Multilingual Question Answering using Lexico-Syntactic Patterns

Nikolay Radoev [a,*], Amal Zouq [a] and Michel Gagnon [a]
[a] *Software and Engineering Deaprtment, Ecole Polytechnique de Montreal, QC, Canada*
E-mail: nikolay.radoev@polymtl.ca

Abstract. Continued work on existing knowledge bases (KBs) has given acces to a large amount of structured data. However, most of the existing QA systems using those KBs are designed to handle questions in a single language (mostly English) and those that handle multiple languages have a rather generic approach, leading to reduced overall performance.

We present a different method for transforming natural language questions into SPARQL queries. Our method focuses on leveraging the syntactic information of questions to generate one or multiple SPARQL triples. We present a QA system aimed at multilingual query processing and describe a set of lexico-syntactic patterns used to generate the SPARQL queries. Our evaluation was done by applying the described patterns on our QA system over DBpedia and measure the impact on the overall performance.

Keywords: Question answering, Lexio-syntactic Patterns, DBpedia, SPARQL Query Generation

1. Introduction

Many applications rely on the SPARQL standard in order to query data based on an RDF model. The use of SPARQL does however come with a significant drawback represented by its steep learning curve. Several systems have been developed to hide SPARQL from the average user by accepting a specific set of inputs that is used to generate specific SPARQL queries. Some systems opt in for a keyword search where every word or group of words is considered as a separate entity and no additional semantic or syntactic properties of the query are being considered. SPARQL queries are then built by trying different combinations of all the words and returning only those which obtain some answer. While those systems are effective, they are reliable only for simple queries and do not deal well with ambiguities. More so, keyword-only approaches can sometimes feel *unnatural* to users. Some more sophisticated systems are able to accept questions entirely written in natural language without imposing additional constraints to the user. Such systems include [1–4] and take into consideration not only keywords but also additional semantic and syntactic representations to build more accurate and complex SPARQL queries.

While natural language questions written without any restrictions can be complex, it can be observed that people often ask similar questions and similar structures can be found in different questions [5]. This paper aims to present the question-answering system LAMA [6] based on various multilingual (French / English) lexico-syntactic patterns that can help generate corresponding SPARQL queries. These patterns can be used in any question-answering (QA) system that wants to leverage the power of syntax and POS-tagging to generate SPARQL queries. We show the relevance and effectiveness of the proposed patterns on two different question-answering datasets [7, 8].

The remainder of the paper is structured as follows. In the next section, we analyze the two datasets and describe our LAMA system. We then detail the two different types of patterns used by the system. This is followed by an evaluation on the datasets, both for determining the patterns' frequency and their impact on a question answering system. After a discussion on the evaluation results and a review of related work, we conclude with some final remarks.

[*]Corresponding author. E-mail: nikolay.radoev@polymtl.ca.

2. Methodology

In this section, we describe in more detail the datasets and the system used to evaluate the proposed patterns. A more in-depth description of the datasets allows for a better understanding of their particularities and the type of questions that can be asked to a QA system by the average user. A description of the *LAMA* system also allows us to give an example of a practical application of the patterns outside of their theoretical description.

While DBpedia is used as the reference knowledge base throughout this paper, the work presented is not limited to DBpedia and can be applied to any other knowledge base. For readability purposes, we have abbreviated some of the URIs by using the prefixes in Table 1. The *dbo:* represents classes and concepts in DBpedia's ontology while *dbp:* and *dbr:* represent properties and resources respectively.

Table 1
DBpedia prefixes

dbo	http://dbpedia.org/ontology/
dbr	http://dbpedia.org/resource/
dbp	http://dbpedia.org/property/

2.1. Dataset and Question Analysis

The work presented in this paper is based on two different corpora containing questions designed to be ran against a particular knowledge base, in this case DBpedia. Each question also contains the expected answer as well as a sample SPARQL query used to obtain that answer. We classify each question as either *simple* or *complex*. A question is defined as simple if it can be translated into a SPARQL query that contains only one triple pattern, otherwise it is considered as a complex question. For example, the question *Who died of malaria?* is a simple question since it can be expressed using the following SPARQL triple pattern:

?x dbo:deathCause dbr:malaria .

The question *Who died from malaria in North Borneo?* is considered a complex question since it requires the 2 following triple patterns for a complete answer:

?x dbo:deathCause dbr:malaria .
?x dbo:deathPlace dbr:North_Borneo .

We now present the two datasets that have been used in our experiments.

Question Type	QALD	SQA
Date	6.5%	1.3%
Number	6.8%	2.7%
Boolean	21.0%	7.4%
Resource	65.8%	88.6%

Table 2
Question Types per Dataset

2.1.1. QALD Dataset

The QALD dataset is a combination of both the QALD7 and QALD8 training datasets provided for the QALD competition [7] in 2017 and 2018, respectively. Given their similarity, both datasets were merged for a total of 560 questions. Duplicate questions appearing in both sets were removed for a final count of *384* unique questions. Filtering was done purely on complete string match and thus questions like *What basketball players were born in X?*, where *X* is a different location name in each dataset, were kept as different queries.

QALD is overwhelmingly composed of simple questions (298 out of the 384 questions), representing 78% overall. The general distribution per type can be seen in Table 2. We partitioned the questions according to the type of the expected answer. The last category (Resource) designates questions for which the expected answer is one or more URIs and for which no other more appropriate category was found The dataset is mostly composed of *Resource* questions but has a non-negligible amount of *Boolean* questions.

The QALD questions are also translated in multiple languages (6 different languages including French, English, Spanish, Italian, German and Danish) but only the French and English translations were considered in our experiments. The provided SPARQL query and answers only contain references to the English version of DBpedia and no alternative answers are provided.

2.1.2. SQA Dataset

The SQA dataset is similar to the QALD in its structure but does only contain questions in English while still providing both the answers to the questions and the SPARQL queries used to obtain those answers.

Despite having a larger number of questions than QALD, the SQA dataset contains many questions that are redundant in their structure. For example, the question *"Who was married to X?"* appears 7 times in the dataset with different entities at position *X*. Just like QALD, those types of questions are kept in the final dataset.

As for the question type distribution, SQA differs from QALD by presenting a high amount (3853 or 77.1%) of complex questions. The general type distribution is provided in Table 2 and shows a significant bias towards *Resource* type questions.

It is to be noted that the SQA dataset contains a non-negligible amount of noise represented by spelling mistakes, wrong capitalization and missing words in some of the provided questions. This can have an impact on the final performance analysis since both syntactic parsing and POS tagging are sensitive to such noise, especially if some words are missing.

2.2. *Overview of the LAMA System*

One of the main reasons for exploring a pattern-based approach to generate SPARQL queries from natural language questions was to enhance our question answering system *LAMA* [6] and reduce its reliance on ad-hoc heuristics and pre-defined rules.

LAMA (Language-Adaptive Method for Question Answering) is a system originally designed to answer simple questions in both French and English. Even though it has been originally targeted to simple questions, the first version of LAMA (AMAL, [9]) was still able to answer a very limited amount of complex questions.The system is modular and offers multiple components that can be modified and replaced with custom ones if a different behaviour is desired, as long as a corresponding adapter is provided for the new endpoint. Some of the components are optional, such as the *Translation Module*, which can be used to translate questions in English and use the answering module for this language, thus leveraging the larger amount of data in English.

Figure 1 shows a high-level representation of the system's architecture. The system adopts a module-oriented structure where every module is responsible for a specific task and can be swapped for a different module if desired. The preprocessing modules (Syntax Parser and Question Classifier) generate additional information (dependency tree, POS tags, question type) that is passed to the core module: the *Question Solver* module.

This module exposes an interface with a single method, *GetAnswer(string question)*. The interface can be implemented based on the needs of the system and can use the different modules represented inside the Question Solver box in the figure. The three main components, *Entity Extractor, Property Extractor* and *SPARQL Generator*, are required for a correct query processing while the *Translation Module* is optional and the *Lexicon Generator* can be omitted if a lexicon is already provided or not used at all. A more detailed explanation of the different modules is given in the following sections.

The architecture also interacts with several external resources: DBpedia, Wikipedia and the Google Translate API. In the case of Wikipedia, it is not an active component of the system since it was used only to train a Word2Vec model for the *Property Extractor* module in both English and French [10]. Other Word2Vec models [11] can be used, but given the wealth of data and the proximity between DBpedia and Wikipedia, Wikipedia remains a good source for model training in multiple languages.

Figure 2 offers a more detailed look of the system's process and how the architecture is translated in a processing pipeline. LAMA's pipeline is composed of 3 main steps, indicated by different colors between the initial question and the final answer : *Pre-Processing*, *Syntax Tree Representation* and *SPARQL Query Generation*. Each step uses one or more of the components shown in Figure 1 with more details given in the following sections.

