POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Multilingual SPARQL Query Generation Using Lexico-Syntactic Patterns

NIKOLAY RADOEV
Département de génie informatique et génie logiciel

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie informatique

Avril 2019

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Ce mémoire intitulé :

Multilingual SPARQL Query Generation Using Lexico-Syntactic Patterns

présenté par **Nikolay RADOEV**
en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

DEDICATION

To all my friends,
Without whom I would've finished a long time ago...

ACKNOWLEDGEMENTS

I would like to thank my supervisors Amal Zouaq and Michel Gagnon for their immense help, guidance and the **many** detailed comments all throughout my work. Their advice and supervision has helped me learn a lot for the past two years. I would also like to acknowledge their patience with my disorganization and habitual lateness.

I would like to thank my friend and coworker Mathieu Tremblay who designed a bit part of the original AMAL system and has helped me many times since then. I thank all my friends who supported me while I worked, especially Yonni Chen and Emilio Rivera with who I started my masters and are the reason I didn't quit two months in.

Finally, I would like to thank my mother who helped me with many of the linguistic questions I had for my work and my dad and sister who had to endure many conversations about things they didn't care about.

RÉSUMÉ

Le Web Semantique et les technologies qui s'y rattachent ont permis la création d'un grand nombre de données disponibles publiquement sous forme de bases de connaissances. Toutefois, ces données nécessitent un langage de requêtes SPARQL qui n'est pas maitrisé par tous les usagers. Pour faciliter le lien entre les bases de connaissances comme DBpedia destinées à être utilisées par des machines et les utilisateurs humains, plusieurs systèmes de question-réponse ont été développés. Le but de tels systèmes est de retrouver dans les bases de connaissances des réponses à des questions posées avec un minimum d'effort demandé de la part des utilisateurs. Cependant, plusieurs de ces systèmes ne permettent pas des expressions en langage naturel et imposent des restrictions spécifiques sur le format des questions. De plus, les systèmes monolingues, très souvent en anglais, sont beaucoup plus populaires que les systèmes multilingues qui ont des performances moindres. Le but de ce travail est de développer un système de question-réponse multilingue capable de prendre des questions exprimées en langage naturel et d'extraire la réponse d'une base de connaissance. Ceci est effectué en transformant automatiquement la question posée en requêtes SPARQL. Cette génération de requêtes repose sur des patrons lexico-syntaxiques qui exploitent la spécificité syntaxique de chaque langue.

ABSTRACT

The continuous work on the Semantic Web and its related technologies for the past few decades has lead to large amounts of publicly available data and a better way to access it. To bridge the gap between human users and large knowledge bases, such as DBpedia, designed for machines, various QA systems have been developed. These systems aim to answer users' questions as accurately as possible with as little effort possible from the user. However, not all systems allow for full natural language questions and impose additional restrictions on the user's input. In addition, monolingual systems are much more prevalent in the field with English being widely used while other languages lack behind. The objective of this work is to propose a multilingual QA system able to take full natural language questions and to retrieve information from a knowledge base. This is done by transforming the user's question automatically into a SPARQL query that is sent to DBpedia. This work relies, among other aspects, on a set of lexico-syntactic patterns that leverage the power of language-specific syntax to generate more accurate queries.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ACRONYMS

AMAL	Ask Me in Any Language
API	Application Programming Interface
CNL	Cloud Natural Language
KB	Knowledge Base
LAMA	Language Adaptive Method for question Answering
LSTM	Long Short-Term Memory
NLP	Natural Language Processing
OWL	Web Ontology Language
POS	Part of Speech
QA	Question Answering
QALD	Question Answering over Linked Data
RDF	Resource Description Framework
RDFS	RDF Schema
RNN	Recurrent Neural Network
SQA	Scalable Question Answering
SPARQL	SPARQL Protocol and RDF Query Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator

LIST OF APPENDICES

CHAPTER 1 INTRODUCTION

Recent developments and democratization of technology have led to an increasing amount of available data as well as increasing audience looking to access this information. An increased audience has also led to more and more less technologically savvy users turning towards the web for information. This has prompted an increased interest in developing systems aimed to answer user questions while abstracting most of the technical details and the different types of data that have to be analyzed to get the answer. This field is also known as Question Answering (QA) and its aim is to build such systems. QA systems can be classified as closed-domain, meaning that they handle questions only about a specific topic or open-domain, meaning that they are not restricted to a single domain and rely on more general knowledge. Open-domain question answering offers a more interesting challenge given that it has to deal with much more diverse information in many different topics and it represents a more realistic use case scenario, making it a more interesting choice for our work.

Generally, an answer can be extracted from different sources using two prevalent methods: information-retrieval (IR) and knowledge-based question answering. IR question answering relies on large quantities of textual information in collections such as Wikipedia or other texts (mainly unstructured data) and aims to find the most relevant documents or passages for a given question and use those to formulate an answer.

Knowledge-based question answering relies on creating a semantic representation of the question and trying to match it to a structured set of data. Such sets, also known as Knowledge Bases (KB) or more recently linked data, represent information as entities connected together by one or more relations. One of the advantages of performing question answering over linked data is that since the data is already structured, a QA system "only" has to transform the user's question in a similar logic structure while IR methods have to handle both the question and answer sources in an unstructured form. This is one of the reasons why we have chosen, in this work, to focus on question answering using KBs and not unstructured data.

Depending on the information source, different approaches can be used. Deep learning techniques [1] such as word embeddings combined with recurrent neural networks (RNN) have gathered increasing popularity recently but many more *classic* deterministic methods such as rules and string matching are still widely used for question answering. While deep learning methods offer promising results, they often use a *black box* approach where their inner processing is not clearly detailed, making their answers difficult to justify or explain. We have opted for a more deterministic approach based on specific rules and patterns with a clearly divided workflow. This *white box* method also makes it easier to explain and understand every part of the process and to evaluate the separate segments.

Another aspect considered in this thesis is the ability to answer questions in different languages, also known as multilingual question answering. Even though there are quite a few popular QA systems, most of them only accept questions in one language, mainly English. For those offering functionality in multiple languages, performance tends to be sub-par, compared to monolingual systems. The work presented in this thesis aims to tackle the issue of balance between general performance and the capability to process questions in multiple languages. We propose a QA system whose main goal is to translate the user's question into a machine-readable query using a specific query language (SPARQL in our case) and return the data resulting from this query. This QA system can handle questions in both French and English. The proposed system is an improved version of our QA system *AMAL* [2] (available in the annex of this thesis), the winner of the QALD-7 Challenge [3].

In addition, we propose a method for the generation of SPARQL queries that applies to multiple languages. This method consists of looking for the presence of various lexico-syntactic patterns in the syntactic representation of the user's queries and generating SPARQL queries based on specific rules associated with each pattern. Patterns are divided into two categories: those based on the question's syntactic dependencies and those based on the parts of speech (POS). Both sets can be used independently or together for better performance. The lexico-syntactic patterns and their SPARQL generation rules are implemented in our QA system and their effect on the system's ability to correctly answer questions is then evaluated.

1.1 Problem Definition

As previously stated, the field of knowledge-based question answering consists in developing systems that often *translate* natural language questions into structured queries that a machine can more easily understand.

Existing QA systems do have several limitations as to the way the user has to formulate their questions. Some systems expect the user to have a certain amount of knowledge of the knowledge base or the query languages. Other QA systems use a specific grammar that restricts the way a question can be worded, leading to *unnatural* questions for the user.

Another issue is the prevalent use of English-only QA systems. While the main goal of Web Semantics is to abstract the semantics (meaning) of information away from other variables such as format, syntax or even language, most systems only handle questions given in English. This can be a disadvantage to users who do not have a working knowledge of English or who prefer to obtain information in their native tongue.

Even for systems that aim for a *multilingual* approach, the methods used are often very generic and aim to fit all languages without adapting to their particularities. This can lead to handling multiple languages at the cost of low performances across all supported languages, reducing the overall utility of the system.

1.2 Concepts and Definitions

Hereafter, we present some background knowledge on the various fields and methods related to question answering systems.

1.2.1 Semantic Web

The Semantic Web is an extension of the World Wide Web and is largely expressed by standards defined by W3C (World Wide Web Consortium). The term was first described by Tim Berners-Lee [4] when talking about a web of data designed to be processed by machines rather than humans. This comes as a contrast to most of the web that was designed to be human readable and does not offer any structured data.

Conceptually, the Semantic Web aims to represent semantically structured knowledge. It can be represented as a vast graph where every node represents any arbitrary thing and every edge represents the relationship between two nodes with the addition of assigning a meaning to the relationship.

A formal graph representation with labeled relationships between entities allows for information to be read and exploited not only by humans but also by machines which are already adapted to work with this form of data structures. Using a common framework to describe data only through semantics allows for sharing of that information between different applications, communities, and even languages.

The Semantic Web is realized by the aggregation of various different technologies. Specifications such as RDF (Resource Description Framework) [5] describe how information is represented, OWL (Web Ontology Language) [6] aims to adapt ontologies of different domains to the World Wide Web. Structured data on the Web can be either represented by metadata tags on web pages or stored more permanently in a knowledge base. Those knowledge bases are mostly accessed by using a query language, SPARQL [7] being the one recognized as an official standard by the W3C.

1.2.2 Knowledge Base

Knowledge bases are a technology used to store complex structured data and are mainly aimed to be used by a machine or a computer system. KBs can be either domain specific or cross-domain oriented. The Linked Open Data Cloud [8] community project describes over 1200 publicly available datasets as of March 2019 with DBpedia [9] and Wikidata being among the largest ones. KBs are also used by commercial products such as Google's Knowledge Graph [10] and Wolfram Alpha's Wolfram Knowledgebase [11], both of which are integrated in their companies' search engines.

Unlike normal relational databases, most KBs store their data in a triple format where each triple contains two entities and a relation, also called property, that links them. This closely follows the general idea of the Semantic Web where information is represented in a directed graph format. Since most KBs use RDF, they can be queried using SPARQL as a query language.

As already mentioned, KBs are designed to be used primarily by machines and not humans with some of them, such as Wolfram Knowledgebase, being only accessible through specific systems. Generally, those systems are designed to facilitate the access to the KBs information by abstracting the way the data is structured to the user and simplifying the way the user needs to formulate their query. Such tools are also known as Question answering systems or QA systems.

Throughout this thesis, examples will be given using DBpedia. DBpedia is a large, community driven project, aimed to extract structured data from information available in Wikipedia. DBpedia stores cross-domain data as an open knowledge graph using RDF. Information can be retrieved using SPARQL queries to the public Virtuoso endpoint[1]. As of the latest stable release in 2017, DBpedia contains 9.5 billion RDF triples with 1.3 billion extracted from the English version of Wikipedia and 5.0 billion originating from other language editions. DBpedia also provides 125 localized versions with English being the most complete version with 4.58 million entities. A more detailed look of DBpedia and why it was chosen is given in section 3.2.

For the sake of readability, Table 1.1 shows the prefixes that will be used for DBPedia URIs throughout the thesis.

[1]http://dbpedia.org/sparql/

Table 1.1 DBpedia prefixes

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

1.2.3 SPARQL

SPARQL, a recursive acronym for SPARQL Protocol and RDF Query Language, is a standard query language designed to manipulate data in RDF format. Given that SPARQL was designed to manipulate RDF data, its syntax is very similar to RDF's triple representation.

Generally, SPARQL queries contain a set of triples and represent a subgraph of the RDF data. Each of the three elements can be a variable. Variables can either be *bound* to specific URI or literal or *free* and take any value in their domain.

A single SPARQL triple can be defined as $(s,p,o) \in (X \cup R) \times (X \cup P) \times (X \cup R \cup L)$ where R designates all entities, P all properties and L all literals in the database.

SPARQL queries are constructed using a set of clauses with different functions. The most basic ones are the *SELECT* clause which describes what variables are in the query result and the *WHERE* clause which contains the SPARQL triples and defines the subgraph that should be matched in the RDF data. This is also called the *SELECT form*. For example, if we want to find who is the wife of Barack Obama, the SPARQL query would look like :

```
SELECT ?wife
WHERE { dbr:Barack_Obama dbo:spouse ?wife . }
```

SPARQL also supports the *ASK form* which does not return information but rather a true value if the subgraph defined by the SPARQL triples can be matched in the RDF data. For instance, asking if Barack Obama is married to Michelle Obama takes the following SPARQL form:

```
ASK
WHERE { dbr:Barack_Obama dbo:spouse dbr:Michelle_Obama . }
```

SPARQL's syntax describes other functions such as *DISTINCT* for eliminating duplicates, *ORBER BY* for result ordering and many others. Users familiar with other query languages, such as SQL, should be able to recognize most of the similar functions.

1.2.4 Lexico-Syntactic Patterns

In linguistics, *syntax* is defined as the set of rules and principles that describe the structure of sentences in a particular language. The concept of lexico-syntactic patterns is to find syntactic structures that occur with enough frequency to be considered significant. Those structures are then combined with the words they represent to form a lexico-syntactic pattern. In this thesis, we focus on two types of syntactic representations. A more detailed explanation of each type of pattern is given in chapter 5 in sections 5.5 and 5.4.

The first syntactic representation is using the *part of speech (POS)*, also known as grammatical categories. POS include nouns, verbs, adjectives, etc. POS patterns are represented as a sequence of POS tags and in order for a segment to match a given pattern, it has to contain the same tags in the same order as the pattern. For example, if we have the following pattern: *NN VB DET NN*, it could match the segment *Tolkien(NN) wrote(VB) The(DET) Hobbit(NN).* since its syntactic representation contains the same POS in the same order as the pattern.

The second representation uses the syntactic dependencies between the words in the question. A dependency between two words is expressed as a function that has those two words as input variables. Table 1.2 shows different syntactic dependencies. Similar to the POS patterns, dependency-based patterns can be matched to a sentence segment only if all of the syntactic dependencies are present. Unlike POS patterns, there is no particular word order to be respected. For example, if we have the pattern *subj(V,S), dobj(V,O)*, it could match the segment *Tolkien(S) wrote(V) The(D) Hobbit(O)* where the dependency *DET(D,O)* is ignored since it is not part of the pattern.

Table 1.2 Dependency details

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 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

1.3 Research Objectives

The challenges highlighted in section 1.1 can lead to a few different observations when it comes to modern QA systems.

- Most QA systems expect users to know how a particular query language work or to abide by restrictive rules when it comes to asking questions.
- Most systems focus partially or entirely on English questions.

Combining those observations, we can formulate the following research question: can an effective multilingual SPARQL query generation be driven by lexico-syntactic patterns?

Based on both our observations and this question, we can define the following research objectives :

1. Build a QA system that is able to process natural language questions in multiple languages.
2. Identify various lexico-syntactic patterns capable to generate SPARQL triples that represent the content of the question.
3. Implement the proposed patterns in the QA system and evaluate their performance.

The proposed QA system is able to process questions in two languages: French and English, making it a bilingual system. We do however hypothesize that the results for French can be representative of other languages.

1.4 Thesis Overview

This thesis is structured in 6 chapters, including two research papers in chapters 4 and 5 as well as an additional paper in Appendix A. Chapter 2 presents a literature review about the state of the art for QA systems as well as other connected topics, such as entity and property disambiguation. Chapter 3 describes the research methodology and explains the link between the two papers included in this thesis. The first one on the *LAMA* (Language Adaptive Method for question Answering) system, in chapter 4, focused on the system's presentation, implementation and evaluation. This directly covers the first research objective, aiming to create a multilingual QA system. This article is followed by a second one in chapter 5 which enriches LAMA with lexico-syntactic patterns in English and French, and evaluates their impact on the performance of the QA system. The conclusion in chapter 6 discusses the system's limitation as well as potential future work.

In addition, Appendix A contains the original article describing the *AMAL* (Ask Me in any Language) system, the predecessor of *LAMA*. As already mentioned, this system was chosen due to its performance, earning it the first place in the competition for which it was created.

