formulations of natural language questions, even having large training sets at one's disposal. Nevertheless, our approach requires no manual reformulation of individual questions, unlike some other controlled language systems. In the future we plan to develop additional methods to provide better semantic analysis. Moreover, the system needs more sophisticated entity linking methods, spell checking techniques, and the implementation needs to be improved to utilise parallel computing in a multi-server, multi-processor environment. In addition, we are planning to exploit GF's multilinguality facilities and extend the system by adding new languages.

References

1. Abujabal, A., Yahya, M., Riedewald, M., Weikum, G.: Automated template generation for question answering over knowledge graphs. In: WWW'17, pp. 1191–1200. New York (2017).
2. Apache Lucene, http://lucene.apache.org/. Last accessed 15 Jun 2018.
3. Berant, J., Chou, A., Frostig, R., Liang, P.: Semantic parsing on Freebase from question-answer pairs. In: EMNLP 2013, pp. 1533–1544 (2013).
4. Bordes, A., Chopra, S., Weston, J.: Question answering with subgraph embeddings. In: Computer Science, pp. 615–620 (2014).
5. Bringert, B., Hallgren, T., Ranta, A.: GF Resource Grammar Library: Synopsis, http://www.grammaticalframework.org/lib/doc/synopsis.html. Last accessed 15 Jun 2018.
6. Damova, M., Dannells, D., Enache, R., Mateva, M., Ranta, A.: Natural Language Interaction with Semantic Web Knowledge Bases and LOD. In: Towards the Multilingual Semantic Web, Paul Buitelaar and Philip Cimiano, eds. Springer (2013).
7. DBpediaSpotlight, https://www.dbpedia-spotlight.org/. Last accessed 15 Jun 2018.
8. DBpedia version 2016-04, http://wiki.dbpedia.org/dbpedia-version-2016-04. Last accessed 15 Jun 2018.
9. Diefenbach, D., Singh, K., Maret, P.: WDAqua-core0: A Question Answering Component for the Research Community. In: Semantic Web Challenges, pp. 84–89. Springer International Publishing (2017).
10. Dong, L., Wei, F., Zhou, M., Xu, K.: Question answering over Freebase with multicolumn convolutional neural networks. In: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing, pp. 260–269 (2015).
11. Dubey, M., Dasgupta, S., Sharma, A., Höffner, K., Lehmann, J.: AskNow: A Framework for Natural Language Query Formalization in SPARQL. In: The Semantic Web. Latest Advances and New Domains, pp. 300–316 (2016).
12. Fader, A., Zettlemoyer, L., Etzioni, O. Paraphrase-driven learning for open question answering. In: Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, pp. 1608–1618 (2013).

13. Ferré, S.: squall2sparql: a translator from controlled English to full SPARQL 1.1. In: Work. Multilingual Question Answering over Linked Data (QALD-3). Valencia, Spain (2013).
14. FOX: Federated knOwledge eXtraction Framework, http://fox-demo.aksw.org/. Last accessed 15 Jun 2018.
15. LC-QuAD: A Corpus for Complex Question Answering over Knowledge Graphs, ISWC 2017, https://project-hobbit.eu/challenges/sqa-challenge-eswc-2018/. Last accessed 15 Jun 2018.
16. Marginean, A., Groza, A., Slavescu, R.R., Letia, I.A.: Romanian2SPARQL: A Grammatical Framework approach for querying Linked Data in Romanian language. In: International Conference on Development and Application Systems (2014).
17. Marginean, A.: Question answering over biomedical linked data with Grammatical Framework. Semantic Web, 8(4):565–580, 2017.
18. Mazzeo, G. M., Zaniolo, C.: Question Answering on RDF KBs using controlled natural language and semantic autocompletion. In: Semantic Web 1, pp. 1–5. IOS Press (2016).
19. Question Answering over Linked Data (QALD), https://qald.sebastianwalter.org/. Last accessed 15 Jun 2018.
20. Christophides, V., Efthymiou, V., Stefanidis, K.: Entity Resolution in the Web of Data, Synthesis Lectures on the Semantic Web: Theory and Technology, Morgan & Claypool Publishers, 2015.
21. Radoev, N., Tremblay, M., Gagnon, M., Zouaq, A.: AMAL: Answering French Natural Language Questions Using DBpedia. In: Semantic Web Challenges, pp. 90–105. Springer International Publishing (2017).
22. Ranta, A.: Grammatical Framework Tutorial, http://www.grammaticalframework.org/doc/tutorial/gf-tutorial.html. Last accessed 15 Jun 2018.
23. Ranta, A.: Grammatical Framework: Programming with Multilingual Grammars. CSLI Publications, Stanford (2011).
24. Roy, S. B., Stefanidis, K., Koutrika, G., Lakshmanan, L. V. S., Riedewald, M.: Report on the Third International Workshop on Exploratory Search in Databases and the Web (ExploreDB 2016). SIGMOD Record, 45(3), pp. 35–38 (2016).
25. Stefanidis, K., Fundulaki, I.: Keyword Search on RDF Graphs: It Is More Than Just Searching for Keywords. In: ESWC, 2015.
26. SPARQL 1.1 Overview – W3C, http://www.w3.org/TR/sparql11-overview/. Last accessed 15 Jun 2018.
27. Unger, C., Cimiano, P.: Pythia: compositional meaning construction for ontology-based question answering on the Semantic Web. In: Natural Language Processing and Information Systems, pp. 153–160. Springer Berlin Heidelberg (2011).
28. Van Grondelle, J., Unger, C. A three-dimensional paradigm for conceptually scoped language technology. In: Towards the Multilingual Semantic Web, pp. 67–82. Springer Berlin Heidelberg (2014).
29. Yao, X., Van Durme, B.: Information extraction over structured data: Question answering with Freebase. In: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, pp. 956–966 (2014).