French and English Question Answering using Lexico-Syntactic Patterns

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

Abstract. The evolution of the Semantic Web has given rise to multiple knowledge bases (KBs) with access to a large amount of structured data. However, querying these knowledge bases through SPARQL remains a challenge for the average user. Question answering systems based on natural language can help alleviate this challenge. In this paper, we present a knowledge base question answering system that answers questions in English and French. Our method is based on transforming natural language questions into SPARQL queries by leveraging the syntactic information of questions. We describe a set of lexico-syntactic patterns used to automatically generate triple patterns and SPARQL queries. Our results over DBpedia show that the use of the lexico-syntactic patterns can improve the results of the system.

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 learning curve. Several systems [1, 2] 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. SPARQL queries are then built by trying different combinations of all the words in the question and returning only those combinations that obtain some answer. While such systems are effective, they are reliable only for simple queries and do not deal well with ambiguities. More so, if additional restrictions are applied to reduce ambiguity, the system 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 [3, 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 is based on the question-answering system LAMA [6], which is 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 QA datasets [7, 8].

The remainder of the paper is structured as follows. In the next section, we analyze the two datasets and describe the LAMA system. We then detail the two different types of patterns implemented in the system. This is followed by an evaluation on question answering datasets to determine 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 the datasets and the LAMA system, which is used as a proof of concept for the interest of 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. 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.

The work presented here used the DBpedia knowledge base as the reference KB throughout this paper. For readability purposes, we have abbreviated some of the URIs by using the prefixes in Table 1. The *dbo:* prefix represents classes, concepts and some properties in DBpedia's ontology while *dbp:* and *dbr:* represent properties and resources respectively.

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

Table 1
DBpedia prefixes

2.1. Dataset and Question Analysis

In this work, we used two different corpora containing questions designed to be answered using 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 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	LC-Quad
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. Since our aim is to compare ourselves to a baseline, the merge allows us to have a bigger and more varied dataset for comparison.

QALD is overwhelmingly composed of simple questions (298 out of the 384 questions), representing 78% overall. We partitioned the questions according to the type (Date, Number, Boolean or Resource) of the expected answer. The general distribution per type can be seen in Table 2. The last category (Resource) designates questions for which the expected answer is one or more URIs and for which no other more appropriate type 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. that is, URIs that refer to English Wikipedia such as *dbr:The_Hobbit* representing The Hobbit (the book). This distinction is important since URIs about the same concept in English and French DBpedia can, sometimes, refer to different triples describing the concept. For example, if we want to find who is the publisher of *The Hobbit*, we can use the *dbo:publisher* property to get *dbr:George_Allen_&_Unwin* in English. However, using the same property in the French version of DBpedia, we get *fr.dbr:Éditions_Stock* (the original publisher of the french version of the book)

and in order to get the same information as in English, we need to use the *dbo:firstPublisher* property.

2.1.2. LC-Quad Dataset

The LC-Quad(SQA) dataset is similar to QALD in its structure but does only contain questions in English. It also provides both the answers to the questions and the SPARQL queries used to obtain those answers on the English DBpedia knowledge base.

Despite having a larger number of questions than QALD, the LC-Quad 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, LC-Quad 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. The questions in the dataset are generated from patterns, which can explain the distribution.

We must note that the LC-Quad 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 at simple questions, the first version of LAMA [9] was still able to answer a very limited amount of complex questions.

Figure 1 shows the system's processing pipeline. The system is modular and based on components that can be modified or replaced with custom ones if a different behaviour is desired.The pre-processing phases (Syntax Parsing and Question Classification) generate additional intermediate structures (dependency tree, POS tags, question type) that are passed to the core processing module, which transforms the syntax tree in an intermediate representation. This representation is parsed to generate one or more triple patterns used in the SPARQL requests to the Knowledge Base.

The system also interacts with several external resources: DBpedia, Wikipedia and the Google Translate API. In the case of Wikipedia, we used it only to train a Word2Vec model for the *Property Detection* phase in both English and French [10]. Other Word2Vec models [11] could be exploited, but given the wealth of data and the proximity between DBpedia and Wikipedia, Wikipedia remains a good source for model training in multiple languages. The Google translate API is an optional module that can be used for questions when working in languages other than English. This module leverages the larger amount of data in English KBs to better answer the question. This is explained in more detail in section 2.2.2.