2.2.1. *Question Parsing.*

There are many different existing frameworks for parsing natural language sentences, focusing on different particularities of the language. We rely on Google's Cloud Natural Language API [12] (CNL), which combines multiple tools for different tasks. Based on the *SyntaxNet* projet, Cloud Natural Language API offers syntactic parsing, POS tagging, dependency parsing and basic entity annotation. Another interesting property of this tool is that it supports many languages.

CNL's parsing is done on pre-trained models, with English being based on the Penn Treebank and OntoNotes corpora [13]. To keep uniformity in this paper and all the given examples, the *Universal Dependencies* project notation is used. As an example, using this notation on the sentence *When was the statue of liberty built?*, we generate the following POS-tags :

When(ADV) was(VERB) the(DET) Statue(NOUN) of(ADP) Liberty(NOUN) built(VERB) ?

As for the dependency parsing, the universal annotation is also being used. As not all dependencies are represented in all languages, Table 3 presents the most used dependencies, as well as a brief explanation for each.

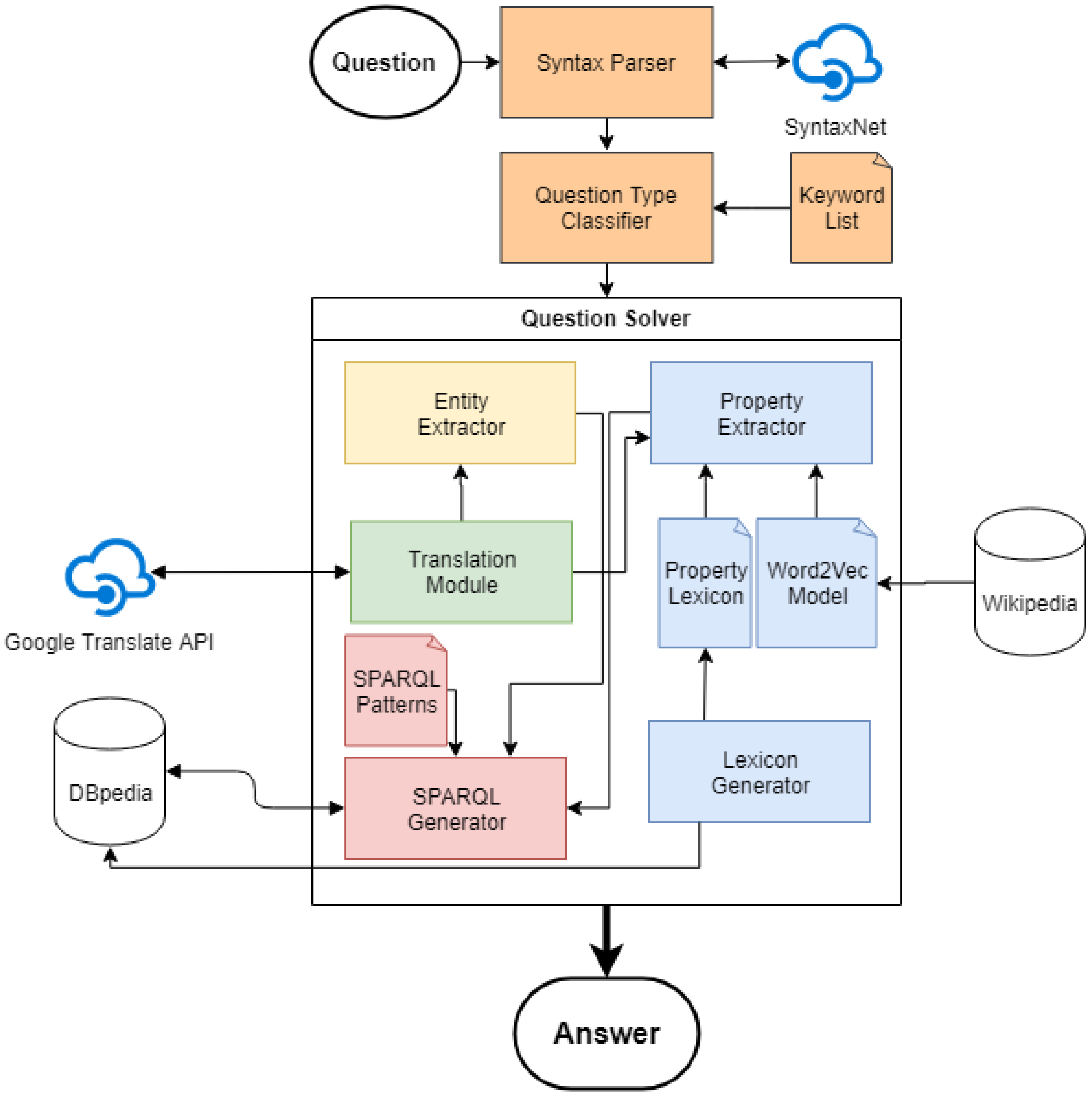


Figure 1. LAMA's Architecture

2.2.2. *Pre-processing and Question Type Classification.*

The Pre-processing step involves parsing the input query to extract the sentence's syntactic tree representation as well as the POS tag for each word. This step is explained in more detail in sections 2.2.1 and 4. Retrieved data is saved and passed forward into the pipeline. Following this step, the system classifies the question into one of the following categories : *Boolean, Date, Number* and *Resource*. An additional subtype, *Aggregation* is applied to questions that require counting or ordering (descending or ascending) of results.

Classification is done by using patterns that have been manually extracted from the QALD6 and QALD7 training data sets [6]. The pattern-based approach is relatively easy to implement and can be adapted to a multilingual setting by requiring a separate set of

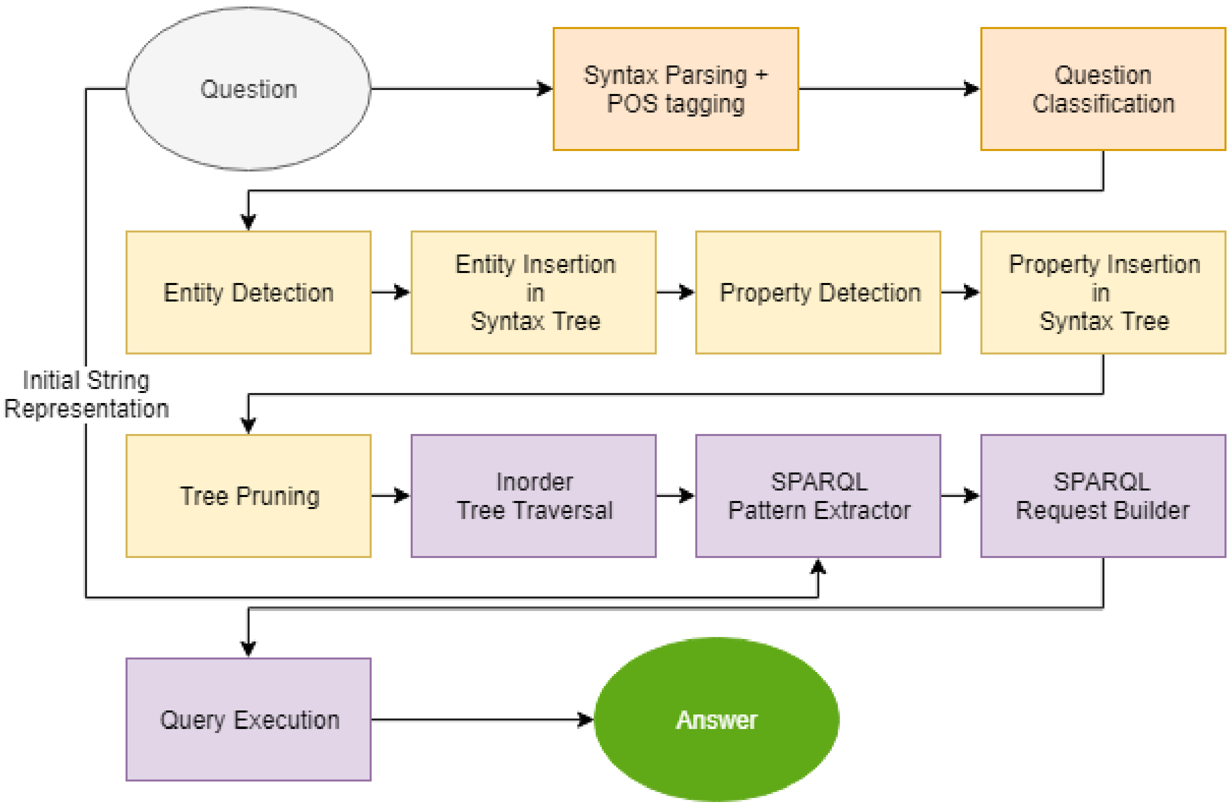


Figure 2. LAMA's Pipeline

Dependency	**Details**	**Example**
subj(V,S)	Denotes a relation between the verb V and its subject S	subj(be,Obama) => Obama is
dobj(V,O)	Dependency between a verb V and its direct object O	dobj(buy,book) => buy a book
amod(N,A)	Dependency between an adjective A and the modified noun N	amod(building,old) => old building
conj(A,B)	Conjunction between two elements	conj(Michel,Amal) => Michel and Amal

Table 3
Dependency details

user-defined patterns for a new language. Applied only to French and English, the question classification was able to accurately predict the question type of 92% of the QALD7 test set. The remaining 8% were instances where a more specific type was not detected and was declared as *Resource* by default. In some cases, information stored in DBpedia uses the wrong format. For instance, when asking for the budget of the Lego Movie, the answer is a string literal and not a *number*. The impact of a wrong classification is minimal as long as a question is not classified as *Boolean*, given that those type of questions are answered with an ASK query that only returns true or false.