CHAPTER 2 LITERATURE REVIEW

This chapter focuses on exploring works related to our research topic at a high-level of abstraction. More specific details and references are given in the related work sections in chapter 4 and 5. In this chapter, we present the research and work done on QA systems in their various forms. We also look at how various systems handle different aspects of the answering process such as entity and property extraction, disambiguation and query generation. Finally, we look at how multilingual systems approach the QA problem.

2.1 Question Answering Systems

Question answering systems have the potential to become one of the most popular ways for human users to access large knowledge bases, designed to be normally used by software agents. QA systems have actually existed for quite some time, with BASEBALL [12] being one of the earliest QA systems in 1961 aimed at answering questions about baseball stats in the USA baseball league. The system was originally limited to single clause questions and constructs implying relations like *most* or *highest* were prohibited. Modern systems, such as IBM's Watson or Facebook's DrQA [1] have greatly expanded the work on QA systems and are able to answer open-domain questions without being restricted to a single topic.

With the advent of more structured data and the existence of challenges such as QALD (Question Answering over Linked Data), many more QA systems have been created, trying to tackle the problem of answering natural language questions using structured knowledge bases. These systems are explored in more detail in section 2.1.1.

QA systems are however not limited to using KBs as their information source. Some systems such as DrQA look for answers in a large corpus of unstructured documents, such as Wikipedia. DrQA and similar systems use a combination of different machine learning techniques to match a set of paragraphs or articles to a given question. Paragraphs and questions are tokenized and encoded into vectors using a deep learning network (e.g. long short-term memory network (LSTM, RNN)) using word embeddings and other features such as POS tokens or term frequency. Finally, some classifiers are trained to predict the answer's span start and end with the vectors produced by the networks explained above (paragraphs and question). The way questions and paragraphs are encoded and then classified varies between systems, but deep-learning-based systems share the same high-level logic.

This thesis focuses on QA based on information extraction in KBs. This is a different approach that does not include document matching but rather communication with structured data. The following section describes in more details the state of the art in QA systems also based on KB knowledge extraction.

2.1.1 QA Systems' Architecture

Since QA systems are not constricted by a particular standard, there is no predefined structure when it comes to their architecture. However, most modern systems share a similar high-level architecture. The research paper [13] presents a detailed survey of over 15 different QA systems that have participated in the different QALD challenges between 2011 and 2016 or retrieve information from DBpedia. The survey excludes systems such as CANaLI [14] which restrict the user's input or Sparkli [15] which relies on a graphical interface for creating its SPARQL queries.

Generally, the processing of a question follows a sequence of steps that can be classified into 4 different categories: Question analysis, Word Mapping, Disambiguation, and SPARQL Query Construction. There is no set order for the steps in the process, varying between systems. For instance, WDAqua-core1 [16] creates its queries before applying disambiguation while Xser [17] does the opposite sequence.

Systems that handle unstructured data have some additional steps including Document Retrieval and Document Selection that are used to determine whether some document contains the answer to the question, but since this thesis focuses on a system interacting with KBs, we will only focus on similar systems.

Question Analysis

In this step, the user's question goes through an analysis aimed to determine how the question should be segmented, which parts correspond to an entity or a property or what are the dependencies between the different words. NLP tools are sometimes used to extract additional information about the question, but some systems, such as SINA [18] do not use any.

An important part of the question analysis is recognizing groups of words corresponding to a specific resource, also known as named entities. For instance, the questions "Who is the director of the Titanic ?" contains *The Titanic* that refers to a film.

There are two main strategies to solve the problem of entity detection. Systems such as SINA and CASIA [19] try to map a group of words, or n-grams, in the question to a specific entity in the underlying KB. The n-grams are considered only as a possible named entity if there is at least one resource found. This has the advantage that each discovered named entity can be mapped to a resource in the KB. However, this method can generate many potential candidates, rendering the disambiguation task more complicated. Some other systems use Entity Linking Tools such as DBpedia Spotlight [20] which provide entity detection as well as entity linking with resources in the KB. Spotlight uses only the context around a potential entity to return the most likely candidate if more than one is available. This approach is limited when it comes to questions since the context is very limited, and only constrained to a single sentence.

Property or relation recognition is handled by much more varied methods by different systems. The most simple way is by having handmade rules that dictate which words or groups of words are linked to a given property. Systems such as PowerAqua [21] use regular expressions to identify question types and map them to a list of templates that indicates which property is to be used based on the question type and/or the named entities in the question.

POS taggers can also be used to detect properties. Often nouns correspond to entities and verbs and adjectives correspond to properties. This is however not always true. For instance, in "Who is the director of the Titanic?", the noun *director* can refer to the property *dbo:director*. Due to this limitation, Xser uses a POS tagger only for building a training set on which it learns what words are most likely to be a property. It does so by employing a custom tagger that labels words as: Entity, Relation(property), Variable or None using rules learned automatically. Rules are learned using POS tags, Named Entity tags and the question itself as its features.

Property detection can also be handled by using dependency parsing which gives the dependencies between all words in a question. This graph can be used to find relations between entities and look for properties in those relations. For instance, gAnswer [22] first identifies named entities and then chooses the shortest path in the dependency graph between them as a relation.

Word Mapping

Since QA systems have to query a knowledge base using specific URIs, they need to know how to map words discovered in the first step to KB specific resources. In many cases, this step is heavily linked with the question analysis, especially in cases where the n-gram method described above is used.

KBs storing their data in an RDF format provide a *label* property used to give a human-readable version for a given resource. All systems described in this chapter try to match a label of at least one resource to segments of the original question.

This method does have some limitations in the case where the segment does not match exactly a label. This can be due to a spelling mistake or a variation of plurality between the two strings. This is handled by different distance metrics with Levenshtein and Jaccard distances being by far the most popular. Systems like PowerAqua that use Lucene to match labels can use the fuzzy search option of the tools to expand the possible matches.

Another limitation with this method comes with cases where the textual segment and desired label are semantically equivalent but very different as strings. For instance, the segment *writer* and label *author* are close semantically but not in spelling. Most systems use lexical databases that are either handcrafted and system-specific or part of available lexicons such as WordNet. In the case of PowerAqua, using WordNet, each segment s is replaced by a vector of synonyms $[s_1 ... s_n]$ where each synonym is used to match a label.

Most systems try to match segments or some of their forms to labels in the KB. However, the BOA [23] framework takes the opposite approach and tries to extract natural language expressions that match a particular propriety P. In the case of BOA, the system extracts the subject-object pairs (X,Y) connected by P. A large text corpus is then scanned and all segments between *label(X)* and *label(Y)* or *label(Y)* and *label(X)* are extracted. The main idea is that those segments are a natural language representation of P and if found in a question, can be mapped to the particular property. This method is however very KB-specific. Its portability can be impacted since different KBs do not always share the same properties.

Disambiguation

Ambiguity can often arise when dealing with the questions, especially if those are asked in natural language. Most often, ambiguity comes when the word mapping returns multiple resources for the same segment of a question. For instance, in the case of "Who is the director of the Titanic?", *Titanic* clearly references the movie and not the ship. However, both the film and the ship can logically be returned when trying to bind "Titanic" to an entity in the KB.

To deal with these ambiguities, all presented systems use at least one of the following methods. The first method relies on checking the string similarity using some distance metric, between the segment and the resource. This method is used to rank resources and those with a high distance correspond to a lower overall ranking. The second method checks if different entities are actually directly linked together in the KB and is used to exclude some resources. For instance, "Titanic" the ship is not linked to *director* while "Titanic" the movie does indeed have a property *dbo:director* so only the second entity is kept.

Systems that rely on a graph representation such as gAnswer and PowerAqua take the second method and explore it much further. In the case of gAnswer, the system generates all possible graphs based on the question and tries to match them to subgraphs in the KB. Graph representations are then ranked based on how similar the labels of the resources and the question segments are. PowerAqua does have a similar approach but proceeds iteratively by individually ranking each potential resource and then trying to match its graphs.

In some cases, ambiguity cannot be tackled completely automatically. This can be due to a very ambiguous question or to the unreliability of the techniques used by the system. In those cases, some systems actually rely on the user's feedback to resolve the ambiguity by choosing from multiple resources. The Freya [24] relies mainly on user feedback when it deals with ambiguity.

SPARQL Query Construction

Information from the KB is extracted by querying it using one or more SPARQL queries. This research focuses on systems using SPARQL but there are some systems [25] that handle information from KBs without actually using SPARQL queries.

A popular approach used by QAKis [26] and PowerAqua is to use template queries. Templates are a set of predefined queries with missing slots that have to be completed with information extracted from the question. QAKiS only allows queries with a single triple while PowerAqua assumes that questions can be expressed using only one or two triples (maximum of two predicates) and applies its templates on those triples. As explained above, PowerAqua applies a graph-based disambiguation process and generates its SPARQL query only after.

Systems such as Freya generate SPARQL queries by parsing, from left to right, a question where some segments are associated with a particular source in the KB. Those resources are gathered in triples according to the order of the segments in the question with some variables being added between segments if a property is followed by another property in order to respect the Subject-Property-Object order.

QA systems like gAnswer and SemGraphQA [27] use a dependency tree. They generate a graph where every node is an entity or a variable and every edge is a property linking the entities. The graph reflects the structure of the final SPARQL query. Disambiguation techniques can be applied before or after generating all SPARQL queries with the choice of having an impact on the number of queries generated.

The SINA system also uses a graph-based approach, but the graph is created by connecting two nodes representing an entity by an edge representing a property only if that connection also exists in the underlying KB. This method is well-suited for systems that focus on keyword queries but since it ignores the question's syntax, some questions can be misinterpreted. For instance, this method does not make a difference between "Who is the father of Barack Obama?" and "Barack Obama is the father of who?".

2.1.2 Multilingual Systems

QA systems are, as described, quite numerous but almost all of them only focus on answering questions in a single language. In most cases, the only accepted language is English given its popularity and a large amount of information already available. The QALD Challenge first offered a multilingual task over DBpedia in its 6th edition with 8 participating systems[1]. Among them, only two answered questions in languages other than English and only one handled more than one language. The following 2 years, the WDAqua-core1 (originally called WDAqua-core0) participated claiming to be language-agnostic and able to handle multiple languages.

WDAqua does indeed handle multiple languages, those being English, French, Spanish, Italian, and German so far. The system tries to minimize the reliance on the specific language used in order to improve portability to other languages. The only language-specific part of the WDAqua system is the creation of a stop word list for each language with stop words being described as filler words that do not carry a lot of semantic meaning and can be removed from a question. The system also has a resource lexicon that binds properties to their KB labels as already described. Given that the system is not limited to a single KB, it uses the fact that other used KB, such as Wikidata, have a much better lexicalization than DBpedia and merge all the labels together.

WDAqua approach is decomposed in 4 steps: question expansion, query construction, query ranking, and response decision. The first step is similar to other systems where all possible entities and properties are extracted from the original question without limiting their number. The system then generates all possible SPARQL query using all of the extracted properties and entities but only for triples that exist in the underlying KB's graph.

Due to the large set of possible combinations, only SPARQL queries with a maximum of 2 triples are kept and no modifiers are used (ORDER BY, LIMIT, COUNT, etc). Resulting queries are then ranked based on several criteria: number of words covered by the query, the edit distance between a word and its associated resource, the number of variables and the

[1]http://qald.aksw.org/6/documents/qald-6_results.pdf

number of triples in the query. The resulting list is then filtered by using a pre-trained logistic regression model that gives a confidence score between 0 and 1 for a query and applying it to the first ranked query. If the first query of the list is below a certain threshold, the list is discarded and the system does not return an answer.

While the method presented by WDAqua-core1 is indeed applicable to multiple languages with minimal portability cost, the results presented in [28] are somehow limited. For the QALD7 Task 1 challenge, the F-score for English and French is 0.25 and 0.16 respectively with other languages being under 0.20. Results for other QALD datasets are somehow better with results of 0.51 for QALD-4 and QALD-5. No details are given about the difference in performance between the different datasets.

Our work's approach to multilingual questions is different than what the WDAqua team proposes. Given that results produced by language-agnostic approaches are clearly below an acceptable performance, we have opted for a more language-specific approach. While this increases portability cost, it is offset by delivering better performance by leveraging the added benefits that can be obtained by exploiting language-specific realities. Unlike most of the systems described in literature, we consider both the semantics and the syntax of the user's questions. While graph-based systems use syntax and dependency graphs to determine if some word segments are related together, they do not use it to determine *how* they are related. Our work aims to identify syntactic patterns that might help improve QA systems' performances when dealing with multiple languages.

CHAPTER 3 RESEARCH METHODOLOGY

This chapter describes the general approach used in order to complete the different research objectives and answer the thesis' main research question. Given that most of the information in this thesis is explained in the various papers presented in chapters 4 and 5, we focus here on how the different papers are related and how they complement each other.

3.1 Multilingual Question Answering

As described in section 1.3, the main objective of this research project is to propose, develop and implement a multilingual QA system that leverages the potential usefulness of different lexico-syntactic patterns. The proposed system has the additional requirements of allowing natural language questions without any constraints and also allowing questions in multiple languages, i.e. French and English in our case.

The final system built for the project and presented in both chapters 4 and 5 is *LAMA* (Language Adaptive Method for question Answering). LAMA is the second version of our QA system *AMAL* (Ask Me in Any Language), a system presented in the paper in Appendix A.

AMAL is a French QA system which was originally designed for Task 1 of the QALD2017 Challenge [3] at the 2017 ESWC Conference. Despite its name, *AMAL* was able to only answer questions in French and had no real native support for other languages in its original form. The system also focused on answering *Simple* questions that are defined as questions that concern only a single entity and property related to it or two entities and a single property linking both of them. Simple and *Complex* questions are described in sections A.3 and 4.3 with more details and examples.

Due to its modular architecture and encouraging results at the QALD 2017 competition, AMAL was chosen as a starting point for the rest of the research project. The system was developed throughout the last 2 years in an iterative and incremental fashion with two major updates, corresponding to the different research objectives being presented in this work.

The first article included in Chapter 4 describes the *LAMA* system, a successor of *AMAL* that was designed for the SQA Challenge [29] at the 2018 ESWC Conference, roughly a year after *AMAL*. Unlike its predecessor, *LAMA* is actually able to process questions in both French and English, making it truly a multilingual system and thus, completing one of the main research objectives of the project.