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 (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 multiple languages.

CNL's parsing is done on pre-trained models, with English being based on the Penn Treebank and OntoNotes corpora [12]. 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.

2.2.1. Pre-processing and Question Type Classification.

The Pre-processing step involves parsing the input question 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 section 4. These representations are 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,

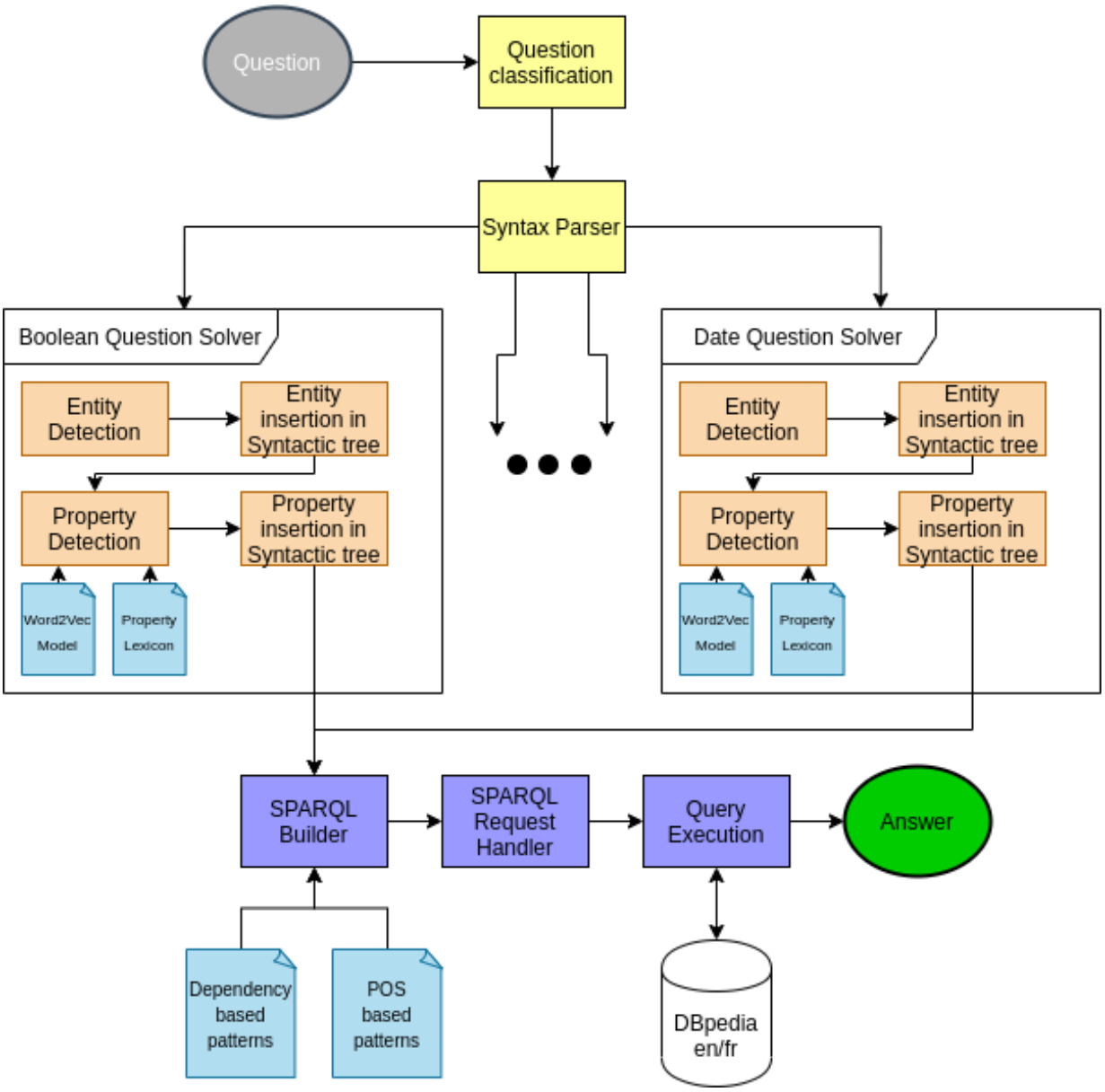


Figure 1. LAMA's Pipeline

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

Aggregation is applied to questions that require counting or ordering (descending or ascending).