Based on the classification result, the question is passed to a specific *Question Solver* that implements custom rules and heuristics to better answer particular question types. For example, The solver for *Date* type questions will try to look for keywords such as *When, What time, What date, etc* or words representing time such as *birthdate, ending, etc*.

2.2.3. Lexicon Generation.

In order to help with the property extraction, we built a property lexicon based on our chosen knowledge base DBpedia. For each property we extracted its URI as well as the corresponding labels in different languages, French and English in our case. Some properties in the *dbo:* domain have labels for both languages, such as *dbo:author* : author(en) and auteur(fr) while others only have English labels. The extraction was ran both on the *dbo:* and *dbp:* domains. Extraction was also ran on the *dbp:* domain of the French version of DBpedia since language-specific properties are defined in this domain and not *dbo:*. For each label, we mapped all corresponding URIs. When several URIs exists in dbo and dbp, we favor the URI of the *dbo:* domain first. For instance, the label "parent" has 2 URIs in the namespaces *dbp:* and *dbo:* and thus lead to the following mapping :

{"parent" : [dbo:parent, dbp:parent] }

The generated lexicon is stored as a hashtable, allowing for a fast lookup with an average complexity of O(1) and reducing the number of network calls required per query analysis. It allows us to find existing properties based on their label and its presence in the question. When working with languages other than English, the translation of the potential property can be used for properties that do not have labels for the original language.

While our lexicon is extracted automatically from DBpedia, it can still be enriched by adding additional bindings that are either generated by other means or created manually. This can be especially helpful in cases where a certain property is expressed using literals that are quite different than the label used in the Knowledge Base. LAMA uses a different approach for such cases relying on word embeddings as explained in the next section.

2.2.4. Entity and Property Extraction.

After the question is parsed and classified, the system tries to extract as much semantic information from the question as possible, starting with the entities.

First, a coarse-grained extraction is done by identifying and removing the question word *(who is, who are, when was, etc.)*. The remaining string is then decomposed into all possible substrings, which are searched in the target knowledge base (DBpedia in this case) after appending the *dbr:* prefix and replacing the space character by the underscore character. All valid entities (i.e. the ones that exist in the knowledge base) are kept as possible candidates. One of the advantages of this method is that only existing entities are kept and the system guarantees that if an entity is used, its URI points to an existing entry in the knowledge base.

For example, *Who is the queen of England?* generates the following sub-strings after removing the question indicator (*who is*): *queen of England, queen, England, queen of, of England*. Out of those, only the first 3 are kept as valid entities since *queen of* and *of England* are not DBpedia entities. Based on the assumption that entities are most likely a noun or a part of a NP (queen of England for example), potential entities extracted from nouns are ranked higher than potential candidates from verbs or other grammatical groups. To increase the set of potential entities, we add lemmatization and capitalization but penalize entities discovered by these transformations. For example, the word *queen* can also lead to *queens, Queen* and *Queens* but all those have a lower score than the original word. If no entities are found using all these methods, DBpedia Spotlight [14] is used as a back-up tool.

In the case of languages other than English, an optional translation step can be taken where the initial question is translated to English using Google Translate. In fact, the French DBpedia chapter is less complete than its English counterpart and all the question answering competitions (QALD, SQA) expect a URI from the English DBpedia. This translation increases the chance of finding an entity, especially in languages with limited presence in a knowledge base (KB), but comes with a potential risk of a false negative since we cannot guarantee that the provided translation matches the English label in the KB.

The different modifications (capitalization, translation, stemming) are combined to calculate a score that is used to rank the entities. The score is computed as follows, with *e* being the entity string:

$$\mathbf{S}(\boldsymbol{e}) = length(e) - T \times \frac{length(e)}{2} + 2 \times U \times nsp(e) - \mathrm{P}$$

where:

$nsp(e)$ is the number of spaces in *e*

P is the number of characters added or removed if plural form was added or removed, respectively.

$T = 1$ if *e* was translated, 0 otherwise

$U = 1$ if *e* has no capitalization applied, 0 otherwise

We consider the number of spaces for entities that span multiple words, often names or titles of movies, paintings, etc. This is however only considered if some of the words in the entity are already capitalized. This is done to reduce the risk of transforming unrelated word into entities. For instance *the creator* can be transformed into *the Creator*, an existing entity in DBpedia but incorrect in this case. Similarly, translated word groups incur a penalty proportional to their length. This is done to prioritize words in the native language of the query and to reduce translation errors. The penalty factor of 2 was determined empirically in the first versions of the system. With this formula and our previous example, the entity *queen of England* (score of 18) is ranked higher than the other two based on its length and the fact that it is composed of many words.

Both dependency (Table 4) and POS (Table 6) patterns rely on identifying particular words as subjects or objects in the sentence, that will be reused in the generated SPARQL query. In order to facilitate the transformation from a string literal to a valid URI, the system offers the function **getEntity(x)**, which matches the label *x* to one of the already detected entities in the question. If more than one valid entity (in the case of multi-word entities) are found, they are all returned along with their respective rankings. For instance, the question *Did Tolkien write the Hobbit?* can be matched with a lexico-syntactic pattern that recognizes the subject *Tolkien* and the object *Hobbit*, which are respectively tagged as the potential subject and object in the SPARQL triple. The application *getEntity(Tolkien)* will return *dbr:J._R._R._Tolkien*, while *getEntity(Hobbit)* will return both *dbr:Hobbit* (the fictional race) and *dbr:The_Hobbit* (the book). Based on the score computed as explained before, *dbr:The_Hobbit* is correctly chosen as the most likely candidate for the SPARQL triple.

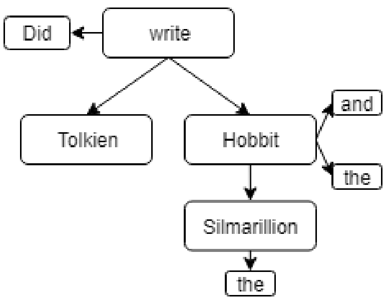
After extracting the entities, the system detects and extracts possible properties from the question. Unlike entities, in most cases, the expression of the property cannot be directly matched with its representation in the knowledge base. For example, using the same question as before, *dbr:J._R._R._Tolkien* and *dbr:The_Hobbit* are connected by the *dbo:author* property in DBpedia, while in the question this property is expressed by the word *write*. We first try to match the label to an existing property in our property lexicon either by a full match to the word label or to its derivative. A derivative is defined as a literal with a Levenstein distance of less than 3, applied only to the end of the initial word label. While this is helpful, it does not cover cases where the desired property is significantly different from the word label.

To alleviate this problem, LAMA uses a Word2Vec word embeddings model trained on Wikipedia to match a potential word to a valid property as long as the cosine similarity between the vector representation of the words is above a certain threshold (0.6 in our case). To reduce the number of false positives, a property is considered valid only if it exists in the target base **and** is used in relation with at least one of the extracted entities. Taking *write* as an example, we cannot find a direct valid property, but its derivative *writer* can be mapped to *dbo:writer*, a valid property in our lexicon. This however does not satisfy the second condition since neither of the two entities uses it as a property. Using the Word2Vec model, we find a cosine similarity of 0.719 between *writer* and *author*, a word that can be mapped to the DBpedia property *dbo:author*, which is also related to both entities. In the case of multiple properties matching all the criteria, they are all saved and ranked based on their cosine similarity.

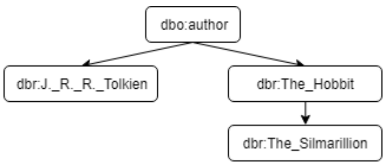
Like the entity extractor, the property extractor module offers a helper function called **getProperty(X)** where *X* is the word denoting the potential property. In some questions, no expression can be targeted as a potential property since the relation between two entities is implicit. In those cases, *X* is a pair of entities and the system tries to find at least one valid property that connects those entities. For example, the question *"Was Margaret Thatcher a chemist?"* contains the entities *dbr:Margaret_Thatcher* and *dbr:Chemist* but no other words denoting a relationship since *Was* is a question word and is thus removed. However, looking into DBpedia, we find that both of those entities are connected by the *dbo:profession* property which is the property returned by the *getProperty()* function.

