Multilingual Question Answering over Linked Data building on a model of the lexicon-ontology interface

Mohammad Fazleh Elahi[a,*] Basil Ell[a,c] Gennaro Nolano[b] Philipp Cimiano[a]

[a] *Cognitive Interaction Technology Center, Universität Bielefeld, Bielefeld, Germany*
E-mails: melahi@techfak.uni-bielefeld.de, cimiano@techfak.uni-bielefeld.de, bell@techfak.uni-bielefeld.de, basile@ifi.uio.no
[b] *UniOr NLP Research Group, University of Naples "L'Orientale", Naples, Italy*
E-mail: nolanogenn@gmail.com
[c] *Department of Informatics, University of Oslo, Norway*
E-mail: basile@ifi.uio.no

Abstract. Multilingual Question Answering (QA) has the potential to make the knowledge available as linked data accessible across languages. Current state-of-the-art QA systems for linked data are nevertheless typically monolingual, supporting in most cases only the English language. As most state-of-the-art systems are based on machine learning techniques, porting such systems to new languages requires a training set for every language that should be supported. Furthermore, most recent QA systems based on machine learning methods lack controllability and extendability, thus making the governance and incremental improvement of these systems challenging, not to mention the initial effort of collecting and providing training data. Towards the development of QA systems that can be ported across languages in a principled manner without the need of training data and towards systems that can be incrementally adapted and improved after deployment, we follow a model-based approach to QA that supports the extension of the lexical and multilingual coverage of a system in a declarative manner. The approach builds on a declarative model of the lexicon ontology interface, *OntoLex lemon*, which enables the specification of the meaning of lexical entries with respect to the vocabulary of a particular dataset. From such a lexicon, in our approach, a QA grammar is automatically generated that can be used to parse questions into SPARQL queries. We show that this approach outperforms current QA approaches on the QALD benchmarks. Furthermore, we demonstrate the extensibility of the approach to different languages by adapting it to German, Italian, and Spanish. We evaluate the approach with respect to the QALD benchmarks on five editions (i.e., QALD-9, QALD-7, QALD-6, QALD-5, and QALD-3) and show that our approach outperforms the state-of-the-art on all these datasets in an incremental evaluation mode in which additional lexical entries for test data are added. For example, on QALD-9, our approach obtains F_1 scores of 0.85 (English), 0.82 (German), 0.65 (Italian), and 0.83 (Spanish) in an incremental evaluation mode in which lexical entries covering test data questions are added. So far there is no system described in the literature that works for at least four languages while reaching state-of-the-art performance on all of them. Finally, we demonstrate the low efforts necessary to port the system to a new dataset and vocabulary.

Keywords: Multilingual Question Answering over Linked Data, Model-based QA Approach, Lemon Lexica, Grammar Generation, Parsing

[*]Corresponding author. E-mail: melahi@techfak.uni-bielefeld.de.

1. Introduction

As the amount of available multilingual linked data keeps growing, intuitive and effective paradigms for accessing and querying this body of knowledge become increasingly important. Previous research has stressed the need to transform linked data into a “global database” in which information is accessible across languages, thus creating a level-playing field in which each language community has equal opportunities to contribute to and benefit from the growing linked data ecosystem [1].

There are approximately $7,000$ languages spoken worldwide and a large amount of multilingual linked data available. Yet, English predominates in research communities focused on question-answering systems over linked data (QALD) [2, 3]. Previous research [4, 5] has examined the availability of multilingual QA systems developed in the last 10 years. Furthermore, the applicability of the machine translation approach to monolingual QA systems [6] was investigated. The findings include that there are a few multilingual QA systems over linked data, but most of them can not be used at the moment due to the unavailability of either the online demo, API, or source code. While it is possible to adopt a QA system to any language using machine translation (MT), in most cases, it yields comparable performance (F_1-measure) only when the source language is translated into English. However, it requires the availability of a machine translation tool for all languages to English, which is not available for many of the world’s $7,000$ languages.

State-of-art approaches to QALD are machine learning-based and can be broadly grouped into three categories [7]: (i) classification-based approaches, where neural models are used to predict one of a fixed set of classes given a question; (ii) ranking-based approaches, where neural networks are used to compare different candidate logical forms with the question to select the best-ranked one; and (iii) translation-based approaches, where the network learns to translate natural language questions (NLQs) into their corresponding logical forms. Many neural network-based approaches have proven effective for the QALD task, ranging from simple neural embedding-based models [8] and attention-based recurrent models [9] to memory-augmented neural controller architectures [10–12].

Recently, transfer learning based QALD [13] have demonstrated that significant improvements can be achieved by fine-tuning a pre-trained language model trained on a large corpus. Yin et al. [14] used three types of neural machine translation models, including RNN-based, CNN-based, and Transformer-based models for translating natural language to SPARQL. Research by Wang et al. [15] has demonstrated that GNNs naturally fit the graph-structured knowledge and show promising results on the QALD task. The QA-GNN model [16], which jointly reasons over knowledge from pretrained language models and knowledge graphs using Graph Neural Networks (GNN), has been shown to yield strong question-answering (QA) performance compared to models that rely solely on Knowledge Graphs (KG) or Language Models (LM). While machine learning-based systems have seen increasing performance in recent years, they suffer from a number of limitations.

The first limitation is the lack of principled extensibility and controllability. If the QA system is to be extended to cover a particular type of question that has not been supported so far, current systems would require to add a number of additional training questions and to retrain the system with an uncertain outcome as there is no guarantee that the additional training questions are sufficient for the approach to learn the new question type. A more principled approach here would support adding a new syntactic construction to the system that will parse or generate all questions involving this new syntactic construction rather than only single instances of such questions. The same holds for an extension of a system with a new lexical element.

Adding new training questions involving the lexical element does not guarantee that the system is able to induce the semantics of that lexical element as a basis to parse all the questions involving this element. In this sense, current systems can not be extended to cover new types of questions in a principled way.

The second limitation is related to the difficulty of increasing the capability of these systems for a new language. The capability of the systems that follow an inductive paradigm can be increased for a new language either by adding training data for that language (which is costly to obtain) or by training systems in a cross-lingual fashion, e.g. transferring a system relying on cross-lingual embeddings [17] or from one language to another via training data projection [18] using machine translation systems. However, it has been demonstrated that such systems require at least some domain-specific data, in this case, in-language data, to perform well. For example, Klinger and Cimiano [19] propose an approach based on projecting training data from one language to another based on machine translation techniques [18]. They show that, given that the translation can introduce errors, the performance of the system on the target language depends on the quality of the translation system and the induced word alignment.

Another approach to multilingual question answering is based on the cross-lingual language model [20, 21] that partially overcomes the lack of a large amount of training data. The approach creates machine-learning models for languages by transferring knowledge from one natural language to another. For example, XLM-Roberta [20] uses self-supervised training techniques to achieve state-of-the-art performance in cross-lingual understanding, in which a model is trained in one language and then used with other languages without additional training data. The approach is deployed on comprehension datasets (i.e., SQuAD benchmark[1]), consisting of questions posed by crowd workers on a collection of Wikipedia articles. To the best of our knowledge, the approach has not been applied to linked data [22, 23], so it remains unclear how far such an approach addresses the challenges of question answering over a knowledge graph [2, 3] and what its performance would be.

We explore a different paradigm to question answering that relies on a model-based approach [24]. The approach relies on a model of the lexicon-ontology interface [25] in which the meaning of lexical entries is specified with respect to a given vocabulary or ontology. We build on the OntoLex-Lemon model[2] [25] in particular. Given a lexicon in a particular language and for a particular vocabulary/ontology, our approach automatically generates a grammar that allows parsing questions in the specific language into SPARQL queries over the corresponding vocabulary/ontology. This approach is model-based or declarative in the sense that it relies on a declarative specification of the meaning of lexical entries. The approach can be extended in a principled and efficient way as follows:

- Adding new lexical items: The approach to extending the QA system is principled and consistent in the sense that adding a lexical item to the lexicon guarantees that all questions involving valid constructions in which the lexical item can be used will be parsed correctly.
- Adding a new syntactic construction: when making available a new syntactic construction to the grammar generation component, then this syntactic construction will be available for the grammars generated for different datasets, so the syntactic construction needs to be added once for a language and it will be available for all datasets.
- Porting the system to a new language: porting a system to a new language requires creating a lexicon and developing sentence templates for that particular language. This is an effort done once for all the datasets that the QA system will work on.

The way to extend the system is thus principled in the sense that porting the system to a new dataset merely requires providing a new lexicon in the languages the system is expected to support. This can be done by engineers who do not need to be aware of how the grammar is generated but

[1]https://rajpurkar.github.io/SQuAD-explorer/
[2]https://www.w3.org/2019/09/lexicog/

only need to understand how to add appropriate lexical entries. The extension with new syntactic constructions and languages is done once by a language expert in the QA system and will be available for all instances of the QA system that would only need to re-run the grammar generation to incorporate the newly added syntactic constructions.

While we have presented preliminary versions of this work [24] before, we have so far not provided evidence of the robustness, controllability, scalability, adaptability, and portability of the system across languages in a systematic evaluation. As one important ingredient towards making information accessible across languages, we investigate how our model-based approach can be extended in terms of coverage in a principled way.

In the current paper, we describe our new approach in full detail and describe its architecture. We further demonstrated that the approach is fit for purpose and outperforms the existing state-of-the-art machine learning-based and rule-based approaches in an incremental evaluation for English on five editions of the QALD dataset, i.e., QALD-9, QALD-7, QALD-6, QALD-5, and QALD-3. We provide a proof for the portability of our system to further languages by describing how we adapted it to German, Spanish, and Italian and also show that it provides state-of-the-art results on all series of QALD including the newest version QALD-9 for these three languages. We also demonstrate that our approach outperforms pre-trained state-of-the-art Large Language Models (LLMs), particularly ChatGPT [26, 27], in an incremental evaluation, both in zero-shot and few-shot settings.

2. General Approach

In this section, we describe our approach to multilingual grammar generation on the basis of a lexicon represented using the OntoLex-Lemon model. For each syntactic frame supported by our system, we describe *generic templates* (i.e., grammar rule templates) for the grammar rules generated and provide examples of how these templates are instantiated for specific lexical entries. Our examples use DBpedia [28] as a reference dataset. However, our approach is dataset-independent and can be adapted to any particular dataset and vocabulary by providing a corresponding lexicon.

2.1. OntoLex-Lemon

In this section, we explain the OntoLex-Lemon model [25], which is an updated version of the Lemon model [29], and which is the core representation used by our model-based approach to grammar generation. The OntoLex-Lemon model is based around the core module, as depicted in Figure 1. The primary element of this is the *lexical entry*, which represents a single word and thus collects together all morphological variants of that word, which correspond to forms in the model, and all possible concepts in the given vocabulary/ontology it can refer to. The meaning of a word is thus given by reference to an ontological concept; this connection is modelled via a *lexical sense* that represents the mapping from a word to an ontological concept.

Words have a clear syntactic behaviour which to a great extent is determined by their syntactic category (verb, noun, adjective, etc.). The way that sentences are composed in a language crucially depends on the syntactic behaviour of the words that make up the sentence. The syntax and semantics module of the OntoLex-Lemon model describes the syntactic behaviour of words in terms of an inventory of syntactic (subcategorization) frames such as transitive verb-frame, intransitive verb-frame requiring a prepositional object etc. In addition, it describes how the syntactic structure relates to a semantic structure described in terms of vocabulary elements of the given vocabulary/ontology. There are other existing sources that comprise of a repository of frames. A relevant example is the Framester hub [31] which acts as a hub between FrameNet [32], WordNet [33], VerbNet [34],

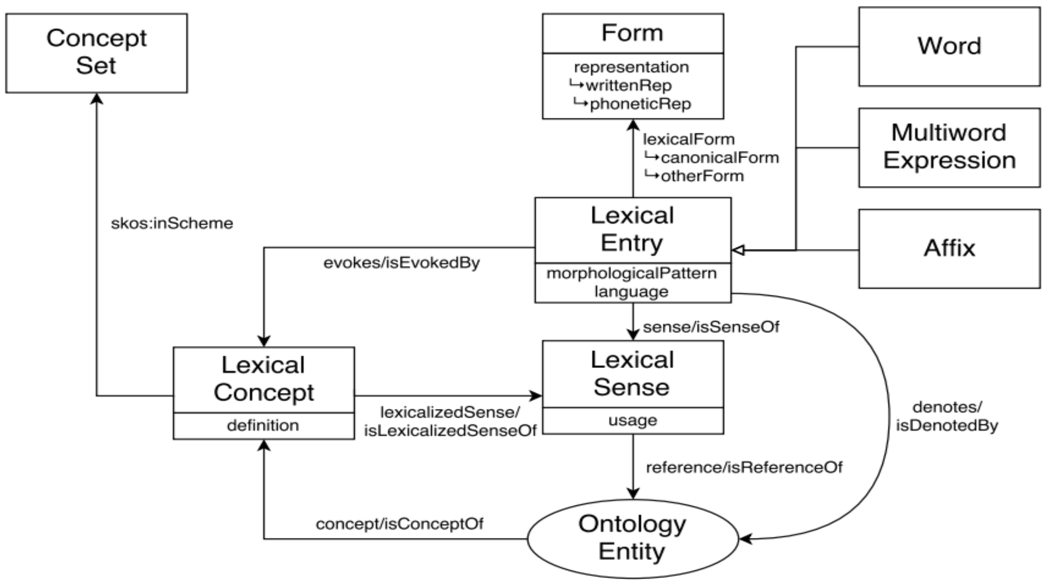


Fig. 1. The Core OntoLex-Lemon Model [25]

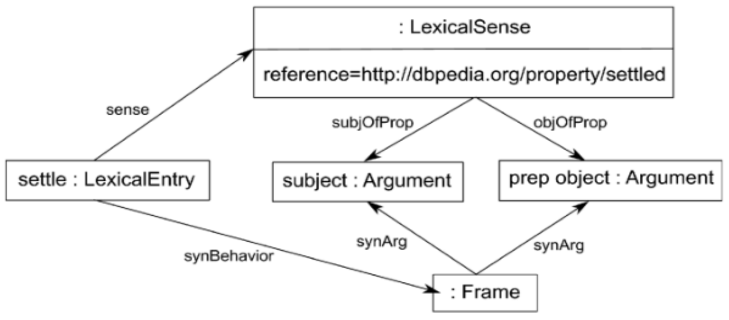


Fig. 2. The Syntax/Semantics mapping in LexInfo [30]

BabelNet[3], DBpedia, Yago, DOLCE-Zero, as well as other resources, while encoding frame-related and lexical knowledge in OWL to support expressive querying and reasoning. In contrast to such resources, the frames we consider in this paper are mainly syntactically defined, getting semantics from a relation to a specific vocabulary or ontology.

The core of the lemon model is insufficient for grammar generation to parse natural language sentences into SPARQL queries. This is because it lacks syntactic information and arguments, such as the specific types of arguments needed for certain lexical entries (e.g., direct and indirect objects), their grammatical functions, or the type of a frame (e.g., intransitive, transitive). `Lexinfo` [30] is an extension of the lemon model, featuring over 600 distinct linguistic categories. This declarative model allows us to associate linguistic information with elements in an ontology with respect to the syntactic frame (as detailed in Section 2.2), which is crucial for constructing grammatically

[3]https://babelnet.org/

correct sentences for that element. For example, Figure 2 shows a syntactic frame of the so-called `InTransitivePPFrame`, which consists of an intransitive verb and a prepositional adjunct.

Consider the intransitive verb with prepositional adjunct *'settle in'* construction: *'`subject` settled in `object`'* and the meaning of which can be captured by the DBpedia property *dbp:settled*, where the `subject` is an instance of the domain of the property and the `object` is an instance of the range of the property. In the general case, however, it will not always be the case that the syntactic subject maps to the domain and the syntactic object to the range of a property. Consider the transitive verb *'(to) write'* with a construction: *'`subject` wrote `object`.'* In this case, *'wrote'* corresponds to the property *dbp:author*, the `subject` corresponds to the range, and the `object` corresponds to the domain of the property. The mapping between syntactic and semantic arguments needs thus to be specified explicitly for each syntactic construction. This can be done via the *syntax and semantics* module of the OntoLex-Lemon model. This is schematically depicted in Figure 2.

2.2. Overall Architecture

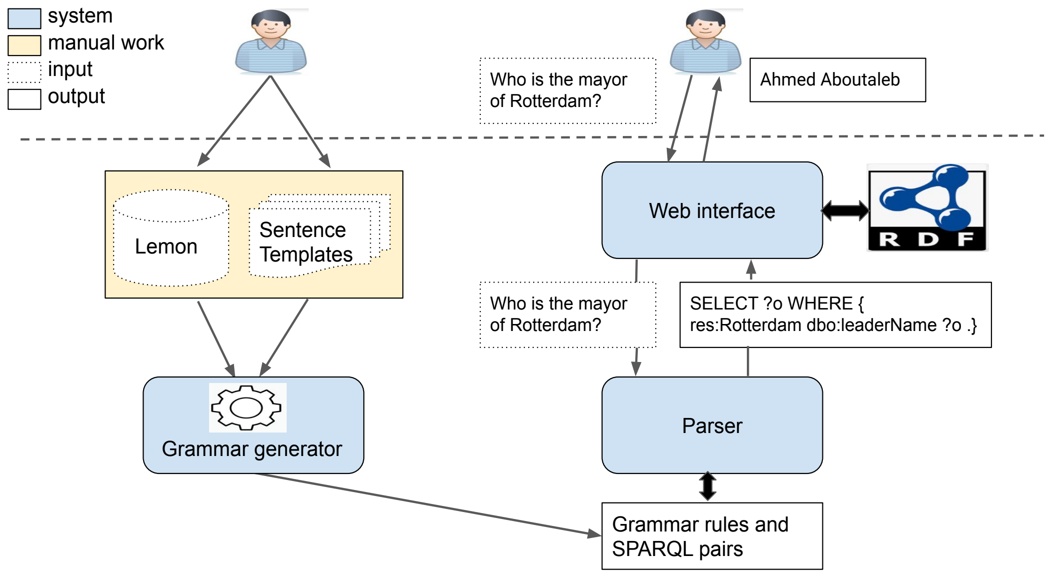

system
manual work
input
output
Who is the mayor of Rotterdam?
Ahmed Aboutaleb
Lemon
Sentence Templates
Web interface
RDF
Who is the mayor of Rotterdam?
SELECT ?o WHERE { res:Rotterdam dbo:leaderName ?o .}
Grammar generator
Parser
Grammar rules and SPARQL pairs

Fig. 3. High-level architecture of the model-based approach

Figure 3 shows a high-level architecture of our model-based approach. The core component of the approach is the grammar generator, which takes a Lemon lexicon and a set of sentence templates (detailed in Section 3.1) as input, accesses an RDF dataset and automatically creates a lexicalized grammar. The grammar is lexicalized in the sense that every grammar rule has a lexical anchor as in Lexicalized Tree Adjoining Grammars (LTAG) [35, 36]. As in LTAG, our rules correspond to full projections of lexical anchors in that all (dependent) arguments are local to each rule.

A necessary prerequisite for our grammar generation approach is the availability of a Lemon lexicon that describes by which lexical entries the elements (classes, properties) of a given dataset can be verbalized in a particular language. In particular, a lexicon is needed for each language to be supported by the QA system. The grammar generation for a lexical entry with a specific syntactic frame is controlled by grammar rule templates (i.e., generic templates) that describe how specific lexicalized grammar rules can be generated for a given lexical entry. We support the following syntactic frames[4]:

- `NounPPFrame`: corresponding to a relational noun that requires a prepositional object such as *'mayor' (of)*, *'capital' (of)*, etc.
- `TransitiveFrame`: correspoding to transitive verbs such as *(to) 'direct'*, *(to) 'marry'*, etc.
- `InTransitivePPFrame`: corresponding to intransitive verbs subcategorizing a prepositional phrase such as *'flow' (through)*,*'born' (on)*, etc.
- `AdjectivePredicateFrame`: covering intersective adjectives such as *'Spanish'*, *'Afghan'*, etc. This frame is used for both attributive and predicative use of the adjective.
- `AdjectiveSuperlativeFrame`: covering gradable adjectives such as *'high'*, *'highest'*, etc.

In the following sections, we first describe the lexical entries (as detailed in Section 2.3) for each of these syntactic frames. Afterwards, we will explain the sentence templates (as detailed in Section 2.4), as they serve as a prerequisite for obtaining the grammar rule templates. Finally, we describe in detail the grammar rule templates (see Section 2.5) and give an instantiated example for a specific lexical entry.

2.3. Creating Lexical Entries

To create a QA system for a given dataset with its associated vocabulary and a specific language, the language expert of that language needs to create lexical entries. In this section, we provide examples of lexical entries for English as language and DBpedia as vocabulary/ontology.

2.3.1. Nouns

Consider the Lemon lexical entry shown in Figure 4 for the relational noun *'mayor' (of)*. The entry states that the canonical written form of the entry is *'mayor'*. It specifies that the syntactic behaviour of the entry corresponds to the one of a `NounPPFrame` in LexInfo. This means that the lexical entry can be used in a copulative construction *'X is the mayor of Y'* with two arguments, where `copulativeArg` corresponds to the copula subject X and the `prepositionalAdjunct` corresponds to the prepositional object Y. The semantics of the relational noun is captured with respect to the property *dbo:mayor*[5], where the subject of the property is realized by the `prepositionalAdjunct` and the object of the property is realized by the `copulativeArg`. The entry also captures that the domain (i.e., the class specified via the property *rdfs:domain*) of the property is *dbo:City* in the context of the lexical entry. Correspondingly, the range (i.e., the class that is specified via the property *rdfs:range*) of the property is *dbo:Person*.

2.3.2. Verbs

Consider the Lemon lexical entry shown in Figure 5 for the transitive verb *(to) 'direct.'* The lexical entry states that the canonical form has the written representation *'direct'*. The simple past form is *'directed'*. The semantics of the verb *(to) 'direct'* is expressed by the property *dbo:director*. The entry specifies that the subject of the property is realized by the direct object of the verb *'direct'* while the object of the property is realized by the syntactic subject of the verb *'direct'*. The entry also captures that the domain of the property is *dbo:Film* and the range of the property is *dbo:Person*.

An example of an intransitive verb with a prepositional adjunct, *'flow' (through)*, is shown in Figure 6. According to the Lemon entry, the verb has a subject and a prepositional adjunct. The seman-

[4]http://www.lexinfo.net/ontology/3.0/lexinfo

[5]@prefix `dbo:` <http://dbpedia.org/ontology/>, @prefix `dbp:` <http://dbpedia.org/property/>,
@prefix `res:` <http://dbpedia.org/resource/>, `rdf:` <http://www.w3.org/1999/02/22-rdf-syntax-ns#>, and
`rdfs:` <http://www.w3.org/2000/01/rdf-schema#>

```
@prefix : <http://localhost:8080/lexicon#> .
@prefix lexinfo: <http://www.lexinfo.net/ontology/2.0/lexinfo#> .
@prefix lemon: <http://lemon-model.net/lemon#> .
@base <http://localhost:8080#> .

:lexicon_en a lemon:Lexicon ;
  lemon:language "en" ;
  lemon:entry :mayor_of ;
  lemon:entry :of .

:mayor_of a lemon:LexicalEntry ;
  lexinfo:partOfSpeech lexinfo:noun ;
  lemon:canonicalForm :mayor_form_1 ;
  lemon:canonicalForm :mayor_form_2 ;
  lemon:synBehavior :mayor_of_nounpp ;
  lemon:sense :mayor_sense_ontomap_1 ;
  lemon:sense :mayor_sense_ontomap_2 .

:mayor_form_1 a lemon:Form ;
  lemon:writtenRep "mayor"@en .

 :mayor_form_2 a lemon:Form ;
  lemon:writtenRep "mayors"@en .

:mayor_of_nounpp a lexinfo:NounPPFrame ;
  lexinfo:copulativeArg :arg1 ;
  lexinfo:prepositionalAdjunct :arg2 .

:mayor_sense_ontomap_1 a lemon:OntoMap, lemon:LexicalSense ;
  lemon:ontoMapping :mayor_sense_ontomap_1 ;
  lemon:reference <http://dbpedia.org/ontology/mayor> ;
  lemon:subjOfProp :arg2 ;
  lemon:objOfProp :arg1 ;
  lemon:condition :mayor_condition .

:mayor_sense_ontomap_2 a lemon:OntoMap, lemon:LexicalSense ;
  lemon:ontoMapping :mayor_sense_ontomap_2 ;
  lemon:reference <http://dbpedia.org/ontology/leaderName> ;
  lemon:subjOfProp :arg2 ;
  lemon:objOfProp :arg1 ;
  lemon:condition :mayor_condition .

:mayor_condition a lemon:condition ;
  lemon:propertyDomain <http://dbpedia.org/ontology/City> ;
  lemon:propertyRange <http://dbpedia.org/ontology/Person> .

:arg2 lemon:marker :of .

:of a lemon:SynRoleMarker ;
  lemon:canonicalForm [ lemon:writtenRep "of"@en ] ;
  lexinfo:partOfSpeech lexinfo:preposition .
```

Fig. 4. The lexical entry for the relational noun *'mayor' (of)*.

```
:direct_frame_transitive a lexinfo:TransitiveFrame ;
  lexinfo:subject :direct_subj ;
  lexinfo:directObject :direct_obj .

:direct_ontomap a lemon:OntoMap, lemon:LexicalSense ;
  lemon:ontoMapping :direct_ontomap ;
  lemon:reference <http://dbpedia.org/ontology/director> ;
  lemon:subjOfProp :direct_obj ;
  lemon:objOfProp :direct_subj ;
  lemon:condition :direct_condition .

:direct_condition a lemon:condition ;
  lemon:propertyDomain <http://dbpedia.org/ontology/Film> ;
  lemon:propertyRange <http://dbpedia.org/ontology/Person> .
```

Fig. 5. The lexical entry for the transitive verb *(to) 'direct'*.

tics of *'X flows through Y'* is captured by the property *dbo:city*, where the subject of the property is realized by the syntactic subject of the *'flow' (through)* verb, and the object of the property is realized by the prepositional adjunct of the verbal construction. The entry states that the domain of the property is *dbo:River* and the range of the property is *dbo:City*.

2.3.3. Adjectives

```
:flow_frame a lexinfo:IntransitivePPFrame ;
 lexinfo:subject :flow_subj ;
 lexinfo:prepositionalAdjunct :flow_pobj .

:flow_sense1 a lemon:OntoMap, lemon:LexicalSense ;
 lemon:ontoMapping :flow_sense1 ;
 lemon:reference <http://dbpedia.org/ontology/city> ;
 lemon:subjOfProp :flow_subj ;
 lemon:objOfProp :flow_pobj ;
 lemon:condition :flow_condition_city .

:flow_condition_city a lemon:condition ;
 lemon:propertyDomain <http://dbpedia.org/ontology/River> ;
 lemon:propertyRange <http://dbpedia.org/ontology/City> .
```

Fig. 6. The lexical entry for the intransitive verb *‘flow’ (through)*.

```
:lexicon_en a lemon:Lexicon ;
 lemon:language "en" ;
 lemon:entry :spanish ;
 lemon:entry :spanish_res .

:spanish a lemon:LexicalEntry ;
 lexinfo:partOfSpeech lexinfo:adjective ;
 lemon:canonicalForm :spanish_lemma ;
 lemon:synBehavior :spanish_attrFrame, :spanish_predFrame ;
 lemon:sense :spanish_sense .

:spanish_lemma lemon:writtenRep "Spanish"@en .

:spanish_predFrame a lexinfo:AdjectivePredicateFrame ;
 lexinfo:copulativeSubject :spanish_PredSynArg .

:spanish_attrFrame a lexinfo:AdjectiveAttributiveFrame ;
 lexinfo:attributiveArg :spanish_AttrSynArg .

:spanish_sense a lemon:LexicalSense ;
 lemon:reference :spanish_res ;
 lemon:isA :spanish_AttrSynArg, :spanish_PredSynArg .

:spanish_res a owl:Restriction ;
 owl:onProperty <http://dbpedia.org/ontology/country> ;
 owl:hasValue <http://dbpedia.org/resource/Spain> .
```

Fig. 7. The lexical entry for the adjective *‘Spanish’*.