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 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 a *Boolean*, given that those types of questions are answered with an ASK query that only returns true or false.

Based on the classification result, the question is then handled by a specific solver that inherits from a base *Question Solver* module. All solvers perform the same steps as seen in Figure 1 and detailed in sections 2.2.2 through 2.2.4. Individual solvers can be modified to allow for custom rules for property and entity extraction. 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*. If no specific rules are given, the described default behaviour is used.

2.2.2. Entity Extraction.

After the question is parsed and classified, the system tries to extract a semantic representation from the question, 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 for in the target knowledge base (DBpedia in this case) after appending the *dbr:* prefix and replacing the space character by the underscore character. A heuristic where definite articles, such as *the*, are always appended to the following word is also used. All valid entities (i.e. the ones that exist in the knowledge base) are kept as possible candidates. Entities are then sorted in a descending order using the computation formula shown below. 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*): *the queen of England, the queen, England, the queen of, of England*. Out of those, only the first 3 are kept as valid entities since *the 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 nominal phrase NP (the 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 capitalization and pluralization but penalize entities discovered after these transformations. For example, the word *queen* can also lead to *Queen, Queens* and *queens* but all those transformations have a lower score than the original word (scores of 4 and 3, respectively, instead of 5 for the original word, as we will see in more detail below) since pluralization and/or capitalization was required to obtain them. If no entities are found using all these methods, the semantic annotator DBpedia Spotlight [13] is used as a back-up tool.

In the case of languages other than English, an optional translation step can take place where the initial question is translated to English. In fact, the French DBpedia chapter is less complete than its English counterpart and all the question answering competitions (QALD, LC-Quad) expect an 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, even if semantically correct, 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:

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

where:

$length(e)$ is the number of letters in e

$nsp(e)$ 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, 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 words into entities. For instance *the creator* can be transformed into *the Creator*, an existing entity in DBpedia but this is 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 multiple words.

Both dependency (Table 4) and POS (Table 6) patterns rely on identifying particular words as subjects or objects in the question, and these words 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 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, sorted in descending order based on their score. 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 triple pattern. The application *getEntity(Tolkien)* will return *dbr:J._R._R._Tolkien*, while *getEntity(Hobbit)* will return both *dbr:The_Hobbit* (the book) and *dbr:Hobbit* (the fictional race). Based on the score computed as explained before, *dbr:The_Hobbit* is correctly chosen as the most likely candidate for the triple pattern.

2.2.3. Property Extraction.

After extracting the entities, the system detects and extracts possible properties from the question. It uses a lexicon that maps various expressions to the same property.

2.2.4. Lexicon Generation.

In order to help with the property extraction, we automatically extract a property lexicon based on our chosen knowledge base DBpedia. The lexicon is built once and used by the system in an *offline* phase. 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 executed both on the *dbo:* and *dbp:* prefixes. Extraction was also run on the *dbp:* domain of the French version of DBpedia since language-specific properties are defined in this domain instead of *dbo:*. For each label, we mapped all corresponding URIs. When several URIs exist in dbo and dbp, we favor the URI of the *dbo:* domain first. For instance, the label "parent" has 2 URIs in both namespaces and thus lead to the following mapping :

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

The generated lexicon is stored as a hash table, allowing for a fast lookup (*O(1)*) and reducing the cost of the lookups required for the query analysis. It allows us to find existing properties based on their label and its presence in the question. We use a combination of string matching and Levenshtein distance, as described in subsection 2.2.2. When working with languages other than English, translations can be used for properties without 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 property is expressed using literals that are quite different from the label in the knowledge base. LAMA uses a different approach for such cases, relying on word embeddings, as explained in the following sections.