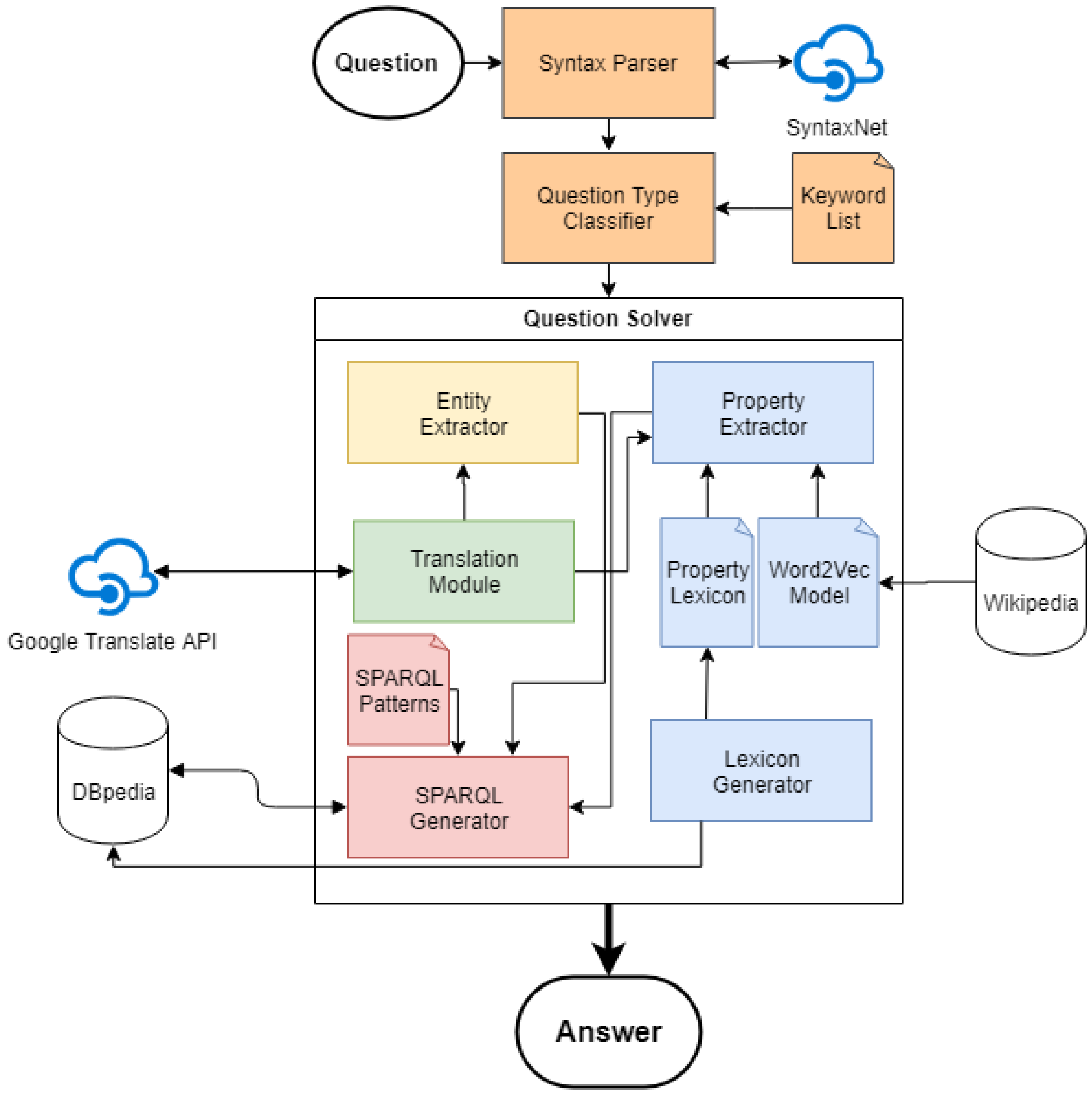


Figure 3.1 LAMA's Architecture

Architecturally, both systems are not very different, *LAMA* being a refinement of *AMAL*. The main differences are the reduced use of manually created rules and a shift towards automatically generated lexicons and resources such as a Property Lexicon and a Word2Vec model for the *Property Extractor* module. Another improvement of the system is that it is also capable of handling more complex questions and is not limited to the simple questions as in *AMAL*. A more detailed description of the system's implementation and inner working is given throughout the paper. Figure 3.1 shows the system's most up to data architecture, as described in Chapter 5. The update adds a Syntax Parser module in the preprocessing steps but keeps the same general structure.

3.2 Knowledge Base Exploitation

The *LAMA* system uses DBpedia, currently one of the biggest open knowledge bases. Using a knowledge base as a source of information can be quite helpful since it offers already structured data as well as a way to query that data in the form of SPARQL queries. Given its size multi-domain nature, DBpedia is better suited as a knowledge source than many other, smaller KBs.

DBpedia also has several other advantages, such as offering a relatively human-readable interface for entities. This is especially helpful in our case since *LAMA* only returns URLs to DBpedia entries, TRUE/FALSE in the case of close-ended questions or simply a number if the question requires some form of counting or aggregation. For instance: "How many official languages does Canada have?" returns 2, while "What are the official languages of Canada?" returns URLs for "French_Language" and "English_Language". The user can then follow those links for more information.

DBpedia is also available in multiple languages, making it very useful if we want to build a multilingual QA system that actually returns information in the same language as the original question.

DBpedia is however not without limitations. While there are indeed versions for languages other than English, the underlying knowledge graph differs from a language to another. For example, the French chapter of DBpedia has a different knowledge base compared to the English chapter, leading to different sets of answers for the same question. Our work shows that this can impact our performance, especially in French as the same information available in English is either missing or different in French DBpedia. This is the reason why French questions were translated to English in *AMAL* and why the Translation module remains available in *LAMA*. In some cases, translating a question or part of it remains a viable option to get an answer.

DBpedia can also be criticized for its limited use of labels when it comes to properties. For instance, *dbo:author* only has two labels : auteur(fr) and author(en). This is quite limiting, considering that the same property in Wikidata has over 20 labels in 4 different languages [1]. This can be important when creating the property lexicon, as shown in section 3.3.2.

[1]https://www.wikidata.org/wiki/Property:P50

3.3 Entity and Property Extraction

An important part of QA systems that interact with KBs using SPARQL is the detection of entities and the properties in the user's questions. A word or group of words can be matched to an existing entity or property in the KB and we use the URIs of the extracted entities and properties in our final SPARQL query that is sent to the KB for a final answer.

3.3.1 Entity Extraction

Entity extraction is performed first and it is overall an easier process than property extraction. We generally look for entities such as people, places, titles or other proper nouns, but we are not limited to those. We give a brief description of the entity extraction process here.

Entities are extracted by taking the user's input, removing the question word or words, and generating all possible substrings by recursively tokenizing the output. All resulting strings are then appended to the *dbr:* prefix and ran against DBpedia in order to filter them and only keep combinations that represent existing DBpedia URIs. This means that if our Entity Extractor returns a set of potential entities, all of those actually exist as resources in the KB. To improve our results, we also modify the used strings to generate more possibilities and handle some cases of ambiguity. A much more detailed description, including examples, is presented in section 4.3.2.

3.3.2 Property Extraction

Both papers in Chapter 4 and Appendix A discuss the inherent complexity of extracting properties from questions. The label and string matching methods described in section 4.3.2 show that entity extraction is quite straightforward, but that applying this to properties does not yield the same effectiveness.

This problem was tackled by creating a custom lexicon that binds specific words to an existing property URI in DBpedia, e.g. ["wrote":"dbo:author"]. Originally, the lexicon was created by hand by manually extracting the most common properties and linking them to the corresponding words in the questions.

To reduce reliance on manually created bindings, we have developed an automated lexicon creation that generates the bindings by parsing DBpedia properties and mapping their provided labels to their URIs. Section 4.3.3 provides a more detailed explanation of how the bindings are created from DBpedia. The method can also be applied for labels in other languages, with French being the one used in our case.

Property extraction and mapping are further helped by word embeddings such as a Word2Vec model to deal with cases where no property can be directly matched to a word in the question. Using the cosine similarity given by the embeddings, we can look if the word in our question is similar enough to other words that are bound to DBpedia properties in our lexicon. For instance, the word *wrote* is not bound to a DBpedia property, but its vector representation is close to *author* which is a label linked to the *dbo:author* property.

3.4 Lexico-syntactic patterns

While the first paper presents in detail *LAMA*'s implementation, the second paper in Chapter 5 focuses much more on the rest of the research's objectives: the presentation and implementation of the lexico-syntactic patterns.

In this paper, we present two types of patterns: dependency-based patterns that rely on the dependency graph of the question and POS-based ones that rely on the POS tags of the different words in the question. Both types of patterns are described in more detail in sections 5.4 and 5.5. For each pattern, we provide an example of a real question taken from the two datasets being used (QALD [3] and SQA [29]). Given that the system is multilingual, French patterns are also provided in the paper's appendix in the same format.

3.5 Datasets

Research and evaluation have both been done on two different datasets: SQA and QALD. As already mentioned, many of the examples given throughout all 3 papers come directly from either one of the datasets. A brief summary is thus well deserved in order to better understand the datasets' particularities and their choice. A more complete and detailed analysis is done in Section 5.3.

3.5.1 QALD

The QALD dataset is actually a combination from both QALD7 and QALD8 training sets where all duplicate questions have been removed, providing a final count of 384 unique questions. Each question is provided with translations in 6 different languages (French, English, Spanish, Italian, German and Danish) alongside a list of keywords for each language. We have only kept the French and English translations and all keywords are ignored since we don't want to impose keywords on the user's input. The training sets also provide SPARQL queries and their answers as a golden standard. Both the query and its answers are only provided for the English version of DBpedia. The QALD dataset is primarily composed of *Simple* questions with a total of 78% of its questions being represented by a single SPARQL triple.

3.5.2 SQA

The SQA dataset is similar in structure to QALD: it also includes the question's literal, a SPARQL query and its answers for each entry. However, SQA is not multilingual and only contains questions in English. SQA is also much bigger than QALD with a total of 5000 questions but unlike QALD, 3853, or 77.1% of the questions are classified as *Complex*.

Those datasets were chosen due to the fact that *AMAL* and *LAMA* were built for challenges on those datasets. QALD is also interesting due to the possibility of having questions that are both in French and English and while SQA lacks multiple languages, the high concentration of complex questions makes it an interesting candidate to test our QA system and use more complex methods to handle those questions.

3.6 LAMA's evaluation

The last research objective: the evaluation and validation of the patterns' impact on the QA system, is also covered in the second paper as well as the evaluation of the system without any patterns as a baseline for comparison.

Even if specified in our main research objective, we also compute the frequency of each pattern to verify that the presented patterns were actually significant enough to have an impact on the system's performance, without aiming for a specific frequency threshold.

Due to the particularities of the different datasets used, the evaluation was performed on both of the datasets separately and both sets of results are kept for the final evaluation.

In all evaluation cases (with or without patterns), evaluation was done by running all questions of each dataset through the system and comparing the system's output to the expected result provided to us by the datasets. Given that results are only given for the English DBpedia, evaluation was done using the English version of questions when ran on the QALD dataset. In order to be consistent throughout our work, closed-ended questions respect the closed-world hypothesis, meaning that the answer is either TRUE or FALSE and for cases where no definitive answer can be given, the answer is assumed to be FALSE.

The impact of the patterns was measured by comparing the system's performance on the same set of questions using four different methods with the first one being used as a baseline:

1. Without any patterns.
2. Using dependency-based patterns only.
3. Using POS-based patterns only.
4. Using a both types of patterns together.

For the evaluation, the F1-score measure was used, defined using the following formula:

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

The multilingual aspect of the pattern-based approach was evaluated differently given that only one of the datasets (QALD) had questions in multiple languages but no answers in French (they were given only in English). Because of this, a more qualitative approach was used to evaluate French patterns where the generated SPARQL triples were compared to the expected answer in French and each generated triple set was evaluated manually by humans (graduate students from the LAMA-WEST Lab) based on two criteria. Both criteria lead to a binary classification (Yes/No). The first one evaluates if the set is semantically equivalent to the question or a part of it, i.e., if the generated triples express the same meaning as some part of or the whole question. The second one evaluates if the set is similar to the English DBpedia triples given in the gold standard. A detailed description of how the evaluation is done as well as an example is given in section 5.6.3.

While not focused on time performance, the evaluation was performed on a 4 core i5-4690K CPU with 8GB of RAM using a hard 5 second timeout for SPARQL queries. This was done to demonstrate that the *LAMA* system can be ran on an average laptop without the need for a particularly powerful hardware.

CHAPTER 4 ARTICLE 1 : A LANGUAGE ADAPTIVE METHOD FOR QUESTION ANSWERING ON FRENCH AND ENGLISH

Authors

Nikolay Radoev[1] <nikolay.radoev@polymtl.ca>
Amal Zouaq[2], 1 <azouaq@uottawa.ca>
Mathieu Tremblay[1] <mathieu-4.tremblay@polymtl.ca>
Michel Gagnon[1] <michel.gagnon@polymtl.ca>

[1] Département de génie informatique et génie logiciel, Polytechnique Montréal
[2] School of Electrical Engineering and Computer Science, University of Ottawa

Presented at : ESWC 2018, Extended Semantic Web Conference, Heraklion, Greece
Published in : Communications in Computer and Information Science, vol 927

Abstract

The LAMA (Language Adaptive Method for question Answering) system focuses on answering natural language questions using an RDF knowledge base within a reasonable time. Originally designed to process queries written in French, the system has been redesigned to also function on the English language. Overall, we propose a set of lexico-syntactic patterns for entity and property extraction to create a semantic representation of natural language requests. This semantic representation is then used to generate SPARQL queries able to answer users' requests. The paper also describes a method for decomposing complex queries into a series of simpler queries. The use of preprocessed data and parallelization methods helps improve individual answer times.

4.1 Introduction

An important aspect of using the Semantic Web to its full potential is providing typical users with an easier way to access the growing amount of structured data available in different databases [3]. General knowledge bases (KBs), such as DBpedia [30], contain information extracted from Wikipedia, while many specialized knowledge bases have curated domain-specific knowledge, such as Dailymed [31]. However, given their reliance on SPARQL, these knowledge bases are difficult to use for the average user. Moreover, querying these KBs without knowledge of their underlying structure is a very complex task. Developing an intuitive interface to allow natural language queries is still an open problem [32].

The Scalable Question Answering over Linked Data Challenge (SQA) requires finding a way to transform a question into a query over a knowledge base, DBpedia being the one used in this particular challenge. Questions range from simple ones, which target a single fact about a single entity, to complex questions that involve many entities and require some kind of additional computation or inference to provide an answer. The training data provided in the challenge consists of 5000 natural language questions, for which one or more answers must be extracted from DBpedia.

In addition to being able to answer questions, systems must also be able to give those answers in a reasonable time and be able to handle a large load of queries in a given time. Assuming that normally, a QA system is not ran locally by the client, network delay in the user's requests must also be taken into account. If the QA system itself acts as a client to external services, the network delay becomes an important factor in the system's final performance.

We present LAMA (Language Adaptive Method for question Answering), an extension to AMAL [33], a system that was originally created for the QALD2017 [3] Challenge at the ESWC2017 [34] conference. The first version of our system was built specifically to handle questions in French. In LAMA, we extend this capability to the English language and handle more complex questions. Given that the AMAL system was based on the French syntactic structures, additional work was done to reduce the importance of language-specific structures. As an example, French often uses a *Subject-Object-Verb* structure (e.g.*Il le mange*, (*He it eats*) in English), compared to English where the *Subject-Verb-Object* structure is used instead (e.g. *He eats the apple*). In addition, the LAMA system does not enforce a controlled natural language [35] for the input queries but handles the questions "as-is".

Our approach aims at minimizing the number of manual features and ad hoc heuristics required to process questions. We also aim to optimize the preprocessing time for each question to improve scalability for a large number of questions or to deal with complex questions which may be costly in terms of computational resources. Finally, we propose a method to decompose complex questions into a series of smaller, simpler queries that can be more easily processed. This paper is a more detailed version of the one initially submitted [29] to the SQA2018 challenge.

While DBpedia is mentioned throughout the article, the LAMA system is not completely specific to the DBpedia knowledge base. Other KBs, such as WikiData [36], can be used to answer questions as long as some conditions are respected, notably the availability of the property configurations files 4.3. For readability purposes, we have abbreviated certain URIs by using the prefixes detailed in Table 4.1. For example, *http://dbpedia.org/ontology/spouse* becomes *dbo:spouse*. The *dbo:* prefix is used to represent classes (aka concepts) from the DBpedia ontology while the *dbr:* prefix is used to identify resources from this KB.

4.2 Related Work

Question Answering over linked data is a problem that has already been explored by different studies in the past. Generally, most QA systems follow a similar approach to produce answers: the user's question is taken as input, parsed to extract the various relations between the entities and then a query (most often written in SPARQL) is generated and submitted to one or more KBs.

Generally, all systems try to build a semantic representation of the question that relies mainly on the identification of entities and their properties in the user's questions. Xser [17] and WDAqua-core1 [16] rely on string matching and generate all possible interpretations for every word in the question, i.e. each word can be considered a potential entity or property. QAnswer [37] uses the Stanford CoreNLP parser to annotate questions and employs a mix of string matching with both text fragments and synonyms extracted from WordNet and the information given by the POS tags and lemma provided by the parser. This method has the added benefit of not relying on a perfect matching, making it more resistant to spelling mistakes.