Adjectives have different behaviour in terms of grammar rules generated compared to the entries discussed before. Consider the example entry for the adjective *‘Spanish’* shown in Figure 7. The entry states that the adjective can be used in an attributive frame (i.e., *‘Spanish movie’*) as well as in a predicative frame (i.e., *‘the movie is Spanish’*). The entry states that the semantics of the adjective *‘Spanish’* can be expressed through a restriction on property *dbo:country* for the value *res:Spain*.

Gradable adjectives are represented using the class `AdjectiveSuperlativeFrame` in `LexInfo` and are modelled along the proposal of McCrae et al. [37] using the lemonOILS vocabulary[6] (the Lemon Ontology for the Interpretation of Lexical Semantics). The entry *‘high’*, shown in Figure 8, is expressed as an instance of the class *oils:CovariantScalar*, indicating that the adjective is covariant with its bound property *dbo:elevation* for the degree *oils:strong*, that is, the *‘higher’* something is, the larger the value of the property *dbo:elevation*.

2.4. Creation of Sentence Templates

The sentence templates are manually created by language experts, as indicated in the high-level diagram as the second step. A sentence template defines the structure of a sentence for a specific

[6]http://lemon-model.net/oils

```
:lexicon_en a lemon:Lexicon ;
  lemon:language "en" ;
  lemon:entry :high_1 ;
  lemon:entry :high_1_res .

:high_1 a lemon:LexicalEntry ;
  lexinfo:partOfSpeech lexinfo:adjective ;
  lemon:canonicalForm :form_high_1 ;
  lemon:otherForm :form_high_1_comperative ;
  lemon:otherForm :form_high_1_superlative ;
  lemon:sense :high_1_sense_1 ;
  lemon:synBehavior :high_1_predFrame .

:form_high_1 lemon:writtenRep "high"@en .
:form_high_1_comperative lemon:writtenRep "higher"@en .
:form_high_1_superlative lemon:writtenRep "highest"@en .

:high_1_predFrame a lexinfo:AdjectiveSuperlativeFrame ;
  lexinfo:copulativeSubject :high_1_PredSynArg .

:high_1_sense_1 a lemon:LexicalSense ;
  lemon:reference :high_1_res ;
  lemon:isA :high_1_PredSynArg ;
  lemon:condition :high_1_sense_1_condition .

:high_1_res a oils:CovariantScalar ;
  oils:boundTo <http://dbpedia.org/ontology/elevation> ;
  oils:degree <http://lemon-model.net/oils/high> .

:high_1_sense_1_condition a lemon:condition ;
  lemon:propertyDomain <http://dbpedia.org/ontology/Mountain> ;
  lemon:propertyRange <http://www.w3.org/2001/XMLSchema#double> .
```

Fig. 8. The lexical entry for the gradable adjective *'high'*.

language, including the order and type of its components, such as the subject, various types of verbs (i.e., main verbs, auxiliary verbs, imperative verbs, etc.), prepositions, objects, and modifiers.

We create sentence templates for each syntactic frame (detailed in Section 3.1.2), not for individual lexical entries. For example, for relative nouns (e.g., `NounPPFrame`) for English, the language expert is instructed to create sentence templates given sample of a relative noun (e.g., *'mayor of'*) and a dictionary with syntactic categories (i.e., *`InterrogativeDeterminer`* → *{'which'}*, *`InterrogativePronounForPerson`* → *{'who'}*, *`InterrogativePronounForThing`* → *{'what'}*, *`AuxVerb(be:present:singular)`* → *{'is'},... etc.*). The dictionary contains a list of words for each syntactic category. For example, in the dictionary, `InterrogativeDeterminer` is a syntactic category that refers to the word `'which'`. The detailed method of creating sentence templates is discussed in Section 3.1.2. Below are examples of sentence templates for relative nouns created by the language expert for English.

sentence templates (ST$_{subject}$):
ST$_{q1}$: InterrogativePronounForPerson AuxVerb(be:present:singular) Determiner(the) RelativeNounSingular Preposition PrepositionalAdjunct?
($q1$: 'Who is the mayor of Paris?')
ST$_{q2}$: InterrogativePronounForPerson AuxVerb(be:past:singular) Determiner(the) RelativeNounSingular Preposition PrepositionalAdjunct?
($q2$: 'Who was the mayor of Paris?')
ST$_{q3}$: ImperativeVerb(give) pronoun(me) Determiner(the) RelativeNounSingular Preposition PrepositionalAdjunct?.
($q3$: 'Give me the mayor of Paris.')
ST$_{q4}$: ImperativeVerb(give) pronoun(me) Determiner(the) RelativeNounPlural Preposition PrepositionalAdjunct?.
($q4$: 'Give me the mayors of Paris.')
ST$_{q5}$: ImperativeVerb(list) Determiner(all) Determiner(the) RelativeNounPlural Preposition PrepositionalAdjunct?.
($q5$:'List all the mayors of Paris.')

Here, the sentence template `InterrogativePronounForPerson AuxVerb(be:present:singular) Determiner(the) RelativeNounSingular Preposition PrepositionalAdjunct?` presents the structure for the question ‘`Who is the mayor of Paris?`’. These five sentence templates describe the structure for questions containing a relative noun. Therefore, these templates (i.e., ST_{q1}, ST_{q2}, ST_{q3}, ST_{q4}, and ST_{q2}) are grouped under $\text{ST}_{subject}$, as they all aim to represent questions seeking information about the subject of the sentence.

2.5. Generating Grammars

In our previous work, we described how grammar generation [24] works for four syntactic frames according to LexInfo [30]. In our current work, we comprehensively describe the grammar generation process that builds on and substitutes the previous work and describe the methodology of porting it to other languages and datasets.

This section explores the transformation process of sentence templates (as discussed in the previous section) into grammar rule templates. In this paper, we use *ST* as the abbreviation for the sentence template, *GRT* as the abbreviation for the grammar rule template, and *GR* for the grammar rule.

When a sentence template (ST) is transformed into a grammar rule template (GRT), three conceptually different types of replacements take place, each serving a distinct purpose. For example, the sentence template ST_{q1}: `InterrogativePronounForPerson AuxVerb(be:present:singular) Determiner(the) RelativeNounSingular Preposition PrepositionalAdjunct?` presents the structure for the question $q1$: *‘Who is the mayor of Paris?’*. The template is converted to a grammar rule template by following three steps:

1. Transformations related to the lexicon: During the process of transforming an ST into a GRT, the parts obtained from the lexical entry are replaced to generalize the resulting template and abstract away from the concrete lexical entry. For example, in the sentence template ST_{q1}, the syntactic categories `RelativeNounSingular` and `Preposition` refer to *‘mayor’* and *‘of’*, respectively. By wrapping these categories in square brackets (i.e., `[NounSingular]` and `[Preposition]`), these symbols can easily be distinguished from other symbols later.
2. Transformations related to the knowledge base: The sentence template outlines the structure of a sentence. Some syntactic categories in the sentence template represent the syntactic subject or object. In our approach, these are obtained from RDF data (i.e., subject or object of the property). Therefore, when transforming an ST into a GRT, those parts are replaced by non-terminal symbols enclosed in <> along with a mapping function (i.e., `SyntacticFunction`). The `SyntacticFunction` is mapped to either the subject or object of the property. For example, in the sentence template ST_{q1}, the syntactic category `PrepositionalAdjunct` refers to *‘Paris’*, which is an entity that can be retrieved from RDF data. Therefore, when transforming the ST into a GRT, `PrepositionalAdjunct` is replaced by non-terminal (i.e., $<\texttt{NP}_{\texttt{map(SyntacticFunction), Property}}>$). When instantiating the GRT with a lexical entry (detailed in Section 2.5.1), `SyntacticFunction` is mapped to either `Domain` or `Range`.
3. Transformations related to language-specific parts: The language-specific components, such as determiners, pronouns, auxiliary verbs, conjunctions, and negation, neither fall under pre-terminals retrieved from the lexicon nor non-terminals retrieved from RDF data. For example, the syntactic category `InterrogativePronounForPerson` refers to *‘who’*, which is a language-specific component independent of the lexicon and RDF data. The purpose of this step is to transform the pre-GRT into an actual GRT that incorporates lexical elements for language-specific components.

The step-by-step process of transformation from a sentence template (ST) via pre-GRTs to a grammar rule template (GRT) is shown below:

ST$_{q1}$: InterrogativePronounForPerson AuxVerb(be:present:singular) Determiner(the) RelativeNounSingular Preposition PrepositionalAdjunct?

Step 1: transformations related to lexicon:
(pre-GRT) S -> InterrogativePronounForPerson AuxVerb(be:present:singular) Determiner(the) **[NounSingular] [Preposition]** PrepositionalAdjunct?

Step 2: transformations related to the knowledge base:
(pre-GRT) S -> InterrogativePronounForPerson AuxVerb(be:present:singular) Determiner(the) [NounSingular] [Preposition] **<NP$_{\mathbf{map(SyntacticFunction), Property}}$>**?

Step3: Transformations related to language-specific parts:
GRT$_{q1}$: S -> **Who is the** [NounSingular] [Preposition] <NP$_{\text{map(SyntacticFunction), Property}}$>?

In the following section, we describe the grammar at two levels of abstraction: (i) we describe generic templates (i.e., grammar rule templates) that are independent of a particular lexical entry, and (ii) we provide example instantiations (i.e., grammar rules) of the grammar rule templates for illustration. We also provide the SPARQL queries to retrieve the value of non-terminals of the grammar rules. The SPARQL queries of the individual grammar rules are provided in the Appendix.

2.5.1. NounPPFrame

From the sentence templates of the relative nouns (i.e., syntactic behaviour called NounPPFrame in LexInfo), our approach obtains ten grammar rule templates presented into five groups (such as GRT$_{\text{subject}}$, GRT$_{\text{object}}$, GRT$_{\text{amount}}$, GRT$_{\text{ask}}$, GRT$_{\text{nounPhrase}}$). For example, the GRT$_{\text{subject}}$ presents the grammar rule templates obtained by transforming sentence templates of group ST$_{\text{subject}}$ (as detailed in Section 2.4). The ST$_{\text{subject}}$ presents the structure for questions seeking information about the subject position.

As discussed in the previous section, symbols enclosed in <> represent non-terminals. The non-terminal <NP$_{\text{map(SyntacticFunction), Property}}$> can be substituted by each label of a URI that appears as the subject (resp. object) of the property Property depending on whether the given syntactic function (Copulative Subject, Prepositional Adjunct, etc.) maps to the domain resp. range of the property. Hereby, map(SyntacticFunction) specifies the semantic argument of the property (domain or range) that the given syntactic function maps to.

Symbols enclosed in [] represent proto-terminals that are to be replaced with content specific to the lexical entry, e.g., [NounSingular] and [NounPlural] refer to the inflectional forms of the lexical entry. [Preposition] refers to the preposition of the lexical entry.

Grammar Rule Templates (GRT$_{subject}$):
S -> Who is the [NounSingular] [Preposition] <NP$_{\text{map(SyntacticFunction), Property}}$>? | Who was the [NounSingular] [Preposition] <NP$_{\text{map(SyntacticFunction), Property}}$>? | Which <NP$_{\text{Class, <map(SyntacticFunction),Property>}}$> is the [NounSingular] [Preposition] <NP$_{\text{map(SyntacticFunction), Property}}$>? | Which <NP$_{\text{Class, <map(SyntacticFunction),Property>}}$> was the [NounSingular] [Preposition] <NP$_{\text{map(SyntacticFunction), Property}}$>? | Give me the [NounSingular] [Preposition] <NP$_{\text{map(SyntacticFunction), Property}}$>. | Give me all [NounPlural] [Preposition] <NP$_{\text{map(SyntacticFunction), Property}}$>. | List all [NounPlural] [Preposition] <NP$_{\text{map(SyntacticFunction), Property}}$>.
GRT$_{object}$:

S -> <NP$_{\text{map(SyntacticFunction), Property}}$> is the [NounSingular] [Preposition] which
    <NP$_{\text{Class, <map(SyntacticFunction),Property>}}$>? | <NP$_{\text{map(SyntacticFunction), Property}}$> was the
    [NounSingular)] [Preposition] which <NP$_{\text{Class, <map(SyntacticFunction),Property>}}$>?
GRT$_{amount}$:
S -> How many [NounPlural] does <NP$_{\text{map(SyntacticFunction), Property}}$> have?
GRT$_{ask}$:
S -> Is <NP$_{\text{map(SyntacticFunction), Property}}$> the [NounSingular] [Preposition]
    <NP$_{\text{map(SyntacticFunction), Property}}$>? | Was <NP$_{\text{map(SyntacticFunction), Property}}$> the [NounSingular]
    [Preposition] <NP$_{\text{map(SyntacticFunction), Property}}$>?
GRT$_{nounPhrase}$:
S -> the [NounSingular] [Preposition] <NP$_{\text{map(SyntacticFunction), Property}}$>?

Here, the first group (i.e., GRT$_{\text{subject}}$) consists of five grammar rule templates, the second group (i.e., GRT$_{\text{object}}$) consists of two grammar rule templates, the third group (i.e., GRT$_{\text{amount}}$) consists of one grammar rule template, the fourth group (i.e., GRT$_{\text{ask}}$) consists of two grammar rule templates, and finally, GRT$_{\text{nounPhrase}}$ consists of one grammar rule template.

The <NP$_{\text{map(SyntacticFunction), Property}}$> is instantiated either to <NP$_{\text{Domain, Property}}$> or <NP$_{\text{Range, Property}}$>, depending on whether map(SyntacticFunction) is Domain or Range. Similarly, the <NP$_{\text{Class, <map(SyntacticFunction),Property>}}$> is instantiated either to <NP$_{\text{Class, Domain,Property>}}$> or <NP$_{\text{Class, Range,Property>}}$>. Therefore, the grammar rule templates with the non-terminal symbols <NP$_{\text{map(SyntacticFunction), Property}}$> on the right are pre-terminal rules and are generated by retrieving the corresponding labels of URIs using the following SPARQL queries:

<NP$_{\text{Domain, Property}}$> -> eval(SELECT ?label WHERE {?Domain Property ?Range .
                                        ?Domain rdfs:label ?label . })
<NP$_{\text{Range, Property}}$> -> eval(SELECT ?label WHERE {?Domain Property ?Range .
                                        ?Range rdfs:label ?label . })
<NP$_{\text{Class, <Domain,Property>}}$> -> eval(SELECT ?label WHERE {?Domain Property ?Range.
                                        ?Domain rdf:type ?Class.
                                        ?Class rdfs:label ?label })
<NP$_{\text{Class, <Range,Property>}}$> -> eval(SELECT ?label WHERE {?Domain Property ?Range.
                                        ?Range rdf:type ?Class.
                                        ?Class rdfs:label ?label })

Here, `eval(query)` returns a list of the bindings of the query separated by the '|' symbol that separates grammar rules for the same pre-terminal symbol on the left.

Consider a particular lexical entry shown in Figure 4 for the relational noun *'mayor' (of)*. [NounSingular] refers to the singular form of the noun (i.e., *'mayor'*) and [NounPlural] refers to the plural form of the noun (i.e.,*'mayors'*). The instantiation of the above grammar rule templates (GRTs) for the lexical entry *'mayor' (of)* is shown below:

Grammar Rules from GRT$_{subject}$:
S -> Who is the mayor of <NP$_{\text{Domain, dbo:mayor}}$>? | Who was the mayor of <NP$_{\text{Domain, dbo:mayor}}$>?|
    Which <NP$_{\text{Class, <Range,dbo:mayor>}}$> is the mayor of <NP$_{\text{Domain, dbo:mayor}}$>? | Which
    <NP$_{\text{Class, <Range,dbo:mayor>}}$> was the mayor of <NP$_{\text{Domain, dbo:mayor}}$>? | Give me the mayor of
    <NP$_{\text{Domain, dbo:mayor}}$>. | Give me all mayors of <NP$_{\text{Domain, dbo:mayor}}$>. | List all mayors
    of <NP$_{\text{Domain, dbo:mayor}}$>.
Grammar Rules from GRT$_{object}$:
S -> <NP$_{\text{Range, dbo:mayor}}$> is the mayor of which <NP$_{\text{Class, <Domain,dbo:mayor>}}$>? | <NP$_{\text{Range, dbo:mayor}}$>
    was the mayor of which <NP$_{\text{Class, <Domain,dbo:mayor>}}$>?

Grammar Rules from GRT_{amount}:
S -> How many mayors does <$NP_{\text{Range, dbo:mayor}}$> have?
Grammar Rules from GRT_{ask}:
S -> Is <$NP_{\text{Range, dbo:mayor}}$> the mayor of <$NP_{\text{Domain, dbo:mayor}}$>? | Was <$NP_{\text{Range, dbo:mayor}}$> the mayor of <$NP_{\text{Domain, dbo:mayor}}$>?
Grammar Rules from $GRT_{nounPhrase}$:
S -> the mayor of <$NP_{\text{Domain, dbo:mayor}}$>?

The values of the non-terminals <$NP_{\text{Domain, dbo:mayor}}$> and <$NP_{\text{Range, dbo:mayor}}$> are retrieved with the following automatically generated SPARQL queries:

```
<NP_Domain, dbo:mayor> -> eval(SELECT ?label WHERE {?Domain dbo:mayor ?object.
                                                ?Domain rdfs:label ?label . })
<NP_Range, dbo:mayor>  -> eval(SELECT ?label WHERE {?subject dbo:mayor ?Range.
                                                ?Range rdfs:label ?label . })

<NP_Class, <Domain,dbo:mayor>> -> eval(SELECT ?label WHERE {?Domain dbo:mayor ?Range.
                                                ?Domain rdf:type ?Class.
                                                ?Class rdfs:label ?label })
<NP_Class, <Range,dbo:mayor>> -> eval(SELECT ?label WHERE {?Domain dbo:mayor ?Range.
                                                ?Range rdf:type ?Class.
                                                ?Class rdfs:label ?label })
```

After executing the SPARQL query, we obtain the following results:

```
<NP_Domain, dbo:mayor> -> Paris | Nice | ...
<NP_Range, dbo:mayor>  -> Anne Hidalgo | Christian Estrosi | ...
<NP_Class, <Domain, dbo:mayor>> -> city | cities ..
<NP_Class, <Range, dbo:mayor>> person | persons ..
```

The first, second, third, and fourth groups ($GRT_{subject}$, GRT_{object}, GRT_{amount}, and GRT_{ask}) contain grammar rule templates of type SENTENCE; that is, they generate a full question. The grammar rules generated by instantiating grammar rule templates of the first group (i.e., $GRT_{subject}$) using the lexical entry *'mayor' (of)* can parse a wide range of questions. For example, 1) *'Who is the mayor of Paris?'*, 2) *'Who was the mayor of Paris?'*, 3) *'Which person is the mayor of Paris?'*, 4) *'Which person was the mayor of Paris?'*, 5) *'Give me the mayor of Paris.'*, 6) *'Give me all mayors of Paris.'*, 7) *'List all mayors of Paris.'*.

The grammar rules generated from the second group (i.e., GRT_{object}) can parse the questions such as 1) *'Anne Hidalgo is the mayor of which city?'*, 2) *'Anne Hidalgo was the mayor of which city?'*, etc.

The grammar rules generated from the third group (i.e., GRT_{amount}) can parse the questions such as 1) *'How many mayors does Paris have?'*, etc. The grammar rule from the fourth template (i.e., GRT_{ask}) can parse boolean questions (i.e., yes–no questions), for example, *'Is Anne Hidalgo the mayor of Paris?'*. The grammar rules from the fifth template (i.e., Grammar Rules from GRT-5) are of type NounPhrase, for example, *'the mayor of Paris'*.

We have presented the grammar rule templates for the relative noun of Agent[7] type, such as *'mayor' (of)*, *'president' (of)*, etc. However, the grammar rule templates and grammar rules for the relative noun of non-Agent[8] type, such as *'capital' (of)*, *'ingredient' (of)*, are not detailed here.

[7]An agent noun refers to a person or entity that performs an action or is associated with a particular action
[8]A non-agent noun refers to something that is not related to or does not possess the characteristics of an agent.

2.5.2. TransitiveFrame

From the sentence templates of a transitive verb (i.e., the syntactic behaviour called `TransitiveFrame` in `LexInfo`), our approach obtains 15 grammar rule templates organized into five groups (such as $\mathrm{GRT}_{\mathrm{subject}}$, $\mathrm{GRT}_{\mathrm{object}}$, $\mathrm{GRT}_{\mathrm{amount}}$, $\mathrm{GRT}_{\mathrm{ask}}$, $\mathrm{GRT}_{\mathrm{nounPhrase}}$).

The non-terminal $<\mathrm{NP}_{\mathrm{map(SyntacticFunction),\ Property}}>$ can be substituted by all labels of URIs that appear as the subject (resp. object) of the property `Property` depending on whether the given syntactic function (`Subject`, `Object`, etc.) maps to the domain resp. range of the property. The lexicalized grammar rules templates are shown below.

```
Grammar Rule Template (GRT_subject):
S -> Who [Verb3rdPerson] <NP_map(SyntacticFunction), Property>? |
     Who [VerbPast] <NP_map(SyntacticFunction), Property>? |
     Which <NP_Class, <Range,Property>> [Verb3rdPerson] <NP_map(SyntacticFunction), Property>? |
     Which <NP_Class, <map(SyntacticFunction),Property>> [VerbPast] <NP_map(SyntacticFunction), Property>?
GRT_object:
S -> What is [VerbParticiple] [Preposition] <NP_map(SyntacticFunction), Property>? |
     What was [VerbParticiple] [Preposition <NP_map(SyntacticFunction), Property>? |
     Which <NP_Class, <map(SyntacticFunction),Property>> is [VerbParticiple] [Preposition]
     <NP_map(SyntacticFunction), Property> | Which <NP_Class, <map(SyntacticFunction),Property>> are
     [VerbParticiple] [Preposition] <NP_map(SyntacticFunction), Property>? | Give me all
     <NP_Class, <map(SyntacticFunction),Property>> [VerbParticiple] [Preposition]
     <NP_map(SyntacticFunction), Property>?
GRT_amount:
S -> How many <NP_Class, <map(SyntacticFunction),Property>> are [VerbPast] [Preposition]
     <NP_map(SyntacticFunction), Property>? | How often did <NP_map(SyntacticFunction), Property>
     [VerbPresent]?
GRT_ask:
S -> Did <NP_map(SyntacticFunction), Property> [VerbPresent] <NP_map(SyntacticFunction), Property>?| Does
     <NP_map(SyntacticFunction), Property> [VerbPresent] <NP_map(SyntacticFunction), Property>?
GRT_nounPhrase:
S -> <NP_Class, <map(SyntacticFunction),Property>> [VerbPresent] [Preposition]
     <NP_map(SyntacticFunction), Property> | <NP_Class, <map(SyntacticFunction),Property>> [VerbPast]
     [Preposition] <NP_map(SyntacticFunction), Property>
```

Consider a particular lexical entry shown in Figure 5 for the transitive verb *(to) 'direct'*. `[VerbPresent]` and `[VerbPast]` refer to the inflectional forms of the lexical entry. `[VerbPresent]` refers to the present form of the verb, e.g., *'direct'*, and `[VerbPast]` refers to the plural form of the verb, e.g., *'directed'*.

```
Grammar Rules from GRT_subject:
S -> Who directs <NP_Domain, dbo:director>? | Who directed <NP_Domain, dbo:director>? | Which
     <NP_Class, <Range,dbo:director>> directs <NP_Domain, dbo:director>? | Which <NP_Class, <Range,dbo:director>>
     directed <NP_Domain, dbo:director>?
Grammar Rules from GRT_object:
S -> What is directed by <NP_Range, dbo:director>? | What was directed by <NP_Range, dbo:director>?
     | Which <NP_Class, <Domain,dbo:director>> is directed by <NP_Range, dbo:director>? | Which
     <NP_Class, <Domain,dbo:director>> are directed by <NP_Range, dbo:director>? | Which
     <NP_Class, <Domain,dbo:director>> was directed by <NP_Range, dbo:director> | Which
     <NP_Class, <Domain,dbo:director>> were directed by <NP_Range, dbo:director>? | Give me all
     <NP_Class, <Domain,dbo:director>> directed by <NP_Range, dbo:director>?
```

Grammar Rules from GRT$_{amount}$:
S -> How many <NP$_{\text{Class, <Domain,dbo:director>}}$> are directed by <NP$_{\text{Range, dbo:director}}$> ? | How often did <NP$_{\text{Range, dbo:director}}$> direct?
Grammar Rules from GRT$_{ask}$:
S -> Did <NP$_{\text{Range, dbo:director}}$> direct <NP$_{\text{Domain, dbo:director}}$>? | Does <NP$_{\text{Range, dbo:director}}$> direct <NP$_{\text{Domain, dbo:director}}$>?
Grammar Rules from GRT$_{nounPhrase}$:
S -> <NP$_{\text{Class, <Domain,dbo:director>}}$> directed by <NP$_{\text{Range, dbo:director}}$>

The grammar rules generated by instantiating grammar rule templates of the first group (i.e., GRT$_{\text{subject}}$) using the lexical entry *(to) 'direct'* can parse a wide range of questions. For example, 1) *'Who directs Wait for Your Laugh?'*, 2) *'Who directed Wait for Your Laugh?'*, 3) *'Which person directs Wait for Your Laugh?'*, 4) *'Which person directed Wait for Your Laugh?'*, etc. The grammar rules generated from GRT$_{\text{object}}$ can parse questions such as 1) *'What is directed by Mario Monicelli?'*, 2) *'What was directed by Mario Monicelli?'*, 3) *'What were directed by Mario Monicelli?'*, 4) *'Which film is directed by Mario Monicelli?'*, 5) *'Which films are directed by Mario Monicelli?'*, 6) *'Which film was directed by Mario Monicelli?'*, 7) *'Which films were directed by Mario Monicelli?'*, 8) *'Give me all films directed by Mario Monicelli?'*, etc.

The grammar rules generated from GRT$_{\text{amount}}$ can parse questions such as 1) *'How many films are directed by Mario Monicelli?'*, 2) *'How often did Mario Monicelli direct?'*, etc. The grammar rule from GRT$_{\text{ask}}$ can parse boolean questions, for example, 1) *'Did Mario Monicelli direct Wait for Your Laugh?'*, 2) *'Does Mario Monicelli direct Wait for Your Laugh?'*, etc. The grammar rule from GRT$_{\text{nounPhrase}}$ can parse the noun phrases such as: 1) *'film directed by Mario Monicelli'*, 2) *'films directed by Mario Monicelli'*, etc.