Unlike entities, in most cases, the expression of a 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 Levenshtein distance of less than 3 (determined empirically), applied only to the end of the initial word label. Levenshtein distance is used instead of stemming and lemmatization since it is a simpler substitution for both and can help with spelling mistakes at the end of words (common in languages such as French). While this is helpful, it does not cover cases where the desired

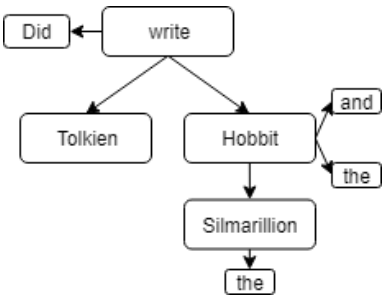
property is significantly different from the word label. To reduce the number of false positives, a property is considered valid only if it exists in the target base **and** is used in relation with at least one of the extracted entities.

To alleviate the problem of a label-property difference, 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). 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.

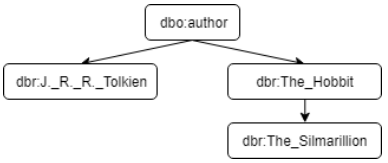
Like the entity extractor, the property extractor module offers a helper function called **getProperty(x)** where *x* is the word or group of words denoting the potential property. In some questions, no expression can be targeted as a potential property since the relation between two entities is implicit. 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 both *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.



(a) Syntactic tree



(b) Semantic representation after entity and property extraction

Figure 2. Tree representation for the question: Did Tolkien write the hobbit and the Silmarillion?

For example, as seen in Figure 2 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* order.

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 queries and triple patterns