2.2.5. Syntax Tree Representation

After each extraction step, the original syntactic tree representation is modified by replacing words with their corresponding entities or properties while maintaining the dependency between those new nodes. In some cases, multiple words are replaced by a single node, most commonly noun phrases representing a single entity. In such cases, dependencies between the words are removed as they are merged into a single node. During this process, the tree is no longer a purely syntactic representation but has semantic information injected into it. Finally, words that are not mapped to an entity or a property are considered as *filler* words and are thus removed. Such words are most often question markers such as *who, when, where, did, etc.* or left-over determinants. It is important to note that for question markers such as *when*, which denote a possible *DATE* type question, their information is still kept, but not in the tree representation.



(a) Syntactic tree



(b) Semantic representation after entity and property extraction

Figure 3. Tree representation for the question: Did Tolkien write the hobbit and the Silmarillion?

For example, as seen in Figure 3 the question *Did Tolkien write the Hobbit and the Silmarillion?* is transformed in *[dbr:J._R._R._Tolkien] [dbo:author] [dbr:The_Hobbit] [dbr:The_Silmarillion]* with the word *Did* being pruned and the determinants *the* being merged into the new entities.

This process allows to simplify the question's representation by removing useless words and gradually building partial semantic data for some segments. This is particularly useful when handling complex questions. Also, for some complex questions, the resulting representation can be analyzed as separate simple questions.The syntax tree traversal is done inorder, i.e. traversing nodes in a *left - root - right* pattern.

For example, the question *What cars made in Canada are electric?* can be represented as [What cars made in Canada] - [are] - [electric], where the left subtree is analyzed separately and mapped to a variable *X* which is the reused as the left node of the rest of the tree: [X] - [are] - [electric].

2.2.6. *SPARQL Query Generation*

The last step of the process pipeline is the building of the SPARQL query. The query generated by LAMA can either take the form of ASK (for *Boolean* questions) or SELECT queries for all other type of questions. The system also supports the ORDER BY modifier and the COUNT function when handling sorted or aggregation-based questions. After building and executing the SPARQL query, only non-null answers are kept, with the exception of ASK queries, which always return either *true* or *false*.

The *SPARQL Pattern Extractor* takes information from the various patterns (these are detailed in the next section) applied to the initial question and generates the corresponding triple or set of triples as represented in Tables 4 and 6. The result of this step along with information generated by previous steps is passed on to the *SPARQL Request Builder* that creates the final SPARQL query. In the case of multiple possible triples and/or combinations, the different possibilities are also generated and stored. For example, if there are 2 different possible properties for a a triple, 2 different SPARQL triples are generated, each with one of the two properties.

The generated queries that are independent from each other are ran in parallel against the standard DBpedia endpoint : *https://dbpedia.org/sparql* with a built-in timeout of 5 seconds to prevent system lock and to limit computation time. A query that times out is considered as returning an empty answer (or false for Boolean questions).

2.3. *SPARQL queries and triple patterns*

A standard SPARQL query, according to the standard definition [15], contains three parts :

– a body section describing the data to be retrieved
– an optional section describing the data that can be retrieved if available
– a modifier section with all the additional modifications to be applied to the data retrieved from the previous sections

In this paper, we are mainly interested in the first section, the *body* of the SPARQL request. The *body* is composed of triple patterns, similar to RDF triples.

Given a set of patterns *P*, we aim to generate the set *S* containing one or more SPARQL triples that represent the semantics of a given question. A single pattern $p \in P$ generates at least one SPARQL triple. When more than one triple are generated, some of the variables can be shared. For example, the question *Who are the people who played a sport in the Olympics?* can be expressed using the following SPARQL triple set where the object of the first triple is also the subject of the second one:

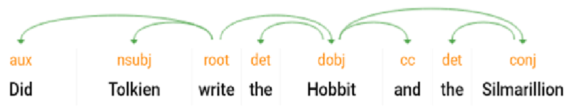


Figure 4. Dependency parse of a simple query

```
?people dbo:playedIn ?x .
?x dbo:sportOf dbr:Olympic_Games .
```

In theory, such *triple chaining* can be done with potentially an unlimited amount of triples. However, in practice, most questions rarely require such a large number of triples. These queries can also be split in individual triple queries that can be ran in a sequence, allowing for all the intermediate results to be available for storage and additional use.

Each of the three elements in a SPARQL triple can be a variable. Variables can either be *bound* to specific URI or literal or *free* and take any value in their domain. In the case of free variables, they are expressed using the *?x* form. Let R be all resources, P all properties and L all literals in the knowledge base being targeted by the SPARQL query and let X be the set of all variables usable in the SPARQL query. A single SPARQL triple can be defined as $(s,p,o) \in (X \cup R) \times (X \cup P) \times (X \cup R \cup L)$. In the following sections, we detail the dependency-based and parts-of-speech patterns used in LAMA.

3. Dependency-based Patterns

3.1. Dependency Parsing

As already stated, our work is based on using SyntaxNet as a dependency parser. SyntaxNet is a transition-based dependency parser [16], meaning it processes data from left to right and it creates *dependency* arcs between the different tokens in the initial query. After the parsing, SyntaxNet produces a single direct acyclic graph representing the dependencies between all words in a given sentence. Figure 4 shows a graphic representation of a simple question and its dependency parse tree.

For every dependency arc in the tree representation, we can create pairs of *head* and *modifier* tokens inside a dependency relation. For example, *write* and *Hobbit* are represented as *dobj(write,Hobbit)* with *dobj(v,o)* being the dependency relation. We do not need to transform all dependency arcs into pairs, since not all dependencies have the same usefulness. Such dependencies can be ignored and even classified as *noisy* and thus be removed in order to simplify the analysis of a question. One frequent case is the dependency between a determinant and its head noun.

3.2. Lexico-Syntactic Pattern Representation

As previously mentioned, the first set of patterns presented are based on the dependency graph generated by the parser. For each pattern, we show both a visual representation as well as the dependency relations based on the universal representation. We also give the generated SPARQL triples representing the semantics of the pattern. The list of patterns is shown in Table 4.

Pattern detection can be best illustrated with a specific example. Using the question *Did Tolkien write the Hobbit and the Silmarillion?*, we can observe the presence of the last pattern described in Table 4.

The dependency tree of the question is already presented in Figure 4 and using that representation, we can extract the following dependency relations (Note that we have here a distributive interpretation of the conjunction):

- subj(Tolkien,write)
- dobj(write, Hobbit)
- dobj(write, Silmarillion)
- conj(Hobbit, Silmarillion)

We now recognize our last pattern where we can use the following mapping :

- S = Tolkien
- V = write
- O_1 = Hobbit
- O_2 = Silmarillion

And using our SPARQL helper functions, we can generate the following SPARQL triples:

dbr:Tolkien dbo:author dbr:Hobbit.
dbr:Tolkien dbo:author dbr:Silmarillion.

The same example can also work if we replace *Did Tolkien write ...* with *Who wrote ...*. In this case, the pattern also applies, but the subject of the generated triple is replaced by the free variable *?x* based on the question word *Who*. The only difference in the final SPARQL query is that the absence of a free variable leads to an ASK form, while its presence indicates the need to use the SELECT form.

#	Diagram	Dependency pattern	SPARQL	Example
1	$$\begin{tikzcd} S & V \arrow[l, "subj"'] \arrow[r, "obj"'] & O \end{tikzcd}$$	subj(v, s) obj(v, o)	getEntity(s) getProperty(v) getEntity(o).	Did Barack marry Michelle ?
2	$$\begin{tikzcd} X \arrow[r, "amod"'] & Y \end{tikzcd}$$	amod(x/$_{ADJ}$,y/$_{NN}$)	getEntity(y) getProperty() getEntity(x).	Canadian athlete
3	$$\begin{tikzcd} V_1 \arrow[r, "subj"'] \arrow[rr, bend left, "dobj"] & S_1 & D_1 \\ V_2 \arrow[rru, "subj"] \arrow[rr, "dobj"'] & & D_2 \end{tikzcd}$$	subj(V_1,S_1) dobj(V_1,D_1) subj(V_2,D_1) dobj(V_2,D_2)	getEntity(s_1) getProperty(v_1) ?x. ?x getProperty(v_2) getEntity(d_2)	Give me people who played a sport that is in the Olympics.
4	$$\begin{tikzcd} S & V \arrow[l, "subj"'] \arrow[dl, "dobj"'] \\ O_1 \arrow[r, "conj"'] & O_2 \end{tikzcd}$$	subj(S,V) dobj(V, O_1), conj(O_1,O_2)	getEntity(s) getProperty(v) getEntity(o_1). getEntity(s) getProperty(v) getEntity(o_2)	Did Tolkien write the Hobbit and the Silmarillion?