2.5.3. IntransitivePPFrame

From the sentence templates of an intransitive verb (i.e., the syntactic behaviour called `IntransitivePPFrame` in `LexInfo`), our approach obtains 15 grammar rule templates organized into four groups (such as GRT$_{\text{subject}}$, GRT$_{\text{object}}$, GRT$_{\text{amount}}$, and GRT$_{\text{ask}}$).

The non-terminal <NP$_{\text{map(SyntacticFunction), Property}}$> can be substituted by all labels of URIs that appear as the subject (resp. object) of the property `Property` depending on whether the given syntactic function (`Subject`, `PrepositionalAdjunct`, etc.) maps to the domain resp. range of the property.

Grammar Rule Template (GRT$_{subject}$):
S -> What [VerbPresent] [Preposition] <NP$_{\text{map(SyntacticFunction), Property}}$>? | Which <NP$_{\text{Class, <map(SyntacticFunction),Property>}}$> [VerbPresent] [Preposition] <NP$_{\text{map(SyntacticFunction), Property}}$>? | Which <NP$_{\text{Class, <map(SyntacticFunction),Property>}}$> [VerbPresent] [Preposition] <NP$_{\text{map(SyntacticFunction), Property}}$>?
GRT$_{object}$:
S -> What does <NP$_{\text{map(SyntacticFunction), Property}}$> [VerbPresent] [Preposition]? | Which <NP$_{\text{Class, <map(SyntacticFunction),Property>}}$> does <NP$_{\text{map(SyntacticFunction), Property}}$> [VerbPresent] [Preposition]? | Which <NP$_{\text{Class, <map(SyntacticFunction),Property>}}$> does <NP$_{\text{map(SyntacticFunction), Property}}$> [VerbPresent] [Preposition]? | [Preposition] which <NP$_{\text{Class, <map(SyntacticFunction),Property>}}$> does <NP$_{\text{map(SyntacticFunction), Property}}$> [VerbPresent]?
GRT$_{amount}$:
S -> How many <NP$_{\text{Class, <map(SyntacticFunction),Property>}}$> [VerbPresent] [Preposition] <NP$_{\text{map(SyntacticFunction), Property}}$>?
GRT$_{ask}$:
S -> Does <NP$_{\text{map(SyntacticFunction), Property}}$> [VerbPresent] [Preposition]

<NP$_{\text{map(SyntacticFunction), Property}}$> | Did <NP$_{\text{map(SyntacticFunction), Property}}$> [VerbPresent] [Preposition] <NP$_{\text{map(SyntacticFunction), Property}}$>

Consider a particular lexical entry shown in Figure 6 for the intransitive verb *'flow through'* with a prepositional adjunct. [VerbPresent] refers to the present form of the verb (i.e., *'flow'* or *'flows'*). The GRTs are instantiated as follows for the lexical entry *'(to) flow through)'*.

Grammar Rules from GRT$_{subject}$:
S -> What flows through <NP$_{\text{Range, dbo:city}}$>? | Which <NP$_{\text{Class, <Domain,dbo:city>}}$> flows through <NP$_{\text{Range, dbo:city}}$>? | Which <NP$_{\text{Class, <Domain,dbo:city>}}$> flow through <NP$_{\text{Range, dbo:city}}$>?
Grammar Rules from GRT$_{object}$:
S -> What does <NP$_{\text{Domain, dbo:city}}$> flow through? | Which <NP$_{\text{Class, <Range,dbo:city>}}$> does <NP$_{\text{Domain, dbo:city}}$> flow through? | Which <NP$_{\text{Class, <Range,dbo:city>}}$> does <NP$_{\text{Domain, dbo:city}}$> flow through? | Through which <NP$_{\text{Class, <Range,dbo:city>}}$> does <NP$_{\text{Domain, dbo:city}}$> flow?
Grammar Rules from GRT$_{amount}$:
S -> How many <NP$_{\text{Class, <Domain,dbo:city>}}$> flow through <NP$_{\text{Range, dbo:city}}$>?
Grammar Rules from GRT$_{ask}$:
S -> Does <NP$_{\text{Range, dbo:city}}$> [VerbPresent] [Preposition] <NP$_{\text{Domain,dbo:city}}$>? | Did <NP$_{\text{Range, dbo:city}}$> [VerbPerfect] [Preposition] <NP$_{\text{Domain, dbo:city}}$>?

The grammar rules generated by instantiating grammar rule templates of the first group (i.e., GRT$_{\text{subject}}$) using the lexical entry *'flow through'* can parse a wide range of questions. For example, 1) *'What flows through Chennai?'*, 2) *'Which river flows through Chennai?'*, 3) *'Which rivers flow through Chennai? '*, etc. The grammar rules generated from GRT$_{\text{amount}}$ allow us to parse the questions such as 1) *'How many rivers flow through Chennai?'*, etc. The grammar rules generated from GRT$_{\text{object}}$ can parse the questions such as 1) *'What does Adyar River flow through?'*, 2) *'Which cities does Adyar River flow through?'*, 3) *'Which city does Adyar River flow through?'*, 3) *'Through which cities does Adyar River river flow?'*, etc. The grammar rules generated from GRT$_{\text{ask}}$ can parse the boolean questions such as 1) *'Does Adyar River flow through Chennai?'*, 2) *'Did Adyar River flow through Chennai?'*, etc.

The grammar rule templates for temporal expressions involving intransitive verbs, such as *'dissolve' (at)*, or when the subject is of Agent type, as in 'born' (in), are not presented here.

2.5.4. AdjectivePredicateFrame

After transforming the sentence templates of intersective adjectives (i.e., the syntactic behaviour referred to as AdjectivePredicateFrame in LexInfo), our approach yields six grammar rule templates organized into two groups (namely, GRT$_{\text{subject}}$ and GRT$_{\text{nounPhrase}}$).

Grammar Rule Template GRT$_{subject}$:
S -> Which is a [AdjectiveProper] <Noun$_{\text{Class, RestrictionClass(Property, Object)}}$>? |
Which are [AdjectiveProper] <Noun$_{\text{Class, RestrictionClass(Property, Object)}}$>? |
Give me [AdjectiveProper] <Noun$_{\text{Class, RestrictionClass(Property, Object)}}$>. |
List all [AdjectiveProper] <Noun$_{\text{Class, RestrictionClass(Property, Object)}}$>.
GRT$_{nounPhrase}$:
S -> a [AdjectiveProper] <Noun$_{\text{Class, RestrictionClass(Property, Object)}}$> |
[AdjectiveProper] <Noun$_{\text{Class, RestrictionClass(Property, Object)}}$>

The values of the non-terminals <Noun$_{\text{Class, RestrictionClass(Property, Object)}}$> are automatically generated via the SPARQL query given below.

<Noun$_{\text{Class, RestrictionClass(Property, Object)}}$> -> eval(SELECT ?label WHERE {
                                                                ?x Property Object .
                                                                ?x rdf:type ?Class .
                                                                ?Class rdfs:label ?label . })

Consider a particular lexical entry shown in Figure 7 for the adjective *'Spanish'*. [AdjectiveProper] refers to the proper form of the adjective (i.e., *'Spanish'*). The nouns that adjectives modify in the rules come from the classes in the ontology that have instances that are related to *res:Spain* via the property *dbo:country*.

Grammar Rules from GRT$_{subject}$:
S -> Which is a Spanish <Noun$_{\text{Class, RestrictionClass(dbo:country, res:Spain)}}$>? | Which are Spanish
      <Noun$_{\text{Class, RestrictionClass(dbo:country, res:Spain)}}$>? | Give me all Spanish
      <Noun$_{\text{Class, RestrictionClass(dbo:country, res:Spain)}}$>. | List all Spanish
      <Noun$_{\text{Class, RestrictionClass(dbo:country, res:Spain)}}$>.
Grammar Rules from GRT$_{nounPhrase}$:
S -> a Spanish <Noun$_{\text{Class, RestrictionClass(dbo:country, res:Spain)}}$> |
      Spanish <Noun$_{\text{Class, RestrictionClass(dbo:country, res:Spain)}}$>

The SPARQL query used to retrieve <Noun$_{\text{Class, RestrictionClass(dbo:country, res:Spain)}}$> is given below.

<Noun$_{\text{Class, RestrictionClass(dbo:country, res:Spain)}}$> -> eval(SELECT ?label WHERE {
                                                                ?x dbo:country res:Spain . ?
                                                                ?x rdf:type ?Class .
                                                                ?Class rdfs:label ?label . })

Once the query is executed, a rule like this is generated:

<Noun$_{\text{Class, RestrictionClass(dbo:country, res:Spain)}}$> -> Athlete | Economist | ...

The grammar rules generated by instantiating grammar rule templates of the first group (i.e., GRT$_{\text{subject}}$) using the lexical entry *'Spanish'* allow us to parse questions such as 1) *'Which is a Spanish Athlete?'*, 2) *'Which are Spanish Athletes?'*, 3) *'Give me all Spanish Athletes.'*, 4) *'List all Spanish Athletes.'*, etc. The grammar rules generated from GRT$_{\text{nounPhrase}}$ can parse questions such as *'a Spanish Athlete'*,*'Spanish Athletes'*, etc.

2.5.5. AdjectiveSuperlativeFrame

From the sentence templates of gradable adjectives (i.e., the syntactic behaviour called AdjectiveSuperlativeFrame in LexInfo), our approach obtains six grammar rule templates organized into three groups (such as GRT$_{\text{baseForm}}$, GRT$_{\text{comparative}}$, and GRT$_{\text{superlative}}$).

Grammar Rule Template (GRT$_{baseForm}$):
S -> How [AdjectivePositive] is <NP$_{\text{map(SyntacticFunction), Property}}$>?
GRT$_{comparative}$:
S -> Which <NP$_{\text{Class, <map(SyntacticFunction),Property>}}$> is [AdjectiveComparative] than
      <NP$_{\text{map(SyntacticFunction), Property}}$>? | Which <NP$_{\text{Class, <map(SyntacticFunction),Property>}}$> are
      [AdjectiveComparative] than <NP$_{\text{map(SyntacticFunction), Property}}$>?
GRT$_{superlative}$:
S -> What is the [AdjectiveSuperlative] <NP$_{\text{Class, <map(SyntacticFunction),Property>}}$>?

Consider the lexical entry shown in Figure 8 for the gradable adjective *'high'*. [AdjectivePositive] refers to the positive form of the adjective, e.g., *'high'*. [AdjectiveSuperlative] refers to the superlative form of the adjective, e.g., *'highest'*. The lexicalized grammar rules are shown below.

```
Grammar Rules from GRT_baseForm:
S -> How high is <NP_{Domain, dbo:elevation}>?
Grammar Rules from GRT_comparative:
S -> Which <NP_{Class, <Domain, dbo:elevation>}> is higher than the <NP_{Domain, dbo:elevation}>? |
     Which <NP_{Class, <Domain, dbo:elevation>}> are higher than the <NP_{Domain, dbo:elevation}>?
Grammar Rules from GRT_superlative:
S -> What is the highest <NP_{Class, <Domain, dbo:elevation>}>?
```

The values of the non-terminals <NP$_{\text{Domain, dbo:elevation}}$> are retrieved with the following automatically generated SPARQL query:

```
<NP_{Domain, dbo:elevation}> -> SELECT ?label WHERE {?Domain dbo:elevation ?object .
                                              ?Domain rdfs:label ?label . }
```

The grammar rules generated by instantiating grammar rule templates of the first group (i.e., GRT$_{\text{baseForm}}$) using the lexical entry *'high'* allow us to parse questions such as *'How high is Zugspitze?'*. The grammar rules generated from GRT$_{\text{comparative}}$ can parse questions such as *'Which mountain is higher than the Zugspitze?'*, *'Which mountains are higher than the Zugspitze?'*, etc. The grammar rule generated from GRT$_{\text{superlative}}$ can parse questions such as *'What is the highest mountain?'*.

2.6. Parsing

In this section, we provide a complete example of parsing a given question (i.e., *'Who is the mayor of Russia?'*) into a SPARQL query using the grammar discussed in the previous section. After that, we explain how the approach parses a complex question (i.e., *'Who is the mayor of the capital of Russia?'*).

For parsing, the lexicalized grammar rules (as detailed in Section 2.5.1) are automatically converted into regular grammar[9] [38, 39] by replacing the non-terminals with regular expressions (i.e., $([A-Za-z])$). For example, the grammar rule *'Who is the mayor of <NP$_{\text{Domain, dbo:mayor}}$>?'* is transformed into *'Who is the mayor of* $([A-Za-z])$*?'*. The regular grammar is shown below:

```
S -> Who is the mayor of ([A-Za-z]*)? | Who was the mayor of ([A-Za-z]*)>? | Which
     ([A-Za-z]*) is the mayor of ([A-Za-z]*)? | Which ([A-Za-z]*) was the mayor of
     ([A-Za-z]*)? | Give me the mayor of ([A-Za-z]*). | Give me all mayors of
     ([A-Za-z]*). | List all mayors of ([A-Za-z]*).
S -> ([A-Za-z]*) is the mayor of which ([A-Za-z]*)? | ([A-Za-z]*) was the mayor of
     which ([A-Za-z]*)?
S -> How many mayors does ([A-Za-z]*) have?
S -> Is  ([A-Za-z]*) the mayor of ([A-Za-z]*)? | Was ([A-Za-z]*) the mayor of
     ([A-Za-z]*)?
S -> the mayor of ([A-Za-z]*)?
```

There are several advantages to using a regular grammar. First, regular grammar is more straightforward compared to state-of-the-art grammar formalisms, such as context-free grammar [40] or tree-adjoining grammar [35, 41]. This simplicity can be advantageous in scenarios where a less complex grammar is sufficient for tasks such as question answering over linked data, as is the case in our approach. Second, regular languages can be parsed using finite automata, which are generally more computationally efficient than the more complex mechanisms employed in top-down parsing [40] for context-free grammar or the substitution and adjunction operations [42] of tree-adjoining grammar.

[9]A regular grammar typically consists of production rules that define how symbols can be combined to form strings in the language. The rules are often expressed in the form of regular expressions, which describe patterns of symbols.

2.7. *Simple Question*

The parser takes a sentence in natural language and produces an analysis of the sentence in the form of one or more SPARQL queries. In our example, the input and output of the parser are shown below:

```
input: Who is the mayor of Moscow?
output: SELECT ?label WHERE {res:Moscow dbo:mayor ?Range . ?Range rdfs:label ?label .}
```

The step-by-step method is as follows:

1. *Find the grammar rule*: Given a question in natural language, the parser iterates through all the grammar rules and retrieves the grammar rule when a match is found.

   (a) If a match is found, it returns the grammar rule that includes two SPARQL queries (as detailed in Section 2.5.1). Once a match is found, proceed to steps 2 and 3.
   For example, the grammar rule matching the question *'Who is the mayor of Moscow?'* is shown below.

   ```
   S -> Who is the mayor of ([A − Za − z]∗)?

   The SPARQL query for retrieving values of the regex ([A − Za − z]∗) is as follows:
   SELECT ?label WHERE {?Domain dbo:mayor ?Range . ?Domain rdfs:label ?label .}

   After executing the SPARQL query, the values are as follow:
   Provisional Government of Belgium | Paris | Moscow | ..

   SPARQL query of the grammar rule (SPARQL_rule):
   SELECT DISTINCT ?label WHERE { ?Arg dbo:mayor ?Range . ?Range rdfs:label ?label .}
   ```

   As explained in Section 2.5.1, the values of non-terminal in a grammar rule is extracted using SPARQL query. In regular grammar, non-terminals are replaced by regular expressions, so the values of the regex $([A - Za - z]*)$ is extracted using the SPARQL query, resulting in a set of strings. The SPARQL query (i.e., SPARQL$_{\text{rule}}$) of the grammar rule contains a variable (i.e., Arg) that needs to be replaced with a resource.

   (b) If there is no grammar rule that matches the given question, it cannot be parsed.

2. *Extract the entity of the question*: After the question *'Who is the mayor of Moscow?'* is matched with the grammar rule *'Who is the mayor of* $([A-Za-z]*)$*?'*, the entity of the question is extracted; in this case, *'Moscow'*.

3. *Match the entity with terminals*: To match the entity *'Moscow'* (of the question) with the terminals (of the grammar rule), the parser uses the Jaccard similarity[10] measure with a threshold set to 0.5. The matching is done as follows:

   (a) Each terminal of step 1a (e.g., *'Provisional Government of Belgium'*) is converted to a set of tokens.

   (b) Afterwards, the Jaccard similarity is computed between each terminal and the entity of the question, in this case, *'Moscow'*. The similarity scores are ranked in descending order.

   (c) If the similarity score is greater than the threshold (i.e., 0.5), then the top match is selected and the corresponding variable (i.e., Arg) of the SPARQL query (i.e., SPARQL$_{\text{rule}}$) is replaced. Otherwise, the question is not parsed. In this case, the score is greater than the threshold and so the parse result is as follows:

[10]The Jaccard similarity method finds the similarity by the size of the intersection divided by the size of the union of the two sets.

```
        SELECT ?label WHERE {res:Moscow dbo:mayor ?Range . ?Range rdfs:label ?label .}
```

2.8. Complex Questions

The grammar rules discussed in the section 2.5 allow parsing questions that can be represented by a SPARQL query that involves a triple pattern with the predicate *rdf:type*, a triple pattern with the predicate *rdfs:label* and one more triple pattern. The approach can thus not parse complex questions involving more triples such as the following one:

```
Question: Who is the mayor of the capital of Russia?
SPARQL: SELECT DISTINCT ?label WHERE { res:Russia dbo:capital ?o .
                                       ?o dbo:mayor ?uri .
                                       ?uri rdfs:label ?label . }
```

In order to allow our QA system to parse such questions correctly, we implemented a limited form of composition that allows certain rules to be nested within others. Specifically, our approach supports the nesting of noun phrases (e.g., *'the capital of Russia'* or *'Russian capital'*) into the rules of other frames.

The parser takes a sentence *'Who is the mayor of the capital of Russia?'* and produces an analysis of the sentence in the form of one or more SPARQL queries. In our approach, the input and output of the parser is shown below:

```
input: Who is the mayor of the capital of Russia?
output: SELECT ?label WHERE {res:Moscow dbo:mayor ?Range . ?Range rdfs:label ?label .}
```

The step-by-step method is as follows:

1. The question 'Who is the mayor of the capital of Russia?' matches with the following grammar rule:

   S -> Who is the mayor of $([A-Za-z]*)$?

   The SPARQL query for retrieving values of the regex $([A-Za-z]*)$ is as follows:
   SELECT ?label WHERE {?Domain dbo:mayor ?Range . ?Domain rdfs:label ?label .}

   After executing the SPARQL query, the values are as follow:
   $([A-Za-z]*)$ -> Provisional Government of Belgium | Paris | Moscow | ..

   SPARQL query of the rule (SPARQL$_{\text{rule}}$):
   SELECT DISTINCT ?label WHERE { ?Arg dbo:mayor ?Range .
                                  ?Range rdfs:label ?label .}

   The SPARQL query of the grammar rule contains a variable (i.e., Arg) that needs to be replaced with a resource.
2. Extract the entity of the question: After the question *'Who is the mayor of the capital of Russia?'* is matched with the grammar rule *'Who is the mayor of* $([A-Za-z]*)$*?'*, the entity of the question is extracted; in this case, *'the capital of Russia'*.
3. Parse simple question: In this case, the parser treats *'the capital of Russia?'* as a noun phrase and parses it similarly to the simple question discussed in the previous section. The grammar rule matched is shown below:

   NP -> the capital of $([A-Za-z]*)$.

   The SPARQL query for retrieving values of the regex $([A-Za-z]*)$ is as follows:

```
SELECT ?label WHERE { ?Domain dbo:capital ?Range . ?Range rdfs:label ?label . }

After executing the SPARQL query, the values are as follow:
Canton of Baden | Germany | Russia | ..
```

The parse result of *'the capital of Russia?'* is

```
SELECT ?Range WHERE {res:Russia dbo:capital ?Range . }
```

The answer retrieved for this noun phrase is *res:Moscow*. Therefore, the semantics of the expression *'the capital of Russia'* can alternatively be expressed by *res:Moscow*.

4. Replace the variable: The variable (i.e., Arg) of SPARQL query (step 1) is replaced by *res:Moscow*. The parse result of the complex question *'Who is the mayor of the capital of Russia?'* is as follows:

```
SELECT ?label WHERE {res:Moscow dbo:mayor ?Range . ?Range rdfs:label ?label .}
```

This simple approach to dealing with complex questions implements a limited form of compositionality that treats the phrase *'the capital of Russia'* as (non-compositionally) referring to the URI *res:Moscow*. While this way of parsing certain types of complex questions is effective as shown by our experimental results, it is limited to two-hop queries involving at most a linear chain of two properties in addition to *rdf:type* (or *rdfs:label*) statement.

3. Multilingual Grammar Generation

In this section, we detail our model-based approach that automatically generates lexicalized grammar rules from a Lemon lexicon. First, we explain the method for English. After that, we describe the methodology we followed to adapt the system to multiple languages—in our case, German, Spanish, and Italian. We elaborate on the process of implementing corresponding grammar rule templates (GRT) (as discussed in the previous section) for each of these languages. These grammar rule templates are independent of a particular dataset. Subsequently, the grammar rules are generated through instantiations of the templates using lexical entries. Further, we explain the process of creating lexical entries for these four languages using DBpedia as proof of concept.

3.1. English Grammar Generation

As described in the high-level architecture (as detailed in Section 2.5), the first step is to create lexical entries manually.

3.1.1. Creating Lexical Entries

To create a lexical entry of a relative noun, such as the one shown in Figure 4, the English language expert needs to provide the following information: (i) the singular and plural form of the reference, i.e., *'mayor'* and *'mayors'* for *dbo:mayor*; and (ii) the preposition, i.e.,*'of'*.

3.1.2. Creating Sentence Templates

The sentence templates are manually created by language experts, as indicated in the high-level diagram as the second step. In Section 2.4, we define sentence templates with examples. In this section, we detail the steps necessary to create sentence templates.

We create sentence templates for each syntactic frame (detailed in Section 2.2). For example, for a relative noun (i.e., the syntactic frame called `NounPPFrame` as detailed in Section 2.5.1), we provide an example of a relative noun of the Agent type, such as *'mayor of'*. An Agent noun refers to a person or entity that performs or is associated with a particular action. The language expert created sentence templates for this sample relative noun. After that, we provided another sample of a relative noun of the non-Agent type, such as *'capital of'*. A non-Agent noun refers to something that is not related to or does not possess the characteristics of an agent. The language expert then created sentence templates for this type. No further samples of relative nouns were provided to the language expert. Note that creating sentence templates is different from creating lexical entries, as sentence templates are created for a syntactic frame only and are independent of lexical entries.

The following provides a detailed method for creating sentence templates for the Agent type. The language expert is provided with the following information:

1) a sample of relative noun of Agent type w (e.g., w=*'mayor of'*)
2) five example questions q_{subject}, q_{object}, $q_{\text{subject-amount}}$, q_{ask}, $q_{\text{nounPhrase}}$ involving that relative noun. The examples are as follows:

   q_{subject}: `'Who is the mayor of Paris?'`
   q_{object}: `'Anne Hidalgo is the mayor of which city?'`
   q_{amount}: `'How many mayors does Paris have?'`
   q_{ask}: `'Is Anne Hidalgo mayor of Paris?'`
   $q_{\text{nounPhrase}}$: `` `the mayor of Paris.' ``

   The example q_{subject} illustrates a question containing a relative noun that seeks information about the subject of a sentence, while the example q_{object} provides a question seeking information about the object of a sentence. The question q_{amount} is quantitative or numerical, and the response would require providing a specific numerical value. The question q_{ask} is a boolean question that typically elicits a binary response. The reason for presenting various types of questions is to address different variations of sentence templates containing relative nouns.
3) a set C of syntactic categories (e.g., C ={`InterrogativeDeterminer`, `InterrogativePronounForPerson`, `InterrogativePronounForThing`, `RelativeNoun(singular)`, `RelativeNoun(plural)`, `Preposition`, `PrepositionalAdjunct`, `AuxVerb(be:present:singular)`,..., etc.})
4) with a dictionary $D_{Category}$, that contains a list of words for each syntactic category in C (e.g., $D_{Category}$ = {`InterrogativeDeterminer` → {'what'}, `InterrogativePronounForPerson` → {'who'}, `AuxVerb(be:present:singular)` → {'is'}, ..., etc. }).

Note that there are two kinds of syntactic categories: (i) categories that take no parameters, such as `InterrogativeDeterminer`, `PrepositionalAdjunct`, etc., and (ii) categories that take one or more parameters, such as `AuxVerb(be:present:singular)`, etc. Therefore, the language expert receives guidelines on which syntactic categories need to be parameterized. For example, the syntactic category `AuxVerb` needs to be parameterized with `be:present:singular`. Here, `be` indicates an auxiliary verb, `present` indicates the tense (in this case, the present tense), and `singular` specifies the grammatical number.

For the given example q_{subject}, the language expert created the following sentence template:

q_{subject}: `` `Who is the mayor of Paris?' ``
$ST_{subject}$: `InterrogativePronounForPerson AuxVerb(be:present:singular) Determiner(the) RelativeNounSingular Preposition PrepositionalAdjunct?`

The language expert also created four variations for sentence templates for the given example, such as:

[$q2$]: `` `Who was the mayor of Paris?' ``
[ST_{q2}]: `InterrogativePronounForPerson AuxVerb(be:past:singular) Determiner(the) RelativeNounSingular Preposition PrepositionalAdjunct?`
[$q3$]: `` `Give me the mayor of Paris.' ``
[ST_{q3}]: `ImperativeVerb(give) pronoun(me) Determiner(the)`

```
RelativeNounSingular Preposition PrepositionalAdjunct?.
[q4]: `Give me the mayors of Paris.'
[ST_q4]: ImperativeVerb(give) pronoun(me) Determiner(the)
RelativeNounPlural Preposition PrepositionalAdjunct?.
[q5]: `List all the mayors of Paris.'
[ST_q5] ImperativeVerb(list) Determiner(all) Determiner(the)
RelativeNounPlural Preposition PrepositionalAdjunct?.
```

Similarly, the language expert created sentence templates for the examples of q_{object}, q_{amount}, q_{ask}, $q_{\text{nounPhrase}}$. These sentence templates are transformed into grammar rule templates (as discussed in Section 2.5) and the grammar rule templates are subsequently instantiated with lexical entries, resulting in lexicalized grammar rules (GR).

A total of 24 sentence templates were manually created for the `Agent` type and 20 sentence templates for the `non-Agent` type of relative noun, providing variations in sentence construction for the `NounPPFrame`.

Similarly, for the `TransitiveFrame` (as discussed in Section 2.5.2), two samples of monotransitive verbs are provided, where the verb takes only one direct object: (i) a sample (i.e., *'to marry'*) when both the subject and object in the sentence are of the Agent type (e.g., people, organizations, etc.), and (ii) a sample (i.e., *'to produce'*) where the subject is of the Agent type and the object is a thing (e.g., Skype, hovercrafts, etc.). Five examples (i.e., q_{active}, q_{passive}, q_{amount}, q_{ask}, and $q_{\text{nounPhrase}}$.) are provided for each of these samples. Here, q_{active} represents an example that contains the active form of a verb (e.g., 'Who developed Skype?').