While the general approach is similar in all the state of the art, the way the questions are parsed and what information is extracted vary among systems. Most systems [17], [27], [38] rely on semantic structures in order to find the answer to a given question. Xser defines a semantic structure as a structure that can be instantiated in the queried data [17]. For instance, SemGraphQA [27] generates direct acyclic graphs that represent all possible and coherent interpretations of the query and only keeps graphs that can be found in DBpedia's graph representation. WDAqua-core1 has a similar approach to [27] that ignores the syntax and focuses on the semantics of the words in a question by extracting as many entities as possible and trying to find relationships between them by exploring the RDF graph of each entity. Relying on partial semantics, where only a part of the question is given a semantic representation, appears to be a reliable approach towards QA given its wide use [17], [16], [27].

Table 4.1 DBpedia prefixes

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

Finally, all of the systems observed used DBpedia or Wikidata as their KB backend, making the use of SPARQL queries ubiquitous. WDAqua-core1 generates multiple SPARQL queries and ranks them based on multiple criteria, mainly the number of words in the question that are linked to resources in the query and how similar the label is to the corresponding resource. In contrast, QAnswer generates a single SPARQL query at the end of its entity-relationship extraction phase.

The approach proposed with LAMA is also based on the identification of entities and properties. However, unlike Xser [17] and others, our entity and property extraction do not rely only on string matching but also on lexico-syntactic patterns, detailed in sections 4.3.2 and 4.3.3. Additionally, our SPARQL query generator creates multiple queries in order to explore different possible interpretations of a question but also has some additional pre-processing to limit unnecessary query generation, unlike WDAqua-core1.

4.3 System Overview

As previously mentioned, LAMA is based on the AMAL system developed for the QALD-7 challenge and thus keeps the same core architecture. The system is developed using a modular approach to separate application logic in different subsystems. Each subsystem can be developed, modified or enabled independently. For example, the *translation* module can be toggled on or off based on the initial language of the query: in the original system, parts of the French requests were translated into English in order to retrieve information from the English version of DBpedia. While we have attempted to reduce the reliance on language-specific rules, all main modules of the LAMA system can be extended and adapted to include such rules if the developers deem it appropriate.

Originally, AMAL's main focus was to interpret and answer *Simple Questions*, which are questions that concern i) only one single entity and a property of this entity, e.g. *Who is the mayor of Montreal?* where *Montreal* is the **Entity** and *dbo:mayorOf* is the **Property** or ii) a single relationship between two entities (e.g *is Valerie Plante the mayor of Montreal?*). The LAMA system now also handles some *Complex Questions* by converting them into a set of *Simple Questions* and processing those smaller parts. The architecture of our system involves a multi-step pipeline: a *Question Type Classifier*, an *Entity Extractor*, a *Property Extractor* and a *SPARQL Query Generator* that generates the final SPARQL query that is used to retrieve the answer(s) from DBpedia.

The first step is to determine the type of the question and the nature of the expected answer. Once the system knows the question type, the query is sent to a specific solver that has been designed for this type of question. For instance, a question such as *Is GIMP written in*

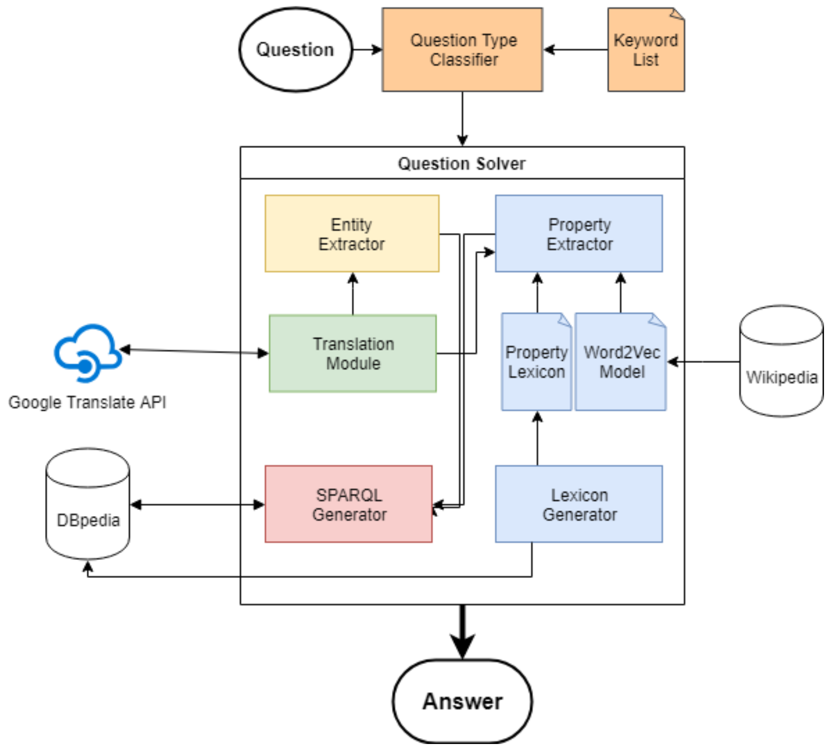


Figure 4.1 System Overview

GTK+? will be sent to the *Boolean* answerer, and *Who died due to Morphine?* is handled by the *Resource* question solver. Every question solver makes use of one or more submodules that function as *extractors*. There are two main extractors: an *entity extractor* and a *property extractor*. Once we obtain a semantic representation of the natural language question using knowledge base entities and properties, we generate one or more SPARQL queries based on these entities and properties. Given that multiple valid entities and/or properties can be found in a single query, all possible SPARQL queries based on these valid combinations are generated and saved in a queue.

The following sections detail the main steps of our system's pipeline: the Question Type Classifier, the Entity Extractor, the Property Extractor and the SPARQL Query Generator and their general implementation. As previously mentioned, each module can be extended and modified separately.

4.3.1 Question Type Classifier

Currently, questions are grouped into the following types: *Boolean, Date, Number* and *Resource*. *Resource* is the most generic type and designates questions for which the expected answer is one or more URIs. It is used as the default type for questions that don't fit any of the other categories. Boolean questions operate on the closed world assumption and always return a *TRUE* or *FALSE* answer, with *FALSE* being returned in the case where the answer is unknown. Date questions refer to dates in a standard *YYYY-MM-DD* format. The question type is determined through pattern matching with manually extracted patterns (roughly 5 patterns per question type) from the QALD6 and QALD7 training sets, totalling more than 400 questions. The different patterns and an example for each are given in more detail in Table 4.2. Questions about location are classified as *Resource* but have their own patterns to further help with classification. For example, classifying the question *Where is the Westminster Palace?* as a location question type, we can look for properties such as *dbo:location* or at least properties that are in the range of *dbo:Place*.

Classification is done by trying to match the questions to different types in the following order: Boolean, Date, Number, and Resource. This order is due to the relative complexity of type matching, with Boolean and Date questions being easier to detect while Number questions can be more complicated, i.e. requiring counting the number of answers in a set (e.g. *How many languages are spoken in Canada?*) or calculating a timespan (e.g. *How long did World War II last?*). In the current version of the system, multiple types are not supported and only the first detected type is considered. The system does however support subtypes, more specifically, the *Aggregation* subtype that applies to questions that require counting or ordering (ascending or descending) of results. Several examples are provided in Table 4.3.

This pattern-based method has the advantage of being easily transferable between languages in the context of a multilingual system, as patterns can be easily expressed in different languages. In fact, after extracting patterns for both French and English, the system was able to accurately predict the question type in more than 92% of the QALD7 training set.

4.3.2 Entity Extraction

As previously mentioned, the original AMAL project focused on *Simple Questions*, limited to at most one entity and one property. For example, a simple question is *Who is the father of Barack Obama?*, where 'Barack Obama' is the entity and 'father of' is the property. To extract the entity from the question, we first try to match the words (or a combination of these words) to existing DBpedia entities. Given that noun groups can contain many more elements than just the entity, such as adjectives or determinants, we start by removing the question keywords and consider the remaining string as the longest possible entity. We then generate all possible substrings by recursively tokenizing the output. All generated combinations are then appended to the *dbr:* prefix (see Table 4.1), e.g. *dbr:Barack_Obama* and are ran against

Table 4.2 Keywords for question classification

Contains	**Example**
	Boolean question
Does	Does Toronto have a hockey team?
Do	Do hockey games half halftime?
Did	Did Ovechkin play for Montreal?
Was	Was Los Angeles the 2014 Stanley Cup winner?
Is	Is Corey Crawford Blackhawks' goalie?
	Date question
When	When was the Operation Overlord planned?
What time	What time is the NHL draft?
What date	What date is mother's day?
Give me the date	Give me the date of the FIFA 2018 final?
Birthdate	What is the birthdate of Wayne Gretzky?
	Location question
Where	Where is the Westminster Palace?
In what country/city	In what country is the Eiffel Tower?
(Birth)place	What is Barack Obama's birthplace?
In which	In which state is Washington DC in?
the location of	What was the location of Desert Storm?

Table 4.3 Number and Aggregation question classification

Contains	**Example**
How many	How many countries were involved in World War 2?
Count	Count the TV shows whose network company is based in New York?
List the [..]	List the categories of the animal kingdom.
Sum the things	Sum the things that are produced by US companies?
the most/the least	What is the most important worship place of Bogumilus?
first/last	What was the last movie with Alec Guinness?
[ADJ]-est (biggest, oldest, tallest, etc.)	Name the biggest city where 1951 Asian Games are hosted? Who is the tallest player in the NBA?

the DBpedia database to find as many valid (i.e. that exist in DBpedia) candidate entities as possible. For example, *Who is the queen of England?* generates the following substrings after removing the question indicator (*Who is*, a *Resource* question indicator): *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. The selected entities are then sorted by length.

In order to increase the set of considered entities, we add variations of the identified nouns using normalization and stemming. For example, the noun *queen* can be transformed to extract the entities *Queens* or *Queen*. If no entities are found using this method, we use the DBpedia Spotlight [39] tool. Spotlight is a tool for entity detection in texts and its performance is limited when applied to single phrases so it is only used as a backup method.

Once all the possible candidate entities have been extracted, we use multiple criteria, such as the length of the entity's string, the number of modifications needed to find it in DBpedia and whether we had to translate it, to determine their likelihood of being the right entity. For example, every modification through normalization and stemming reduces the likelihood of the entity to be chosen. In our previous example, since *Queens* requires 2 modifications (pluralization and capitalization), it is less likely to be selected as the correct entity by the entity extractor. The formula we currently use to combine these factors is the following, where e is the entity string:

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

where:

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

P is the number of characters added/removed to add/remove pluralization

$T = 1$ if e has been translated, 0 otherwise

$U = 1$ if e has not been capitalized, 0 otherwise

According to the formula, the score is penalized by half the length of the entity string if we used a translation. This only applies for languages other than English since the translation step is entirely skipped when analyzing an English query. Also, the more words it contains (a word is delimited by the use of spaces) the higher the score will be, if it has not been capitalized. For example, *In which city is the Palace of Westminster in?* we can extract *Palace of Westminster* and *Westminster*. However, using the above formula, *Westminster* has a score of 11 while *Palace of Westminster* has a score of 25. Therefore, we consider it more likely that the correct entity is *Palace of Westminster.*

During the entity extraction phase, there is very often the issue of ambiguity when deciding which resource in the knowledge base we are looking for. For example, *Victoria* can refer to: *Person, Ship, City, Lake, etc.* To determine which resource to use to answer the query, LAMA has a multi-step approach which relies not only on candidate entities but also on properties, as explained in section 4.3.2 and 4.3.3:

1. For every candidate entity being a disambiguation (*dbr:Victora,dbr:Victoria_(name), dbr:Victoria_(ship), dbr:Queen_Victoria_(ship), etc.*) for the word *Victoria*, we extract possible disambiguation entities following the disambiguation link provided in DBpedia and store them in a vector V where the first element is the original entity (i.e. *dbr:Victoria*).

2. For each entity in the vector V, we generate a SPARQL query that involves the entity and the properties extracted by the *Property Extractor* (see section 4.3.3). We keep only the entities whose queries return non-empty answers for an *ASK* SPARQL query looking for an existing link between the entity and the property.

3. In the case of multiple entities having the same property in their DBpedia description, we find the intersection between the entity's label and the original query's string representations (character per character) and compute its length (See the example below). Results are then sorted by a distance factor (from 0 to 1) based on the ratio of the intersection's value and the entity length using the following representation:

$$distance\ vector = \frac{|intersection\ set|}{entity's\ label\ length}$$

This can be best illustrated with an example using the following question taken from the SQA training set : *Ship Victoria belongs to whom?* The property in the query is: *dbo:owner* (representing the expression *belongs to*) but there are 2 candidate entities: *dbr:Ship* and *dbr:Victoria* and as already mentioned, *dbr:Victoria* has several possible disambiguations (more than a 100). Most of those can be eliminated as they are not linked to any other entity by the *dbo:owner* property. However, some resources do have the property and can be valid possibilities, most notably: *Victoria_(ship)* and *Queen_Victoria_(ship)* where both entities have a *dbo:owner* and are both *ships.* In this case, calculating the intersection's length between those entities and the original question's string, we get an intersection length of **12** (i.e. there are 12 letters shared between the two strings) for *Victoria_(ship)* and **13** for *Queen_Victoria_(ship).* However, the distance factor is **12/12** and **13/17** respectively.

Using those metrics, we return a list of ranked entities where *Victoria_(ship)* is the top entity.

4.3.3 Property Extraction

Besides the entity extraction, another step for the request's semantic representation involves the extraction of properties from the original question to generate SPARQL queries. First, the potential entities tagged by the *Entity Extractor* are removed from the query and the remaining tokens are parsed for potential properties. The process is actually run in parallel with the entity extraction since some parts of it depend on the existence of entity-property relations. This also helps with the processing time, since we can do both extractions at the same time.

To increase performance, we have automatically extracted more than 20000 DBpedia properties along with their labels, thus building a property lexicon. The information is stored in a hash table, allowing for a fast ($O(1)$) access to a property URI based on its label. Without this lexicon, verifying if a possible label is related to a valid DBpedia property would require sending a network query to DBPedia, which incurs additional time and computational investment. While much of the *label-URI* pairs can be considered noise, e.g. *dbp:125TotalPoints*, some can be quite useful for question answering tasks, e.g. ["author": dbo:author], ["spouse": dbo:spouse], ["birth place": dbo:birthDate] etc. In the case of properties that exist in both **dbo** and **dbp** prefixes 4.1, both properties are added to the *label-URI* pair with the **dbo** URI having priority, e.g. *"parent"* is a label with both *dbp:* and *dbo:* prefixes, but most entities use the *dbo:* prefix. Querying for the properties and their labels, with the additional option to only get labels in a specific language (English in this case) can be done using this SPARQL query to DBpedia :

```
SELECT ?pred ?label WHERE {
    ?pred a rdf:Property .
    ?pred rdfs:label ?label
FILTER (
    LANG(?label)="" || LANGMATCHES(LANG(?label), "en"))
}
```

This allows us to easily find existing properties based on their label and its presence as a text fragment in the query. If the entities identified in the natural language request do not have any of the potential properties in their description, they are simply discarded.

The original AMAL system relied on manually annotated mappings between properties and entities to tackle ambiguities in property selection. This initial reliance on manual bindings has been reduced in LAMA by adding some basic rules for valid entity-property relations. For example, *the tallest* can be ambiguous given that it can both mean *height* or *elevation*, with the first meaning used in a relation with the DBpedia entity type *Person*, and the second used with entities that are instances of *Natural Place*. A valid rule for *the tallest* would thus be: *the tallest* : *{height: dbo:Person}, {elevation: dbo:NaturalPlace}*

Having such type-based rules can improve performance and speed by exploring only properties that are part of a valid rule. While those rules are still manually introduced in the system, generalizing them to the most generic type eliminates the need to add a rule for each instance of the targeted class.