Table 4

Lexico-Syntactic Patterns

4. POS-based Patterns

4.1. POS Tagging

In addition to dependency patterns, the Request Builder module of our system also uses POS-based patterns.

We rely on a POS (Part of Speech) tagger that assigns a tag that specifies the grammatical function of each of the words in a given sentence. For words that may have many functions, depending on the context, the tagger will select the correct one. For example, the word *saw* that can be a verb as in *I saw the Hobbit movie last night* or in *He saw the table in half* or a noun as in *I bought a new saw*.

In our experiments, we used the SyntaxNet tagger, with the default configuration. It is based on the *Universal Dependencies* project with some minor modifications in notation [17]. A full list can be see in Table 5. For the sake of readability, *NN* and *VB* are used instead of *NOUN* and *VERB* throughout this paper. It is however important to note that *NN* does not represent only singular nouns as in the *Penn Treebank Project* but all type of nouns. The tagger used offers a coarse-grained level of POS tagging represented by the tags in Table 5.

POS tag	Meaning
ADJ	Adjective
ADP	Adposition (preposition and postposition)
ADV	Adverb
CONJ	Conjunction
DET	Determiner
NOUN/NN	Noun
NUM	Cardinal number
PRON	Pronoun
PRT	Particle
PUNCT	Punctuation
VERB/VB	Verb (all tenses and modes)
WP	Wh-pronoun
X	Others
AFFIX	Affix

Table 5

POS tags

4.2. POS Pattern Representation

In this section, we show a few POS-based patterns that can be mapped to one or more SPARQL triples in order to generate a SPARQL query representing the semantics of the original question. The patterns are presented in more detail in Table 6.

POS patterns can be used by themselves or in conjunction with dependency-based patterns, as it is the case in LAMA. Using POS patterns can help by covering additional use cases, confirming already generated triples or generating a correct tagging when the sentence is inaccurately handled by the syntactic parser. For instance, both first patterns in Tables 4 and 6 detect similar representations so only one type of pattern is necessary to cover these specific cases. However, this redundancy can help by extracting patterns using POS that are missed due to an incorrect dependency parse. POS tagging using SyntaxNet (Parsey McParseface) has a very high rate of accuracy : 96.27% for French and 95.34% for English.

For each pattern, we give the pattern itself represented in the format X_{tag} where X is the token and *tag* is the POS tag associated with it. We used the □ symbol to represent tokens that may be ignored. We sometimes associate a tag to this symbol to specify a word that must be present with this tag, but that will not be used in the query.

For the POS-based patterns, the order of the words matters and has to be as is in the original question for the pattern to be recognized. However, some tags can be ignored, most notably the *DET* tag which often does not change the meaning of the question. Such elements are denoted by using the [X] notation where X is the tag that can be ignored. For example, in the question : *"Who invented the plane?"* we have *WP VERB [DET] NN* as tags but the word **the** represented by *[DET]* can be dropped without altering the original question. In some cases, a pattern requires to have specific tokens to be present and are represented by using $\langle X \rangle$ notation where X is the token of set of tokens required for the pattern.

For each pattern, we also show how the SPARQL request is built, using the helper functions already described in a previous section. The use of the pattern is also illustrated using a question from one of the two corpora.

As an example, we can take the following question from the QALD-7 dataset: *Who developed Skype?* along with its representation given by the POS tagger: *[WP VERB NN]*. Based on the tags, we are able to apply the second pattern given in Table 6. In this case, we can map specific words of our question to the different variables of the pattern:

- X_{WP} = Who
- Y_{VB} = developed
- Z_{NN} = Skype

Using our SPARQL helper functions and replacing our question word *Who* with a SPARQL variable, we generate the following SPARQL triple :

dbr:Skype dbo:developer ?X .

Which gives us the following result when ran on DBpedia :

- http://dbpedia.org/resource/Microsoft
- http://dbpedia.org/resource/Skype_Technologies

Based on the gold standard provided for the question, we can affirm that the generated triple can indeed provide the correct answer to the question.

While the first example was quite simple and the input question matches exactly the used pattern, our patterns can be chained together in order to present more complex queries. For instance, starting from the previous question and adding more information : *Was the developer of Skype and Windows founded before 2010 ?*, we get a more complex question that is a combination of our 2nd and 7th pattern. We have our initial question while adding a conjunction that targets both *Skype* and *Windows* and additional information about a time limit (*before 2010*). As per the 7th pattern, we apply the same triple twice by replacing the first target of the conjunction by the second. Here, *Skype* is the subject of the triple, so we generate a second triple where the subject is *Windows* while the predicate and object remain *dbo:developer* and *?x* respectively.

The new SPARQL triples generated now are as follows :

dbr:Skype dbo:developer ?X .
dbr:Windows dbo:developer ?X .

While the time restriction is not handled by our patterns, they help simplify the question so it can be analyzed by the rest of the system. After our patterns, the original question can be written as *Was ?x founded before 2010?* which leads us to create the following SPARQL segments :

?X dbo:foundingYear ?Y .
FILTER (?Y < 2010)

#	Pattern	SPARQL	Example

1	$X_{NN}\ Y_{VB}\ [\square_{DET}]\ Z_{NN}$	getEntity(X)
		getProperty(Y)
		getEntity(Z)
			Did Barack marry Michelle ?

2	$X_{WP}\ Y_{VB}\ Z_{NN}$	getEntity(Z)
		getProperty(Y)
		?*answer*
			Who developed Skype ?

3	$X_{ADV}\ \square_{VB}\ Y_{NN}\ Z_{VB}$	getEntity(Y)
		getProperty(Z)
		?*answer*
			When was the Statue of Liberty built?

4	$X_{DET}\ Y_{NN}\ \square$	?answer *typeOf* getEntity(X)	Which presidents were born after 1945?

5	WP ⟨**TO BE**⟩ X_{NN} ⟨**OF**⟩ Y_{NN}	getEntity(Y)
		getProperty(X)
		?answer
			What is the official color of
			the University of Oxford?

6	⟨**TO BE**⟩ $Y_{NN}\ [\square_{DET}]\ Z_{NN}$	getEntity(Y)
		getProperty(Y, Z)
		getEntity(Z)
			Was Margaret Thatcher a chemist?

7	**[...]** $X_{CONJ\langle \textbf{BOTH} \mid \textbf{AND} \rangle}$ OR $X_{ADV\langle \textbf{BOTH} \mid \textbf{AND} \rangle}$ **[...]**	The RDF triple is repeated and the entity targeted by the conjunction or adverb X is replaced in each triple	What cars are fabricated in Canada AND the USA?

Table 6
POS TAG Patterns

Combining both segments, we get the final complete SPARQL query that can answer the question :

```
ASK WHERE {
    dbr:Skype dbo:developer ?X .
    dbr:Windows dbo:developer ?X .
    ?X dbo:foundingYear ?Y .
    FILTER (?Y < 2010)
}
```

The ability to use multiple patterns on the same query can help to more accurately understand the question and generate as much of the final request as possible. The SPARQL triples retrieved after applying the patterns can be applied in a system's pipeline as its done in LAMA or can be used to test if an automatically generated question contains valid semantic structures.

5. Experiments and Evaluation Results

We evaluate our lexico-syntactic patterns based on two different criteria : i) the presence of each pattern in the two datasets QALD and SQA, and ii) the relative impact of the patterns on the LAMA system's performance.

5.1. Pattern presence in datasets

The first evaluation aims to verify the presence of the different patterns in both the QALD and SQA datasets and thus their usefulness. The aim of this experiment is not to obtain a presence of 100% for every pattern since it would indicate that either i) the pattern is too generic and matches almost anything or ii) the dataset lacks variety and is not very representative of the real world. Patterns are also not mutually exclusive, as multiple patterns can be present in the same question. For example : *What french athletes won a gold medal?* has both the first and the second dependency-based patterns with {athletes,won,medal} matching the first one and {french athletes} matching the second one.

Both QALD and SQA are described in detail in sections 2.1.1 and 2.1.2 which show the particularities for both datasets.

Table 7 shows the distribution between the different dependency-based patterns in the datasets. A pattern is counted as long as it is detected in a question and patterns detected multiple times in the same question are only counted once.

5.1.1. QALD analysis

Looking at the results for the QALD dataset, we can see that as far as the dependency-based patterns are considered, pattern 1 is much more frequent than others while the last two occur less than 20% of the time.