For the `InTransitivePPFrame` (as discussed in Section 2.5.3), five samples of intransitive verbs are provided: (i) a sample (i.e., *'flow through'*) where the subject and the object in the prepositional adjunct in the sentence are things, (ii) a sample (i.e., *'born in'*) where the subject is of the Agent type and the object is a thing, (iii) a sample (i.e., *'born in'*) where the subject is of the Agent type and the object is a place, (iv) a sample (i.e., *'founded in'*) where the subject is of the Agent type and the object is a date, and (v) a sample (i.e., *'start in'*) where the subject and the object contain a place.

For the `AdjectiveAttributiveFrame` and `AdjectiveSuperlativeFrame` (as discussed in Section 2.5.4 and Section 2.5.5), two samples are provided: (i) a sample of an intersective adjective (i.e., *'Spanish'*), and (ii) a sample of a gradable adjective (i.e., *'high'*). Note that, in order to adapt to another language, we only need to write sentence templates for that language using the same syntactic categories, but the process of grammar generation remains the same.

Frame	**# Lexical Entry**	**# Grammar Rule Templates (GRT)**	**# Grammar Rule (GR)**
NounPPFrame	311	44	8,708
TransitiveFrame	96	101	3,078
InTransitivePPFrame	143	91	2,183
AdjectivePredicateFrame	28	5	910
AdjectiveSuperlativeFrame	21	8	168
Total	599	249	15,047

Table 1

An overview of the number of manually created lexical entries, grammar rule templates, and grammar rules for different frame types of `LexInfo` for the English language. The table also shows the number of grammar rules that are generated automatically after instantiations of grammar rule templates with lexical entries.

As shown in Table 1, a total of 44 grammar rule templates for `NounPPFrame`, 101 templates for `TransitiveFrame`, and 91 templates for `IntransitivePPFrame` were created, resulting in a total of

249 grammar rule templates for the English language. After instantiating these grammar rule templates with 599 lexical entries of various syntactic frames, a total of 15, 047 grammar rules were automatically generated. The manually created lexical entries, sentence templates, grammar rule templates and grammar rules are available here[11] for all four languages: English, German, Italian, and Spanish. The English grammar is created from English DBpedia[12].

Note that the lexicon presented here includes lexical entries that also cover the training data of QALD-9 benchmarks. The lexicon is evaluated with test data in two modes (as detailed in Section 5): (i) *inductive evaluation*, and (ii) *incremental evaluation*.

The time required for creating a lexical entry is approximately 5–10 minutes depending on the syntactic frame. The time required for creating a sentence template is approximately 10–15 minutes. However, the majority of sentence templates are similar to one of the templates with minor changes (i.e., inflectional forms, tense, etc.). The creation of these templates took approximately 1–2 minutes. The total time required to create all sentence templates for entries of type `NounPPframe` took approximately 1 hour, sentence templates for entries of type `TransitiveFrame` took approximately 3 hours, and sentence templates for entries of type `IntransitivePPframe` took approximately 5 hours.

3.2. Porting to other languages

Porting the QA system to other languages is straightforward in our approach, which follows two steps: (i) adding new lexical items by providing language-specific properties, and (ii) adding grammar rules for that language. In this section, we detail the lexicon creation for other languages (e.g., German, Italian, and Spanish), focusing on language-specific features and how we accommodate those in our model-based approach.

3.2.1. German Grammar Generation

Unlike in English, German nouns are gender-specific (e.g., either masculine, feminine, or neuter) and the endings of the nouns vary with the corresponding case (e.g., nominative, accusative, dative, and genitive). To add a lexical entry of type `NounPPFrame` in German, the language expert needs to provide not only the gender but also the written form of the reference in singular and plural forms in all cases. Consider the relative noun *'Bürgermeister' (von)* (i.e.,*'mayor' (of)*) in German. The semantics is captured by the property *dbo:mayor* (as shown in Figure 4) and the written forms of the noun are presented in the Lemon dictionary as follows:

```
:Buergermeister_von_singular_form a lemon:Form ;
  lemon:writtenRep "Bürgermeister"@de ;
  lexinfo:gender   lexinfo:masculine ;
  lexinfo:number   lexinfo:singular ;
  lexinfo:case     lexinfo:nominativeCase .

:Buergermeister_von_plural_form a lemon:Form ;
  lemon:writtenRep "Bürgermeister"@de ;
  lexinfo:gender   lexinfo:masculine ;
  lexinfo:number   lexinfo:plural ;
  lexinfo:case     lexinfo:nominativeCase .

:Buergermeister_von_accusative_form a lemon:Form ;
  lemon:writtenRep "Bürgermeister"@de ;
  lexinfo:gender   lexinfo:masculine ;
```

[11]https://github.com/fazleh2010/Journal-Paper/blob/master/README.md
[12]https://downloads.dbpedia.org/2016-10/core-i18n/en/

```
  lexinfo:number   lexinfo:singular ;
  lexinfo:case     lexinfo:accusativeCase .

:Buergermeister_von_dative_form a lemon:Form ;
  lemon:writtenRep "Bürgermeister"@de ;
  lexinfo:gender   lexinfo:masculine ;
  lexinfo:number   lexinfo:singular ;
  lexinfo:case     lexinfo:dativeCase .

:Buergermeisterin_von_singular_form a lemon:Form ;
  lemon:writtenRep "Bürgermeisterin"@de ;
  lexinfo:gender   lexinfo:feminine ;
  lexinfo:number   lexinfo:singular ;
  lexinfo:case     lexinfo:nominativeCase .

:Buergermeisterin_von_plural_form a lemon:Form ;
  lemon:writtenRep "Bürgermeisterinnen"@de ;
  lexinfo:gender   lexinfo:feminine ;
  lexinfo:number   lexinfo:plural ;
  lexinfo:case     lexinfo:nominativeCase .

:Buergermeisterin_von_accusative_form a lemon:Form ;
  lemon:writtenRep "Bürgermeisterin"@de ;
  lexinfo:gender   lexinfo:feminine ;
  lexinfo:number   lexinfo:singular ;
  lexinfo:case     lexinfo:accusativeCase .

:Buergermeisterin_von_dative_form a lemon:Form ;
  lemon:writtenRep "Bürgermeisterin"@de ;
  lexinfo:gender   lexinfo:feminine ;
  lexinfo:number   lexinfo:singular ;
  lexinfo:case     lexinfo:dativeCase .
```

Frame	**# Lexical Entry**	**# Grammar Rule Templates (GRT)**	**# Grammar Rule (GR)**
NounPPFrame	101	45	5,102
TransitiveFrame	63	113	5,345
InTransitivePPFrame	74	121	4,972
AdjectivePredicateFrame	15	15	501
AdjectiveSuperlativeFrame	10	7	70
Total	263	301	15,990

Table 2

An overview over the number of manually created lexical entries, grammar rule templates, and grammar rules for different frame types of LexInfo for the German language. The table also shows the number of grammar rules that are generated automatically after instantiations of grammar rule templates with lexical entries.

The German verb has infinitive forms, conjugated forms depending on the subject, reflexive forms, and separable forms (i.e., verbs with separable prefixes). To add a lexical entry for a German intransitive verb, e.g.,*'aufhören'*, which is captured by the property *dbo:routeEnd*, the following information needs to be provided: (i) the verb type (i.e., either conjugate or separable or reflexive), (ii) the infinite form, i.e.,*'aufhören'*, (iii) the past form, i.e.,*'hörte auf'*, and (iv) the perfect form, i.e.,*'hat aufgehört'*.

We created a lexicon for the German language and the DBpedia dataset (Release 2016-10; core, links)[13]. As shown in Table 2, there are a total of 301 grammar rule templates for German. After instantiating these grammar rule templates with 263 lexical entries of various syntactic frames, a total of $15,990$ grammar rules were automatically generated.

Similar like English, the lexicon presented here includes lexical entries that also cover the training data of QALD-9 benchmarks.

The time required for creating a lexical entry is approximately 10–15 minutes depending on the syntactic frame. The time required for creating a sentence template is approximately 10–25 minutes. However, the majority of sentence templates are similar to one of the templates with minor changes in cases and genders. The creation of these templates took approximately 1–2 minutes. The total time required to create all sentence templates for entries of type `NounPPframe` took approximately 2 hours, sentence templates for entries of type `TransitiveFrame` took approximately 4 hours, and sentence templates for entries of type `IntransitivePPframe` took approximately 5 hours.

Note that to adapt to German, we only added language-specific properties to the lexicon and adapted the sentence templates with respect to the features of German, but the process of grammar generation remains the same.

3.2.2. Italian Grammar Generation

Unlike German, the Italian noun does not depend on the cases but instead on the first letter of the noun they are preceding. To create a Lemon entry for a relative noun in Italian, the language expert needs to provide the gender information of the reference, domain, and range in both singular and plural forms. That is, the language expert needs to provide one of the four forms (i.e., *'il'* or *'la' 'l"* or *'l'*) in the case of a singular noun.

Frame	**# Lexical Entry**	**# Grammar Rule Templates (GRT)**	**# Grammar Rule (GR)**
NounPP	291	55	$3,820$
TransitiveFrame	35	89	$3,001$
InTransitivePPFrame	37	102	$1,593$
AdjectivePredicateFrame	74	13	$6,13$
AdjectiveSuperlativeFrame	4	7	34
Total	441	266	$9,061$

Table 3

An overview over the number of manually created lexical entries, grammar rule templates, and sentence templates for different frame types of `LexInfo` for the Italian language. The table also shows the number of grammar rules that are generated automatically after instantiations of grammar rule templates with lexical entries.

The presence of auxiliary verbs, either *'avere'* ('have') or *'essere'* ('be'), in compound tenses, needs to be explicitly specified. The generation of the specific interrogative pronoun is governed by the following rules as (i) *'chi' ('who')* is invariable and refers only to people; (ii) *'quale' ('which')* is used for the masculine and feminine singular, and *'quali'* is used for masculine and feminine plural; (iii) *'quanto'* and *'quanti'* have feminine forms, etc. The use of different prepositions, either simple or articulated, is based on domain/range (e.g., *'toponyms'* might require different prepositions). The presence of a determiner/articulated preposition based on domain/range semantics (e.g., *'toponyms'*

[13]https://downloads.dbpedia.org/2016-10/core-i18n/de/

are preceded by a determiner when the noun refers to a country) also impacts the final choice in lexical elements. In adapting the grammar generation to Italian, we accommodate all these language-specific properties in the sentence template.

We created a lexicon for the Italian language and the DBpedia dataset (Release 2016-10; core, links)[14]. As shown in Table 3, there are a total of 266 grammar rule templates. After instantiating these grammar rule templates with 441 lexical entries of various syntactic frames, a total of $9,061$ grammar rules were automatically generated.

The time required for the creation of a lexical entry is approximately 10–25 minutes and the time required for creating a sentence template is 20–30 minutes. However, the majority of sentence templates are similar to one of the templates with minor changes in inflectional forms and tense. The creation of these templates took approximately 1–2 minutes. The total time required to create all sentence templates for entries of type `NounPPframe` took approximately 2 hours, sentence templates for entries of type `TransitiveFrame` took approximately 3 hours, and sentence templates for entries of type `IntransitivePPframe` took approximately 5 hours.

3.2.3. Spanish Grammar Generation

The properties of Spanish nouns and verbs are not significantly different from those for Italian. Therefore, lexical entries are added in the same way as for Italian, providing a masculine and feminine form for each noun. However, the gradable adjective in Spanish differs from Italian. To add a lexical entry for an adjective such as *'alto'* (i.e., *'high'* in English), the language expert needs to provide the written form of superlative in the singular masculine form (i.e., *'el más alto'*), the plural masculine form (i.e., *'los más altos'*), the singular feminine form (i.e., *'la más alta'*), and the plural feminine form (i.e., *'las más altas'*).

Frame	**# Lexical Entry**	**# Grammar Rule Templates (GRT)**	**# Grammar Rule (GR)**
NounPPFrame	287	57	$3,512$
TransitiveFrame	81	85	$2,987$
InTransitivePPFrame	78	97	$1,503$
AdjectivePredicateFrame	13	14	519
AdjectiveSuperlativeFrame	11	9	27
Total	470	262	$8,548$

Table 4

An overview over the number of manually created lexical entries, grammar rule templates, and grammar rules for different frame types of `LexInfo` for the Spanish language. The table also shows the number of grammar rules that are generated automatically after instantiations of grammar rule templates with lexical entries.

We created a lexicon for the Spanish language and the DBpedia dataset (Release 2016-10; core, links)[15]. As shown in Table 4, there are a total of 262 grammar rule templates. After instantiating these grammar rule templates with 470 lexical entries of various syntactic frames, a total of $8,548$ grammar rules were automatically generated.

The time required for creating a lexical entry is approximately 5–10 minutes depending on the syntactic frame. The time required for creating a sentence template is approximately 10–25 minutes. The total time required to create all sentence templates for entries of type `NounPPframe` took approximately 2 hours, sentence templates for entries of type `TransitiveFrame` took approximately 4 hours, and sentence templates for entries of type `IntransitivePPframe` took approximately 6 hours.

[14]https://downloads.dbpedia.org/2016-10/core-i18n/it/
[15]https://downloads.dbpedia.org/2016-10/core-i18n/es/

The total construction time for the lexicons in four languages was approximately 60 hours.

4. Implementation

Our approach consists of three components: (i) a grammar generator (as shown on the left side of Figure 3), (ii) a parser, and (iii) a web application (as shown on the right side of Figure 3).

The core component is the grammar generator tool, which takes a Lemon lexicon and sentence templates as input and automatically creates lexicalized grammar rules. For example, the tool takes the sentence templates (ST) of a syntactic frame (e.g., `NounPPFrame`) and transforms them into grammar rule templates (GRT) (as detailed in Section 2.5). These grammar rule templates are independent of the particular lexical entry. After that, these templates are instantiated with lexical entries (e.g., *'mayor' (of)*) of `NounPPFrame`, resulting in lexicalized grammar rules. The grammar generation tool is a command-line tool developed in Java 15. Its source code is available here[16].

The second component is the parser tool, which takes a question in natural language and grammar and provides one or more SPARQL queries. The parser is also a command line tool developed in Java 15, and its source code is available here[17].

The third component is the web application (as shown on the right side of Figure 3), which is implemented using the Spring7 framework. Given a question, the application is able to present an answer (or a set of answers) given by the SPARQL endpoint in a comprehensive manner. The previous version of the QA system was limited to English [43] and limited to answering a million questions. We extended the system for German, Spanish, and Italian and also improved the system by implementing the parsing method, thus the capacity of the QA system is extended to cope with an unlimited number of questions as the questions do not need to be indexed. The web-based demonstration of the QA system is available online here[18], and the source code can be obtained from here[19].

5. Evaluation

In this section, we present the evaluation of our approach using the well-known QALD shared task [44] as a benchmark. We evaluate the system on five editions of QALD (i.e., QALD-9, QALD-7, QALD-6, QALD-5, and QALD-3) and two modes: inductive and incremental. The inductive setting corresponds to the standard evaluation mode in which a model is induced from training data and its generalization ability is evaluated on unseen test data. In our case, the inductive aspects come from the fact that the lexicon entries are created to correspond to lexical elements used in questions in the training dataset.

In our work, we are interested however mainly in an incremental setting, showing that the performance of a QA system can be systematically improved by adding new lexical entries for uncovered questions. In the incremental evaluation mode, we thus add further lexical entries to cover the questions in the test split of the data.

In addition to evaluating our system with respect to QALD datasets, we also discuss how 3rd parties could successfully adapt the system to new datasets. Further, we show the consistency and controllability of our approach by analyzing the impact of removing or adding single lexical items on the performance of the QA system.

[16]grammar generator: https://github.com/fazleh2010/multilingual-grammar-generator
[17]parser: https://github.com/ag-sc/grammar-rules.git
[18]web-based demonstration: https://webtentacle1.techfak.uni-bielefeld.de/quegg-l/
[19]web interface: https://github.com/ag-sc/QueGG-web/tree/extension

5.1. Benchmark

We evaluate our approach using the QALD challenge series [45] (i.e., QALD-9, QALD-7, QALD-6, QALD-5, and QALD-3), adopting standard evaluation metrics such as precision, recall, and F_1 for the DBpedia dataset. QALD benchmarks [46] allow us to measure the performance for multiple languages and over natural language questions on DBpedia KB elements. The dataset contains questions in multiple languages and corresponding SPARQL queries and answers from DBpedia.

Language	**Micro P**	**Micro R**	**Micro F_1**	**Macro P**	**Macro R**	**Macro F_1**
English	0.88	0.74	0.80	0.55	0.56	0.55
German	0.95	0.71	0.81	0.51	0.51	0.51
Italian	0.89	0.53	0.66	0.45	0.46	0.45
Spanish	0.90	0.52	0.66	0.43	0.43	0.43

Table 5

Results for QALD-9 training dataset. P refers to Precision, R refers to Recall, and F_1 refers to F_1-Measure.

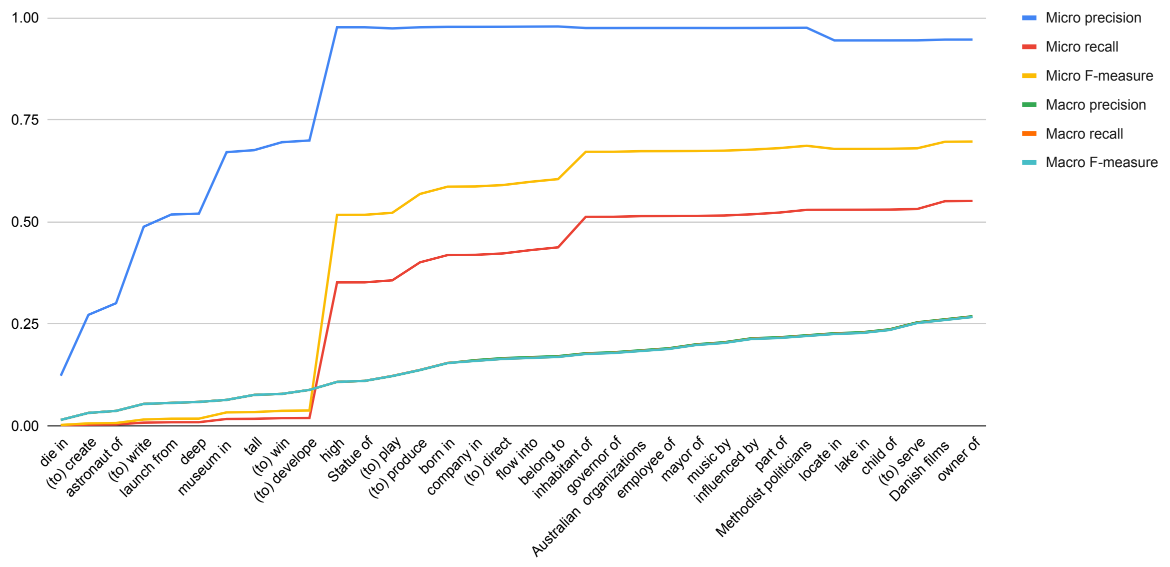


Fig. 9. Results of the incremental evaluation using QALD-9 training data. After each extension of the lexicon, we evaluate precision, recall, and F_1-measure.

Table 5 shows the results of the evaluation against QALD-9 [47] training data for English, German, Italian, and Spanish questions. Figure 9 shows an example of how the F_1-measures, the precision measures, and the recall measures improve for each lexical entry added to the lexicon in our evaluation. As can be seen from Table 5: (i) the micro F_1-measure is 0.80 for English; (ii) the score is 0.81 for German; (iii) the score is 0.66 for Italian; and (iv) for Spanish, the score is 0.66.

The evaluation with test data is conducted in two modes: inductive and incremental.

5.2. Inductive Evaluation

The goal of the inductive evaluation mode is to evaluate our approach's ability to generalize to unseen data or tasks. For the *inductive evaluation*, we used only the training data to create lexical

Language	**Evaluation mode**	**# Lexical Entries**	**Micro P**	**Micro R**	**Micro F_1**	**Macro P**	**Macro R**	**Macro F_1**
English	Inductive	61	0.72	0.18	0.29	0.22	0.21	0.21
	Incremental	+14	0.91	0.79	**0.85**	0.31	0.30	**0.30**
German	Inductive	54	0.99	0.13	0.23	0.19	0.18	0.19
	Incremental	+11	0.99	0.70	**0.82**	0.28	0.28	**0.27**
Italian	Inductive	51	0.98	0.14	0.24	0.21	0.20	0.21
	Incremental	+13	0.99	0.49	**0.65**	0.27	0.26	**0.27**
Spanish	Inductive	53	0.98	0.13	0.24	0.2	0.19	0.19
	Incremental	+21	0.99	0.71	**0.83**	0.28	0.27	**0.27**

Table 6

Results of the inductive and incremental evaluation using the QALD-9 test dataset. P refers to Precision, R refers to Recall, and F_1 refers to F_1-Measure.

entries. As explained in Section 3.1.2, we provide samples for each syntactic frame to the language expert, who then creates the sentence templates. These samples were taken from the QALD-9 training data, ensuring that the test data remains unseen by the language expert. Thus, in our inductive evaluation, we follow the same procedure typically used to evaluate machine learning approaches.

Table 6 shows the results for the QALD-9 test dataset for English, German, and Italian. As can be seen from the Table: (i) the micro F_1-measure in inductive mode is 0.29 for English; (ii) the score is 0.23 for German; (iii) the score is 0.24 for Italian; and (iv) for Spanish, the score is 0.24.

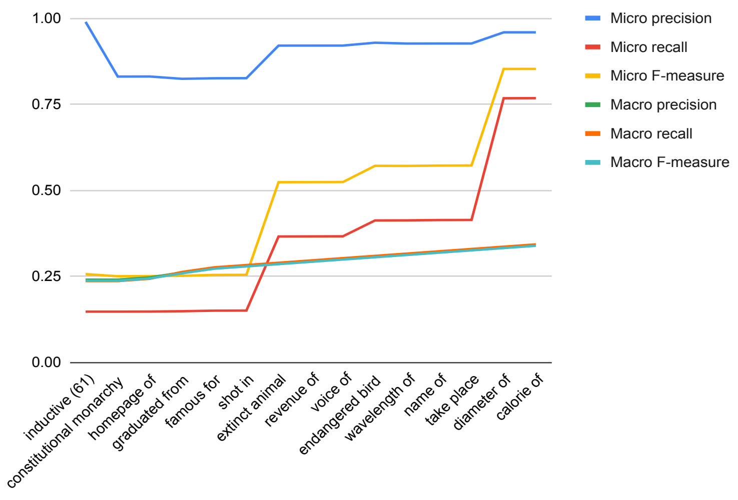
5.3. *Incremental Evaluation*

In an *incremental evaluation*, we start with the lexicon created for the inductive evaluation and then add lexical entries incrementally until the questions in the test data are covered. The benefits of the incremental evaluation are that (i) a working version of the QA system for a particular language is ready for use after adding just a few lexical entries, and (ii) each lexical entry added to the lexicon improves the system. The goal of incremental evaluation is to maintain and enhance system performance in a dynamic and efficient manner, ensuring that the system remains robust and effective as new information becomes available.

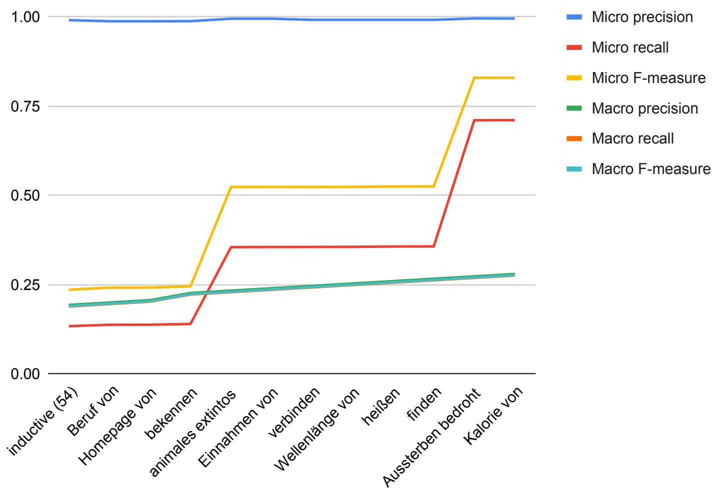
We argue that our approach is indeed generalizable, as we use only the sentence templates created by the language expert for inductive evaluation, in which the test data remains unseen by the language expert. In the incremental evaluation, we only extend the lexicon at each step for the test data, but we never extend the set of sentence templates, so the grammar remains the same. Nevertheless, the performance improves when we extend the lexicon. Thus, the sentence templates created for the training data allow the approach to generalize and perform well on test data.

An important aspect is that we assume that there is a lexicon so all our results hinge on the availability of such a lexicon. While one could argue that in the incremental evaluation we are relying on test data to create the lexicon (not the sentence templates), we view this as simulating that there is a fairly complete lexicon by incorporating lexical entries relevant for test data. In our view, this is not different from assuming the availability of some other lexical resource such as WordNet [33], FrameNet [32], ConceptNet [48] etc. The only difference is that we are "simulating" the availability of such a resource. We believe that our approach outperforms ML-based approaches under the assumption that a well-populated lexicon exists. Because, once we add the missing entries to the lexicon, our approach outperforms existing ML-based approaches.

Table 6 shows the results for the QALD-9 test dataset for English, German, and Italian. The table also shows the lexical entries common in both the training and test datasets and additional lexical



(a) English



(b) German

Fig. 10. Results of the incremental evaluation using QALD-9 test data for English (top) and German (bottom). The Figure shows precision, recall, and F_1-measure after adding a lexical element.

entries created to cover cases in the test data. For example, for English, 61 lexical entries are required both for the training dataset and the test dataset and an additional 14 lexical entries are created to cover the test data.

Figure 10 shows the results of the incremental evaluation that begins with the lexicon created for the inductive evaluation and shows the impact of each additional lexical entry added to the lexicon. Note that, no sentence templates are added, so the grammar remains the same. For English, the evaluation begins with the result of the inductive evaluation (i.e., F_1-measure 0.29) and then the result is improved (i.e., F_1-measure 0.85) by incrementally adding 14 lexical entries. Similarly, the F_1-measure for German is improved from 0.23 to 0.82 by incrementally adding 11 lexical entries.

As can be seen from Table 6: (i) the micro F_1-measure in incremental mode is 0.85 for English; (ii) the score is 0.82 for German; (iii) the score is 0.65 for Italian; and (iv) for Spanish, the score is 0.83. So far there is no system described in the literature that works for four languages while reaching state-of-the-art performance on all of them.

5.4. Case Analysis

In order to illustrate the workings of our system, we analyze its behaviour in more detail by discussing four types of cases. We sample all questions from the QALD-9 training set and classify them into four cases[20] to highlight interesting aspects of its behaviour.

Case 1 (Exact SPARQL Query): There are many cases for which the parsing result of the question is exactly the same as specified in the QALD-9 dataset. This is the case for 196 questions (i.e., 45%) in QALD-9. An example question is *'Who developed Skype?'*. Given that we model the lexical entry *(to) 'develop'* (i.e., `TransitiveFrame`) pointing to the property *dbo:developer* as semantics, specify that the syntactic object maps to domain of the property, the QA system is able to parse the question using the grammar rule (as detailed in Section 2.5.2) `Who [VerbPast] <NP`$_{\texttt{map(SyntacticFunction), Property}}$`>?` and returns the exact SPARQL query as specified in the QALD-9 dataset. Table 7 shows the questions in three languages and the grammar rules for each of these questions.