A basic SPARQL query, according to the standard definition, 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 lexico-syntactic patterns P, we aim to generate the set S containing one or more triple patterns that represent the semantics of a given question. A single pattern $p \in P$ generates at least one triple pattern. 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 triple patterns set where the object of the first triple is also the subject of the second one:

```
?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 a large number of triples. These queries can also be split in individual triple queries that can be run in a sequence, allowing for all the intermediate results to be available for storage and additional use.

Each of the three elements in a triple pattern 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 by *?x*.

A single triple pattern can be defined as $(s,p,o) \in (X \cup R) \times (X \cup P) \times (X \cup R \cup L)$ where R are all resources, P all properties and L all literals in the knowledge base being targeted by the SPARQL query and X is the set of all variables usable in the query.

2.3. SPARQL Query Generation

The last step of the process pipeline is the generation of the SPARQL query. Each query can either take the form of ASK queries (for *Boolean* questions) or SELECT queries for all other types 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 Builder* 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 Handler* 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 triple, 2 different triple patterns 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).

In the following sections, we detail the patterns used in LAMA.

3. Dependency-based Patterns

As already stated, our work is based on using SyntaxNet as a dependency parser. SyntaxNet is a transition-based dependency parser [14] [15], 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 3 shows a graphic representation of a question and its dependency parse tree.

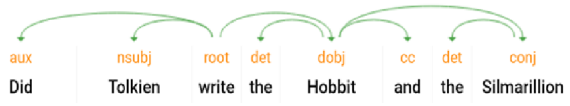


Figure 3. Dependency parse of a simple query

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*

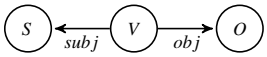
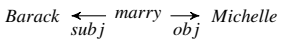
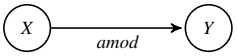
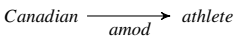
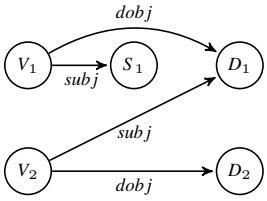
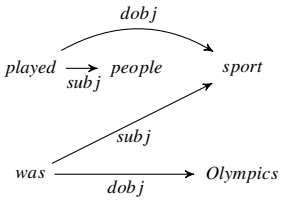
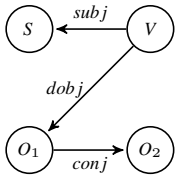
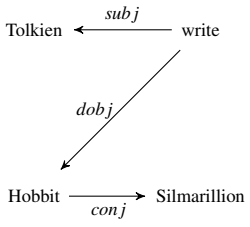
#	Diagram	Dependency pattern	SPARQL	Example	Syntax Tree
1		subj(v, s) obj(v, o)	getEntity(s) getProperty(v) getEntity(o).	Did Barack marry Michelle ?	
2		amod(x/$_{ADJ}$,y/$_{NN}$)	getEntity(y) getProperty() getEntity(x).	Canadian athlete	
3		subj(V_1,S_1) dobj(V_1,D_1) subj(V_2,D_1) dobj(V_2,D_2)	getEntity(s_1) getProperty(v_1) ?x. ?x getProperty(v_2) getEntity(d_2)	Give me people who played a sport that is in the Olympics.	
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)	Did Tolkien write the Hobbit and the Silmarillion?	

Table 4

Lexico-Syntactic Patterns

and thus removed in order to simplify the analysis of a question. One frequent such a case is the dependency between a determinant and its head noun.

As previously mentioned, the first set of patterns are based on the dependency graph generated by the parser. Those patterns are shown in Table 4. 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 triple patterns representing the semantics of the pattern.

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 3 from which 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)

And using our dependency patterns, we can generate the following triple patterns:

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 is also instantiated, 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.

4. POS-based Patterns

In addition to dependency patterns, the system also uses POS-based patterns.

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

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.

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 the notation [16]. A full list can be seen 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 types of nouns. The tagger offers a coarse-grained level of POS tagging represented by the tags in Table 5.

Our POS patterns are presented in detail in Table 6. For each pattern, we show 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. Here, we show a few POS-based patterns that can be mapped to one or more triple patterns to generate a SPARQL query representing the semantics of the original question.

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 the POS-based patterns, the order of the words matters and has to appear 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 obtain the tags *WP VERB [DET] NN* but the word **the** represented by *[DET]* can be dropped without altering the original question. In some cases, a pattern requires some specific tokens which are represented by the $\langle X \rangle$ notation where *X* is the token or set of tokens that is required.

For each pattern in Table 6, we also show how the SPARQL request is built, using the 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 dataset: *Who developed Skype?* along with its representation given by the POS tagger: *[Who$_{WP}$ developed$_{VB}$ Skype$_{NN}$]*. Based on the tags, we are able to instantiate the second pattern shown 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 triple pattern :

dbr:Skype dbo:developer ?answer .

Which gives us the following result :

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

#	Pattern	SPARQL	Example	POS Tags
1	$X_{NN}\ Y_{VB}\ [\square_{DET}]\ Z_{NN}$	getEntity(X) getProperty(Y) getEntity(Z)	Did Barack marry Michelle ?	$Barack_{NN}\ marry_{VB}\ Michelle_{NN}$
2	$X_{WP}\ Y_{VB}\ Z_{NN}$	getEntity(Z) getProperty(Y) ?*answer*	Who developed Skype ?	$Who_{WP}\ developed_{VB}\ Skype_{NN}$
3	$X_{ADV}\ \square_{VB}\ Y_{NN}\ Z_{VB}$	getEntity(Y) getProperty(Z) ?*answer*	When was the Statue of Liberty built?	$When_{ADV}\ was_{VB}$ $Statue\ of\ Liberty_{NN}\ built_{VB}$
4	$X_{DET}\ Y_{NN}\ \square$	?answer *typeOf* getEntity(Y)	Which presidents were born after 1945?	$Which_{DET}\ presidents_{NN}$ [...]
5	WP ⟨**TO BE**⟩ X_{NN} ⟨**OF**⟩ Y_{NN}	getEntity(Y) getProperty(X) ?answer	What is the official color of the University of Oxford?	$What_{WP}$ ⟨**is**⟩ [...] $color_{NN}$ ⟨**of**⟩ $University\ of\ Oxford_{NN}$
6	⟨**TO BE**⟩ Y_{NN} $[\square_{DET}]\ Z_{NN}$	getEntity(Y) getProperty(Y, Z) getEntity(Z)	Was Margaret Thatcher a chemist?	⟨**Was**⟩ Margaret Thatcher $_{NN}$ $chemist_{NN}$
7	[...] X_{CONJ}⟨ **BOTH | AND** ⟩ OR X_{ADV}⟨**BOTH | AND**⟩ [...]	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?	What cars are fabricated in [Canada] AND_{CONJ} [USA]

Table 6
POS TAG Patterns

Based on the gold standard provided for the question, we can check 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 5^{th} and 7^{th} pattern.

To see how this works, let us first see the result of POS tagging:

$$Was_{VB}\ the_{DET}\ developer_{NN}\ of_{ADP}\ Skype_{NO}\ and_{CONJ}\ Windows_{NN}\ founded_{VB}\ before_{ADP}\ 2010_{NUM}$$

We can see that the segment

$$Was_{VB}\ the_{DET}\ developer_{NN}\ of_{ADP}\ Skype_{NO}$$

matches the 5^{nd} pattern, resulting in the following triple pattern:

dbr:Skype dbo:developer ?answer .

Now considering the whole segment $Was_{VB}\ the_{DET}\ developer_{NN}\ of_{ADP}\ Skype_{NO}\ and_{CONJ}\ Windows_{NN}$, we see that it matches the 7^{th} pattern, whose effect is to repeat the triple pattern, replacing the target entity by the one that is represented by the second part of the coordination, thus resulting in this final form:

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

After processing the question with these patterns, what remains is $Was_{VB}\ founded_{VB}\ before_{ADP}\ 2010_{NUM}$, and we know that its subject corresponds to the variable *?answer*. This time restriction is handled by the base version of LAMA which matches the keywords *before* and *2010* to a simple *less than(<)* FILTER constraint. This leads us to create the following SPARQL segment:

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

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