This can be explained by looking at the composition of the QALD dataset: 78% is represented by simple questions that very often match the *subject, verb, object* pattern directly. Even complex questions can often contain the same pattern. For example, the question *Did Rowling write the first book of the Harry Potter series?* matches the first pattern with {Rowling,write, book}. The relatively low occurrence of the last two patterns can also be explained by the bias towards simple questions in QALD and the fact that those patterns generate two SPARQL triples and are thus exclusively targeted towards complex questions. However, it is interesting to note that 32% of the QALD questions are classified as complex and patterns **3** and **4** collectively cover 29.6% of questions, meaning that almost all complex questions in the QALD dataset are covered by those patterns.

Analysis for the POS tag-based patterns for QALD shows similar results with patterns **1** and **2** much more present than the rest. This is most likely due to the higher occurrence of simple questions. Since both *Did Gustave build the Eiffel Tower* (pattern **1**) and *Who built the Eiffel Tower* (pattern **2**) match the first dependency-based pattern, it can explain their lowered frequency, but also shows that using both patterns can offer some redundancy and increased accuracy. As for the patterns **4** and **5** we observe a much lower frequency, around 5% for both. As explained above, a low frequency does not correlate directly to a bad pattern as the examples given in Table 4 show questions that can occur naturally.

5.1.2. SQA analysis

Compared to QALD, dependency-based patterns frequency in SQA is more balanced, especially when it comes to patterns **3** and **4**. This is explained by the larger presence of complex questions in the dataset as well as more general variation between questions. This indicates that dependency-based patterns are more present and can be potentially more useful in a context where the questions are more complex and varied. Increased complexity in questions also means that there is an increased chance of questions containing more than one pattern, allowing for a combination of patterns to produce more complex SPARQL queries.

Frequeencies of POS tagging patterns for SQA are similar to the results obtained for QALD. There are however some differences for pattern **3** which are explained by the fact that questions that match the pattern *What/Which X [...]* are much more frequent. Similar to QALD, patterns **4** and **5** have a lower frequency but as explained in the previous section, should still be considered representative. Finally, the last pattern is much more present in SQA, mostly due to the higher presence of complex questions and the fact that this pattern targets specifically those types of questions.

Pattern	**QALD**	**SQA**
1	0.714	0.573
2	0.341	0.472
3	0.122	0.308
4	0.174	0.445

Table 7

Dependency-based pattern frequency

Pattern	**QALD**	**SQA**
1	0.443	0.568
2	0.331	0.447
3	0.247	0.365
4	0.376	0.342
5	0.065	0.095
6	0.054	0.106
7	0.154	0.378

Table 8

POS tag-based pattern frequency

5.2. Patterns Impact on the LAMA system

In order to verify that lexico-syntacic patterns are not only present but can be actually useful for answering natural language questions, they were introduced in the LAMA's development pipeline. The system was then tested with both datasets using dependency-based and POS-tag-based patterns separately, as well as a combination of both pattern sets. In all cases, the F-score was computed on the final answers returned by the question answering system and not only by considering the generated SPARQL query. In fact, only a single SPARQL query was provided in the gold standard and there can be multiple valid SPARQL queries for the same question. As per LAMA's original design, partial answers were not accepted, i.e., if the number of items in the answers returned by the system is a subset of the answers in the golden standard, the question is considered as wrongly answered.

Table 9 shows the F-score for the different experiments separated by dataset. Looking at the data for *QALD* we can see that using one of the two types of patterns to the pipeline leads to a small increase in performance but the combination of both approaches leads to an improvement of 10% or 8% globally.

Method	F-score
QALD	
No patterns	0.844
Dependency patterns	0.886
POS-tag patterns	0.872
Both patterns	0.905
SQA	
No patterns	0.535
Dependency patterns	0.783
POS-tag patterns	0.754
Both patterns	0.816

Table 9

Impact of LAMA on SQA and QALD

Results for the *SQA* dataset are however more interesting. We can observe a significant improvement in F-score when adding patterns to the answering process. This is most likely due to the increased proportion of complex questions in the dataset compared to QALD. Also, as seen in the previous section, around 40% of the questions in SQA match patterns based on the presence of conjunctions. As already established, the SQA dataset offers a more varied set of questions and using syntax and POS-tag patterns seems to significantly improve the performance of the system, especially when it comes to complex questions.

5.3. Multilingual analysis

As already mentioned, the aim of the presented patterns is to apply common patterns to different languages in order to extract semantic information from questions. Throughout this article, we have used both English and French as example languages and thus, we need to evaluate the patterns' performance in both languages. While the English evaluation is relatively straightforward and based on measuring the impact of patterns on LAMA's performance, evaluating French queries is a bit more complicated. The additional challenge is brought by both the available datasets and the knowledge base being used. Among both datasets, only QALD offers questions in more that one language but the answers and the SPARQL queries are only given based on the English version of DBpedia (e.g. property labels are in English). Since different versions of DBpedia do not contain the same data or the same properties between entities, we cannot guarantee that all questions from the dataset can be answered correctly or even at all using French DBpedia. Translating the entities in the provided answers can't be guaranteed to be correct for the same reasons.

For example, if we take the question *Which museum exhibits The Scream by Munch? / Dans quel musée est exposé Le Cri de Munch?* from the QALD dataset, we get *dbr:National_Gallery_(Norway)* for the English DBpedia (same as the answer provided in the dataset) and *dbr:Musée_Munch* for the French one. While the French answer is different than what is provided, it is correct since the entity *Le_Cri* (The Scream) is indeed related to *Musée_Munch* by the *dbo:museum* property.

In order to focus on evaluating the viability of pattern-generated SPARQL triples without the limitations of the knowledge base, the multilingual evaluation is done differently. For each pattern, we look at the SPARQL triples generated from questions in French. Each triple or set of triples is compared to the generated triple in English. The comparison is a binary classification (yes/no) that the generated triples are i) semantically equivalent to the question or a part of it and ii) similar to the ones generated in English. This qualitative evaluation is done by asking a person familiar with SPARQL and DBpedia, but not with the patterns being used, to determine if the two criteria have been respected.

For instance, the question *Qui est connu pour le projet Manhattan et le prix Nobel de la paix?/ Who is known for the Manhattan Project and Nobel peace prize* generated the following triples :

?x dbo:knownFor dbr:Projet_Manhattan .
?x dbo:knownFor dbr:Prix_Nobel_de_la_paix .

This satisfies both criteria since it the triples convey the same meaning as the original question and are similar to the triples generated in English : same property and same entities (similarity can be proved by the fact that they are linked using the *sameAs* property). In the case of the *POS tag patterns*, the 5th one is not used since it does not exist in French and the 6th one is replaced by the following pattern :

$$\langle \textbf{Est-ce que} \rangle\ X_{NN}\ \langle \textbf{ÊTRE} \rangle\ Y_{NN}$$

where *être* is the infinitive of the verb *to be*

Results from the evaluation are presented in Table 10 and 11. Results show that SPARQL triples generated in French are quite close to the expected results with POS-based patterns being a bit less accurate than dependency-based ones. This is mostly due to the fact that French tends to be more verbose than English and adding additional words can generate more POS tags and reduce the accuracy of pattern matching. Some frequent cases, such as ignoring the auxiliary verb *avoir (have)* used in French's past tense, are handled but the parsing does not cover some other uses cases.

Pattern	SPARQL Accuracy
1	0.933
2	0.905
3	0.916
4	0.925

Table 10

Dependency-based patterns in French

Pattern	SPARQL Accuracy
1	0.892
2	0.904
3	0.860
4	0.854
5	N/A
6	0.917
7	0.931

Table 11

POS tag patterns in French

5.4. Error analysis

While syntax and POS-tagging approaches have shown to be a promising tool in improving QA-systems' performance, they are not infallible and pose certain limitations, especially when it comes to generating the SPARQL triples associated to each pattern.

First of all, given that both approaches rely on an accurate parsing of the question, they are directly dependant on the accuracy of the parser. The parser can be affected by the quality of the model on which it was trained as well as the quality of the original question. While most syntax parsers in English are quite accurate [12, 18], other languages do not have such high quality tools. This can be sometimes corrected by translating the question in English but it can be a source of error if the translation is erroneous or modifies the semantics of the question. This can be however the only option for languages that do not have any syntax parsers available. The CoNLL Shared Task [18] evaluates the performance of the ParseySaurus (now just called Parsey) dependency parser with an average labeled attachemend score (LAS) of 77.93%. While languages such as English and French have a score of 84.45% and 83.1% respectively, some other languages such as Latvian are at 52.52%.

The quality of the question can also negatively affect the pattern detection. By quality, we consider the amount of grammatical and orthographic errors that can change how the parser or POS tagger interprets the words. Sometimes, missing only one letter can change how an entire sentence is interpreted.