Lang.	**QALD-9**	**Lexical Entry**	**Grammar Rule Template (GRT)**
English	*'Who **developed** Skype?'.*	*(to) 'develop'*	`Who [VerbPast] <NP`$_{\texttt{map(SyntacticFunction), Property}}$`>?`
German	*'Wer **entwickelt** Skype?'*	*'entwickeln'*	`Wer [VerbPast] <NP`$_{\texttt{map(SyntacticFunction), Property}}$`>?`
Italian	*'Chi **sviluppato** Skype?'*	*'sviluppare'*	`Chi [VerbParticiple] <NP`$_{\texttt{map(SyntacticFunction), Property}}$`>?`
Spanish	*'Quien **desarrollado** Skype?'*	*'desarrollar'*	`Quien [VerbParticiple] <NP`$_{\texttt{map(SyntacticFunction), Property}}$`>?`

Table 7

Case 1: An example of a lexical entry for which 100% F_1-measure is obtained. In this case, the parsing result is the same SPARQL query as specified in the QALD-9 dataset.

Case 2 (Similar labels): A second case is one where our system is able to parse a question that is equivalent to the question in QALD-9 apart from the labels used for entities. This is the case for 31 questions (i.e., 8%). An example is the QALD-9 question *'List all boardgames by GMT.'* The lexical entry involved in generating the grammar rule is *'boardgame (by)'*. The semantics of the entry is captured with respect to the property *dbo:publisher*, the subject of the property is *dbo:VideoGame*, and the object of the property is *dbo:Company*. The question is parsed using the grammar rule
S -> *'List all boardgames by <NP$_{\text{Range, dbo:publisher}}$>?'*. For this grammar rule, the values of the non-terminal <NP$_{\texttt{Range, dbo:publisher}}$> are retrieved with the following automatically generated SPARQL query:

```
<NP_Range, dbo:publisher> -> eval(SELECT ?label WHERE {
                                 ?subject dbo:publisher ?object.
                                 ?object rdfs:label ?label . })
```

```
<NP_Range, dbo:publisher> -> Cambridge University Press | GMT GAMES |...
```

For matching the entity *'GMT'* of the question *'List all boardgames by GMT.'* with these terminals, we used the Jaccard similarity measure (as detailed in Section 2.7) and set a similarity threshold of 0.5. Each terminal (e.g., *'Cambridge University Press'*, *'GMT GAMES'*, etc.) is converted into a set of

[20] 32% of the questions do not belong to any of these classes.

tokens, which is then compared with the set of tokens (i.e., *'GMT'*) from the entity of the question. The Jaccard similarity method finds the similarity by the size of the intersection divided by the size of the union of the two sets. The best match of the token *'GMT'* is *'GMT Games'* since the similarity score is greater than/equal to the threshold 0.5. The parse result is as follows:

```
SELECT ?uri WHERE { ?uri dbo:publisher res:GMT_Games . }
```

Case 3 (Multiple properties): There are cases where a question in QALD-9 is parsed by our QA system, but the corresponding SPARQL query differs. This is the case for 23 questions (i.e., 6%) in QALD-9. The question *'Which organizations were founded in 1950?'* is parsed by our system, relying on the grammar rule S -> *Which organizations were founded in <$NP_{\text{Range, dbo:formationYear}}$>?'* The lexical entry is *'founded' (in)*, the subject of the property is *dbo:Organisation*, and the object of the property is *xsd:gYear*.

The SPARQL query contained in the QALD-9 dataset for the question consists of UNION operations involving multiple properties:

```
SELECT DISTINCT ?uri WHERE { ?uri rdf:type dbo:Organisation
                           { ?uri dbo:formationYear ?date } UNION
                           { ?uri dbo:foundingYear ?date } UNION
                           { ?uri dbp:foundation ?date } UNION
                           { ?uri dbp:formation ?date }
                           FILTER regex(?date, "^1950") }
```

In contrast, the SPARQL query generated by our approach consists of one property. The parse result of the question is as follows:

```
SELECT ?uri WHERE {
       ?uri rdf:type dbo:Organisation .
       ?uri dbo:formationYear "1950"^^<http://www.w3.org/2001/XMLSchema#gYear> . }
```

Case 4 (ASK queries): Our approach can parse boolean questions (i.e., ASK queries that have a Yes/No answer). Consider the question *'Is Rita Wilson the spouse of Tom Hanks?'* Our approach can parse this question by the grammar rule: *Is <$NP_{\text{Range, dbo:spouse}}$> spouse of <$NP_{\text{Domain, dbo:spouse}}$>?* The rule is automatically generated by instantiating the grammar rule template (as detailed in Section 2.5.1) using the lexical entry *'spouse' (of)*. The semantics of the entry is captured with respect to the property *dbo:spouse*, the syntactic subject maps to the object of the property *dbo:Person*, and the syntactic object maps to the subject of the property *dbo:Person*. This is the case for 37 questions (i.e., 9%) in QALD-9.

To the best of our knowledge, there is only one knowledge graph based QA system described in the literature that can handle limited forms of boolean questions (i.e., temporal questions) [49] In contrast, the automatically generated grammar rules in our approach allows us to parse a wide range of boolean questions from RDF data.

5.5. *Error Analysis*

In order to better understand the potential errors that our system produces, we manually analysed errors on the QALD-9 training data, yielding nine types of errors:

- **Complex queries:** Our approach supports a limited form of composition (as detailed in Section 2.8) by nesting rules that generate noun phrases into the rules for the other frames. 31% of errors are due to the limited support of some variations of complex queries. Therefore, the approach can not handle certain type of complex questions such as *'Give me all people that were born in Vienna and died in Berlin.'* The SPARQL query is as follows:

```
SELECT DISTINCT ?uri WHERE { ?uri dbo:birthPlace res:Vienna .
                             ?uri dbo:deathPlace res:Berlin . }
```

While in principle the question could be parsed by the grammar rule S -> *Give me all people that were born in* <*NP*$_{\text{Range, dbo:birthPlace}}$>., our approach can at the moment not handle conjunctions, neither at the verb phase nor noun phrase level.

- **Abbreviated label**: 5% of errors are due to the abbreviated label of entities in the QALD-9 question. Consider the QALD-9 question *'Where was JFK assassinated?'* and its SPARQL query:

```
SELECT DISTINCT ?uri WHERE { res:John_F._Kennedy dbo:deathPlace ?uri }
```

We used the Jaccard similarity measure (as detailed in Section 2.7) to match the entity (of the question) with the label of the DBpedia entity and set a similarity threshold of 0.5. Our system fails to parse the question as the similarity score of string *'JFK'* and *'John F. Kennedy'* is below 0.5.

- **Wrong label**: 7% of errors are due to the wrong label of entities in the QALD-9 question. Consider the QALD-9 question *'Who wrote the book Les Piliers de la terre'* and its SPARQL query:

```
SELECT DISTINCT ?uri WHERE {res:The_Pillars_of_the_Earth dbo:author ?uri }
```

The question contains the French label (i.e., *'Les Piliers de la terre?'*) of the entity *res:The_Pillars_of_the_Earth* instead of the English label (i.e., *'The Pillars of the Earth'*).

- **SPARQL query without property:** There are cases (i.e., 8%) where the SPARQL query of QALD-9 question does not contain any property. Consider the question *'How many airlines are there?'* and its SPARQL query:

```
   SELECT (COUNT(DISTINCT ?uri) AS ?c) WHERE { ?uri a dbo:Airline }
```

In our approach, however, the semantics of all lexical entries for frames is specified in relation to one property. Thus, in order to successfully parse a question, the question needs to have at least one noun or verb (as detailed in Section 2.3) referring to a property. This issue could be solved by a simple grammar rule of the form *'How many X are there'* where X refers to all possible classes in DBpedia.

- **Entity-less questions:** All our grammar rules assume that at least one of the syntactic positions is filled with a specific entity at subject or object position of the corresponding property. There are however questions in QALD that mention no specific entity. An example is the following question: *'What is the most frequent cause of death?'* and its SPARQL query:

```
   SELECT DISTINCT ?x WHERE {
                   ?uri dbo:deathCause ?x }
                   ORDER BY DESC(COUNT(DISTINCT ?uri)) OFFSET 0 LIMIT 1
```

Our grammar rules generated for the lexical entry *'death cause' (of)* could instead parse questions involving at least one entity (that died), e.g.
'What is the cause of death of Calel Perechodnik?' as

```
   SELECT DISTINCT ?x WHERE {
                   res:Calel_Perechodnik dbo:deathCause ?x }
```

- **Adjective questions:** 7% of errors are due to the missing support for different variations of superlative questions (e.g., the question *'Which Indian company has the most employees?'*,*'What* *is the bridge with the longest span?'* Our approach can only process the canonical forms: *'What is the highest mountain in Australia?'* or *'What is the longest river?'*, respectively.

- **missing templates:** 14% of errors are due to missing templates. This is the case for questions such as *'In which countries can you pay using the West African CFA franc?'* or *'Which ingredients do I need for carrot cake?'*, among others. The sentence templates are created by language expert, so it is unclear why certain templates were not created. To assist in creating sentence templates, we provide a few examples along with a dictionary of syntactic categories (detailed in Section 3.1.2) to the language expert. However, no examples are provided using the first person *'I'* or second person *'You'*, which may discourage the language expert from creating such templates. For example, the language expert created templates for questions like *'What are the ingredients of carrot cake?'* and *'Give me all the ingredients of carrot cake?'* (detailed in Section 2.5.1), but not for *'Which ingredients do I need for carrot cake?'* or *'Which ingredients do you need for carrot cake?'*.
  Another example of a missing template involves temporal questions, such as the question *'Give me all libraries established earlier than 1400.'*. 4% of errors are due to the missing support for different variations of temporal questions. Instead, our approach can parse the question *'Give me all libraries established in 1400'* using the grammar rule: S -> *'Give me all libraries established in <NP$_{\text{Range, dbo:established}}$>?'* relying on the lexical entry *'establish' (in)*.
- **Spurious matching:** 8% of errors are due to the fact that we retrieve the wrong entity using the Jaccard similarity. Consider the QALD-9 question *'Where was Bach born?'* The QA system parses the question using the grammar rule S -> *'Where was <NP$_{\text{Domain, dbo:birthPlace}}$>?'* relying on the lexical entry *'born' (in)* The parse is as follows:

  ```
  SELECT ?uri WHERE { res:Bach_(actor) dbo:birthPlace ?uri }
  ```

  But the SPARQL query of the question contained in QALD-9 is:

  ```
  SELECT DISTINCT ?uri WHERE { res:Johann_Sebastian_Bach dbo:birthPlace ?uri }
  ```

  Note that the entity *'Bach'* of the question does not match with the entity *'Johann Sebastian Bach.'* Instead, the highest ranked entity according to the Jaccard similarity score is *res:Bach_(actor)*.

5.6. *Comparison to state-of-the-art English QA systems*

In this section, we discuss question-answering systems evaluated on the QALD benchmark and compare their performance (i.e., F$_1$-Measure) with our approach. To compare our system with these approaches, we evaluate our system both in inductive and incremental modes.

The goal of most QALD systems is to map natural language questions to corresponding SPARQL queries. State-of-the-art question-answering systems use machine learning to learn a model to map natural language questions into SPARQL queries. These systems support in most cases only the English language.

5.6.1. *Comparision with ML-based approach*

There is a fundamental difference between how machine learning models are extended and how our model-based approach is extended. While machine learning-based approaches require extensive training data to learn to handle phenomena that are either absent or not sufficiently prominent in the training set, our approach operates independently of training data. Instead, the capabilities of our system can be increased by extending the lexicon, which is a completely different paradigm. This allows us to improve our system step by step (as detailed in Section 5.3).

The incremental improvement of our approach can be compared to experiments on the continuous improvement of machine learning approaches; however, such experiments are typically outside the

QALD	**Dataset**	**System**	**Micro P**	**Micro R**	**Micro F_1**
QALD-9	Test	Elon [47]	0.04	0.05	0.10
		gAnswer [50]	0.29	0.32	0.43
		NSQA [51]	0.31	0.32	0.45
		WDAqua-core1 [52]	0.26	0.26	0.28
		QASystem [47]	0.09	0.11	0.20
		Zheng et al. [53]	0.45	0.47	0.46
		Falcon 1.0 [54]	0.23	0.23	0.23
		SLING [55]	0.39	0.50	0.44
		GenRL [56]	0.49	0.61	0.53
		EDGQA [57]	0.31	0.40	0.32
		KGQAN [58]	0.49	0.39	0.43
		Galactica [59]	0.14	0.02	0.03
		TeBaQA [60]	0.24	0.24	0.37
		our approach (inductive, L=61)	0.98	0.14	0.25
		our approach (incremental (L=+14)	**0.95**	**0.76**	**0.85**
QALD-7	Train	WDAqua-core1 [52]	0.37	0.39	0.39
		our approach	**0.79**	**0.93**	**0.86**
QALD-6	Test	AMUSE [61][21]	-	-	0.34
		our approach (inductive, L=53)	0.42	0.50	0.45
		our approach (incremental (L=+47)	**0.79**	**0.83**	**0.82**
QALD-5	Test	Xser [62]	0.74	0.72	0.73
		UTQA [63]	0.55	0.53	0.54
		AskNow [64]	0.32	0.34	0.33
		SemGraphQA [65]	0.19	0.20	0.20
		YodaQA [66]	0.18	0.17	0.18
		our approach (incremental, L=38)	**0.76**	**0.75**	**0.75**
QALD-3	Test	gAnswer [65]	0.40	0.40	0.40
		CASIA [67]	0.35	0.36	0.36
		Scalewelis [68]	0.33	0.33	0.33
		RTV [69]	0.32	0.34	0.33
		Intui2 [70]	0.32	0.32	0.32
		SWIP [71]	0.16	0.15	0.16
		DEANNA [72]	0.21	0.21	0.21
		our approach (inductive, L=23)	0.41	0.46	0.44
		our approach (incremental, L=+17)	**0.75**	**0.83**	**0.78**

Table 8

Results for machine learning and rule-based systems for English in different QALD series. L refers to the number of lexical entries, P refers to Precision, R refers to Recall, and F_1 refers to F_1-Measure.

scope of ML papers, and we lack evidence on how effectively ML methods can be incrementally improved by modifying the training data. There are approaches, such as [73], that support incremental improvement of QA systems without re-training. However, to the best of our knowledge, this approach has not been applied to linked data, so it remains unclear how machine learning-based QALD systems can be incrementally improved. Therefore, we explain existing ML-based QALD systems and their limitations, highlighting the advantages of our approach for incremental improvements.

In neural network-based QALD systems [7, 16, 74, 75], the problem of transforming natural language into a SPARQL query is typically framed as a semantic parsing problem. For example, the transfer learning-based QALD approach [13] uses a fine-tuned BERT model [76] and decomposes question interpretation into two separate learning tasks: (1) entity span detection, which identifies the spans in the question that mention the entity, and (2) relation classification, which predicts the relation used in the question. In the question *'Who is the mayor of New York City?'*, the entity span would be *'New York City'*, and the relation span would be *'mayor'*. The limitation is that linked data uses a formal structure, with specific ontologies, technical terminologies, and relationships that are less common in natural language text. In the context of incremental improvement, our approach can be easily extended for questions like *'When was Carlo Giuliani shot?'* by adding a lexical entry such as *'shot in'* for the property *dbo:deathDate* (detail in Section 5.3). However, pre-trained models may not have encountered enough examples of these specific patterns during training, even though they are trained on large corpora of text. Moreover, while transfer learning-based approaches are typically employed only for simple questions, our approach supports nesting grammar rules (detail in Section 2.8) and a limited set of compositionality.

There are approaches that translate natural language questions into SPARQL queries [14, 77–79], which are vocabulary-dependent and require training on samples derived from manually created QA template pairs. Some approaches use a model for joint query interpretation and response ranking [80]; others learn semantic parsers [81, 82] given a knowledge base and a set of question/answer pairs. Other approaches, such as TeBaQA [60], map natural language questions to SPARQL queries by learning templates from the benchmark (e.g., the QALD-9 dataset [47]). It classifies natural language questions into isomorphic basic graph patterns and then uses this classification to map a question to a template. However, the training dataset is small, consisting of only 403 questions, which limits the ability to learn templates. These approaches rely on either manually created training data, which is time-consuming and expensive, or the availability of benchmarks. On the other hand, our model-based approach can be incrementally improved by adding lexical entries to the lexicon; the time required for creating a lexical entry is approximately 5–10 minutes (detail in Section 3.1.1).

5.6.2. *Systems evaluated on QALD-9*

TeBaQA [60], map natural language questions to SPARQL queries by learning templates from the benchmark. As shown in Table 8, TeBaQA achieves an overall F_1-Measure of 0.37 without any manual effort, whereas our model-based approach achieves 0.85 with very little manual effort. gAnswer [50] is a graph-based approach that maps a natural language question into a semantic query graph containing an edge for each relation mentioned in the question. After that, it simplifies the transformation of natural language into a SPARQL query by converting it into a subgraph matching problem. Similarly, EDGQA [57] introduces a graph structure called the Entity Description Graph (EDG), which provides a hierarchical decomposition of a natural language question for generating and composing SPARQL queries. The graph-based approaches achieve an overall F_1-Measure ranging from 0.30 to 0.45, which is better than the machine learning approaches. The incremental evaluation shows that our approach achieves a 30-40% improvement over this.

SLING [55] and GenRL [56] are relationship linking frameworks developed for question-answering over linked data. The former approach employs semantic parsing using Abstract Meaning Representation (i.e., a rooted, directed, acyclic graph expressing a question) to extract entities and relations between them, utilizing linguistic cues and information from DBpedia. The latter approach utilizes pre-trained sequence-to-sequence models, using only the question text to generate a sequence of relations based on the question. It also employs knowledge integration and validation mechanisms to address the nuances of the knowledge bases. However, the work does not explain the process of constructing a SPARQL query from the sequence of relations. The relation linking-based approaches achieve the highest overall F_1-Measure (i.e., ranging from 0.40 to 0.55) among all existing systems evaluated on the QALD-9 dataset, whereas our model-based approach outperforms these methods when evaluated in incremental mode.

5.6.3. Systems evaluated on QALD-7 to QALD-3

We also compare our approach with systems evaluated on older QALD benchmarks, such as QALD-7 [45] to QALD-3. CASIA [67] is a machine learning-based QA system that translates natural language questions into SPARQL by the following pipeline: (i) first, it decomposes the question and detects the candidate phrases; (ii) second, it maps the phrases to semantic items; (iii) finally, it groups the mapped items into semantic triples using a Markov Logic Network (MLN). A similar approach is RTV [69] that selects key ontological information from the question (i.e., predicate, arguments and properties) using a Hidden Markov Model.

These approaches, together with others (i.e., AMUSE [61], Scalewelis [68], Intui2 [70], SWIP [71], ForecastTKGQuestions [49], and DEANNA [72]), were evaluated by inducing a model from training data and testing the model on test data. Table 8 shows the comparison of our approach with these machine learning-based and rule-based systems for English. The incremental evaluation shows that our approach outperforms all machine learning approaches.

Furthermore, Table 8 also shows the results of the evaluation of our system in comparison to rule-based systems such as Xser [62], SemGraphQA [65], and YodaQA [83] for English. Xser [62] takes as input a natural language question in English and retrieves an answer. In this system, the user query is linguistically analyzed to detect predicate-argument structures through a semantic parser. SemGraphQA [65] transforms natural language questions into SPARQL queries. YodaQA [83] represents the input question as a bag of features and then generates a set of candidate answers by performing a search in the knowledge base for these features. All these rule-based QA systems are evaluated over the QALD-5 test dataset in incremental mode and we see that our system outperforms all systems.

5.7. Performance on other languages

A survey [84] of the multilingual question-answering systems over linked data shows that 10% of them were applied to other than the English language and 5% of them were applied to more than one RDF dataset. Table 9 lists multilingual QA systems over linked data that were developed in the last 10 years.

System	**Language other than English**
QALL-ME [85]	German, Italian, and Spanish
XKnowSearch! [86]	German and Chinese
UTQA [63]	Spanish, Finnish, and Swedish
BreXearch [87]	German and Spanish
AMUSE [61]	German and Spanish
DeepPavlov [88]	Russian
Platypus [89]	French and Spanish
WDAqua-core1 [52]	German, French, Italian, and Spanish
QAnswer [90]	English, German, French, Italian, Spanish, Portuguese, Arabic, and Chinese
Zhou Y. [91]	Farsi, German, Romanian, Italian, Russian, French, Dutch, Spanish, Hindi, and Portuguese

Table 9

The table presented in the Multilingual Question Answering System survey [92] illustrates the development of multilingual question-answering systems over the last 10 years.

QALL-ME [85] is a multilingual Question-answering (QA) system that operates on structured data and has been developed for cinema/movie events in the tourism domain. Multilinguality is achieved by an ontology modelling the domain, which also represents a limitation of the framework since an available ontology has to be previously specified. XKnowSearch! [86] and BreXearch [87] are multilingual semantic search systems that extract knowledge from various media sources, including online news sites, social media, and live TV, and integrate it with additional background knowledge from DBpedia. The approach allows to ask questions not through natural language questions but with keyword queries or by using entities or SPARQL queries. These systems are focused on keyword-based searches instead of mapping natural language questions to SPARQL queries, and no evaluation is provided.

The UTQA [63] system uses a QA pipeline of several language-dependent components, such as keyword extraction and type identification, entity linking, and answer extraction. The DeepPavlov [88] KBQA system works based on a set of large language-dependent deep learning models that perform such tasks as query type prediction, entity detection, relation detection and path ranking. The Platypus system [89] transforms a natural language question into a custom logical representation using a language-dependent grammatical analyzer and a set of predefined rules. From a broad perspective, all these systems are focused on solving the QA problem using language-independent tools, which attracts a wider NLP audience instead of the Question Answering over Linked Data (QALD) community. Therefore, these systems lack evaluations against benchmarks on QALD.

To our knowledge, AMUSE [61] is the first machine learning-based multilingual QA system developed for languages other than English. The questions are first parsed by a dependency parser and then the model learns mappings between the dependency parse tree for a given question text and RDF nodes in the SPARQL query. AMUSE relies on the one hand on lexicalizations of DBpedia properties as extracted by M-ATOLL [93] for multiple languages. In particular for Spanish and German. However, M-ATOLL uses two methods for lexicalization: a label-based approach to extract lexicalizations using ontology labels (i.e., *rdfs:label*) and a dependency-based approach to extract lexicalizations of ontology properties from an available text corpus (i.e., Wikipedia text). The former method suffers from a lexical gap between natural language terms and the content of KB elements, and the latter method produces every other lexical entry inappropriate. Moreover, the approach is not designed to be deployed in a fully unsupervised mode to a new language.

Table 10 shows the results of the inductive evaluation and the incremental evaluation of our approach. The table also shows the lexical entries common in both the training and test datasets and additional lexical entries created over training data to cover the test. For instance, in the case of German, 41 lexical entries are required both for the training dataset and the test dataset, and an additional 45 lexical entries are created for incremental evaluation. Our approach outperforms the AMUSE system for both German and Spanish. The results show that the results measured in the inductive evaluation for German are improved from 0.44 to 0.75 by adding 45 additional lexical entries incrementally. Similarly, for Spanish, it is improved from 0.40 to 0.83 by adding 38 lexical entries.

WDAqua-core1 [52] is a rule-based approach that translates natural language questions to SPARQL queries. The approach is implemented for different languages other than English namely German, French, Italian, and Spanish. As can be seen from Table 10, our approach outperforms the WDAqua-core1 system for all languages.

5.8. Comparsion with ChatGPT

ChatGPT [26] is a pre-trained large language model that can answer questions, ask for clarification, create dialogues with the user, and deal with follow-up questions. It is trained using reinforcement learning with human feedback, by using a combination of a new dialogue dataset and the InstructGPT dataset. Previous research [95] shows that language models can, to some extent,

QALD	**Language**	**QA System**	**Micro P**	**Micro R**	**Micro F_1**
QALD-6 (Test)	German	AMUSE [61][22]	-	-	0.37
		our approach (ind*) (L=41)	0.41	0.46	0.44
		our approach (inc*) (L=+45)	**0.77**	**0.73**	**0.75**
	Spanish	AMUSE [61]	-	-	0.42
		our approach (ind*) (L=+36)	0.40	0.41	0.40
		our approach (inc*) (L=+41)	**0.78**	**0.88**	**0.83**
QALD-7 (Train)	German	WDAqua-core1 [52]	0.27	0.28	0.27
		our approach	**0.74**	**0.71**	**0.73**
	Italian	WDAqua-core1 [52]	0.90	0.20	0.18
		Nolano et al. [94]	0.48	0.22	0.30
		our approach	**0.72**	**0.85**	**0.78**
	Spanish	WDAqua-core1 [52]	0.31	0.32	0.31
		our approach	**0.84**	**0.78**	**0.81**

Table 10

Results for machine learning and rule-based approaches for German, Italian, and Spanish. ind* refers to inductive, inc* refers to incremental, and L refers to the number of lexical entries, P refers to Precision, R refers to Recall, and F_1 refers to F_1-Measure.

be considered as knowledge bases (KBs) to support downstream natural language processing (NLP) tasks. This has sparked a growing interest in exploring whether ChatGPT and other large language models (LLMs) can replace traditional Question Answering over Knowledge Graphs [96] approaches. Benefiting from extensive training on information derived from Wikipedia and showcasing impressive natural language understanding capabilities, ChatGPT has been demonstrated to be a powerful question-answering system [27, 97].