```
ASK WHERE {
    dbr:Skype dbo:developer ?answer .
    dbr:Windows dbo:developer ?answer .
    ?answer 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 triple patterns 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 query 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 LC-Quad, 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 LC-Quad datasets and thus their usefulness. The aim of this experiment is not to obtain a presence of 100% for each 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 LC-Quad 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 triple patterns 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.

The analysis of the POS-based patterns for QALD shows similar results with patterns **1**, **2** and **4** much more present than the rest. This is most likely due to the higher occurrence of simple questions.

Interestingly, both *Did Gustave build the Eiffel Tower* (pattern **1**) and *Who built the Eiffel Tower* (pattern **2**) also match the first dependency-based pattern,which shows that using both patterns can offer some redundancy and increased accuracy. As for the patterns **5** and **6** we observe a much lower frequency, around 5% for both. As explained above, a low frequency does not correlate necessarily with a bad pattern as the examples given in Table 4 show questions that can occur naturally.

5.1.2. LC-Quad analysis

Compared to QALD, dependency-based patterns frequency in LC-Quad 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.

Frequencies of POS tagging patterns for LC-Quad are roughly 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. Similarly, pattern **7** which matches complex questions using conjunction is more frequent in LC-Quad than QALD given QALD's composition. Similar to QALD, patterns **5** and **6** 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 LC-Quad, mostly due to the higher presence of complex questions and the fact that this pattern targets specifically those types of questions.

Pattern	QALD	LC-Quad
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	LC-Quad
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 integrated in our question answering pipeline LAMA [6]. 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 Macro 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, we have opted for a simpler but more conservative evaluation where 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 incorrectly 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 (about 4% for QALD and 22-25% for LC-Quad) but the combination of both approaches leads to an additional improvement of 3% for QALD and 6% for LC-Quad.

Results for the *LC-Quad* dataset are the most 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 LC-Quad match patterns based on the presence of conjunctions. As already established, the LC-Quad 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.

Method	F-score
QALD	
No patterns (base system)	0.844
Dependency patterns	0.886
POS-tag patterns	0.872
Both patterns	0.905
LC-Quad	
No patterns (base system)	0.535
Dependency patterns	0.783
POS-tag patterns	0.754
Both patterns	0.816

Table 9
Impact of LAMA on LC-Quad and QALD

5.3. French questions 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 cannot 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)* from the English DBpedia (same as the answer provided in the dataset) and *dbr:Musée_Munch* for the French DBpedia. 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. However, if we use the French DBpedia, the answer returned will be considered as incorrect due to these differences.