Unlike entity detection, where entities are often detectable using simple string matching with the use of some lemmatization techniques, properties are rarely available in their literal form in a question. For example, the question: *Who played Agent Smith in Matrix?* can be answered by using the DBpedia property *dbo:portrayer*, a word not present in the original query.

To address this issue, LAMA uses a Word2Vec [40] model to expand the possible properties that can be extracted from a natural language query. The Word2Vec model is trained on data from Wikipedia in both French and English. The trained model provides a list of words that are similar to a given word along with their cosine distance. Currently, the Word2Vec model only applies to single words so complex expressions are only handled by the lexicon. If a word is selected as a possible property label, but it does not match one in the property lexicon, LAMA explores the similarity vector of the word, looking for a label that is contained in the lexicon. Only words with a cosine distance above 0.5 are kept to minimize false positives.

As an example, using the question *Is Michelle Obama the wife of Barack Obama?*, the word *wife* is identified as the label of a potential property but is not a label present in the property lexicon. Using the Word2Vec similarity vector for *wife*, we find *spouse* with a cosine distance of 0.723, a word that is also an existing label (mapped to *dbo:spouse*) in the lexicon. Using this, the following SPARQL query can be generated to find the answer to the question:

```
ASK WHERE {
    <dbr:Barack_Obama dbo:spouse dbr:Michelle_Obama.>
}
```

4.3.4 SPARQL Query Generation

The last step is building and sending SPARQL queries to DBpedia. The system uses the standard DBpedia endpoint: *http://dbpedia.org*. For now, LAMA supports basic SPARQL queries such as *ASK* (for checking for existing entities and *Boolean* questions) and *SELECT*. The *SPARQL Query Generator* supports the **ORDER BY** modifier, which allows for sorted answers or a single answer, e.g. "What is the *highest* mountain...?" or "Who was the *youngest* prime minister of Canada?". Queries exist as basic templates in which specific values are injected in the form of RDF triples. When a question has multiple possible *entity-property* combinations, the generated SPARQL queries are sent in parallel since the different combinations are independent from each other. Only queries that return a non-null answer are kept, with the exception of *ASK* queries where an empty answer is considered **FALSE**.

4.4 Complex Question Simplification

As previously indicated, AMAL focused on *Simple Questions* and as defined, a *Simple Question* is a query that can be represented as a single RDF triple, e.g. *Is Valerie Plante the mayor of Montreal?* is represented by:

<dbo:Valerie_Plante dbo:mayor dbo:Montreal>

However, some natural language queries can be more complex, containing multiple relationships between entities in the same sentence. In LAMA, we aim to also analyze and answer such complex questions. In fact, many of those *complex* questions can be represented as a chain of smaller, simpler queries. For example, the question and its respective RDF triple representation:

Who were involved in the wars where Jonathan Haskell battled?
<dbo:Jonathan_Haskell dbo:battle **?wars**>
<**?wars** dbo: combatant **?answer**>

can be transformed into :

What battles was Jonathan Haskell in?
<dbo:Jonathan_Haskell dbo:battle **?wars**>
and
Who were the combatants in those wars?.
<**?wars** dbo: combatant **?answer**>

The question is thus separated in two sub-queries with *wars* being the only shared word. Currently, deciding when to split a *complex question* into multiple *simple questions* is based on the presence of a keyword 4.4, *where* being the separation marker in this example. The split is done right before the targeted keyword and the keyword is kept in the second sub-question. The following table shows the different keywords used to detect *Complex Questions*.

A special case is made for questions that involve the following structure: [...] **both** X **and** Y [...]. In such questions, two RDF triples are automatically generated that have the same structure, but the subject of both is **X** and **Y** respectively. For example, the question:

> Which university was attended by both Richard H. Immerman and Franklin W. Olin?

generates the following RDF triples :

> <dbo:Richard_H._Immerman dbo:education **?university**>
> <dbo:Franklin_W._Olin dbo:education **?university**>

By having a set of *Simple Questions*, each question can be analyzed separately and with a reduced amount of tokens in each phrase, the system has a lower probability to generate erroneous entity-property combination, e.g. linking *Jonathan Haskell* and *involved* (dbo:combatant) together in our example *Who were involved in the wars where Jonathan Haskell battled?*. Currently, the system waits for all triples to be generated by each *Simple Question* and combines them in a single SPARQL query. This saves on processing and answer time, since only a single SPARQL request is sent for the whole question, instead of an individual request for each RDF triple.

4.5 Parallel Processing and Scalability

The original version of the AMAL system had a procedural processing cycle which could bring scalability issues. The system had to wait for every SPARQL query sent to DBpedia to finish its execution before trying a new one and there was no predefined timeout for a request. Originally, all answers were expected to be in English, even if the system was designed to

Table 4.4 Complex Question Keywords

Contains	**Example**
Where	Who were involved in the wars where Jonathan Haskell battled?
Whose	What are the TV shows whose network is also known as the CW?
Which	Give me everything owned by networks which are lead by Steve Burke?
Also	Name the scientist whose supervisor also supervised Mary Ainsworth?
Who	List the resting place of the people who served in Norwalk Trainband

handle French queries only, thus requiring an extra layer of translation for every question. Questions where multiple translations were required were the biggest problem, since the Google API does not allow for batch querying.

For the SQA challenge, we propose a redesign of the system, focusing on optimizing scalability. The following points are the main focus of the optimization techniques and target different parts of the process in order to maximize results.

- The extra step of translating French into English is no longer required, given that the questions are in English only. This saves around 15 to 20% of computation time, depending on the complexity of the question and the amount of translations necessary to process the equivalent question in French.
- Some parts of the question's processing have been moved to the client side, most notably the lookup of valid DBpedia properties, which no longer requires sending requests to DBpedia. This reduces networking time overhead and access overhead since a hashtable has an $O(1)$ complexity.
- SPARQL queries for the same question are run in parallel. Since the generated SPARQL queries are independent, running a query as soon as it is generated by the solver, without waiting to see if the previous one was successful or not, helps with the scalability of the system for multiple queries ran at the same time.

4.6 Evaluation

The evaluation of our system was done on the training set for the SQA2018 challenge. The base benchmark was run on the 5000 provided questions in the training set. The training set contains roughly 22% of *Simple Questions*. We have evaluated both the accuracy of the returned answers and the time to execute all queries. The evaluation was performed on a 4 core i5-4690K CPU with 8GB of RAM using a hard 5 second timeout on SPARQL queries. The tests were run on a limited environment to demonstrate that the LAMA system can be run on an average user's machine without requiring any particularly powerful hardware. The system was able to answer 2654 out of the 5000 questions, which gives us a 53.3% accuracy for the answers with a success rate of 74% for *Simple Questions* and 33.5% for *Complex Questions*. The evaluation took a total of 15603 seconds, giving us an average of 3.12 seconds per query.

When our keyword-based query separation method was applied on the SQA training set, we noted a 20 to 22% improvement in question answering. Not all returned answers sets were complete, given that the method to separate the complex queries is rather basic, but since the previous system had no way to analyze such questions, it is a notable improvement. More details on how this can be improved in future are given in Section 4.7.

While the results are not as good as the original AMAL system (74% accuracy), we have to note that the new version relies much less on case-by-case exception handling and it has transitioned from a system aimed specifically at French to a multilingual system that handles French and English questions.

Finally, the SQA challenge also involves evaluating the competing systems on a test set. We observed that, in contrast to the training set, the test set includes many errors, such as spelling errors, case errors or incorrectly tokenized questions. We expect a decrease of LAMA's performance due to these errors. To evaluate the impact of spelling mistakes, we have taken 200 questions from the SQA training set and randomly added the same type of errors (as detailed in section 4.7) to some of the questions. This gives us an accuracy of 41% and precision of 45%, since some questions were only partially correctly answered. Based on this, it is clear that spelling has an important impact on the system's performance.

4.7 System Limitations and Future Work

One of the current limitations of the system is its difficulty to correctly split complex questions into multiple simple questions. Most of the errors in complex questions are due to the fact that the keyword used to split the sentence is not in fact a correct separator. For example, the sentence: *Give me the places where people who worked for HBO died in?* has been split into ;

{Give me the places} , *{where people}* and *{who worked for HBO died in?}*

This ends up giving 3 sub-queries where the first one (*Give me the places*) is correct, while the second one (*where people*) does not make sense and the last one (*who worked for HBO died in?*) is interpreted by LAMA as looking for somebody who work for HBO. The final result is thus obviously incorrect.

We plan on improving those results with the processing of the *Complex Questions* using a syntactic analysis instead of relying on simple keywords. Using a syntactic parser, such as SyntaxNet, will allow for easier division of queries. Figure 4.2 shows the syntactic analysis of our example above. Following the POS tagging and relations, the question can be split into : *[...] people who worked for HBO [...]* and *Give me the places where X [...] died in* where X is the result of the first sub-question. This separation corresponds to the correct interpretation of the question, which can thus be answered more accurately.

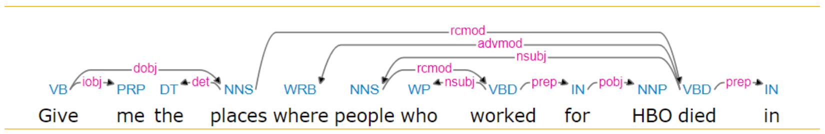

rcmod
advmod
nsubj
dobj
rcmod
iobj
det
nsubj
prep
pobj
prep
VB PRP DT NNS WRB NNS WP VBD IN NNP VBD IN
Give me the places where people who worked for HBO died in

Figure 4.2 Syntactic parsing example

Another limitation of LAMA is that it relies on string matching for entity and property detection. This means that the system is not resistant to typographic mistakes which can easily produce false results. While working on the system, we discovered that using Google Translate can sometimes correct spelling mistakes for some words even if asked to translate from English to English. However, this method is not reliable, providing spelling correction for only around 30% of the misspelled words and can also incur a non- negligible time cost if all words are to be translated for spell checking.

Future work for the system should include a spell check module that can correct most common mistakes. The testing set for SQA2018 has quite a few spelling mistakes and over 75% of those are simple letter inversions (*What* becomes *Waht*) or single missing letter (*hve* instead of *have*). Most of those mistakes can be detected and corrected by a spell checker and thus, improve entity and property extraction in the users' questions.

4.8 Conclusion

In this paper we presented the LAMA system, a modification of a currently existing QA system redesigned for multilingual support and scalability performance. We have modified it by adding emphasis on scalability and reduced its dependence on ad hoc heuristics and case by case exception handling. Additional work was done for handling *Complex* queries that are defined as questions requiring a SPARQL query with multiple RDF triples. A method for splitting those complex questions in a series of simple ones was also proposed.

There is still some work to be done, most notably by improving the way complex queries are decomposed in a sequence of simpler ones. Our approach relies on a string and label matching approach for entity and property extraction in the users' questions but suffers from accuracy issues in case of spelling mistakes. In order to reduce this problem, we plan on adding a spell checking module to the system's pipeline.

Acknowledgements. This research has been partially funded through Canada NSERC Discovery Grant Program.

CHAPTER 5 ARTICLE 2 : MULTILINGUAL QUESTION ANSWERING USING LEXICO-SYNTACTIC PATTERNS

Authors

Nikolay Radoev <nikolay.radoev@polymtl.ca>
Amal Zouaq <amal.zouaq@polymtl.ca>
Michel Gagnon <michel.gagnon@polymtl.ca>

Submitted to : Semantic Web - Interoperability, Usability, Applicability

5.1 Abstract

The abstract should briefly summarize the contents of the paper in 15–250 words.

5.2 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 [16, 27, 38, 41] 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 [42]. This paper aims to present the question-answering system LAMA [29] 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 [3, 43].

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.

5.3 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 5.1. The *dbo:* represents classes and concepts in DBpedia's ontology while *dbp:* and *dbr:* represent properties and resources respectively.

Table 5.1 DBpedia prefixes

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

5.3.1 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.

QALD Dataset

The QALD dataset is a combination of both the QALD7 and QALD8 training datasets provided for the QALD competition [3] 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 5.2. 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.

Table 5.2 Question Types per Dataset

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

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.

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 5.2 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.

5.3.2 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* [29] 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, [33]) 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

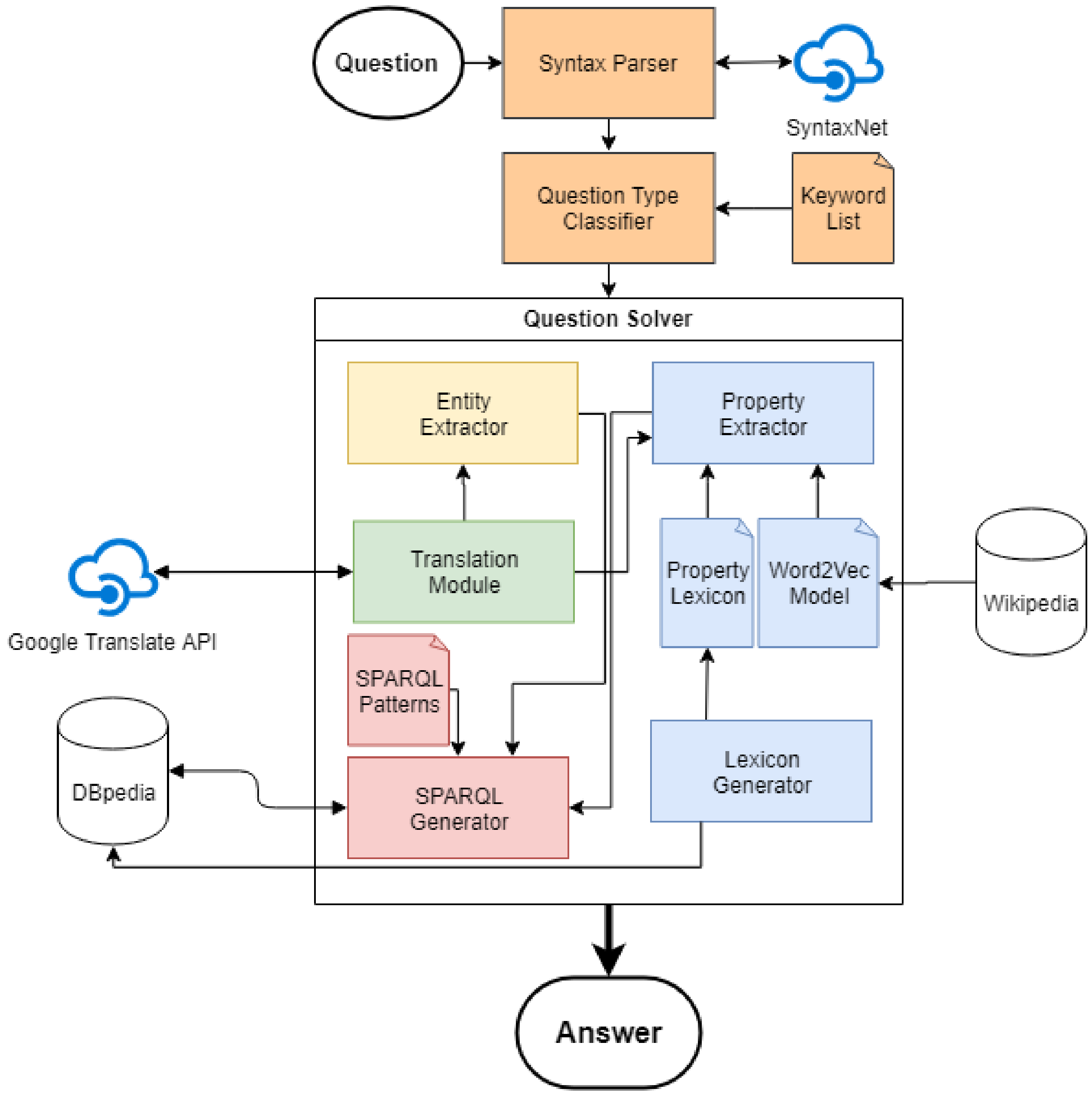


Figure 5.1 LAMA's Architecture

larger amount of data in English.