5.8.1. Prompt

In this section, we compare how ChatGPT (i.e., a language model) performs compared to our approach of question answering over linked data. In particular, we evaluated our approach against ChatGPT using the well-known QALD-9 benchmarks on September 5, 2024. We used ChatGPT 4 (i.e., GPT-4) in a zero-shot and few-shot scenario. In zero-shot scenario, we prompted it with an instruction to generate a SPARQL query for a given question with respect to DBpedia without providing further examples. The prompt for English is shown below:

```
Generate a SPARQL query for DBpedia to answer the following question: "QUESTION".
Ensure the query retrieves relevant information efficiently, using appropriate
filters, properties, and namespaces. Do not explain anything.
```

In a few-shot scenario, we prompted it with an instruction to generate a SPARQL query for a given question in relation to DBpedia, providing one example for each of the five syntactic frames (i.e., `NounPPFrame`, `TransitiveFrame`, `InTransitivePPFrame`, `AdjectivePredicateFrame`, `AdjectiveSuperlativeFrame`) in each language. Here is an example for English. The prompts for other languages (German, Italian, and Spanish) are provided in the Appendix.

```
You are an assistant that has the task to generate a SPARQL query for DBpedia to
answer a given question. Before I show you the question, I show you a couple of
examples of questions and the corresponding SPARQL queries.

Question: Who is the mayor of New York City?
```

```
SPARQL: SELECT DISTINCT ?uri WHERE { <http://dbpedia.org/resource/New_York_City>
<http://dbpedia.org/ontology/leaderName> ?uri }

Question: Give all swimmers that were born in Moscow.
SPARQL: SELECT DISTINCT ?uri WHERE { ?uri a <http://dbpedia.org/ontology/Swimmer> ;
<http://dbpedia.org/ontology/birthPlace> <http://dbpedia.org/resource/Moscow> }

Question: Who created Goofy?
SPARQL: SELECT DISTINCT ?uri WHERE { <http://dbpedia.org/resource/Goofy>
<http://dbpedia.org/ontology/creator> ?uri }

Question: Give me all Danish films.
SPARQL: SELECT DISTINCT ?uri WHERE { ?uri
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://dbpedia.org/ontology/Film>
; <http://dbpedia.org/ontology/country>  <http://dbpedia.org/resource/Denmark> }

Question: What is the highest mountain?
SPARQL: SELECT DISTINCT ?uri WHERE { ?uri a <http://dbpedia.org/ontology/Mountain>
; <http://dbpedia.org/ontology/elevation> ?elevation } ORDER BY DESC(?elevation)
OFFSET 0 LIMIT 1

Now generate a SPARQL query for DBpedia to answer the following question:
"QUESTION". Ensure the query retrieves relevant information efficiently,
using appropriate filters, properties, and namespaces. Do not explain anything.
```

Language	**System**	**Micro P**	**Micro R**	**Micro F_1**	**Macro P**	**Macro R**	**Macro F_1**
English	ChatGPT (zero shot)	0.35	0.06	0.11	0.05	0.05	0.05
	ChatGPT (few shot)	0.90	0.19	0.32	0.29	0.28	0.28
	our approach (ind*)	0.72	0.18	0.29	0.22	0.21	0.21
	our approach (inc*)	**0.91**	**0.79**	**0.85**	**0.31**	**0.30**	**0.30**
German	ChatGPT (zero shot)	0.41	0.01	0.02	0.02	0.02	0.02
	ChatGPT (few shot)	0.92	0.13	0.23	0.22	0.21	0.21
	our approach (ind*)	0.99	0.13	0.23	0.19	0.18	0.19
	our approach (inc*)	**0.99**	**0.70**	**0.82**	**0.28**	**0.28**	**0.27**
Italian	ChatGPT (zero shot)	0.77	0.07	0.14	0.06	0.05	0.05
	ChatGPT (few shot)	0.82	0.21	0.33	0.19	0.18	0.18
	our approach (ind*)	0.98	0.14	0.24	0.21	0.20	0.21
	our approach (inc*)	**0.99**	**0.49**	**0.65**	**0.27**	**0.26**	**0.27**
Spanish	ChatGPT (zero shot)	0.007	0.01	0.01	0.05	0.05	0.05
	ChatGPT (few shot)	0.93	0.17	0.29	0.19	0.19	0.18
	our approach (ind*)	0.98	0.13	0.24	0.2	0.19	0.19
	our approach (inc*)	**0.99**	**0.71**	**0.83**	**0.28**	**0.27**	**0.27**

Table 11

Results for ChatGPT for QALD-9 test dataset. ind* refers to inductive, inc* refers to incremental, P refers to Precision, R refers to Recall, and F_1-Measure refers to F_1-Measure.

Table 11 shows the results for the QALD-9 test dataset for four languages. The table further illustrates the evaluation of our approach to the QALD-9 test dataset in both inductive and incremental modes. The table highlights a comparative performance between our approach and ChatGPT across various scenarios, specifically focusing on zero-shot and few-shot learning across different languages. As shown in the table, our approach outperforms ChatGPT in zero-shot scenarios across four languages, in both inductive and incremental modes.

However, in the few-shot scenario, ChatGPT outperforms our approach in the inductive mode. ChatGPT is trained on an extensive corpus of diverse, high-quality data from various domains, enabling it to understand a wide range of topics and questions. It is unclear whether the QALD-9 dataset was part of the data used to train ChatGPT. Nevertheless, our approach outperforms ChatGPT in the few-shot scenario in the incremental evaluation mode.

5.8.2. *Case Analysis*

In order to compare the results of ChatGPT with those produced by our system, we manually analyzed the errors and non-errors in the first 100 questions of the QALD-9 test data for English in a few-shot scenario, identifying six types [23] of cases:

Case 1 (Exact SPARQL query): There are only a few instances (i.e., 21%) where the SPARQL query generated by ChatGPT is an exact match with the one specified in the QALD-9 dataset. This holds true for 45% of the questions in our approach. An example of such a question is *'What is the time zone of Salt Lake City?'*

```
SELECT DISTINCT ?uri WHERE { res:Salt_Lake_City dbo:timeZone ?uri }
```

Case 2 (Multiple properties): In many cases (i.e., 41%), the SPARQL query generated by ChatGPT is different but represents a correct interpretation of the QALD-9 question. This is the case for 6% of the questions in our approach. Consider the question *'Which airports are located in California, USA?'* and its SPARQL query from the QALD-9 dataset and ChatGPT:

```
QALD-9: SELECT DISTINCT ?uri WHERE { ?uri a dbo:Airport
                   { ?uri dbo:location res:California }
                   UNION { ?uri dbo:city res:California }
                   UNION { ?uri dbo:city ?city . ?city dbo:isPartOf res:California}
                   UNION { ?uri dbo:operator res:California } }
ChatGPT: SELECT DISTINCT ?uri WHERE { ?uri a dbo:Airport ;
                                      dbo:location res:California }
```

As can be seen, the SPARQL query in QALD-9 uses several UNION operations with four properties (i.e., *dbo:location*, *dbo:city*, *dbo:isPartOf*, and *dbo:operator*), whereas the SPARQL query generated by ChatGPT contains only one property (i.e., *dbo:location*), does not use any UNION, yet still provides the correct interpretation of the question.

Case 3 (Different but correct property): There are instances (i.e., 16%) where the SPARQL query generated by ChatGPT contains a different property than the QALD-9 query. Consider the question *'What are the nicknames of San Francisco?'* and its SPARQL query from the QALD-9 dataset and ChatGPT:

```
QALD-9: SELECT DISTINCT ?uri WHERE { ?uri dbo:spokenIn res:Estonia }
ChatGPT: SELECT DISTINCT ?uri WHERE { res:Estonia dbo:language ?uri }
```

[23] 7% do not belong to any of these cases.

As can be seen, the SPARQL query generated by ChatGPT contains the property *dbo:spokenIn* instead of *dbo:language*.

Case 4 (Invented KB elements): In other cases (i.e., 8%), ChatGPT generates a SPARQL query that includes one or more properties, resources, or classes that do not exist in the corresponding RDF data. Consider the question *'When did Finland join the EU?'* along with its corresponding SPARQL query in the QALD-9 dataset and the query generated by ChatGPT:

```
QALD-9: SELECT DISTINCT ?date WHERE { res:Finland dbp:accessioneudate ?date }
ChatGPT: SELECT DISTINCT ?date WHERE { res:Finland dbo:euAccession ?date }
```

The SPARQL query of ChatGPT contains the property (i.e., *dbo:euAccession*), which does not exist in DBpedia.

Case 5 (Incorrect SPARQL query): In a few instances (i.e., 3%), the SPARQL query generated by ChatGPT fails to accurately represent the intended interpretation of the QALD-9 questions. For example, the SPARQL query generated by ChatGPT fails to capture the interpretation of Boolean questions (i.e., ASK queries that require a Yes/No answer) in QALD-9. Consider the question *'Are there any castles in the United States?'* and its SPARQL query from the QALD-9 dataset and ChatGPT:

```
QALD-9:  ASK WHERE { ?uri dct:subject dbc:Castles_in_the_United_States }
ChatGPT: SELECT DISTINCT ?uri WHERE { ?uri a dbo:Castle ;
                                      dbo:location res:United_States }
```

In some cases, ChatGPT generates a structurally incorrect SPARQL query. Consider the QALD-9 question *'Which countries have places with more than two caves?'* and its SPARQL query from the QALD-9 dataset and ChatGPT:

```
QALD-9: SELECT DISTINCT ?uri WHERE { ?cave rdf:type dbo:Cave ; dbo:location ?uri .
                ?uri rdf:type dbo:Country } GROUP BY ?uri HAVING (COUNT(?cave) > 2)
ChatGPT: SELECT DISTINCT ?country WHERE {?place dbo:cave ?cave ;
                                         dbo:isPartOf ?country .{SELECT ?place
                                         WHERE {?place dbo:cave ?cave }
                                         GROUP BY ?place HAVING (COUNT(?cave) > 2)}
                                        }
```

In this example, the SPARQL query generated by ChatGPT includes a nested SELECT statement, which is structurally incorrect.

Case 6 (Incorrect subject/object/property): In some cases (i.e., 2%), the SPARQL query generated by ChatGPT places the object in the subject position and vice versa. Consider the question *'Butch Otter is the governor of which U.S. state?'* and its SPARQL query from the QALD-9 dataset and ChatGPT:

```
QALD-9: SELECT DISTINCT ?uri WHERE {res:Butch_Otter dbo:governor ?uri}
ChatGPT: SELECT DISTINCT ?uri WHERE {?uri dbo:governor res:Butch_Otter}
```

In this example, the entity *res:Butch_Otter* represents a person and should be in the object position for the property *dbo:governor*. However, in the SPARQL query generated by ChatGPT, it appears in the subject position, which is incorrect.

There are also instances (i.e., 2%) where the SPARQL query generated by ChatGPT contains a different property than the QALD-9 query. Consider the question *'Who killed Caesar?'* and its SPARQL query from the QALD-9 dataset and ChatGPT:

```
QALD-9:  SELECT DISTINCT ?uri WHERE {?uri dct:subject dbc:Assassins_of_Julius_Caesar}
ChatGPT: SELECT DISTINCT ?uri WHERE {res:Julius_Caesar dbo:deathCause ?uri }
```

DataSet	**Porting Time (person-hours)**	**Language Frame**	**NounPP Frame**	**Transitive Frame**	**InTransitivePP**	**Total**	**Questions**
ArCo	9 hours	English	3	4	4	11	648
		Italian	3	2	2	7	352
Wikidata	12 hours	English	15	12	17	44	8, 203
		Italian	5	7	3	15	5, 043
		Bangla	10	9	2	21	3, 043
AIFB	8 hours	English	13	2	1	16	7, 087
		German	5	2	2	9	7, 022
LexInfo	4 hours	English	2	2	1	5	13
		Spanish	2	1	1	4	15

Table 12

The results of porting four new datasets (i.e., ArCo [98], Wikidata, AIFB dataset, and LexInfo) to the QA system for five languages (i.e., English, Italian, German, Spanish, and Bangla).

As can be seen, the SPARQL query generated by ChatGPT contains the property *dbo:deathCause* instead of the property-object pair (i.e., *dct:subject* and *dbc:Assassins_of_Julius_Caesar*) used in QALD-9. None of these cases are present in our approach.

5.9. *Portability to new datasets and languages*

The model-based approach can be easily adapted to a new RDF dataset and languages. To test the portability, we organized a one-week *Hackathon on question answering over RDF data*[24] and let users port our system to different datasets and languages. We provided a detailed guideline for adding a new dataset and language. The participants with different backgrounds tested the portability of our system for four datasets:

- ArCo dataset[25] [98] aggregates knowledge of Italian Cultural Heritage through the use of several conceptual ontologies and data from the Italian National Catalogue of Cultural properties. The dataset consists of a network of 7 vocabularies and 169 million triples about 820 thousand cultural entities, providing a SPARQL endpoint to access the data.
- Wikidata[26] contains 2.9 billion triples and supports some Indian languages such as Bangla and Tamil, which are not available in DBpedia.
- AIFB dataset[27] contains information on scientific publications such as publication titles, abstracts, authors, professors, date of publication, place of publication, etc.
- LexInfo[28] is an ontology that defines data categories for the Lemon model[29].

The QA system was ported to all these datasets by the participants of the Hackathon. One group of participants created lexical entries for both English and Italian on three frame types for the ArCo dataset. Another group of participants created lexical entries for three languages (i.e., Italian, English, and Bangla) for Wikidata. As per our knowledge, this is the first question-answering system over linked data for the Bangla language. The third group of participants created lexical entries for English and Spanish for the AIFB dataset and `LexInfo`.

[24]https://github.com/fazleh2010/Journal-Paper/blob/master/README.md
[25]https://dati.beniculturali.it/arco-rete-ontologie
[26]https://www.wikidata.org/wiki/Wikidata:Main_Page
[27]https://raw.githubusercontent.com/fazleh2010/question-grammar-generator/general2/dataset/aifbfixed_complete.ttl
[28]https://lexinfo.net/
[29]https://lemon-model.net/

Table 12 shows the results of porting these four new datasets to the QA system for 5 languages (i.e., English, Italian, German, Spanish, and Bangla).

As shown in the table, the approach allows non-expert users to port the question-answering system to a new RDF dataset by adding lexical entries for that dataset in only a few hours and with little training.

5.10. Consistency and Controllability

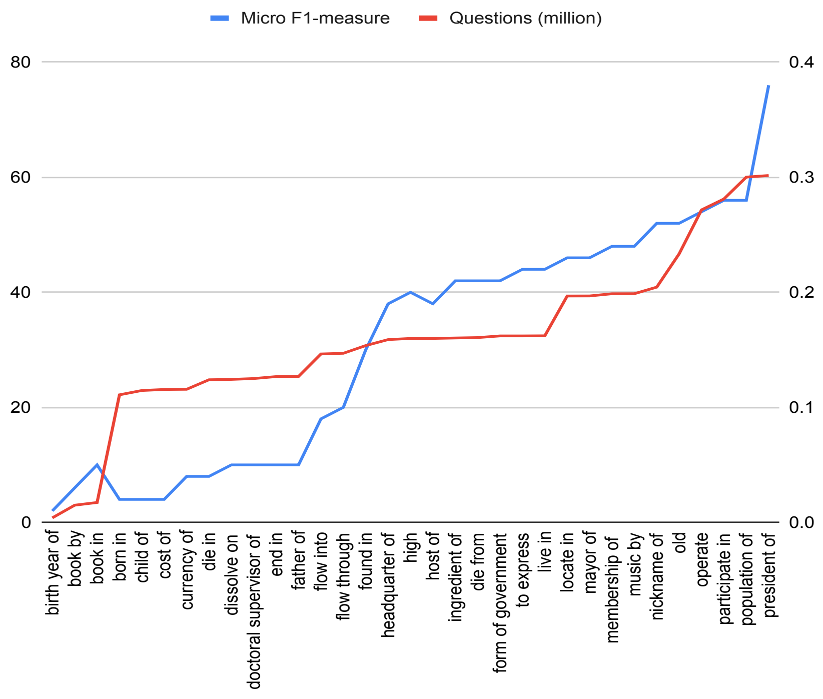


Fig. 11. Consistency: Result of the incremental extension of the QA system for 33 lexical entries. After each extension of the lexicon, we calculate the number of questions in millions (as shown on the left Y-axis) that can be parsed for each lexical entry and evaluate the F_1-measure (as shown on the right Y-axis).

Our QA system is consistent and controllable because if we add a lexical item to the lexicon, it guarantees that all questions involving a syntactic construction supported by the system in which the lexical item can be used will be parsed correctly. Figure 11 shows the number of questions that can be parsed for each lexical entry and the F_1-measure for the QALD-9 training dataset. As can be seen from the Figure, each lexical entry added to the lexicon allows the QA system to parse tens of thousands of questions and so the F_1-measure is improved incrementally.

Consider the lexical entry *'born' (in)* in Figure 11, which is an intransitive verb
(i.e., `IntransitivePPFrame`). The semantics of the entry is captured by the property *dbo:birthPlace*, the domain of the property is *dbo:Place*, and the range of the property is *dbo:Person*.

In Section 2.5, we provide details, along with an example, of how the grammar rule templates (GRT) are obtained from sentence templates. The five grammar rule templates for the intransitive verb of type `Agent` are presented in two groups (i.e., $\text{GRT}_{\text{subject}}$ and $\text{GRT}_{\text{object}}$), as shown below:

$GRT_{subject}$:
S -> Who was [VerbPerfect] [Preposition] <$NP_{\text{map(SyntacticFunction), Property}}$>? | Which <$NP_{\text{Class, <map(SyntacticFunction),Property>}}$> was [VerbPerfect] [Preposition] <$NP_{\text{map(SyntacticFunction), Property}}$>? | Which <$NP_{\text{Class, <map(SyntacticFunction),Property>}}$> were [VerbPerfect] [Preposition] <$NP_{\text{map(SyntacticFunction), Property}}$>?
GRT_{object}:
S -> Where was <$NP_{\text{map(SyntacticFunction), Property}}$> [VerbPerfect]? | [Preposition] which <$NP_{\text{Class, <map(SyntacticFunction),Property>}}$> was <$NP_{\text{Domain,dbo:birthDate}}$> [VerbPerfect]?

The grammar rules (GR) are then automatically generated by instantiating these grammar rule templates (GRT) using the lexical entry *'born' (in)*, as shown below.

grammar rules for $GRT_{subject}$:
S -> Who was born in <$NP_{\text{<Domain,dbo:birtPlace>}}$>? | Which <$NP_{\text{Class, <map(SyntacticFunction),Property>}}$> was born in <$NP_{\text{map(Domain,dbo:birtPlace}}$>? | Which <$NP_{\text{Class, <map(SyntacticFunction),Property>}}$> were born in <$NP_{\text{map(Domain,dbo:birtPlace}}$>?
grammar rules for GRT_{object}:
S -> Where was <$NP_{\text{Range,dbo:birthDate}}$> born? | In which <$NP_{\text{Class, Domain, Property>}}$> was <$NP_{\text{Range,dbo:birtPlace}}$> born?

When adding the lexical entry *'born' (in)* to the lexicon (as shown in Figure 11), the number of parseable questions increased from 3.46 million to 22.18 million. As seen from this example, the addition of a single lexical entry to the lexicon enables the QA system to parse tens of thousands of questions and significantly increase the F_1-measure.

Another example of consistency and controllability is shown in Table 13, where we illustrate the effects of removing a top-ranked lexical entry, a syntactic frame, and a grammar rule from the lexicon and the grammar. We select the top-ranked element by counting the number of questions covered by an element in the QALD series training datasets (QALD-9, QALD-8, QALD-7, QALD-6, and QALD-5) and ranking them in descending order. Following that, we demonstrate the number of questions that cannot be parsed in our approach when removing these elements from the lexicon, along with the resulting decrease in F_1-measure.

	# Questions	**%**	**Micro F_1 (ind*)**	**Macro F_1 (ind*)**	**Micro F_1 (inc*)**	**Macro F_1 (inc*)**
English lexicon	5416.03 m	–	0.29	0.22	0.85	0.30
Remove *'high'* 5.10.1	5410.61 m	0.1%	0.28	0.19	0.85	0.28
Remove Rule 5.10.2	5401.75 m	0.26%	0.27	0.17	0.84	0.24
Remove Template 5.10.3	4978.06 m	8.08%	0.27	0.16	0.84	0.21
Remove `NounPPFrame` 5.10.4	4195.95 m	22.52%	0.27	0.15	0.84	0.19

Table 13

The impact of removing a top-ranked lexical entry, grammar rule (GR), grammar rule template (GRT), and syntactic frame from the English lexicon (as detailed in 3.1). The evaluation is conducted on the QALD-9 test data set. Here, ind* refers to inductive mode, inc* refers to incremental mode of evaluation, Template* refers to grammar rule template (as detailed in Section 5.10.3), Rule* refers to grammar rule (as detailed in Section 5.10.2), % refers to the number questions that cannot be parse after removing these elements, and m refers to millions. P refers to Precision, R refers to Recall, and F_1-Measure refers to F_1-Measure.

5.10.1. Impact of lexical entry removal

Consider the top-ranked lexical entry *'high'* (i.e., `AdjectiveSuperlativeFrame`) in the QALD series, covering a total of 87 questions (i.e., 25 questions in QALD-9, 8 questions in QALD-8, 9 questions in QALD-7, 23 questions in QALD-6, and 22 questions in QALD-5). The lemon entry has four senses *dbo:height*, *dbo:elevation*, *dbp:height*, and *dbp:elevation*.

In DBpedia, there are 202, 143 triples for *dbo:height*, 247, 825 triples for *dbo:elevation*, 6, 863 triples for *dbp:height*, and 3605 triples for *dbp:elevation*. The grammar rule templates (GRT) for gradable adjectives (i.e., `AdjectiveSuperlativeFrame`) are detailed in Section 2.5.5. As can be seen from Table 13, a total of 5, 421, 864 questions (i.e., 0.1% questions) cannot be parsed in our approach when removing this lexical entry from the English lexicon.

5.10.2. Impact of grammar rule (GR) removal

To demonstrate the impact of removing a grammar rule, we consider the top-ranked grammar rule in the QALD series. The top-ranked grammar rule is as follows:

S -> How high is <NP$_{\text{Domain,dbo:height}}$>?

As can be seen in Table 13, after removing this rule, approximately 14, 284, 166 questions (i.e., 0.26% of questions) cannot be parsed by the QA system.

5.10.3. Impact of grammar rule template removal

Furthermore, to demonstrate the consistency and controllability of our approach, we remove the top-ranked grammar rule template (GRT) in the QALD series. The template is shown below:

Grammar Rules Template (top-GRT):
S -> What is the [NounSingular] [Preposition] <NP$_{\text{map(SyntacticFunction), Property}}$>?

The grammar rule generated by instantiating the template with the lexical entry of relative (e.g., *'death date' (of)*) is shown below:

S -> What is the death date of <NP$_{\text{Domain,dbo:deathDate}}$>?

Therefore, the template can be instantiated by a total of 189 lexical entries (e.g., *'capital' (of)*, *'currency' (of)*, *'death date' (of)*, *'birth date' (of)*, etc.) out of 311 (as shown in Table 1). As seen in Table 13, after removing this template, approximately 437, 978, 324 questions (i.e., 8.08% questions) cannot be parsed by the QA system.

5.10.4. Impact of syntactic frame removal

We repeated the experiment for the top-ranked syntactic frame (i.e., `NounPPFrame`) in the QALD series. An example of a lexical entry (detailed in Section 2.3.1), grammar rule templates (GRT), and grammar rules for `NounPPFrame` are detailed in Section 2.5.1.

There are a total of 44 grammar rule templates (as shown in Table 1), and our approach generates a total of 8, 708 grammar rules by instantiating these templates using lexical entries for this frame.

There are 311 lexical entries and 31, 284, 166 triples containing the properties associated with these lexical entries in DBpedia. As seen in Table 13, a total of 1, 220, 082, 474 questions (i.e., 22.52% questions) cannot be parsed in our approach when removing the `NounPPFrame`.

In contrast, for Machine Learning (ML) systems in general, it is unclear what the impact of adding one example is in terms of the behaviour of the system. Usually, several examples of the same or similar type need to be provided for the ML system to learn a pattern, leading to high redundancy in training data.

5.11. Discussion

Most work on question answering over linked data currently builds on systems that use machine learning to learn a model to map natural language questions into SPARQL. Our approach has the following advantages over these systems:

These systems can not be extended to cover new types of questions in a principled way. Our approach overcomes the limitation in a more principled way by simply adding a new syntactic construction to the system, which takes only $10–25$ minutes depending on the language. For example, adding a sentence template[30] that supports sentences such as *'What is Angela Merkel's date of birth?'*, etc., will enable the QA system to parse an additional 19.87 million questions. Furthermore, we demonstrate the consistency and controllability of our approach by removing the top-ranked grammar rule template obtained from the sentence template in the QALD series, as illustrated in Table 13 in Section 5.10.

Another limitation of machine learning approaches is that the addition of new training questions involving a new lexical element does not guarantee that the system can induce the semantics of that lexical element correctly. For example, adding a lexical entry for `InTransitivePPFrame`, e.g.,*'born' (in)*, will make 18.72 million questions for queries that involve *dbo:birtPlace* parsable.

Further, the machine learning-based approaches follow an inductive paradigm to port the system to a new language. The paradigm is costly and time-consuming since it requires adding training data for that language. In our approach, porting the system to a new language involves two steps: (i) creating lexical entries and (ii) creating sentence templates considering the features of that particular language. For example, we ported the system to German by adding lexical entries for all syntactic frames considering features of the German language such as the gender and case information for nouns in Lemon format (as shown in Section 3.2.1). We also ported the system to Italian (Section 3.2.2) and Spanish (Section 3.2.3) and showed that we only needed to add language-specific properties to the lexicon and adapted the sentence templates with respect to the features of these languages, but the process remains the same.

The machine learning-based approaches, e.g., [99], rely on algorithms having a large number of learning parameters and require a significant amount of training data. Creating a new training dataset for the new dataset is very expensive [81]. Finally, our approach operates independently of training data. Instead, the capabilities of our system can be increased by extending the lexicon, which is a completely different paradigm. This allows us to improve our system step by step (as detailed in Section 5.3).

In the 'Hackathon on question answering over RDF data' event, the participants were able to create a basic version of the QA system for the ArCo dataset (i.e., Italian Cultural Heritage linked data) [98] for English and Italian in 9 person-hours, for the Wikidata dataset for English, Italian, and Bangla in 12 person-hours, for the AIFB dataset for English and German in 8 person-hours, and for the Lexinfo dataset for English and Spanish in 4 person-hours (as shown in Section 5.9).

Vollmers [100] presented a question-answering over knowledge graph (KGQA) approach that requires less effort to create training data compared to the state-of-art machine learning approaches as it uses existing benchmarks as training data. The approach learns templates from existing benchmarks (in particular, QALD-8 and QALD-9) by employing machine learning and feature engineering to learn to classify natural language questions into isomorphic basic graph patterns. A disadvantage of the approach is that it depends heavily on benchmarks which limit the system to the language and RDF dataset of the existing benchmarks. Moreover, the research in the field of KGQA suffers from the lack of multilingual benchmarks [101, 102], and only a few benchmarks (i.e., QALD series[31] [47], LC-QuAD[32] [103], CWQ [104], RuBQ [105], etc.) exist that involve multiple languages. In contrast, our approach can be adapted to a new language without the availability of benchmarks. For example, to the best of our knowledge, there are no benchmarks available for the Bangla language, yet the

[30]InterrogativePronounForThing AuxVerb(be:present:singular) PrepositionalAdjunct's [NounSingular]?
[31]the newest version QALD-9 contains 558 questions in 11 languages in DBpedia and Wikidata
[32]LC-QuAD2.0 contains 30,000 questions in English and German

participants of the ’Hackathon on question answering over RDF data’ event were able to create a basic version of the QA system for Wikidata for the Bangla language in 4 person-hours.

There are also question-answering systems over linked data (QALD) that exploit large language models [26, 106]. One promising system in this approach is ChatGPT [27, 97]. Our approach outperforms ChatGPT in zero-shot scenarios across four languages, in both inductive and incremental modes. However, in the few-shot scenario, ChatGPT outperforms our approach in the inductive mode. It is unclear whether the QALD-9 dataset was part of the data used to train ChatGPT. Nevertheless, our approach outperforms ChatGPT in the few-shot scenario in the incremental evaluation mode.

We compare how ChatGPT performs in comparison to our approach and outline the cases where our approach performs better. As an example, when evaluating with the QALD benchmark (i.e., QALD-9 dataset), we found that only a few instances (i.e., 21%) have the SPARQL query generated by ChatGPT as an exact match with the one specified in the QALD-9 dataset. This holds for 45% of the questions in our approach. Moreover, when using ChatGPT, the generated SPARQL query from a natural language question may contain invented KB elements, such as resources, properties, or classes (as detailed in Section 5.8), that do not exist in the RDF data. We have provided several examples of this case using DBpedia.