In order to focus on evaluating the interest of triple patterns without the aforementioned limitations of the knowledge base, the evaluation is done differently for French. For each pattern, we look at the SPARQL 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 the 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 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 proven by the fact that they are linked using the *sameAs* property). The accuracy of the SPARQL patterns is calculated by applying the two criteria to the generated triple patterns : (i) semantic equivalence to English (yes/no) and (ii) for those considered equivalent, we count those with correct elements (entities and properties).

Results from the evaluation are presented in Table 10 and 11. We can notice that some of the *POS tag patterns* are not used, such as the 5^{th} since it does not exist in French and the 6^{th} which is replaced by the following pattern :

⟨**Est-ce que**⟩ X_{NN} ⟨**ÊTRE**⟩ Y_{NN}
where *être* is the infinitive of the verb *to be*

These results show that the triple patterns generated in French are quite close to the expected results. POS-based patterns are a bit less accurate than dependency-based patterns. 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.

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 syntactic 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 triple patterns 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 syntactic parsers in English are quite accurate [14], other languages do not have such high quality tools. This can be sometimes corrected by translating the question in English but such a method can be a source of errors 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 syntactic parsers available. The CoNLL Shared Task [14] evaluates the performance of the ParseySaurus (now just called Parsey) dependency parser with an average labeled attachment 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 pattern detection. By quality, we mean 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 any 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 systems are not recent inventions [17], progress has been made in the field of Question Answering over Linked Data, especially in the last few years [18]. Pattern-based approaches have been used in QA systems as well as other parts of the Semantic Web field. This section aims to explore some of the related work done in both QA systems and pattern-based approaches and does not cover question answering systems based mostly on deep learning techniques, which adopt a very different approach to the problem.

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 [1, 2, 6]. 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 run against the target KBs. Systems such as WDAqua-core1 [2] and Xser [1] 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 [19] that uses the Stanford CoreNLP parser for POS tagging and HAWK [20] that makes use of clearNLP[21] 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 [1–4, 20] rely on semantic structures in order to find answers to a given question. SemGraphQA [3] 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 [2, 22] 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. This system is indeed truly multilingual with only very few adjustments required to add a new language, but 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 1 benchmark [7].

While question answering systems generally rely on a semantic representation of the question and use POS tags for some of them [20, 21], 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 Semantic Web field, not all related to question answering, have adopted a pattern-based approach for solving different issues. The BOA [23] 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. It 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 [24] 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 inspired some of LAMA's own dependency patterns. However, since LAMA works with natural language queries, it can also leverage the use of POS patterns for the 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) patterns. We also present LAMA, a QA system that leverages the use of these patterns to improve its performance. Our evaluation on the LC-Quad 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 could also explore the possibility of checking for logical coherence between the system's output and the expected answers. For example, if we are looking for someone's date of birth, we expect the answer to be in the form of a date. Finally, future development should also take into account the increasing use of "smart assistants" that consider spoken questions and not just written queries.

References

[1] K. Xu, S. Zhang, Y. Feng and D. Zhao, C. Zong and J.-Y. Nie, eds, Springer Berlin Heidelberg, Berlin, Heidelberg, 2014, pp. 333–344. ISBN 978-3-662-45924-9.

[2] D. Diefenbach, P.H. Migliatti, O. Qawasmeh, V. Lully, K. Singh and P. Maret, QAnswer: A Question Answering Prototype Bridging the Gap Between a Considerable Part of the LOD Cloud and End-users, in: *The World Wide Web Conference*, WWW '19, ACM, New York, NY, USA, 2019, pp. 3507–3510. ISBN 978-1-4503-6674-8. doi:10.1145/3308558.3314124.

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

[4] 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.

[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] R. Usbeck, A.-C. Ngonga 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. https://svn.aksw.org/papers/2017/ESWC_2017_QALD/public.pdf.