Figure 5.1 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

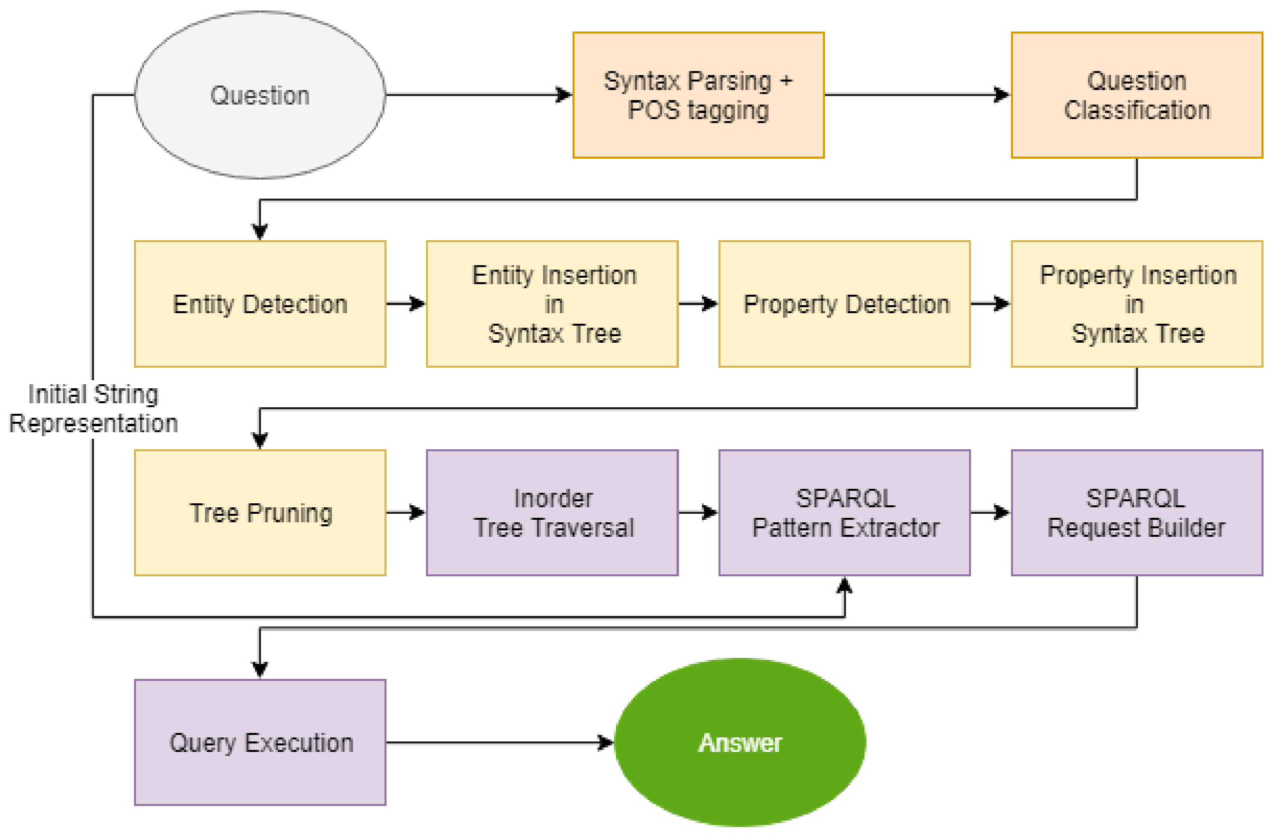


Figure 5.2 LAMA's Pipeline

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 [44]. Other Word2Vec models [45] 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 5.2 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 5.1 with more details given in the following sections.

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 [46] (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 [47]. 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 5.3 presents the most used dependencies, as well as a brief explanation for each.

Table 5.3 Dependency details

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 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

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 5.3.2 and 5.5. 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 [29]. 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 *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.*

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.

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 [39] 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:

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

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 5.4) and POS (Table 5.6) 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.

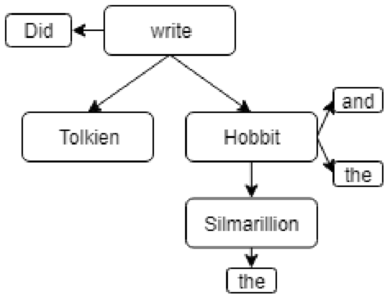
After extracting the entities, the system detects and extracts possible properties from the question. Unlike entities, in most cases, the expression of the property cannot be directly matched with its representation in the knowledge base. For example, using the same question as before, *dbr:J._R._R._Tolkien* 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.

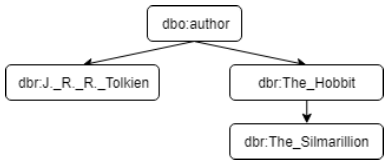
Like the entity extractor, the property extractor module offers a helper function called **getProperty(X)** where X is the word denoting the potential property. In some questions, no expression can be targeted as a potential property since the relation between two entities is implicit. In those cases,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

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, such as 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 removed. Such words are most often question markers such as *who, when, where, etc.* or left-over determinants. It is important to note that for question markers such as *when*, which denote a *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 5.3 Tree representation for the question: Did Tolkien write the hobbit and the Silmarillion?

For example, as seen in Figure 5.3 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. For some complex questions, the resulting representation can be analyzed as separate simple questions.The syntax tree traversal is done inorder, i.e. traversing it 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 sub-tree 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].

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 5.4 and 5.6. 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).

5.3.3 SPARQL queries and triple patterns

A standard SPARQL query, according to the standard definition [7], 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:

```
?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.

5.4 Dependency-based Patterns

5.4.1 Dependency Parsing

As already stated, our work is based on using SyntaxNet as a dependency parser. SyntaxNet is a transition-based dependency parser [48], 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 5.4 shows a graphic representation of a simple question and its dependency parse tree.

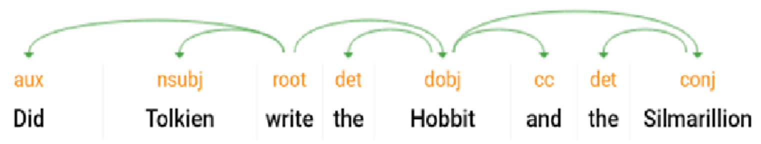


Figure 5.4 Dependency parse of a simple query

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* 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.

5.4.2 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 5.4.

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 5.4.

The dependency tree of the question is already presented in Figure 5.4 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 :

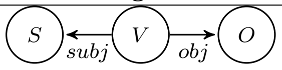
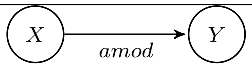
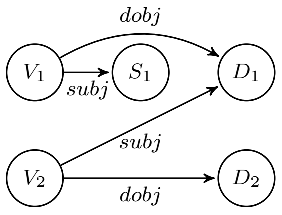
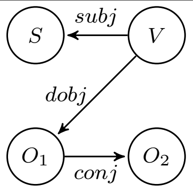
- S = Tolkien
- V = write
- O_1 = Hobbit
- O_2 = Silmarillion

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.

Table 5.4 Lexico-Syntactic Patterns

#	Diagram	Dependency pattern	SPARQL	Example
1		subj(v, s) obj(v, o)	getEntity(s) getProperty(v) getEntity(o).	Did Barrack marry Michelle ?
2		amod(x/$_{ADJ}$,y/$_{NN}$)	getEntity(y) getProperty() getEntity(x).	Canadian athlete
3	 dobj V_1 S_1 D_1 subj subj V_2 D_2 dobj	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?

5.5 POS-based Patterns

5.5.1 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 [49]. A full list can be see in Table 5.5. 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 5.5.

5.5.2 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 5.6.

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 5.4 and 5.6 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 $\square$ 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

Table 5.5 POS tags

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 5.6 POS TAG Patterns

#	Pattern	SPARQL	Example

1	$X_{NN}\ Y_{VB}\ [\square_{DET}]\ Z_{NN}$	getEntity(X)
		getProperty(Y)
		getEntity(Z)
			Did Barrack 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}⟨ **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?

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 5.6. 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 2^{nd} and 7^{th} 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 7^{th} 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)
```

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.

5.6 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.

5.6.1 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 5.3.1 and 5.3.1 which show the particularities for both datasets.

Table 5.7 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.

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 5.4 show questions that can occur naturally.

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.

5.6.2 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

Table 5.7 Dependency-based pattern frequency

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

Table 5.8 POS tag-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

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 5.9 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.

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.

Table 5.9 Impact of LAMA on SQA and QALD

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**

5.6.3 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 the following entity: *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 5^{th} one is not used since it does not exist in French and the 6^{th} 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 5.10 and 5.11. 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.

5.6.4 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 [46, 50], 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 [50] evaluates the performance of the ParseySaurus (now just called Parsey) dependency parser with an average labeled

Table 5.10 Dependency-based patterns in French

Pattern	**SPARQL Accuracy**
1	0.933
2	0.905
3	0.916
4	0.925

Table 5.11 POS tag 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

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.

5.7 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.

5.8 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 also been in 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 [16, 17, 29]. These systems try to answer questions relying mainly on the identification of entities and their properties and then trying to form coherent SPARQL requests that can be ran against the target KBs. Systems such as WDAqua-core1 [16] and Xser [17] 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 [37] that uses the Stanford CoreNLP parser for POS tagging and HAWK [51] that makes use of clearNLP [52] 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 [16, 17, 27, 38, 51] rely on semantic structures in order to find answers to a given question. SemGraphQA [27] 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 [16, 28] ignores the syntax and focuses on the semantic of the extracted entities of the question and explores the RDF graphs of the different entities to determine the appropriate relationships.

WDAqua-core1 is also a multilingual system than handles questions in five different languages: English,French,Spanish,Italian and German. It does not take in consideration the language of the original query and makes no use of NLP tools which makes it resistant to malformed questions. While the 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.

While those systems rely on a semantic representation and POS tags for some of them [51,52], LAMA also uses dependency parsing in addition to POS tags. Additionally, the semantic representation in LAMA derives from an initial syntax tree representation that is modified after entity and property extraction.

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 [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. 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 [53] 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.

5.9 Conclusion

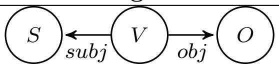
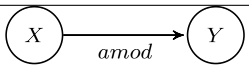
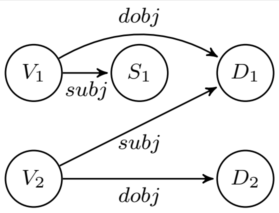
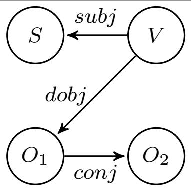
In this paper we present lexico-syntactic patterns aimed at improving question answering systems. The patterns are separated in two different categories : dependency-based patterns and POS (part of speech) tag patterns. We also present LAMA, a multilingual QA system that leverages the use of the shown 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.

Table 5.12 POS TAG 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	[...] $X_{CONJ\langle \text{ AINSI QUE} \mid \text{ET} \rangle}$ 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 5.13 Lexico-Syntactic 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?

CHAPTER 6 GENERAL DISCUSSION AND CONCLUSION

This final chapter concludes the presentation of our research project. We give a brief summary of our work as well as the limitations of our system and the proposed methods. Finally, we present some potential future works or research that can be done on the topic of multilingual QA systems.

6.1 Contribution

In summary, this thesis aims to complete the following objectives: create a multilingual QA system using multiple lexico-syntactic patterns. Taking a question in natural language (English/French) as input, LAMA was able to automatically generate an equivalent SPARQL query based on the DBpedia ontology and knowledge base. Throughout our work, we followed an incremental and iterative process to complete each of those objectives. This process is presented in the 3 papers in Chapters 4 and 5 and Appendix A, each giving an approximate snapshot of the work done roughly every 8 months since 2017. Our work was also presented at the ESWC conferences in 2017 and 2018 resulting in two published conference papers [2, 29].

The creation of our QA system was done in three steps. First, the *AMAL* system was created, aimed at answering simple questions with the help of some lexical patterns and without real support for multiple languages, being able to only answer questions in French. Second, the AMAL system led to *LAMA*, a more complex QA system that incorporates multilingual support by allowing questions in both French and English and improves various components of the original system. Finally, *LAMA* was enhanced by adding a dependency parsing module and a pattern-based processing step in its pipeline and introduced lexico-syntactic patterns. The presented patterns are separated into two different groups: dependency-based and POS-based ones. Each pattern is presented with its corresponding generated SPARQL query. An evaluation was done over two different datasets for each pattern type separately as well as a combination of both pattern types. Results show that the impact of using patterns in our QA system is noticeable, especially on the SQA dataset where an improvement of 0.28 on the F-score can be observed (from 0.535 to 0.816). Experiments on the QALD dataset show a smaller impact of 0.061, mostly due to the type of questions in the dataset. In addition to this, an was done to compute the relative frequency of each pattern in order to determine if they were really representative enough to be considered.

Comparison with other multilingual QA systems shows that while more generic approaches that are language-agnostic and that do not depend on NLP tools are easier to adapt to new languages, their performance is much lower and is hindered by their generic nature. There is an inherent complexity added to portability when focusing on language-specific patterns and methods. Some patterns can be transferred between languages but this does not apply all the time. In our case, while French and English are similar, their syntax is different enough to require changes to the applied patterns. This can be observed when comparing Tables 5.12 and 5.13 to the English patterns in Chapter 5. Language portability is however still a possibility, provided that the developer has a good enough understanding of the target language's syntax and rules and access to sample questions. Data sets such as QALD that provide the question as well as the corresponding SPARQL query can be particularly helpful. Our approach relies much more on NLP tools, takes into account the language of the question, and leads to better overall performance.

6.2 Limitations

While the system has improved over time, allowing for the processing of more complicated questions, there are some limitations to our work.

Having no context awareness, the system is thus limited to individual questions only: each question is treated as an independent event without knowledge of prior queries. For instance, a user cannot ask : *Who created Batman? When were they born?* as two sequential questions. Instead, he would have to create a single sentence: *When were the creators of Batman born?*. This has an impact on how questions are structured and can lead to some potentially unnatural question structures when the user is forced to rephrase everything in a single question.

Due to the use of various NLP tools such as a syntactic parser and a POS tagger, our QA system's performance is directly linked to the tools performance and accuracy. While both the POS tagger Parsey and SyntaxNet have rather good performances for languages such as French or English, this is not the case for less popular ones. Relying on syntax and POS limits the input questions to phrases that have correct spelling and especially, correct syntactic structure. This eliminates keyword queries or ill-formed questions that can lead to errors. In the case of the patterns, the ones that we propose were handcrafted and, as already described, do not cover all possible cases in the available datasets. This limits to some extent the generation of some SPARQL queries where the system requires additional processing in order to be able to fully translate questions into SPARQL. For example, questions that deal with counting or ordering are not handled by patterns and need further processing. Additionally, patterns are language-specific, meaning that they are limited to their language and there is no guarantee that they can be used with other languages.

6.3 Future Research

Limitations described in Section 6.2 show that there are potential improvements to the work presented in the thesis. The external tools as well as our Entity and Property Extractors were shown to be vulnerable to spelling errors and incorrect syntax. While correcting the syntax of the user's questions can be hard without imposing restrictive rules onto them, spelling can be more easily managed with various spell check methods and tools. This can be done directly with the original questions or assumptions can be made throughout the process, similar to how Google's search engine deals with spelling mistakes.