Besides systems using machine learning techniques, there are rule-based systems that rely on a set of rules to map natural language (NL) questions into a representation that can be either directly evaluated over the knowledge graph or rely on graph exploration to map the representation of the NL question into a full-fledged SPARQL query. Although no training data is required by the rule-based systems, there are limitations to porting these systems to a new language. The rule-based systems and their limitations compared to our approach are as follows:

To the best of our knowledge, WDAqua-core1 [52] is the only QA system that is implemented for languages other than English such as German, Spanish, French, and Italian and ported for several RDF datasets [90]. For lexicalizations, WDAqua-core1 maps n-grams (of natural language sentences) to Knowledge Base (KB) elements by comparing an n-gram with an entity’s *rdfs:label*, which is limited to the languages available in the dataset. Moreover, there is a well-known lexical gap between content expressed in the form of natural language (NL) texts and content stored in an RDF knowledge base (KB).

Similar to WDAqua-core1, some approaches [107, 108] use *rdfs:label* to bridge the lexical gap between natural language terms and KB elements, enhancing the quality of lexicalization by analyzing SPARQL queries and determining their correctness with respect to the given question. This correction is performed using the VANiLLa Dataset [109], which contains both correct and incorrect answers to questions. However, the dataset is limited to English and Spanish. Furthermore, the approach was not evaluated with respect to the QALD benchmark, leaving its effectiveness in the question-answering system unknown.

In contrast, in our approach, the Lemon model of lexicalization (detailed in Section 2.5) is not limited to any language, and the questions are more natural as the lexicalization is provided by a language expert. For example, consider the lexical entry *‘high’* (as shown in Figure 8); the semantics of this gradable adjective is captured with respect to the property *dbo:elevation*. However, such lexicalization is not possible when considering the WDAqua-core1 approach, which uses `rdfs:label` for mapping natural language terms to KB elements.

WDAqua-core1 is based on the observation that many questions can be understood from the semantics of the words but the syntax of the question has less importance. The authors claimed that the system is adaptable to any language by changing only the stop word removal and stemming, but provide no evidence of how language language-specific features (i.e., gender information, grammatical cases, word order, verb types, etc.) are incorporated into their approach. In contrast, we create a lexicon for each language (as detailed in Section 3) by incorporating language-specific features in Lemon format and sentence templates.

Similar to WDAqua-core1, there are pattern-based approaches [110–112] that make use of a number of patterns (i.e., a pattern library) that have been defined to map natural language to SPARQL queries. But these works are either in the preliminary stage [110] and provide only two patterns as proof of concept or have been developed for the English language [111] only. We adopted the underlying idea of using patterns but we use automatically generated grammar rules, which can be easily extended to another language and RDF dataset.

SemGraphQA [65], gAnswer [50], and Xser [62] use external tools (i.e., parts-of-speech taggers, dependency parsers, semantic parsers, etc.) and other approaches such as AskNow [64] and DEANNA [72] use language resources or dictionaries (such as WordNet, BOA, etc.) to map natural language terms to KB elements. Both the language tools and resources are limited to a few languages and it is very difficult (and time-consuming) to adapt them to a new language. In contrast, our approach is independent of any such tool or external dictionary and porting to a new language is straightforward by creating lexical entries (as explained in Section 3) and by writing sentence templates for that language.

Creating a question answering system on top of a new RDF dataset is still challenging and cumbersome [84]. The main open challenge remains portability [90], i.e., the ability to easily apply a QA algorithm to new and previously untested RDF datasets. Qanary [113] achieved limited portability to a new dataset by integrating a consistent number of tools. Diefenbach et al. [90] provided a QA system and argued that the system can be ported to a new RDF dataset with minimal investments, but provide no evidence of how much time it requires to port to a new dataset. Our approach allows us to port our system to a wide range of RDF datasets in hours, not days (as detailed in Section 5.9).

6. Conclusion

We present a multilingual question-answering system that relies on a model of the lexicon-ontology interface in which the meaning of lexical entries is specified with respect to a given vocabulary or ontology. More specifically, our model-based approach automatically generates a lexicalized grammar given a lexicon and sentence templates in a particular language and given an RDF dataset. We present the high-level architecture of our multilingual QA system.

Following this, we describe the automatically generated grammar for five syntactic frames at two levels of abstraction. First, we describe grammar rule templates that are independent of specific lexical entries. Secondly, we provide example instantiations of these templates using lexical entries for illustration. We also include SPARQL queries to retrieve the values of non-terminals in the grammar rules. Further, we demonstrate the methods and implementations of creating a lexicon for English, also adapting the system to other languages by creating lexical entries and sentence templates for these languages.

To compare with machine learning-based approaches, we evaluated our approach in two modes: inductive and incremental, showing that our approach outperforms all state-of-the-art approaches in the incremental evaluation mode. So far there is no system described in the literature that works for four languages (i.e., English, German, Italian, and Spanish) while reaching state-of-the-art performance on all of them for QALD-9 benchmarks and outperforming current QA systems.

A necessary prerequisite for our grammar generation approach is the availability of a Lemon lexicon for each supported language, which is manually created in our approach. Previous work [114] has proposed a method for the fully automatic creation of a Lexicon for QA systems using LexExMachina [115] that uses association rules to identify correspondences between lexical elements on the one hand and ontological vocabulary elements on the other hand. Although this method significantly reduces the human effort involved in creating a lexicon for a language and linked data, it causes a $34\% - 56\%$ performance (F_1-measure on QALD benchmark) degradation with respect to a manually created lexicon. In future work, we will extend this automatic creation of a lexicon for multilingual settings and enhance the performance in comparison to state-of-the-art QALD systems.

Furthermore, our approach can handle a limited form of composition that allows us to nest some rules into others. In future work, we will investigate how to implement a more principled approach to semantic composition by Flexible Semantic Composition with DUDES [116]. We also intend to enable the community to contribute to and adapt the grammar generation to other languages. More specifically, we intend to cover Indian and Arabic languages, as there is no QA system for an RDF dataset described in the literature that works for these languages. In this research, we focus on an established task—question answering over linked data [2, 84] —which does not involve semi-structured or unstructured data.This task gained popularity through the QALD challenge, established in 2012, which has undergone 10 iterations. In future work, we will investigate applying the methodology to semi-structured or unstructured data.

Acknowledgments

The work described here is supported by “Linked-data aware NLP services (WG2)” of the Nexus Linguarum (https://nexuslinguarum.eu/) as well as “Lexical and conceptual linking (WG3)” of Prêt-à-LLOD (the European Commission under grant 825182). B. Ell has been partially supported by the SIRIUS centre: Norwegian Research Council project No 237898. Thanks to David Schmidt of the University of Bielefeld for developing a tool for the evaluation of results with ChatGPT. Thanks to the participants of the ’Hackathon on question answering over RDF data’. Special thanks to participant Japesh Methuku from ADAPT, the world-leading SFI Research Centre for AI-Driven Digital Content Technology in Ireland, for actively participating in the Hackathon and contributing to the system. We also extend our appreciation to participants Gennaro Nolano and Maria Pia di Buono from the University of Naples ’L’Orientale,’ Italy, for their valuable contributions to the Italian language. Additionally, we want to acknowledge and thank Wasim Mahmud Surjo from BRAC University, Bangladesh, and Subhana Mahmud Toshi from Arab Open University, Bahrain, for their contributions to the Bangla language.

References

[1] J. Gracia, E. Montiel-Ponsoda, P. Cimiano, A. Gómez-Pérez, P. Buitelaar and J. McCrae, Challenges for the multilingual Web of Data, *Journal of Web Semantics* **11** (2012), 63–71.

[2] C. Unger, A. Freitas and P. Cimiano, An introduction to question answering over linked data, in: *Reasoning Web*, Springer, Athens, Greece, 2014, pp. 100–140.

[3] S. Shekarpour, K.M. Endris, A. Jaya Kumar, D. Lukovnikov, K. Singh, H. Thakkar and C. Lange, Question answering on linked data: Challenges and future directions, in: *Proceedings of the 25th International Conference Companion on World Wide Web (WWW)*, 2016, pp. 693–698.

[4] A. Perevalov, A.-C. Ngonga and A. Both, Enhancing the Accessibility of Knowledge Graph Question Answering Systems through Multilingualization, in: *Proceedings of the 16th International Conference on Semantic Computing (ICSC)*, 2022, pp. 251–256.

[5] A. Perevalov, A. Both and A.-C.N. Ngomo, Multilingual Question Answering Systems for Knowledge Graphs—A Survey, *Journal of Web Semantics* **9**(1) (2018), 29–51.

[6] A. Perevalov, A. Both, D. Diefenbach and A.-C. Ngonga Ngomo, Can Machine Translation Be a Reasonable Alternative for Multilingual Question Answering Systems over Knowledge Graphs?, in: *Proceedings of the ACM Web Conference 2022*, WWW ’22, Association for Computing Machinery, New York, NY, USA, 2022, pp. 977–986–.

[7] N. Chakraborty, D. Lukovnikov, G. Maheshwari, P. Trivedi, J. Lehmann and A. Fischer, Introduction to Neural Network based Approaches for Question Answering over Knowledge Graphs, *CoRR* **abs/1907.09361** (2019).

[8] A. Bordes, S. Chopra and J. Weston, Question Answering with Subgraph Embeddings, in: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[9] J. Cheng, S. Reddy, V. Saraswat and M. Lapata, Learning an Executable Neural Semantic Parser, *Computational Linguistics* **45**(1) (2019), 59–94.

[10] C. Liang, J. Berant, Q. Le, K.D. Forbus and N. Lao, Neural Symbolic Machines: Learning Semantic Parsers on Freebase with Weak Supervision, in: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Association for Computational Linguistics, Vancouver, Canada, 2017, pp. 23–33.

[11] A. Neelakantan, Q.V. Le, M. Abadi, A. McCallum and D. Amodei, Learning a Natural Language Interface with Neural Programmer, *ArXiv* **abs/1611.08945** (2016).

[12] F. Yang, Z. Yang and W.W. Cohen, Differentiable Learning of Logical Rules for Knowledge Base Reasoning, in: *Neural Information Processing Systems*, 2017.

[13] D. Lukovnikov, A. Fischer and J. Lehmann, Pretrained Transformers for Simple Question Answering over Knowledge Graphs, in: *Proceedings of the 18th International Semantic Web Conference (ISWC)*, 2019, pp. 470–486.

[14] X. Yin, D. Gromann and S. Rudolph, Neural Machine Translating from Natural Language to SPARQL, *Future Generation Computer Systems* **117** (2021), 510–519. doi:10.1016/j.future.2020.12.013.

[15] K. Wang, Y. Zhang, D. Yang, L. Song and T. Qin, GNN is a Counter? Revisiting GNN for Question Answering, *ArXiv* **abs/2110.03192** (2021).

[16] M. Yasunaga, H. Ren, A. Bosselut, P. Liang and J. Leskovec, QA-GNN: Reasoning with Language Models and Knowledge Graphs for Question Answering, in: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Association for Computational Linguistics, 2021, pp. 535–546.

[17] S. Jebbara and P. Cimiano, Zero-Shot Cross-Lingual Opinion Target Extraction, in: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2019.

[18] R. Klinger and P. Cimiano, Instance selection improves cross-lingual model training for fine-grained sentiment analysis, in: *Proceedings of the 19th Conference on Computational Natural Language Learning*, 2015.

[19] R. Klinger and P. Cimiano, The USAGE review corpus for fine grained multi lingual opinion analysis, in: *Proceedings of the 9th International Conference on Language Resources and Evaluation (LREC)*, Reykjavik, Iceland, 2014, pp. 2211–2218.

[20] A. Conneau, K. Khandelwal, N. Goyal, V. Chaudhary, G. Wenzek, F. Guzmán, E. Grave, M. Ott, L. Zettlemoyer and V. Stoyanov, Unsupervised Cross-lingual Representation Learning at Scale, in: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, Association for Computational Linguistics, 2020, pp. 8440–8451.

[21] A. CONNEAU and G. Lample, Cross-lingual Language Model Pretraining, in: *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 32, Curran Associates, Inc., 2019.

[22] C. Bizer, T. Heath and T. Berners-Lee, Linked data - the story so far, *International Journal on Semantic Web and Information* **5**(3) (2009), 1–22–.

[23] T. Berners-Lee, Linked Data - Design Issues, *W3C* (2006).

[24] V. Benz, P. Cimiano, M.F. Elahi and B. Ell, Generating Grammars from Lemon Lexica for Questions Answering over Linked Data: a Preliminary Analysis, in: *Proceedings of the 6th Natural Language Interfaces for the Web of Data Workshop (NLIWOD) co-located with the 19th International Semantic Web Conference (ISWC)*, CEUR Workshop Proceedings, Vol. 2722, 2020, pp. 40–55.

[25] P. Cimiano, J.P. McCrae and P. Buitelaar, Lexicon Model for Ontologies: Community Report, in: *W3C Community Group Final Report*, 2016.

[26] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Gray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike and R. Lowe, Training language models to follow instructions with human feedback, in: *Advances in Neural Information Processing Systems*, 2022.

[27] B. Faria, D. Perdigão and H. Gonçalo Oliveira, Question Answering over Linked Data with GPT-3, in: *12th Symposium on Languages, Applications and Technologies (SLATE 2023)*, Open Access Series in Informatics (OASIcs), Vol. 113, Dagstuhl, Germany, 2023, pp. 1:1–1:15. ISSN 2190-6807. ISBN 978-3-95977-291-4.

[28] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P.N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer and C. Bizer, DBpedia - A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia, *Semantic Web Journal* **6**(2) (2015), 167–195.

[29] J.P. McCrae, D. Spohr and P. Cimiano, Linking Lexical Resources and Ontologies on the Semantic Web with Lemon, in: *Proceedings of the 8th extended semantic web conference on The semantic web: research and applications (ESWC)*, Vol. 6643, 2011, pp. 245–259.

[30] P. Cimiano, P. Buitelaar, J.P. McCrae and M. Sintek, LexInfo: A declarative model for the lexicon-ontology interface, *Journal of Web Semantics* **9**(1) (2011), 29–51.

[31] A. Gangemi, M. Alam, L. Asprino, V. Presutti and D.R. Recupero, Framester: A Wide Coverage Linguistic Linked Data Hub, in: *Proceedings of the 20th International Conference on Knowledge Engineering and Knowledge Management*, Lecture Notes in Computer Science, Vol. 10024, Bologna, Italy, 2016, pp. 239–254.

[32] C.F. Baker, C.J. Fillmore and J.B. Lowe, The Berkeley FrameNet Project, in: *36th Annual Meeting of the Association for Computational Linguistics, Volume 1*, 1998, pp. 86–90.

[33] C. Fellbaum, *WordNet: An electronic lexical database*, MIT press, 1998.

[34] K. Kipper, A. Korhonen, N. Ryant and M. Palmer, A large-scale classification of English verbs, *Language Resources and Evaluation* **42** (2008), 21–40.

[35] A.K. Joshi and Y. Schabes, *Tree-Adjoining Grammars*, in: *Handbook of Formal Languages: Volume 3 Beyond Words*, G. Rozenberg and A. Salomaa, eds, Springer Berlin Heidelberg, Berlin, Heidelberg, 1997, pp. 69–123.

[36] Y. Schabes and A.K. Joshi, An Earley-Type Parsing Algorithm for Tree Adjoining Grammars, in: *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics*, Association for Computational Linguistics, Buffalo, New York, USA, 1988, pp. 258–269.

[37] J.P. McCrae, F. Quattri, C. Unger and P. Cimiano, Modelling the Semantics of Adjectives in the Ontology-Lexicon Interface, in: *Proceedings of the 4th Workshop on Cognitive Aspects of the Lexicon (CogALex)*, Association for Computational Linguistics and Dublin City University, 2014, pp. 198–209.

[38] K. Simov, M. Kouylekov and A. Simov, Cascaded Regular Grammars over XML Documents, in: *Proceedings of the 2nd Workshop on NLP and XML (NLPXML)*, 2002.

[39] K. Simov and P. Osenova, A Hybrid Strategy For Regular Grammar Parsing, in: *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC)*, European Language Resources Association (ELRA), Lisbon, Portugal, 2004.

[40] D. Jurafsky and J.H. Martin, *Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition*, Pearson Prentice Hall, 2009. ISBN 9780131873216 0131873210.

[41] T. Lichte and L. Kallmeyer, Tree-Adjoining Grammar: A Tree-Based Constructionist Grammar Framework for Natural Language Understanding, in: *The AAAI 2017 Spring Symposium on Computational Construction Grammar and Natural Language Understanding*, AAAI Press, 2017.

[42] K. Vijay-Shanker, Using Descriptions of Trees in a Tree Adjoining Grammar, *Computational Linguistics* **18**(4) (1992), 481–518.

[43] M.F. Elahi, B. Ell, F. Grimm and P. Cimiano, Question Answering on RDF Data based on Grammars Automatically Generated from Lemon Models, in: *Proceedings of the 17th International Conference on Semantic Systems (SEMANTiCS)*, 2021.

[44] R. Usbeck, X. Yan, A. Perevalov, L. Jiang, J. Schulz, A. Kraft, C. Möller, J. Huang, J. Reineke, A. Ngomo, M. Saleem and A. Both, QALD-10 — The 10th Challenge on Question Answering over Linked Data, *Semantic Web* (2023).

[45] R. Usbeck, A.-C.N. Ngomo, B. Haarmann, A. Krithara, M. Röder and G. Napolitano, 7th Open Challenge on Question Answering over Linked Data (QALD-7), in: *Semantic Web Evaluation Challenge*, Springer International Publishing, 2017, pp. 59–69.

[46] R. Usbeck, M. Röder, M. Hoffmann, F. Conrads, J. Huthmann, A.N. Ngomo, C. Demmler and C. Unger, Benchmarking Question Answering Systems, *Semantic Web* **10**(2) (2019), 293–304.

[47] R. Usbeck, R.H. Gusmita, A.N. Ngomo and M. Saleem, 9th Challenge on Question Answering over Linked Data (QALD-9), in: *Joint proceedings of the 4th Workshop on Semantic Deep Learning (SemDeep-4) and NLIWoD4: Natural Language Interfaces for the Web of Data (NLIWOD-4) and 9th Question Answering over Linked Data challenge (QALD-9) co-located with 17th International Semantic Web Conference (ISWC)*, California, United States of America, 2018, pp. 58–64.

[48] R. Speer, J. Chin and C. Havasi, ConceptNet 5.5: an open multilingual graph of general knowledge, in: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, AAAI Press, 2017, pp. 4444–4451–.

[49] Z. Ding, Z. Li, R. Qi, J. Wu, B. He, Y. Ma, Z. Meng, S. Chen, R. Liao, Z. Han and V. Tresp, ForecastTKGQuestions: A Benchmark for Temporal Question Answering and Forecasting over Temporal Knowledge Graphs, in: *Proceedings of the 22nd International Semantic Web Conference (ISWC)*, Springer Nature Switzerland, Cham, 2023, pp. 541–560.

[50] L. Zou, R. Huang, H. Wang, J.X. Yu, W. He and D. Zhao, Natural language question answering over RDF: a graph data driven approach, in: *2014 ACM SIGMOD international conference on Management of data*, 2014.

[51] P. Kapanipathi, I. Abdelaziz, S. Ravishankar, S. Roukos, A. Gray, R. Fernandez Astudillo, M. Chang, C. Cornelio, S. Dana, A. Fokoue, D. Garg, A. Gliozzo, S. Gurajada, H. Karanam, N. Khan, D. Khandelwal, Y.-S. Lee, Y. Li, F. Luus, N. Makondo, N. Mihindukulasooriya, T. Naseem, S. Neelam, L. Popa, R. Gangi Reddy, R. Riegel, G. Rossiello, U. Sharma, G.P.S. Bhargav and M. Yu, Leveraging Abstract Meaning Representation for Knowledge Base Question Answering, in: *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, Association for Computational Linguistics, 2021, pp. 3884–3894.

[52] D. Diefenbach, A. Both, K. Singh and P. Maret, Towards a question answering system over the Semantic Web, *Semantic Web* **11**(3) (2020), 421–439.

[53] W. Zheng and M. Zhang, Question answering over knowledge graphs via structural query patterns, *ArXiv* **abs/1910.09760** (2019).

[54] A. Sakor, I.O. Mulang', K. Singh, S. Shekarpour, M.E. Vidal, J. Lehmann and S. Auer, Old is Gold: Linguistic Driven Approach for Entity and Relation Linking of Short Text, in: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Association for Computational Linguistics, Minneapolis, Minnesota, 2019, pp. 2336–2346.

[55] N. Mihindukulasooriya, G. Rossiello, P. Kapanipathi, I. Abdelaziz, S. Ravishankar, M. Yu, A.G.S. Roukos and A. Gray, Leveraging Semantic Parsing for Relation Linking over Knowledge Bases, in: *Proceedings of the 19th International Semantic Web Conference (ISWC)*, Springer International Publishing, 2020, pp. 402–419.

[56] G. Rossiello, N. Mihindukulasooriya, I. Abdelaziz, M. Bornea, A. Gliozzo, T. Naseem and P. Kapanipathi, Generative Relation Linking for Question Answering over Knowledge Bases, in: *Proceedings of the 20th International Semantic Web Conference (ISWC)*, Springer International Publishing, 2021, pp. 321–337.

[57] X. Hu, Y. Shu, X. Huang and Y. Qu, EDG-Based Question Decomposition for Complex Question Answering over Knowledge Bases, in: *Proceedings of the 20th International Semantic Web Conference (ISWC)*, Springer International Publishing, 2021, pp. 128–145.

[58] R. Omar, I. Dhall, P. Kalnis and E. Mansour, A universal question-answering platform for knowledge graphs, in: *In Proceedings of the ACM SIGMOD/PODS international conference of Management of Data*, 2023.

[59] A. Glaese, N. McAleese, M. Trębacz, J. Aslanides, V. Firoiu, T. Ewalds, M. Rauh, L. Weidinger, M. Chadwick and P. Thacker, Improving alignment of dialogue agents via targeted human judgements, *ArXiv* **abs/2209.14375** (2022).

[60] D. Vollmers, R. Jalota, D. Moussallem, H. Topiwala, A.-C.N. Ngomo and R. Usbeck, Knowledge Graph Question Answering using Graph-Pattern Isomorphism, in: *Proceedings of the 17th International Conference on Semantic Systems (SEMANTiCS)*, 2021.

[61] S. Hakimov, S. Jebbara and P. Cimiano, AMUSE: Multilingual Semantic Parsing for Question Answering over Linked Data, in: *Proceedings of the 16th International Semantic Web Conference (ISWC)*, 2017, pp. 329–346.

[62] K. Xu, Y. Feng and D. Zhao, Xser@QALD-4: Answering Natural Language Questions via Phrasal Semantic Parsing, in: *Communications in Computer and Information Science*, Vol. 496, 2014.

[63] A.P.-e. veyseh, Cross-Lingual Question Answering Using Common Semantic Space, in: *Proceedings of the 15th Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2016.

[64] M. Dubey, S. Dasgupta, A. Sharma, K. Höffner and J. Lehmann, AskNow: A Framework for Natural Language Query Formalization in SPARQL, in: *The Semantic Web. Latest Advances and New Domains*, Springer International Publishing, 2016, pp. 300–316.

[65] R. Beaumont, B. Grau and A.-L. Ligozat, SemGraphQA@QALD-5, in: *LIMSI participation at QALD-5@CLEF. CLEF*, 2015.

[66] P. Baudiš and J. Šedivy, QALD Challenge and the 'YodaQA System, in: *Prototype Notes*, 2015.

[67] S. He, Y. Zhang, K. Liu and J. Zhao, CASIA@ V2: a MLN-based question answering system over linked data, in: *Conference and Labs of the Evaluation Forum*, 2014.

[68] J. Guyonvarch and S. Ferr´e, Scalewelis: a scalable query-based faceted search system on top of sparql endpoints, in: *In: Work. Multilingual question answering over linked data (QALD-3)*, 2013.

[69] C. Giannone, V. Bellomaria and R. Basili, A HMM-based approach to question answering against linked data, in: *Proceedings of the Question Answering over Linked Data lab (QALD-3) at CLEF (2013)*, 2013.

[70] P. Cimiano, V. Lopez, C. Unger, E. Cabrio, A.-C.N. Ngomo and S. Walter, Intui2: a prototype system for question answering over linked data, in: *Proceedings of the Question Answering over Linked Data lab (QALD-3) at CLEF*, 2013.

[71] C. Pradel, O. Haemmerlé and N. Hernandez, A Semantic Web Interface Using Patterns: The SWIP System, in: *Graph Structures for Knowledge Representation and Reasoning*, M. Croitoru, S. Rudolph, N. Wilson, J. Howse and O. Corby, eds, Springer Berlin Heidelberg, 2012, pp. 172–187.

[72] M. Yahya, K. Berberich, S. Elbassuoni, M. Ramanath, V. Tresp and G. Weikum, Natural language questions for the web of data, in: *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, Association for Computational Linguistics, 2012, pp. 379–390.

[73] M. Barz and D. Sonntag, Incremental Improvement of a Question Answering System by Re-ranking Answer Candidates using Machine Learning, in: *International Workshop on Spoken Dialogue Systems Technology*, 2019.

[74] D. Lukovnikov, A. Fischer, J. Lehmann and S. Auer, Neural Network-Based Question Answering over Knowledge Graphs on Word and Character Level, in: *Proceedings of the 26th International Conference on World Wide Web (WWW)*, 2017, pp. 1211–1220–.

[75] S. Mohammed, P. Shi and J. Lin, Strong Baselines for Simple Question Answering over Knowledge Graphs with and without Neural Networks, in: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, Association for Computational Linguistics, New Orleans, Louisiana, 2018, pp. 291–296.

[76] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L.u. Kaiser and I. Polosukhin, Attention is All you Need, in: *Advances in Neural Information Processing Systems*, Vol. 30, I. Guyon, U.V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan and R. Garnett, eds, Curran Associates, Inc., 2017.

[77] F.F. Luz and M. Finger, Semantic Parsing Natural Language into SPARQL: Improving Target Language Representation with Neural Attention, *ArXiv* **abs/1803.04329** (2018).

[78] T. Soru, E. Marx, D. Moussallem, G. Publio, A. Valdestilhas, D. Esteves and C.B. Neto, SPARQL as a Foreign Language, in: *Proceedings of the 13th International Conference on Semantic Systems - SEMANTiCS2017 Posters and Demos*, Amsterdam, The Netherlands, 2017.

[79] T. Soru, E. Marx, A. Valdestilhas, D. Esteves, D. Moussallem and G. Publio, Neural Machine Translation for Query Construction and Composition, in: *ICML Workshop on Neural Abstract Machines & Program Induction (NAMPI v2)*, 2018.

[80] U. Sawant and S. Chakrabarti, Learning joint query interpretation and response ranking, in: *Proceedings of the 22nd international conference on World Wide Web*, 2013, pp. 1099–1110.

[81] J. Berant, A. Chou, R. Frostig and P. Liang, Semantic Parsing on Freebase from Question-Answer Pairs, in: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2013.

[82] T. Kwiatkowski, E. Choi, Y. Artzi and L. Zettlemoyer, Scaling Semantic Parsers with On-the-Fly Ontology Matching, in: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, Seattle, Washington, USA, 2013, pp. 1545–1556.

[83] S. Ruseti, A. Mirea, T. Rebedea and S. Trausan-Matu, QALD Challenge and the ‘YodaQA System, in: *Prototype Notes*, 2015.

[84] D. Diefenbach, V. López, K.D. Singh and P. Maret, Core techniques of question answering systems over knowledge bases: a survey, *Knowledge and Information Systems* **55** (2018), 529–569.