For example, by only removing the letter **e** in our verb we get the question : *Who creatd the Eiffel Tower?*. Instead of matching the first lexico-syntactic pattern (*subj(v,s), obj(v,s)*), we now have *Tower* as the *subject* and *Who* as an *attribute* to the verb, a structure that does not match that pattern.

In this case, it is also important to note that using both types of patterns (dependency and POS) can help reduce the risk of incorrect pattern detection. In our previous example, the spelling mistake has caused the dependency parsing of the question to change, however the POS tagging has remained the same and this question is still matched with the second POS-tag pattern.

There is however no guarantee that one or more mistakes will not significantly alter the correct syntax and POS parsing and if a pattern is missed or wrongly recognized, it can lead to an incorrect SPARQL query generation and thus, to a wrong answer.

6. Future Work

The evaluation of our pattern-based approach shows that there is a net benefit in using lexico-syntactic patterns, based both on dependency and POS, in order to translate natural language questions into more formal and structured SPARQL query representations. The patterns presented in this paper, with a few exceptions, also aim at covering more than just the English language. Our LAMA system is now able to answer simple and complex questions in both English and French. Additional work can be done to enrich the set of existing patterns by either targeting more cross-language patterns that apply to as many languages as possible or by focusing on language specific patterns. Language specific patterns can be especially powerful when trying to analyze spoken questions rather than written ones. With the recent development in the field of *smart assistants* and voice recognition, questions are more often spoken than written. In fact, spoken questions often exhibit much less formal or sometimes even incorrect syntax and word structures. This can be seen in questions such as *Qui a gagné le mondial 2018? (Who won the 2018 world cup?)* that are very likely to be phrased as follows when spoken : *C'est qui qui a gagné le mondial 2018 ? (Who is it that won the 2019 world cup?)*, changing the sentence's dependency parse.The question is whether question answering systems should adapt to incorrect phrase structures. Deep learning networks might be better able to handle these cases.

7. Related Work

While Question Answering over Linked Data systems are not recent inventions, progress has been made in the field, especially in the last few years. Pattern-based approaches have been used in QA systems as well as other parts of the Web Semantics field. This section aims to explore some of the related work done in both QA systems and pattern-based approaches.

Generally, QA systems follow a similar approach to produce answers: the user's question is taken as input, parsed to extract the different relations between the entities and then a query (most often written in SPARQL) is generated and submitted to one or more KBs [3, 6, 19]. These systems try to answer questions by relying mainly on the identification of entities and their properties and then trying to form coherent SPARQL queries that can be ran against the target KBs. Systems such as WDAqua-core1 [3] and Xser [19] make use of string matching to generate different possible interpretations for the words in a given question, i.e. considering each word as a potential entity or property. Some other systems use parsers to annotate questions such as QAnswer [20] that uses the Stanford CoreNLP parser for POS tagging and HAWK [21] that makes use of clearNLP[22] for its POS tags. Using information from parsers can help improve the system's performance and reduce vulnerability to spelling mistakes and other shortcomings of string matching methods.

Most of the promising systems [1–3, 19, 21] rely on semantic structures in order to find answers to a given question. SemGraphQA [1] generates direct acyclic graphs representing possible interpretations of the query and only keeps the graphs that can be found in DBpedia's graph representation. In a similar fashion, WDAqua-core1 [3, 23] focuses on the semantics of the extracted entities from the question, and explores the RDF graphs of the entities to determine the appropriate relationships.

WDAqua-core1 is the closest system to LAMA in terms of objectives, as it is a multilingual system. It handles questions in five different languages: English,French,Spanish,Italian and German. It does not take into consideration the language of the original query and makes no use of NLP tools which allows it to be robust to ill-formed questions. While this system is indeed truly multilingual with only very few adjustments required to add a new language, the performance is quite limited with an F-score of 0.37 and 0.27 for French and English respectively on their QALD-7, Task 4 benchmark (REF).

While question answering systems generally rely on a semantic representation of the question and use POS tags for some of them [21, 22], LAMA also uses dependency parsing in addition to POS tags. Additionally, the semantic representation in LAMA is derived from an initial syntax tree representation that is modified after entity and property extraction. Compared to WDAqua-core1, LAMA obtains a much higher performance but is limited to English and French.

Several works in the Web Semantics field, not all related to question answering, have adopted a pattern-based approach for solving different issues. The BOA [24] system aims to extract structured data as RDF from unstructured data. BOA has a set of manually crafted patterns but also presents an algorithm for generating new patterns by training a supervised machine learning model over different corpora or knowledge bases. Patterns are generated for each property p by looking for a pair of entities or labels {s,o} such as that the triple {s,p,o} is found in the used knowledge base. Patterns are only saved as such if they are above a certain threshold for the number of occurrences in the training dataset. This method is again based on semantics only and does not take in consideration syntax or part of speech. This method works well for generating RDF data from text, but it has a limited use for QA systems such as LAMA since both s and o need to be existing entities in the KB and questions often generate triples that contain free variables, not bound to a particular entity.

SPARQL2NL [25] is a system that aims to verbalize SPARQL queries, i.e., convert them into natural language and it uses syntax dependencies in order to do so. Query verbalization is done based on on the predicate p of the {s,p,o} triple. Depending if p's realization is a verb, a noun or a variable, different dependency patterns are applied to the triple. For instance, if p is a verb, an equivalent of our 1^{st} dependency pattern is applied to the triple. While SPARQL2NL does the inverse of what LAMA aims to do, its dependency rule have helped extract some of LAMA's own dependency patterns. However, since LAMA works with natural language queries, it can leverage the use of POS patterns for SPARQL triple generation.

8. Conclusion

In this paper we present lexico-syntactic patterns aimed at improving question answering systems. The patterns are separated in two different categories : dependency-based patterns and POS (part of speech) tag patterns. We also present LAMA, a multilingual QA system that leverages the use of the shown patterns to improve its performance. Our evaluation on the SQA and QALD datasets shows that the use of patterns does indeed increase the performance of the system, especially in the case of complex queries. In addition, we have evaluated the use of patterns in languages other than English, more precisely in French.

There are potential improvements that could be made to the system, most notably the enrichment of dependency and POS patterns. Through our error analysis, we have identified that spelling mistakes and grammatical errors can negatively impact the system's performance. In future work, we should aim to reduce the impact of such factors on the system and enrich the set of available patterns. We will also explore the semantics of the returned URIs and see whether they are compliant to the expected output. Finally, future development should also take in account the increasing use of "smart assistants" that consider spoken questions and not just written queries.

References

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

[2] D. Sorokin and I. Gurevych, End-to-End Representation Learning for Question Answering with Weak Supervision, in: *Semantic Web Challenges*, M. Dragoni, M. Solanki and E. Blomqvist, eds, Springer International Publishing, Cham, 2017, pp. 70–83. ISBN 978-3-319-69146-6.

[3] D. Diefenbach, K. Singh and P. Maret, WDAqua-core1: A Question Answering Service for RDF Knowledge Bases, in: *Companion Proceedings of the The Web Conference 2018*, WWW '18, International World Wide Web Conferences Steering Committee, Geneva, Switzerland, 2018, pp. 1087–1091. ISBN 978-1-4503-5640-4. doi:10.1145/3184558.3191541.

[4] A. Both, D. Diefenbach, K. Singh, S. Shekarpour, D. Cherix and C. Lange, Qanary–a methodology for vocabulary-driven open question answering systems, in: *International Semantic Web Conference*, Springer, 2016, pp. 625–641.

[5] P. Achananuparp, X. Hu, X. Zhou and X. Zhang, Utilizing Sentence Similarity and Question Type Similarity to Response to Similar Questions in Knowledge-Sharing Community.

[6] N. Radoev, A. Zouaq, M. Tremblay and M. Gagnon, A Language Adaptive Method for Question Answering on French and English, in: *Semantic Web Challenges*, D. Buscaldi, A. Gangemi and D. Reforgiato Recupero, eds, Springer International Publishing, Cham, 2018, pp. 98–113. ISBN 978-3-030-00072-1.

[7] QALD2017 Challenge – ESWC 2017 – HOBBIT, (Accessed on 03/29/2018).

[8] M. Dubey, LC-QuAD QALD format, figshare, 2018. doi:10.6084/m9.figshare.5818452.v6. https://figshare.com/articles/LC-QuAD_QALDformat/5818452/6.

[9] N. Radoev, M. Tremblay, M. Gagnon and A. Zouaq, AMAL: Answering French Natural Language Questions Using DBpedia, in: *Semantic Web Challenges*, M. Dragoni, M. Solanki and E. Blomqvist, eds, Springer International Publishing, Cham, 2017, pp. 90–105. ISBN 978-3-319-69146-6.

[10] C. Schöch, A word2vec model file built from the French Wikipedia XML Dump using gensim., Zenodo, 2016. doi:10.5281/zenodo.162792.