Evaluation of the described patterns shows that there is a noticeable impact of their use on a QA system. Improving performance further can be achieved by either focusing on more language-specific patterns, trying to cover as many use cases as possible per language or by trying to find more generic patterns that can be applied to as many languages as possible. In the first case, the focus on specific languages can improve performance on individual languages while sacrificing portability while the second option offers easier cross-language portability but has shown lower performance gains if used exclusively.

The patterns currently described have been extracted manually and the effort required to expand them without any automation can be considerable. Additional work and research can be done to automatically extract new patterns from existing questions. With the continued work of challenges such as SQA and QALD that have enriched the set of questions alongside with their SPARQL query representation, a large enough corpus can be created from which significant data could be extracted.

REFERENCES

[1] D. Chen, A. Fisch, J. Weston, and A. Bordes, "Reading wikipedia to answer open-domain questions," *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2017.

[2] N. Radoev, M. Tremblay, M. Gagnon, and A. Zouaq, "Answering Natural Language Questions on RDF Knowledge base in French," in *7th open challenge in Question Answering over Linked Data (QALD-7)*, Portoroz, Slovenia, May 2017.

[3] "Qald2017 challenge – eswc 2017 – hobbit," https://project-hobbit.eu/challenges/qald2017/, (Accessed on 03/29/2018).

[4] T. Berners-Lee and O. Lassila, "The semantic web," *Scientific American*, May 2001.

[5] "Rdf 1.1 concepts and abstract syntax," Feb 2014. [Online]. Available: https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/

[6] "Owl 2 web ontology language document overview (second edition)," Dec 2012. [Online]. Available: https://www.w3.org/TR/owl2-overview/

[7] "Sparql 1.1 query language," Mar 2013. [Online]. Available: https://www.w3.org/TR/sparql11-query/

[8] J. P. McCrae, "The linked open data cloud." [Online]. Available: https://lod-cloud.net/

[9] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, and Sören Auer and Christian Bizer, "Dbpedia - a large-scale, multilingual knowledge base extracted from wikipedia," *Semantic Web*, vol. 6, pp. 167–195, 2015.

[10] X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun, and W. Zhang, "Knowledge vault," *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD 14*, 2014.

[11] S. Wolfram, "Alpha: Making the world's knowledge computable." [Online]. Available: https://www.wolframalpha.com/

[12] B. F. Green, A. K. Wolf, C. Chomsky, and K. Laughery, "Baseball," *Papers presented at the May 9-11, 1961, western joint IRE-AIEE-ACM computer conference on - IRE-AIEE-ACM 61 (Western)*, 1961.

[13] D. Diefenbach, V. Lopez, K. Singh, and P. Maret, “Core techniques of question answering systems over knowledge bases: a survey,” *Knowledge and Information Systems*, vol. 55, no. 3, pp. 529–569, Jun 2018. [Online]. Available: https://doi.org/10.1007/s10115-017-1100-y

[14] G. M. Mazzeo, “Canali : A system for answering controlled natural language questions on rdf knowledge bases ucla csd technical report number : 160004,” 2016.

[15] S. Ferré, “Sparklis: an expressive query builder for sparql endpoints with guidance in natural language,” *Semantic Web*, vol. 8, no. 3, pp. 405–418, 2017.

[16] 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*, ser. WWW ’18. Geneva, Switzerland: International World Wide Web Conferences Steering Committee, 2018, pp. 1087–1091. [Online]. Available: https://doi.org/10.1145/3184558.3191541

[17] 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. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 333–344.

[18] S. Shekarpour, E. Marx, A.-C. N. Ngomo, and S. Auer, “Sina: Semantic interpretation of user queries for question answering on interlinked data,” *Journal of Web Semantics*, vol. 30, p. 39–51, 2015.

[19] H. Shizhu, Z. Yuanzhe, L. Kang, Z. Jun *et al.*, “Casia@ v2: A mln-based question answering system over linked data,” 2014.

[20] 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*. ACM, 2013, pp. 121–124.

[21] V. Lopez, M. Fernández, E. Motta, and N. Stieler, “Poweraqua: Supporting users in querying and exploring the semantic web,” *Semantic Web*, vol. 3, no. 3, pp. 249–265, 2012.

[22] L. Zou, R. Huang, H. Wang, J. X. Yu, W. He, and D. Zhao, “Natural language question answering over rdf: a graph data driven approach,” in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 2014, pp. 313–324.

[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. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 87–96.

[24] D. Damljanovic, M. Agatonovic, and H. Cunningham, “Freya: An interactive way of querying linked data using natural language,” in *Extended Semantic Web Conference*. Springer, 2011, pp. 125–138.

[25] N. Aggarwal and P. Buitelaar, “A system description of natural language query over dbpedia,” *Proc. of Interacting with Linked Data (ILD 2012)[37]*, pp. 96–99, 2012.

[26] E. Cabrio, J. Cojan, F. Gandon, and A. Hallili, *Querying Multilingual DBpedia with QAKiS*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 194–198. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-41242-4_23

[27] R. Beaumont, B. Grau, and A.-L. Ligozat, “Semgraphqa@qald-5: Limsi participation at qald-5@clef,” 09 2015.

[28] D. Diefenbach, A. Both, K. Singh, and P. Maret, “Towards a question answering system over the semantic web,” *Semantic Web*, p. 1–19, 2019.

[29] 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. Cham: Springer International Publishing, 2018, pp. 98–113.

[30] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives, *DBpedia: A Nucleus for a Web of Open Data*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 722–735. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-76298-0_52

[31] N. I. of Health *et al.*, “Daily med,” 2014. [Online]. Available: https://www.healthdata.gov/dataset/dailymed

[32] P. Gupta, “A survey of text question answering techniques,” International Journal of Computer Applications, 2012.

[33] 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. Cham: Springer International Publishing, 2017, pp. 90–105.

[34] “14th eswc 2017 |,” https://2017.eswc-conferences.org/, (Accessed on 03/29/2018).

[35] G. Mazzeio, “Answering controlled natural language questions on RDF knowledge bases,” https://openproceedings.org/2016/conf/edbt/paper-259.pdf, 2016.

[36] D. Vrandečić and M. Krötzsch, “Wikidata: A free collaborative knowledgebase,” *Commun. ACM*, vol. 57, no. 10, pp. 78–85, Sep. 2014. [Online]. Available: http://doi.acm.org/10.1145/2629489

[37] S. Ruseti, A. Mirea, T. Rebedea, and S. Trausan-Matu, “Qanswer - enhanced entity matching for question answering over linked data,” in *CLEF*, 2015.

[38] 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. Cham: Springer International Publishing, 2017, pp. 70–83.

[39] 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.

[40] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” *CoRR*, vol. abs/1301.3781, 2013. [Online]. Available: http://arxiv.org/abs/1301.3781

[41] 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.

[42] 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.”

[43] M. Dubey, “Lc-quad qald format,” Feb 2018. [Online]. Available: https://figshare.com/articles/LC-QuAD_QALDformat/5818452/6

[44] C. Schöch, “A word2vec model file built from the french wikipedia xml dump using gensim.” Oct. 2016.

[45] 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.

[46] S. Petrov, “Announcing SyntaxNet: The world’s most accurate parser goes open source,” https://research.googleblog.com/2016/05/announcing-syntaxnet-worlds-most.html, 2016.

[47] D. Andor, C. Alberti, D. Weiss, A. Severyn, A. Presta, K. Ganchev, S. Petrov, and M. Collins, “Globally normalized transition-based neural networks,” *CoRR*, vol. abs/1603.06042, 2016. [Online]. Available: http://arxiv.org/abs/1603.06042

[48] J. Nivre, "Algorithms for deterministic incremental dependency parsing," *Computational Linguistics*, vol. 34, no. 4, pp. 513–553, 2008. [Online]. Available: https://doi.org/10.1162/coli.07-056-R1-07-027

[49] Petrov, Das, Dipanjan, Ryan, and McDonald, "A universal part-of-speech tagset," Apr 2011. [Online]. Available: https://arxiv.org/abs/1104.2086v1

[50] Alberti, Chris, Daniel, Ivan, Collins, Michael, Gillick, Dan, Kong, Koo, and et al., "Syntaxnet models for the conll 2017 shared task," Mar 2017. [Online]. Available: https://arxiv.org/abs/1703.04929

[51] 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. Cham: Springer International Publishing, 2015, pp. 353–368.

[52] 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*, ser. HLT '11. Stroudsburg, PA, USA: Association for Computational Linguistics, 2011, pp. 687–692. [Online]. Available: http://dl.acm.org/citation.cfm?id=2002736.2002869

[53] 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*, ser. WWW '13. New York, NY, USA: ACM, 2013, pp. 977–988. [Online]. Available: http://doi.acm.org/10.1145/2488388.2488473

[54] 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*, vol. 21, pp. 3–13, 2013.

[55] C. Unger, A.-C. N. Ngomo, and E. Cabrio, *6th Open Challenge on Question Answering over Linked Data (QALD-6)*. Cham: Springer International Publishing, 2016, pp. 171–177. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-46565-4_13

[56] E. Cabrio, J. Cojan, B. Magnini, F. Gandon, and A. Lavelli, "QAKiS @ QALD-2," in *2nd open challenge in Question Answering over Linked Data (QALD-2)*, Heraklion, Greece, May 2012. [Online]. Available: https://hal.inria.fr/hal-01171122

[57] R. Usbecka, M. Rödera, M. Hoffmanna, F. Conradsa, J. Huthmanna, A.-C. Ngonga-Ngomoa, C. Demmlera, and C. Ungerb, "Benchmarking question answering systems," *Semantic Web*, 2016.

APPENDIX A AMAL : ANSWERING FRENCH NATURAL LANGUAGE QUESTIONS USING DBPEDIA

Authors

Nikolay Radoev[1] <nikolay.radoev@polymtl.ca>
Mathieu Tremblay[1] <mathieu-4.tremblay@polymtl.ca>
Michel Gagnon[1] <michel.gagnon@polymtl.ca>
Amal Zouaq[2] <azouaq@uottawa.ca>
[1] Département de génie informatique et génie logiciel, Polytechnique Montréal
[2] School of Electrical Engineering and Computer Science, University of Ottawa

Presented at : ESWC 2017, Extended Semantic Web Conference, Portoroz, Slovenia
Published in : Communications in Computer and Information Science, vol 769

Abstract

While SPARQL is a powerful way of accessing linked data, using natural language is more intuitive for most users. A few question answering systems already exist for English, but none focus specifically on French. Our system allows a user to query the DBpedia knowledge by asking questions in French separated in specific types, which are automatically translated into SPARQL queries. To our knowledge, this is the first French-based question answering system in the QALD competition.

A.1 Introduction

A crucial aspect of making the Semantic Web relevant is providing typical Web users with an intuitive and powerful interface to access the growing amount of structured data. Among the datasets that are currently accessible to the public, there are general knowledge bases (KBs), such as DBpedia [30], which contains information extracted from Wikipedia, and many specialized knowledge bases that have curated domain-specific knowledge, such as Dailymed [31]. However, given their reliance on SPARQL [7], they are difficult to use for the average user. Moreover, querying these KBs without knowledge of their underlying structure is a very complex task. Developing an intuitive interface to allow natural language queries is a problem that has been explored in some of the previous QALD challenges [54].

QALD is a series of evaluation campaigns on question answering over linked data. Multiple systems have already been developed in the past years to solve this problem with up to 0.89 F-Score for English on Task 1 of the QALD competition [55] (Multilingual question answering over DBpedia). In this competition, most systems have been focused on English, primarily because of the amount of resources available in existing KBs [9] and its popularity worldwide. However,we have chosen to tackle questions asked in the French language using both the French and English chapters of DBpedia.

Because of French's specific language-dependent syntactic structures, it is not possible to directly reuse techniques that are available for the English language. For example, adjective placement in English precedes the noun while most of the time in French the adjective follows the noun it describes. Also, the syntactic rules to express verb tenses in French differ greatly from English: simple past in English may be expressed in French by a simple past form or a compound form (*talked* vs *parla* and *a parlé*), there is no gerund form in French (*is talking* vs *est en train de parler*), future tense is expressed by an auxiliary in English and by a suffix in French (*will talk* vs *parlera*), etc.

Interpreting a question given in a natural language is a well-known but unsolved problem [32]. In general, it requires the extraction of a semantic representation that is the result of a multiple-phase approach. The question must be processed to extract keywords and terms that may represent some entities available in different KBs. Then, those keywords and terms must be mapped to resources, classes, and properties in the KBs. This is a complex task given the fact that (i) those keywords and terms might not exist as resources in the KBs and (ii) natural language syntax creates ambiguity that cannot be resolved without proper context. Questions such as *Who made Titanic?* (We are looking for the producer of the 1997 movie) are a good example of an ambiguity between the ship and the film and a need to infer the property *http://dbpedia.org/ontology/producer* from the verb *made*.

Some previous works on the problem used controlled natural language (CNL) [35] approaches to restrict grammar and syntax rules of the input question. Such approaches have the merit of reducing ambiguity and increasing the accuracy of the proposed answers. However, we consider the rigidity of an imposed grammar to be awkward for an average user and we do not impose any constraints on the questions given as input. This paper is a more detailed version of the one initially submitted [2] to the QALD-7 challenge. During the Open Challenge on Question Answering over Linked Data, the AMAL (Ask Me In Any Language) system distinguished itself for being the only system that focuses specifically on French.

For readability purposes, we have abbreviated certain URIs by using the prefixes detailed in Table A.1. For example, *http://dbpedia.org/ontology/spouse* becomes *dbo:spouse*. The *dbo:*

prefix is used to represent classes (aka concepts) from the DBpedia ontology while the *dbr:* prefix is used to identify resources from this KB. The *yago:* prefix refers to entities defined in the Yago knowledge base.

A.2 Related Work

Previous work has already been done to answer natural language questions on a multilingual KB. We have focused our work on answering questions with the DBpedia [30] KB. This KB has been extracted from Wikipedia to represent general knowledge in multiple languages. Some systems also use Wikidata [36] to answer questions. The main difference between these two KBs is that DBpedia is automatically extracted from Wikipedia while Wikidata is manually created and supports the knowledge contained in Wikipedia.

QAKIS is a system that answers questions in English with the use of the French, English, Italian and German chapters of DBpedia [26]. It uses the WikiFramework repository (which contains relational patterns automatically extracted from Wikipedia) [56] to solve the problem of finding lexicalizations of properties from the DBpedia ontology. Finding lexicalizations is a common problem in question answering since different wordings can be used to ask the same question.

WDAqua-core0 is another system presented at the 2017 ESWC conference that accepts many languages: French, English, German and Italian, with English being used with both DBpedia and Wikidata, whereas only Wikidata is used for other languages. The system is integrated in the Qanary Ecosystem [41] and uses some of its other features, most notably the speech recognition module. Just like QAKIS, WDAqua-core0 focuses on simple questions by translating natural language queries to SPARQL queries. Multilingual queries do not implement custom rules but rely on generic ones.

Our system differs from these because it is able to process questions in French and answer over the knowledge contained in DBpedia. QAKIS only processes English queries and while WDAqua-core0 can process questions in French, only knowledge from Wikidata is used and not DBpedia as specified in the QALD challenge. As far as we know, AMAL is the only system that specifically targets the French language.

Table A.1 DBpedia prefixes

dbo	http://dbpedia.org/ontology/
dbr	http://dbpedia.org/resource/
yago	http://dbpedia.org/class/yago/

A.3 System Overview

AMAL (Ask Me in Any Language) was developed using a modular approach to separate application logic in different systems. Each subsystem can be developed, modified and improved independently. With our system, users can ask questions in French that are analyzed and answered with information found in the English DBpedia, as specified in the description of Task 1 of the QALD challenge, for which this system was created.

Our system is the first version of a work in progress. It focuses on *Simple Questions*, which we define as questions that concern only one single entity and a single property of this entity, such as *Qui est le père de Barack Obama?* (Who is the father of Barack Obama?), where 'Barack Obama' is the entity and 'father of' is the property. We are still working on handling more complex questions involving multiple entity/property relations.