[85] Óscar Ferrández, C. Spurk, M. Kouylekov, I. Dornescu, S. Ferrández, M. Negri, R. Izquierdo, D. Tomás, C. Orasan, G. Neumann, B. Magnini and J.L. Vicedo, The QALL-ME Framework: A specifiable-domain multilingual Question Answering architecture, in: *Journal of Web Semantics*, 2011.

[86] L. Zhang, M. Färber and A. Rettinger., BreXearch: Exploring Brexit Data Using Cross-Lingual and Cross-Media Semantic Search, in: *Proceedings of the 25th ACM International Conference on Information and Knowledge Management*, 2016.

[87] L. Zhang, M. Acosta, M. Färber, S. Thoma and A. Rettinger, BreXearch: Exploring Brexit Data Using Cross-Lingual and Cross-Media Semantic Search, in: *Proceedings of the 16th International Semantic Web Conference (ISWC)*, Vol. 1963, 2017.

[88] M. Burtsev, A. Seliverstov, R. Airapetyan, M. Arkhipov, D. Baymurzina, N. Bushkov, O. Gureenkova, T. Khakhulin, Y. Kuratov, D. Kuznetsov, A. Litinsky, V. Logacheva, A. Lymar, V. Malykh, M. Petrov, V. Polulyakh, L. Pugachev, A. Sorokin, M. Vikhreva, and M. Zaynutdinov, DeepPavlov: Open-Source Library for Dialogue Systems, in: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2018.

[89] T.P. Tanon, M.D. de Assuncao, E. Caron and F.M. Suchanek, Demoing Platypus - A Multilingual Question Answering Platform for Wikidata, in: *The Semantic Web: ESWC 2018 Satellite Events*, Springer International Publishing, 2018, pp. 111–116.

[90] D. Diefenbach, J. Giménez-García, A. Both, K. Singh and P. Maret, QAnswer KG: Designing a Portable Question Answering System over RDF Data, in: *Proceedings of the 17th European Semantic Web Conference (ESWC)*, 2020, pp. 429–445.

[91] Y. Zhou, X. Geng, T. Shen, W. Zhang and D. Jiang, Improving Zero-Shot Cross-lingual Transfer for Multilingual Question Answering over Knowledge Graph, in: *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Association for Computational Linguistics, 2021, pp. 5822–5834.

[92] A. Perevalov, A. Both, D. Diefenbach and A.-C. Ngonga Ngomo, Can Machine Translation Be a Reasonable Alternative for Multilingual Question Answering Systems over Knowledge Graphs?, in: *Proceedings of the ACM Web Conference 2022*, WWW ’22, Association for Computing Machinery, 2022, pp. 977–986–.

[93] S. Walter, C. Unger and P. Cimiano, M-ATOLL: A Framework for the Lexicalization of Ontologies in Multiple Languages, in: *Proceedings of the 13th International Semantic Web Conference (ISWC)*, 2014.

[94] G. Nolano, M.F. Elahi, M.P. di Buono, B. Ell and P. Cimiano, An Italian Question Answering System based on grammars automatically generated from ontology lexica, in: *Proceedings of the 18th Italian Conference on Computational Linguistics (CLiC-it)*, 2022.

[95] F. Petroni, T. Rocktäschel, S. Riedel, P. Lewis, A. Bakhtin, Y. Wu and A. Miller, Language Models as Knowledge Bases?, in: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, Association for Computational Linguistics, Hong Kong, China, 2019, pp. 2463–2473.

[96] W.-t. Yih, M. Richardson, C. Meek, M.-W. Chang and J. Suh, The Value of Semantic Parse Labeling for Knowledge Base Question Answering, in: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*, Association for Computational Linguistics, Berlin, Germany, 2016, pp. 201–206.

[97] Y. Tan, D. Min, Y. Li, W. Li, N. Hu, Y. Chen and G. Qi, Can ChatGPT Replace Traditional KBQA Models? An In-Depth Analysis of the Question Answering Performance of the GPT LLM Family, in: *International Semantic Web Conference*, Springer, 2023, pp. 348–367.

[98] V.A. Carriero, A. Gangemi, M.L. Mancinelli, L. Marinucci, A.G. Nuzzolese, V. Presutti and C. Veninata, ArCo: The Italian Cultural Heritage Knowledge Graph, in: *Proceedings of the 18th International Semantic Web Conference (ISWC)*, Springer International Publishing, 2019.

[99] Y. Zhang, K. Liu, S. He, G. Ji, Z. Liu, H. Wu and J. Zhao, Question Answering over Knowledge Base with Neural Attention Combining Global Knowledge Information, in: *https://arxiv.org/abs/1606.00979*, 2016.

[100] D. Vollmers, R. Jalota, D. Moussallem, H. Topiwala, A.-C.N. Ngomo and R. Usbeck, Knowledge graph question answering using graph-pattern isomorphism, in: *Proceedings of the 17th International Conference on Semantic Systems*, Vol. 53, 2021, p. 103.

[101] E. Loginova, S. Varanasi and G. Neumann, Towards end-to-end multilingual question answering, *Information Systems Frontiers* **23** (2021), 227–241.

[102] A. Neves, A. Lamúrias and F.M. Couto, Biomedical Question Answering using Extreme Multi-Label Classification and Ontologies in the Multilingual Panorama, in: *Proceedings of the Workshop on Semantic Indexing and Information Retrieval for Health from heterogeneous content types and languages co-located with 42nd European Conference on Information Retrieval*, CEUR Workshop Proceedings, Vol. 2619, 2020.

[103] M. Dubey, D. Banerjee, A. Abdelkawi and J. Lehmann, LC-QuAD 2.0: A Large Dataset for Complex Question Answering over Wikidata and DBpedia, in: *Proceedings of the 18th International Semantic Web Conference (ISWC)*, 2019.

[104] R. Cui, R. Aralikatte, H. Lent and D. Hershcovich, Compositional Generalization in Multilingual Semantic Parsing over Wikidata, *Transactions of the Association for Computational Linguistics* (2022).

[105] I. Rybin, V. Korablinov, P. Efimov and P. Braslavski, RuBQ 2.0: an innovated Russian question answering dataset, in: *Proceedings of the 18th European Semantic Web Conference (ESWC)*, Springer, 2021, pp. 532–547.

[106] D. Banerjee, P.A. Nair, R. Usbeck and C. Biemann, GETT-QA: Graph Embedding Based T2T Transformer for Knowledge Graph Question Answering, in: *Proceedings of the 20th European Semantic Web Conference (ESWC)*, Berlin, Heidelberg, 2023, pp. 279–297–.

[107] A. Gashkov, A. Perevalov, M. Eltsova and A. Both, Improving Question Answering Quality Through Language Feature-Based SPARQL Query Candidate Validation, in: *Proceedings of the 19th European Semantic Web Conference (ESWC)*, Berlin, Heidelberg, 2022, pp. 217–235–.

[108] A. Perevalov, A. Gashkov, M. Eltsova and A. Both, Towards Knowledge Graph-Agnostic SPARQL Query Validation for Improving Question Answering, in: *Proceedings of the 19th European Semantic Web Conference (ESWC)*, Springer-Verlag, Berlin, Heidelberg, 2022, pp. 78–82–.

[109] B. Magnini, M. Negri, R. Prevete and H. Tanev, Is It the Right Answer? Exploiting Web Redundancy for Answer Validation, in: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, Association for Computational Linguistics, Philadelphia, Pennsylvania, USA, 2002, pp. 425–432.

[110] M. Rico, C. Unger and P. Cimiano, Sorry, I only speak natural language: a pattern-based, data-driven and guided approach to mapping natural language queries to SPARQL, in: *Proceedings of the 4th International Workshop on Intelligent Exploration of Semantic Data (IESD) co-located with the 14th International Semantic Web Conference (ISWC)*, 2015.

[111] N. Steinmetz, Ann-Katrin, Arning and K.-U. Sattler, From Natural Language Questions to SPARQL Queries: A Pattern-based Approach, in: *Datenbanksysteme für Business, Technologie und Web (BTW)*, 2019.

[112] A. Abujabal, M. Yahya, M. Riedewald and G. Weikum, Automated Template Generation for Question Answering over Knowledge Graphs, in: *Proceedings of the 26th World Wide Web Conference (WWW)*, 2017.

[113] K. Singh, A. Both, D. Diefenbach and S. Shekarpour, Towards a message-driven vocabulary for promoting the interoperability of question answering systems, in: *Proceedings of the 10th International Conference on Semantic Computing (ICSC)*, 2016.

[114] M.F. Elahi, B. Ell and P. Cimiano, Bridging the Gap Between Ontology and Lexicon via Class-Specific Association Rules Mined from a Loosely-Parallel Text-Data Corpus, in: *Proceedings of the 4th Conference on Language, Data and Knowledge (LDK)*, 2023.

[115] B. Ell, M.F. Elahi and P. Cimiano, Bridging the Gap Between Ontology and Lexicon via Class-Specific Association Rules Mined from a Loosely-Parallel Text-Data Corpus, in: *Proceedings of the 3rd Conference on Language, Data and Knowledge (LDK)*, 2021.

[116] P. Cimiano, Flexible Semantic Composition with DUDES (short paper), in: *Proceedings of the 8th International Conference on Computational Semantics (IWCS)*, Association for Computational Linguistics, Tilburg, The Netherlands, 2009, pp. 272–276.

Appendix A. SPARQL queries for grammar rules of lexicon entries of LexInfo frames

Grammar rules template (GRT)	**SPARQL Query**
GRT$_{\textbf{subject}}$: S ->Who is the [NounSingular] [Preposition] <NP$_{\text{map(SyntacticFunction), Property}}$>? | Who was the [NounSingular] [Preposition] <NP$_{\text{map(SyntacticFunction), Property}}$>? | Which <NP$_{\text{Class, <map(SyntacticFunction),Property>}}$> is the [NounSingular] [Preposition] <NP$_{\text{map(SyntacticFunction), Property}}$>? | Which <NP$_{\text{Class, <map(SyntacticFunction),Property>}}$> was the [NounSingular] [Preposition] <NP$_{\text{map(SyntacticFunction), Property}}$>? | Give me the [NounSingular] [Preposition] <NP$_{\text{map(SyntacticFunction), Property}}$>. | Give me all [NounPlural] [Preposition] <NP$_{\text{map(SyntacticFunction), Property}}$>. | List all [NounPlural] [Preposition] <NP$_{\text{map(SyntacticFunction), Property}}$>.	`SELECT ?label WHERE { ?Domain Property ?Range . ?Domain rdfs:label ?label .}` or `SELECT ?label WHERE { ?Domain Property ?Range . ?Range rdfs:label ?label .}`
GRT$_{\textbf{object}}$: S -> <NP$_{\text{map(SyntacticFunction), Property}}$> is the [NounSingular] [Preposition] which <NP$_{\text{Class, <map(SyntacticFunction),Property>}}$>? | <NP$_{\text{map(SyntacticFunction), Property}}$> was the [NounSingular)] [Preposition] which <NP$_{\text{Class, <map(SyntacticFunction),Property>}}$>?	`SELECT ?label WHERE { ?Domain Property ?Range . ?Domain rdfs:label ?label .}` or `SELECT ?label WHERE { ?Domain Property ?Range . ?Range rdfs:label ?label .}`
GRT$_{\textbf{amount}}$: S -> How many [NounPlural] does <NP$_{\text{map(SyntacticFunction), Property}}$> have?	`SELECT Count(?Domain) WHERE { ?Domain Property ?Range }` or `SELECT Count(?Range) WHERE { ?Domain Property ?Range }`
GRT$_{\textbf{ask}}$: S -> Is <NP$_{\text{map(SyntacticFunction), Property}}$> the [NounSingular] [Preposition] <NP$_{\text{map(SyntacticFunction), Property}}$>? | Was <NP$_{\text{map(SyntacticFunction), Property}}$> the [NounSingular] [Preposition] <NP$_{\text{map(SyntacticFunction), Property}}$>?	`ASK WHERE { Domain Property Range . }`
GRT$_{\textbf{nounPhrase}}$: S -> the [NounSingular] [Preposition] <NP$_{\text{map(SyntacticFunction), Property}}$>?	`SELECT ?label WHERE { ?Domain Property ?Range . ?Domain rdfs:label ?label .}` or `SELECT ?label WHERE { ?Domain Property ?Range . ?Range rdfs:label ?label .}`

Table 14

SPARQL queries for grammar rule templates (as detailed in Section 2.5.1 that involve lexicon entries of type `NounPPFrame`.

Grammar rules template (GRT)	**SPARQL Query**
$\text{GRT}_{\text{subject}}$: S -> Who [Verb3rdPerson] <$\text{NP}_{\text{map(SyntacticFunction), Property}}$>? | Who [VerbPast] <$\text{NP}_{\text{map(SyntacticFunction), Property}}$>? | Which <$\text{NP}_{\text{Class, <Range,Property>}}$> [Verb3rdPerson] <$\text{NP}_{\text{map(SyntacticFunction), Property}}$>? | Which <$\text{NP}_{\text{Class, <map(SyntacticFunction),Property>}}$> [VerbPast] <$\text{NP}_{\text{map(SyntacticFunction), Property}}$>?	`SELECT ?label WHERE { ?Domain Property ?Range . ?Domain rdfs:label ?label .}` or `SELECT ?label WHERE { ?Domain Property ?Range . ?Range rdfs:label ?label .}`
$\text{GRT}_{\text{object}}$: S -> What is [VerbParticiple] [Preposition] <$\text{NP}_{\text{map(SyntacticFunction), Property}}$>? | What was [VerbParticiple] [Preposition <$\text{NP}_{\text{map(SyntacticFunction), Property}}$>? | Which <$\text{NP}_{\text{Class, <map(SyntacticFunction),Property>}}$> is [VerbParticiple] [Preposition] <$\text{NP}_{\text{map(SyntacticFunction), Property}}$> | Which <$\text{NP}_{\text{Class, <map(SyntacticFunction),Property>}}$> are [VerbParticiple] [Preposition] <$\text{NP}_{\text{map(SyntacticFunction), Property}}$>? | Give me all <$\text{NP}_{\text{Class, <map(SyntacticFunction),Property>}}$> [VerbParticiple] [Preposition] <$\text{NP}_{\text{map(SyntacticFunction), Property}}$>?	`SELECT ?label WHERE { ?Domain Property ?Range . ?Domain rdfs:label ?label .}` or `SELECT ?label WHERE { ?Domain Property ?Range . ?Range rdfs:label ?label .}`
$\text{GRT}_{\text{amount}}$: S -> How many <$\text{NP}_{\text{Class, <map(SyntacticFunction),Property>}}$> are [VerbPast] [Preposition] <$\text{NP}_{\text{map(SyntacticFunction), Property}}$>? | How often did <$\text{NP}_{\text{map(SyntacticFunction), Property}}$> [VerbPresent]?	`SELECT Count(?Domain) WHERE { ?Domain Property ?Range }` or `SELECT Count(?Range) WHERE { ?Domain Property ?Range }`
GRT_{ask}: S -> Did <$\text{NP}_{\text{map(SyntacticFunction), Property}}$> [VerbPresent] <$\text{NP}_{\text{map(SyntacticFunction), Property}}$>?| Does <$\text{NP}_{\text{map(SyntacticFunction), Property}}$> [VerbPresent] <$\text{NP}_{\text{map(SyntacticFunction), Property}}$>?	`ASK WHERE { Domain Property Range }`
$\text{GRT}_{\text{nounPhrase}}$: S -> <$\text{NP}_{\text{Class, <map(SyntacticFunction),Property>}}$> [VerbPresent] [Preposition] <$\text{NP}_{\text{map(SyntacticFunction), Property}}$> | <$\text{NP}_{\text{Class, <map(SyntacticFunction),Property>}}$> [VerbPast] [Preposition] <$\text{NP}_{\text{map(SyntacticFunction), Property}}$>	`SELECT ?label WHERE { ?Domain Property ?Range . ?Domain rdfs:label ?label .}` or `SELECT ?label WHERE { ?Domain Property ?Range . ?Range rdfs:label ?label .}`

Table 15

SPARQL queries for grammar rules template (as detailed in Section 2.5.2) that involve lexicon entries of type `TransitiveFrame`.

Grammar rules template (GRT)	**SPARQL Query**

GRT$_{subject}$:
S -> What [VerbPresent] [Preposition] <NP$_{map(SyntacticFunction), Property}$>? | Which <NP$_{Class, <map(SyntacticFunction),Property>}$> [VerbPresent] [Preposition] <NP$_{map(SyntacticFunction), Property}$>? | Which <NP$_{Class, <map(SyntacticFunction),Property>}$> [VerbPresent] [Preposition] <NP$_{map(SyntacticFunction), Property}$>?
	`SELECT ?label WHERE`
	`{ ?Domain Property ?Range .`
	`?Domain rdfs:label ?label .}`
	`or`
	`SELECT ?label WHERE`
	`{ ?Domain Property ?Range .`
	`?Range rdfs:label ?label .}`

GRT$_{object}$:
S -> What does <NP$_{map(SyntacticFunction), Property}$> [VerbPresent] [Preposition]? | Which <NP$_{Class, <map(SyntacticFunction),Property>}$> does <NP$_{map(SyntacticFunction), Property}$> [VerbPresent] [Preposition]? | Which <NP$_{Class, <map(SyntacticFunction),Property>}$> does <NP$_{map(SyntacticFunction), Property}$> [VerbPresent] [Preposition]? | [Preposition] which <NP$_{Class, <map(SyntacticFunction),Property>}$> does <NP$_{map(SyntacticFunction), Property}$> [VerbPresent]?
	`SELECT ?label WHERE`
	`{ ?Domain Property ?Range .`
	`?Domain rdfs:label ?lable .}`
	`or`
	`SELECT ?label WHERE`
	`{ ?Domain Property ?Range .`
	`?Range rdfs:label ?lable .}`

GRT$_{amount}$:
S -> How many <NP$_{Class, <map(SyntacticFunction),Property>}$> [VerbPresent] [Preposition] <NP$_{map(SyntacticFunction), Property}$>?
	`SELECT Count(?Domain) WHERE`
	`{ ?Domain Property ?Range }`
	`or`
	`SELECT Count(?Range) WHERE`
	`{ ?Domain Property ?Range }`

GRT$_{ask}$:
Does <NP$_{map(SyntacticFunction), Property}$> [VerbPresent] [Preposition] <NP$_{map(SyntacticFunction), Property}$> | Did <NP$_{map(SyntacticFunction), Property}$> [VerbPresent] [Preposition] <NP$_{map(SyntacticFunction), Property}$>
	`ASK WHERE`
	`{ Domain Property Range . }`

Table 16

SPARQL queries for grammar rules template (as detailed in Section 2.5.3) that involve lexicon entries of type `InTransitivePPFrame`.

Grammar rules template (GRT)	**SPARQL Query**
$\text{GRT}_{\textbf{subject}}$: S -> Which is a [AdjectiveProper] <$\text{Noun}_{\text{Class, RestrictionClass(Property, Object)}}$>? | Which are [AdjectiveProper] <$\text{Noun}_{\text{Class, RestrictionClass(Property, Object)}}$>? | Give me [AdjectiveProper] <$\text{Noun}_{\text{Class, RestrictionClass(Property, Object)}}$>. | List all [AdjectiveProper] <$\text{Noun}_{\text{Class, RestrictionClass(Property, Object)}}$>.	`SELECT ?Domain WHERE {?Domain Property Object . ?Domain rdf:type ?Class . }`
$\text{GRT}_{\textbf{nounPhrase}}$: S -> a [AdjectiveProper] <$\text{Noun}_{\text{Class, RestrictionClass(Property, Object)}}$> | [AdjectiveProper] <$\text{Noun}_{\text{Class, RestrictionClass(Property, Object)}}$>	`SELECT ?Domain WHERE {?Domain Property Object . ?Domain rdf:type ?Class . }`

Table 17

SPARQL queries for grammar rules template (as detailed in Section 2.5.4) that involve lexicon entries of type `AdjectivePredicateFrame`.

Grammar rules template (GRT)	**SPARQL Query**
$\text{GRT}_{\textbf{baseForm}}$: S -> How [AdjectivePositive] is <$\text{NP}_{\text{map(SyntacticFunction), Property}}$>?	`SELECT ?Range WHERE { ?Domain Property ?Range . }`
$\text{GRT}_{\textbf{comparative}}$: S -> Which <$\text{NP}_{\text{Class, <map(SyntacticFunction),Property>}}$> is [AdjectiveComparative] than <$\text{NP}_{\text{map(SyntacticFunction), Property}}$>? | Which <$\text{NP}_{\text{Class, <map(SyntacticFunction),Property>}}$> are [AdjectiveComparative] than <$\text{NP}_{\text{map(SyntacticFunction), Property}}$>?	`SELECT ?label WHERE { ?Domain Property ?Range1 . ?uri rdf:type ?Class . ?uri Property ?Range2 . ?uri rdfs:label ?label . FILTER ( ?Range2 > ?Range1 ) }`
$\text{GRT}_{\textbf{superlative}}$: What is the [AdjectiveSuperlative] <$\text{NP}_{\text{Class, <map(SyntacticFunction),Property>}}$>?	`SELECT ?Domain WHERE { ?Domain rdf:type ?Class . ?Domain Property ?Range } order by desc(?Range) limit 1` or `SELECT ?Domain WHERE { ?Domain rdf:type ?Class . ?Domain Property ?Range } order by asc(?Range) limit 1`

Table 18

SPARQL queries for grammar rules template (as detailed in Section 2.5.5) that involve lexicon entries of type `AdjectiveSuperlativeFrame`.

Prompt

For German:

```
Du bist ein Assistent, dessen Aufgabe es ist, eine SPARQL-Abfrage für DBpedia zu generieren,  um eine gegebene Frage zu
beantworten. Bevor ich dir die Frage zeige, zeige ich dir einige Beispiele von Fragen und den dazugehörigen SPARQL-Abfragen.
Frage: Wer ist der Bürgermeister von New York City?
SPARQL: SELECT DISTINCT ?uri WHERE { <http://dbpedia.org/resource/New␣York␣City> <http://dbpedia.org/ontology/leaderName> ?uri}

Frage: Zeig mir alle Schwimmer, die in Moskau geboren wurden.
SPARQL: SELECT DISTINCT ?uri WHERE { ?uri a <http://dbpedia.org/ontology/Swimmer> ; <http://dbpedia.org/ontology/birthPlace>
                                     <http://dbpedia.org/resource/Moscow> }

Frage: Wer hat Goofy erfunden?
SPARQL: SELECT DISTINCT ?uri WHERE { <http://dbpedia.org/resource/Goofy> <http://dbpedia.org/ontology/creator> ?uri}

Frage: Gib mir alle dänischen Filme.
SPARQL: SELECT DISTINCT ?uri WHERE { ?uri <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://dbpedia.org/ontology/Film>;
                                     <http://dbpedia.org/ontology/country> <http://dbpedia.org/resource/Denmark> }

Frage: Was ist der höchste Berg?
SPARQL: SELECT DISTINCT ?uri WHERE { ?uri a <http://dbpedia.org/ontology/Mountain> ; <http://dbpedia.org/ontology/elevation>
                                    ?elevation } ORDER BY DESC(?elevation) OFFSET 0 LIMIT 1

Finde jetzt eine SPARQL-Abfrage für DBPEDIA für die folgende Frage: Erstelle nun eine SPARQL-Abfrage für DBpedia, um die
folgende Frage zu beantworten: "FRAGE".  Stelle sicher, dass die Abfrage relevante Informationen effizient abruft, indem du
geeignete  Filter, Eigenschaften und Namensräume verwendest. Erkläre nichts.
```

For Italian:

```
Sei un assistente il cui compito è generare una query SPARQL per DBpedia per rispondere a una domanda data. Prima di mostrarti
la domanda, ti mostro alcuni esempi di domande e le relative query SPARQL.
Domanda: Chi è il sindaco della città di New York?
SPARQL: SELECT DISTINCT ?uri WHERE { <http://dbpedia.org/resource/New␣York␣City> <http://dbpedia.org/ontology/leaderName> ?uri}

Domanda: Dammi tutti i nuotatori che sono nati a Mosca.
SPARQL: SELECT DISTINCT ?uri WHERE { ?uri a <http://dbpedia.org/ontology/Swimmer> ; <http://dbpedia.org/ontology/birthPlace>
                                     <http://dbpedia.org/resource/Moscow> }

Domanda: Chi è il creatore di Pippo?
SPARQL: SELECT DISTINCT ?uri WHERE { <http://dbpedia.org/resource/Goofy> <http://dbpedia.org/ontology/creator> ?uri}

Domanda: Dammi tutti i film danesi.
SPARQL: SELECT DISTINCT ?uri WHERE { ?uri <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://dbpedia.org/ontology/Film>;
                                     <http://dbpedia.org/ontology/country> <http://dbpedia.org/resource/Denmark> }

Domanda: Qual è la montagna più alta?
SPARQL: SELECT DISTINCT ?uri WHERE { ?uri a <http://dbpedia.org/ontology/Mountain> ; <http://dbpedia.org/ontology/elevation>
                                    ?elevation } ORDER BY DESC(?elevation) OFFSET 0 LIMIT 1

Ora genera una query SPARQL per DBpedia per rispondere alla seguente domanda: "DOMANDA". Assicurati che la query recuperi le
informazioni rilevanti in modo efficiente, utilizzando filtri, proprietà e namespace appropriati. Non spiegare nulla.
```

For Spanish:

```
Eres un asistente cuya tarea es generar una consulta SPARQL para DBpedia para responder a una pregunta dada. Antes de
mostrarte la pregunta, te mostraré algunos ejemplos de preguntas y las consultas SPARQL correspondientes.
Pregunta: ¿Quién es el alcalde de la cuidad de Nueva York?
SPARQL: SELECT DISTINCT ?uri WHERE { <http://dbpedia.org/resource/New␣York␣City> <http://dbpedia.org/ontology/leaderName> ?uri}

Pregunta: Dame todos los nadadores nacidos en Moscú.
SPARQL: SELECT DISTINCT ?uri WHERE { ?uri a <http://dbpedia.org/ontology/Swimmer> ; <http://dbpedia.org/ontology/birthPlace>
                                     <http://dbpedia.org/resource/Moscow> }

Pregunta: ¿Quién es el creador de Goofy?
SPARQL: SELECT DISTINCT ?uri WHERE { <http://dbpedia.org/resource/Goofy> <http://dbpedia.org/ontology/creator> ?uri}

Pregunta: Dame todas las películas danesas.
SPARQL: SELECT DISTINCT ?uri WHERE { ?uri <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://dbpedia.org/ontology/Film>;
                                     <http://dbpedia.org/ontology/country> <http://dbpedia.org/resource/Denmark> }

Pregunta: ¿Cuál es la montaña más alta?
SPARQL: SELECT DISTINCT ?uri WHERE { ?uri a <http://dbpedia.org/ontology/Mountain> ; <http://dbpedia.org/ontology/elevation>
                                    ?elevation } ORDER BY DESC(?elevation) OFFSET 0 LIMIT 1

Ahora genera una consulta SPARQL para DBpedia para responder a la siguiente pregunta: "PREGUNTA". Asegúrate de que la
consulta recupere la información relevante de manera eficiente, utilizando los filtros, propiedades y espacios de nombres
apropiados. No expliques nada.
```

Table 19

ChatGPT prompts for few-shot scenarios in German, Italian, and Spanish.