[8] P. Trivedi, G. Maheshwari, M. Dubey and J. Lehmann, LC-QuAD: A Corpus for Complex Question Answering over Knowledge Graphs, in: *International Semantic Web Conference*, Springer International Publishing, Cham, 2017, pp. 210–218.

[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] 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.

[13] 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.

[14] 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.

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

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

[17] L. Hirschman and R. Gaizauskas, Natural Language Question Answering: The View from Here, *Nat. Lang. Eng.* **7**(4) (2001), 275–300. doi:10.1017/S1351324901002807.

[18] D. Diefenbach, V. Lopez, K. Singh and P. Maret, Core Techniques of Question Answering Systems over Knowledge Bases: A Survey, *Knowl. Inf. Syst.* **55**(3) (2018), 529–569–. doi:10.1007/s10115-017-1100-y.

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

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] S. Han, H. Shim, B. Kim, S. Park, S. Ryu and G.G. Lee, Keyword question answering system with report generation for linked data, in: *2015 International Conference on Big Data and Smart Computing (BIGCOMP)*, 2015, pp. 23–26.

#	Pattern	SPARQL	Example	POS Tags
1	$X_{NN}\ Y_{VB}\ [\square_{DET}]\ Z_{NN}$	getEntity(X) getProperty(Y) getEntity(Z)	Est-ce que Barack a marié Michelle ?	Barack$_{NN}$ marié$_{VB}$ Michelle$_{NN}$
2	$X_{WP}\ Y_{VB}\ Z_{NN}$	getEntity(Z) getProperty(Y) ?*answer*	Qui a développé Skype ?	Qui$_{WP}$ développé$_{VB}$ Skype$_{NN}$
3	$X_{ADV}\ Y_{NN}\ \square_{VB}\ Z_{VB}$	getEntity(Y) getProperty(Z) ?*answer*	Quand la Statue de Liberté a été construite ?	$Quand_{ADV}$ Statue de Liberté$_{NN}$ été$_{VB}$ construite$_{VB}$
4	$X_{DET}\ Y_{NN}\ \square$	?answer *typeOf* getEntity(X)	Quels presidents sont nés après 1945?	Quels$_{DET}$ présidents$_{NN}$ [...]
5	⟨**EST-CE QUE**⟩ Y_{NN} ⟨**ÊTRE**⟩ Z_{NN}	getEntity(Y) getProperty(Y,) getEntity(Z)	Est-ce que Margaret Thatcher a été chimiste ?	⟨**Est-ce que**⟩ Margaret Thatcher $_{NN}$ chimiste$_{NN}$
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 ?	Quelles voitures sont fabriquées au [Canada] ET_{CONJ} [État-sUnis]

Table 12
POS TAG Patterns

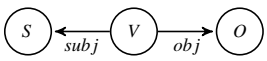
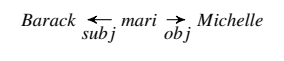
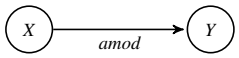
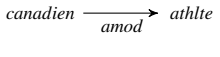
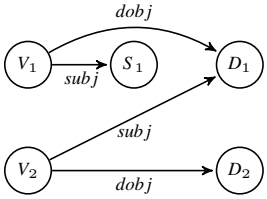
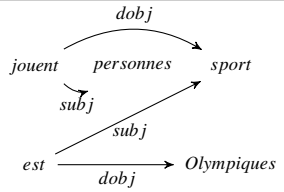
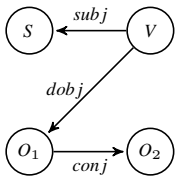
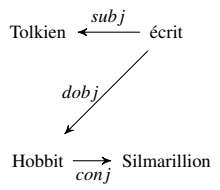
#	Diagram	Dependency pattern	SPARQL	Example	Syntax Tree
1		subj(v, s) obj(v, o)	getEntity(s) getProperty(v) getEntity(o).	Est-ce que Barack a marié Michelle ?	
2		amod(x/$_{ADJ}$,y/$_{NN}$)	getEntity(y) getProperty() getEntity(x).	athlète canadien	
3		subj(V_1,S_1) dobj(V_1,D_1) subj(V_2,D_1) dobj(V_2,D_2)	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