[11] M. Fares, A. Kutuzov, S. Oepen and E. Velldal, Word vectors, reuse, and replicability: Towards a community repository of large-text resources, in: *Proceedings of the 21st Nordic Conference on Computational Linguistics, NoDaLiDa, 22-24 May 2017, Gothenburg, Sweden*, Linköping University Electronic Press, Linköpings universitet, 2017, pp. 271–276. ISSN 1650-3740.

[12] .

[13] D. Andor, C. Alberti, D. Weiss, A. Severyn, A. Presta, K. Ganchev, S. Petrov and M. Collins, Globally Normalized Transition-Based Neural Networks, *CoRR* **abs/1603.06042** (2016). http://arxiv.org/abs/1603.06042.

[14] J. Daiber, M. Jakob, C. Hokamp and P.N. Mendes, Improving Efficiency and Accuracy in Multilingual Entity Extraction, in: *Proceedings of the 9th International Conference on Semantic Systems (I-Semantics)*, 2013.

[15] .

[16] J. Nivre, Algorithms for Deterministic Incremental Dependency Parsing, *Computational Linguistics* **34**(4) (2008), 513–553. doi:10.1162/coli.07-056-R1-07-027.

[17] Petrov, Das, Dipanjan, Ryan and McDonald, A Universal Part-of-Speech Tagset, American Physical Society, 2011. https://arxiv.org/abs/1104.2086v1.

[18] Alberti, Chris, Daniel, Ivan, Collins, Michael, Gillick, Dan, Kong, Koo and et al., SyntaxNet Models for the CoNLL 2017 Shared Task, 2017. https://arxiv.org/abs/1703.04929.

[19] K. Xu, S. Zhang, Y. Feng and D. Zhao, Answering Natural Language Questions via Phrasal Semantic Parsing, in: *Natural Language Processing and Chinese Computing*, C. Zong and J.-Y. Nie, eds, Springer Berlin Heidelberg, Berlin, Heidelberg, 2014, pp. 333–344. ISBN 978-3-662-45924-9.

[20] S. Ruseti, A. Mirea, T. Rebedea and S. Trausan-Matu, QAnswer - Enhanced Entity Matching for Question Answering over Linked Data, in: *CLEF*, 2015.

[21] R. Usbeck, A.-C.N. Ngomo, L. Bühmann and C. Unger, HAWK – Hybrid Question Answering Using Linked Data, in: *The Semantic Web. Latest Advances and New Domains*, F. Gandon, M. Sabou, H. Sack, C. d'Amato, P. Cudré-Mauroux and A. Zimmermann, eds, Springer International Publishing, Cham, 2015, pp. 353–368. ISBN 978-3-319-18818-8.

[22] J.D. Choi and M. Palmer, Getting the Most out of Transition-based Dependency Parsing, in: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: Short Papers - Volume 2*, HLT '11, Association for Computational Linguistics, Stroudsburg, PA, USA, 2011, pp. 687–692. ISBN 978-1-932432-88-6. http://dl.acm.org/citation.cfm?id=2002736.2002869.

[23] D. Diefenbach, A. Both, K. Singh and P. Maret, Towards a question answering system over the Semantic Web, *Semantic Web* (2019), 1–19–. doi:10.3233/sw-190343.

[24] D. Gerber and A.-C.N. Ngomo, Extracting Multilingual Natural-Language Patterns for RDF Predicates, in: *Knowledge Engineering and Knowledge Management*, A. ten Teije, J. Völker, S. Handschuh, H. Stuckenschmidt, M. d'Acquin, A. Nikolov, N. Aussenac-Gilles and N. Hernandez, eds, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 87–96. ISBN 978-3-642-33876-2.

[25] A.-C. Ngonga Ngomo, L. Bühmann, C. Unger, J. Lehmann and D. Gerber, Sorry, I Don'T Speak SPARQL: Translating SPARQL Queries into Natural Language, in: *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, ACM, New York, NY, USA, 2013, pp. 977–988. ISBN 978-1-4503-2035-1. doi:10.1145/2488388.2488473.

[26] G. Mazzeio, Answering Controlled Natural Language Questions on RDF Knowledge Bases, 2016.

[27] P. Gupta, A survey of text question answering techniques, 2012.

[28] N.I. of Health et al., Daily Med, 2014. https://www.healthdata.gov/dataset/dailymed.

[29] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak and Z. Ives, *DBpedia: A Nucleus for a Web of Open Data*, in: *The Semantic Web: 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007. Proceedings*, K. Aberer, K.-S. Choi, N. Noy, D. Allemang, K.-I. Lee, L. Nixon, J. Golbeck, P. Mika, D. Maynard, R. Mizoguchi, G. Schreiber and P. Cudré-Mauroux, eds, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 722–735. ISBN 978-3-540-76298-0. http://dx.doi.org/10.1007/978-3-540-76298-0_52.

[30] V. Lopez, C. Unger, P. Cimiano and E. Motta, Evaluating question answering over linked data, *Web Semantics Science Services And Agents On The World Wide Web* **21** (2013), 3–13. doi:10.1016/j.websem.2013.05.006.

[31] .

[32] D. Diefenbach, A. Both, K.D. Singh and P. Maret, Towards a Question Answering System over the Semantic Web, *CoRR* **abs/1803.00832** (2018). http://arxiv.org/abs/1803.00832.

[33] T. Mikolov, K. Chen, G. Corrado and J. Dean, Efficient Estimation of Word Representations in Vector Space, *CoRR* **abs/1301.3781** (2013). http://arxiv.org/abs/1301.3781.

[34] 14th ESWC 2017 |, (Accessed on 03/29/2018).

[35] D. Vrandečić and M. Krötzsch, Wikidata: A Free Collaborative Knowledgebase, *Commun. ACM* **57**(10) (2014), 78–85. doi:10.1145/2629489.

French Patterns

#	Pattern	SPARQL	Example

1	X_{NN} Y_{VB} [$\square_{DET}$] Z_{NN}	getEntity(X)
		getProperty(Y)
		getEntity(Z)
			Est-ce que Barack
			a marié Michelle ?

2	X_{WP} Y_{VB} Z_{NN}	getEntity(Z)
		getProperty(Y)
		?*answer*
			Qui a développé Skype ?

3	X_{ADV} Y_{NN} $\square_{VB}$ Z_{VB}	getEntity(Y)
		getProperty(Z)
		?*answer*
			Quand la Statue de Liberté
			a été construite ?

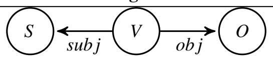
4	X_{DET} Y_{NN} $\square$	?answer *typeOf* getEntity(X)	Quels presidents sont nés après 1945?

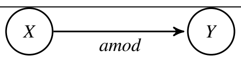
5	⟨**EST-CE QUE**⟩ Y_{NN} ⟨**ÊTRE**⟩ Z_{NN}	getEntity(Y)
		getProperty(Y,)
		getEntity(Z)
			Est-ce que Margaret Thatcher
			a été chemiste ?

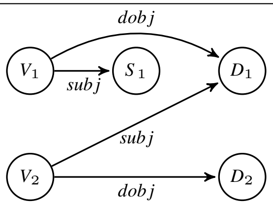
6	**[...]** X_{CONJ}⟨ **AINSI QUE | ET** ⟩ OR X_{ADV}⟨**AINSI QUE | ET**⟩ **[...]**	The RDF triple is repeated and the entity targeted by the conjunction or adverb X is replaced in each triple	Quelles voitures sont fabriquées au Canada ET les États-Unis ?

Table 12
POS TAG Patterns

#	Diagram	Dependency pattern	SPARQL	Example

1		subj(v, s)
		obj(v, o)
			getEntity(s)
			getProperty(v)
			getEntity(o).
				Est-ce que Barack
				a marié Michelle ?

2		amod(x/$_{ADJ}$,y/$_{NN}$)	getEntity(y)
			getProperty()
			getEntity(x).
				athlète canadien

3		subj(V_1,S_1) dobj(V_1,D_1)
		subj(V_2,D_1) dobj(V_2,D_2)
			getEntity(s_1)
			getProperty(v_1)
			?answer.
			
			?answer
			getProperty(v_2)
			getEntity(d_2)
				Donne-moi les personnes
				qui jouent un sport
				qui est dans les Olympiques.

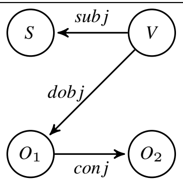
4		subj(S,V)
		dobj(V, O_1),
		conj(O_1,O_2)
			getEntity(s)
			getProperty(v)
			getEntity(o_1).
			
			getEntity(s)
			getProperty(v)
			getEntity(o_2)
				Est-ce que Tolkien a écrit
				le Hobbit et
				le Silmarillion?

Table 13
Lexico-Syntactic Patterns