Our approach consists of a multiple-step pipeline: question type identification, entity extraction, property identification and question answering through a SPARQL query builder. The first step determines the type and possible subtype of the question using a *Question Type Analyzer*. Our system currently supports the following types: *Boolean, Date, Number and Resource*. Boolean questions are questions that can only be answered by *TRUE* or *FALSE*. Date questions refer to specific dates in a standard *YYYY-MM-DD* format. Number questions are questions that have answers in a numeric literal that is a value of one single property and not derived by using arithmetic manipulation. Finally, Resource questions are questions with answers given under the form of a DBpedia URI such as *dbr:Barack_Obama*. Two subtypes may be added to some of those main types: *List* and *Aggregation*. *List* questions, which can be subtype of both *Resource* and *Date* questions, are questions whose answer contains several elements. *Aggregation* questions include (i) questions that require an ascending or descending order such as looking for the most or least of something and (ii) questions that require counting, most often the number of elements that satisfy one or more conditions. *Date, Number and Resource* questions are the ones that may have *Aggregation* as a subtype. Our approach to classify questions is explained in Section A.4.1.

Once the system knows the question type, the query is sent to specific *Question Solvers*. For instance, a question such as *Is Michelle the name of Barack Obama's wife?* will be sent to the boolean solver, and *When was Barack Obama born?* is handled by the date question solver. Aggregation Questions necessitate additional computation which is detailed in Section A.5.3.Every question solver makes use of one or more submodules that function as *extractors*. There are two main extractors: an *entity extractor* and a *property extractor*, as shown on Figure A.1, which are used to identify the entities and properties in a given question. Specific question solvers require specific property extractor heuristics (more details are given in Section A.4.3). Before the question is passed to its specific question solver, all question indicators, as seen in Tables A.5, A.6, A.7 are stripped from the question. In the case of Aggregation questions, the ordering indicators from Tables A.2, A.3 and A.4 are also removed.

The AMAL system was created specifically for the 2017 QALD Task 1 challenge, where all answers are required to be extracted from the English version of DBpedia. Given that we focus on French questions, we need a way to translate the entities and/or properties found by our system. The *Translator* module handles the translation from French to English using Google Translate API for most queries. However, in our experience, Google Translate does sometimes give a different translation term than the one used by DBpedia. In those cases, the system uses custom translations to obtain the correct terms for the final query. As an example, the term *portée* (which means *span* when used as an attribute of a bridge or a road) is translated as *scope* instead of *span* and thus requires a specific rule to be correctly used. This is the last step before constructing the final SPARQL query. The additional overhead of French to English translation was added in order to conform with the competition's rules.

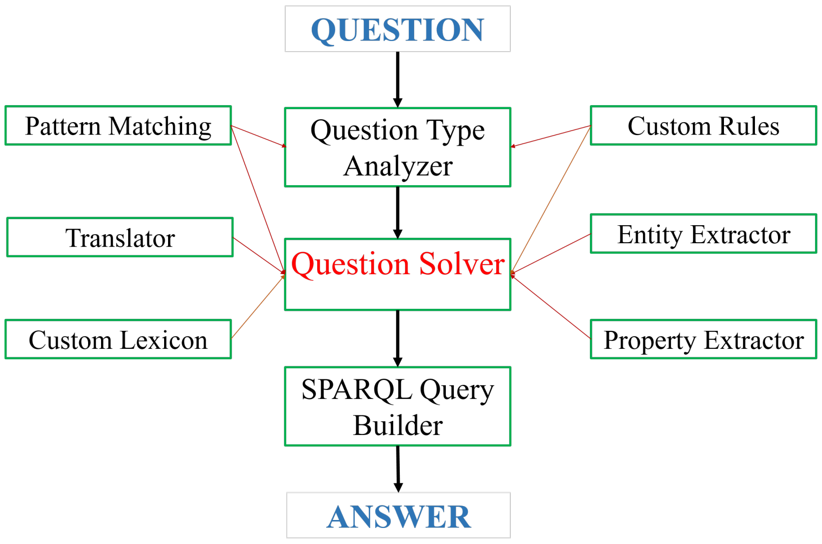


Figure A.1 System Overview.

A.4 System details

This section presents the details of the four main steps of our pipeline, which are question classification, entity extraction, property extraction and SPARQL query building. The description of each specific type of question solver is the subject of Section A.5.

A.4.1 Question classification

By analyzing the questions given in the 2016 and 2017 train datasets, we extracted various keywords and patterns that occur most often in a given question type. The extracted patterns rely both on lexical matching and positioning (start of a sentence or just their presence at any position in the question). For example, pronoun inversions such as *existe-t-il(elle)* (where the general form is *VERB-t-il(elle)*) appear only in close-ended (*Boolean*) questions.

Classification is made by matching the question string against the list of extracted patterns and if a match is found, the question gets assigned a specific type. The type matching is done by trying to match the questions to different types in the following order: Boolean, Number, Date and Resource. This order is due to the relative complexity of type matching with Boolean questions being easier to detect than Date questions, which might require additional work (see Section A.5.1). In the current version of the system, multiple types are not supported and the first detected type is considered the only one. The system does however support subtypes, as explained later.

If no match is found after going through all extracted keywords and patterns, we assign a default value of *Resource* question type. The same method is then applied for the subtypes with a few additional tweaks. For *List* questions, we look for question words or verbs that indicate multiple answers, such as *Qui sont les* (Who are the). For *Aggregation*, we try to determine whether the answers require being sorted in a descending or an ascending order. This classification is made using a list of keywords and patterns extracted from the training datasets. Some patterns require the expression to occur only in the beginning of the sentence or anywhere in the sentence as highlighted by the first row of each table.

To determine if a question is an aggregation question, we use the rules described in Tables A.2, A.3 and A.4. The first column of each table describes the patterns used to classify a question in that category and the second one contains examples of questions. We only check whether a sentence contains this expression to determine if it is an aggregation question.

In the same fashion, Tables A.5, A.6, A.7 shows our rules to classify a question as *Boolean*, *Date* or *Location*. For example, *Est-ce que les grenouilles sont des amphibiens?* is a boolean question starting with the expression *Est-ce que*, but *Quand est-ce que le Carey Price est né?* would not be considered as such, since it does not start with the *est-ce que* pattern, but rather in the middle of the question. According to Table A.5, it is classified as a *Date* question.

Table A.2 Lexical patterns for Aggregation count questions classification

Starts with	**Example**
Combien de	Combien de livres a écrit Isaac Asimov
Combien d'	Combien d'enfants a eu Barack Obama
Combien y	Combien y a-t-il de magasins Aldi

Table A.3 Lexical patterns for Aggregation Descending order questions classification

Contains	**Example**
le plus vieux/la plus vieille	Donnez-moi le plus vieux président des États-Unis
le plus âgé/la plus âgée	Donnez-moi l'enfant le plus âgé de Barack Obama
le premier/la première	Donnez-moi le premier enfant de Barack Obama

A.4.2 Entity extraction

Once our system has identified the type of the question, it extracts the entity from the sentence. For example, with *Quand est-ce que Carey Price est né?*, we extract *Carey Price* from the sentence. To do so, we use a syntactic parser to identify the noun groups and we then find the ones that correspond to an existing entity in DBpedia. Given that noun groups can contain many more elements than just the entity, such as adjectives or determinants, we start by taking the longest string in the question and generate all possible substrings by recursively tokenizing the output. All generated combinations are then ran against the DBpedia database to generate as many valid entities as possible. For example, *Who is the queen of England?* generates the following substrings 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. Having multiple possible entities can help increase the chance of obtaining more accurate answers. Note that we translated the example "Who is the queen of England" in English for readability purposes but this step is actually done on a French question.

Table A.4 Lexical patterns for Aggregation Ascending order question classification

Contains	**Example**
le plus jeune/la plus jeune	Qui est le plus jeune enfant de Barack Obama
le plus haut/la plus haute	Quelle est la plus haute montagne d'Australie
le plus long/la plus longue	Quel est le pont le plus long
le dernier/la dernière	Qui est le dernier enfant de Barack Obama
le plus grand/la plus grande	Quelle est la plus grande montagne d'Australie

Table A.5 Date question classification

Contains	Example
Quand	Quand a eu lieu l'Opération Overlord
Quelle est la date	Quelle est la date de naissance de Rachel Stevens
Donne moi la date	Donne moi la date de naissance de Rachel Stevens
À quelle date	À quelle date est née Rachel Stevens
Quelle est l'année/le mois	Quel est l'année de naissance de Rachel Stevens

Table A.6 Boolean question classification

Starts with	Contains	Example
Est-ce que		Est-ce que le titanic est un bateau
Ont/Sont		Sont les grenouilles un type d'amphibien
Peut-on		Peut-on trouver des frèsques en Crète
	est-il/est-elle/ sont-ils/sont-elles	Les grenouilles sont-elles des amphibiens

Table A.7 Location question classification

Contains	Example
Où	Où est situé le Palais de Westminster
Dans quel pays/quelle ville	Dans quelle ville est la Tour Eiffel
Lieu	Quel est le lieu de naissance de Barack Obama
Endroit	À quel endroit se trouve la Tour Eiffel

Since the question is in French, the *sameAs* link is used to find the corresponding URI in the English version of DBpedia, considering that only URIs from that DBpedia version are considered valid answers for the task 1 of QALD.

If no such link is available, we use the previously described *Translator module* to generate a possible English entity. For the question: *Donne-moi les ingrédients d'un biscuit aux brisures de chocolat* (Give me the ingredients of a chocolat chip cookie), there is no French page *dbr:Cookie_aux_brisures_de_chocolat*, but using Google translate we are able to find *dbr:Chocolate_chip_cookie*, which is a valid DBpedia entity. Entities extracted this way however have a lower chance of being selected as the right entity given the uncertainty created by the translation.

To improve our results, we add variations of the identified nouns by applying modifications such as plurality indicators and capitalization. Every modification reduces the chance of the entity to be chosen as the main one. For example, the entity *queen* can also be manipulated in order to extract the entity *Queens*. However, since *Queens* requires 2 modifications (pluralization and capitalization), it is less likely to be selected than *queen*.

Once all the possible entities have been extracted, we use multiple criteria, such as the length of the entity's string, the number of modifications needed to extract it from DBpedia and whether we had to translate it, to determine their likelihood of being the right entity. The formula we currently use to combine these factors is the following, where e is the entity string:

$$length(e) - T \times \frac{length(e)}{2} + 3 \times U \times nsp(e)$$

where

$nsp(e)$ is the number of spaces in e
$T = 1$ if e has been translated, 0 otherwise
$U = 1$ if e has not been capitalized, 0 otherwise

According to this formula, the score is penalized by half the length of the entity string if we used a translation. Also, the more words it contains (whose value is obtained by counting the number of spaces) the higher the score will be, if it has not been capitalized. For example, *Dans quelle ville se trouve le Palais de Westminster?* we can extract *Palais de Westminster* and *Westminster*. However, *Westminster* by itself has a score of 11 while *Palais de Westminster* has a score of 27. Therefore, we consilder it more likely that the correct entity is *Palais de Westminster*. For the question *Quels sont les ingrédients d'un biscuit au* *brisures de chocolat* we can extract *biscuit* and *chocolat*, with scores of 7 and 8. By translating the question to *What are the ingredients of a chocolate chip cookie?*, we can extract *chocolate chip cookie* with a score of 10.5, *chocolate chip* with a score of 10, *chocolate* with a score of 4.5 and *cookie* with a score of 3.

A.4.3 Property Extraction

Once the entity is extracted from the sentence, the property is found by removing the entity from the question and analyzing the remaining tokens to find all nouns or verbs. To find properties in the question, we use the DBpedia ontology and the RDF description of the selected entity in DBpedia. A property is found when we find a label that matches an URI in DBpedia and that is also present in the selected entity's description. As an example, in the question *Qui a créé Batman?* (Who created Batman?), the selected entity is *Batman* and we test if *dbr:Batman* contains a property that is also present in the phrase. In this specific case, the verb *created*, mapped to *dbo:creator* in our custom lexicon, does indeed appear in the description of *dbr:Batman*.

To facilitate the identification of the predicates to be used in the SPARQL query that corresponds to the question sentence, we built a lexicon by mapping a list of common DBpedia properties to French expressions, in addition to manually adding bindings that were not present in the French DBpedia. The creation of such a lexicon was necessary given the lack of similar resources in the French DBpedia or other external solutions that focus on French. For example, *dbo:spouse* and *http://fr.dbpedia.org/property/conjoint* are both mapped to *conjoint* (spouse), *épouse* (wife), *femme* (wife) and *mari* (husband). With such bindings, the system is able to take into account the various ways of expressing the property in French: *Qui est la conjointe/l'épouse/la femme de Barack Obama?* (Who is the spouse/wife/ wife of Barack Obama?)

A.4.4 SPARQL Query Builder

The last step is building and sending SPARQL queries to DBpedia. The system uses the standard DBpedia endpoint: *http://dbpedia.org*. For now, AMAL supports basic SPARQL queries such as *ASK* and *SELECT*. In addition, *COUNT* and *ORDER BY* queries are supported for *Aggregation* questions. Queries exist as basic templates in which specific values are injected in the form of RDF triples. For example, in *Qui est l'épouse de Barack Obama?*, once we have extracted *dbr:Barack_Obama* and *dbo:spouse*, our system will build the following query: *SELECT DISTINCT ?uri WHERE {dbr:Barack_Obama dbo:spouse ?uri}* and send it to DBpedia, which will return the answer. The library used is available on the following

GitHub link: `https://www.github.com/Mathos1432/dotNetSPARQL`.

A.5 Simple Question Analysis

As previously mentioned, in our current implementation, we deal mostly with simple questions, limited to at most one entity and one property. In the case of boolean questions, we also process *Entity - Entity* relations, for example *Est-ce que la femme de Barack Obama est Michelle Obama?* (Is Michelle Obama the wife of Barack Obama?).

We start by identifying the type of the question, as explained in Section A.4.1. Once this is done, we extract properties and entities. From there, we proceed to answering the question. To do so, we start with the most likely entity that was extracted, using the likelihood formula defined in Section A.4.2, and execute SPARQL queries to determine whether an element in the rest of the question is a property of the entity. If the property exists, the object of the property is usually our answer.

When the entity does not have any of the extracted properties, we attempt disambiguation on the entity by following the Wikipage disambiguation link, if such a link is available. This link provides a way to find other entities that can be a possible match. For example, in *Who is the producer of Titanic?*, our system would extract the entity *Titanic*. Using the disambiguates link in *Titanic*, we can find *Titanic(film)* ,*Titanic(album)* and other possible entities. The property *dbo:producer* can be found in the *Titanic(film)* resource description, thus allowing the system to select the correct entity.

A.5.1 Date and Location Questions

Some queries require that our system find dates or locations for events such as the birth of a person or the place of a war. For most of these questions, the label for the property is directly present in the query string. For example, in the question *Quelle est la date de naissance de Rachel Stevens* (What is Rachel Stevens birth date?), the property is given explicitly in the question (birth date). There are also questions where this is not the case and the property must be inferred, for example in *Quand Rachel Stevens est-elle née?* (When was Rachel Stevens born?). Our solution for this problem is using a lexicon that maps commonly used words or expressions such as *"année de naissance"* (year of birth) or *"est né(e)"* (was born) to the *dbo:dateOfBirth* property. Other types of dates, such as death date and end of career date, can also easily be handled in the same way.

Date questions require some additional work when comparison and ordering are involved and this is explained with more details in Section A.5.3. Tables A.8 and A.9 show the lexicons we built to handle date and location questions.

Location Questions

An additional step is done for location questions. Once our system found answers, it filters them to make sure we only consider the relevant ones. For example, the question *Dans quelle ville est situé le Palais de Westminster?* (In which city is the Palace of Westminster located?), if we query *dbr:Palace_of_Westminster* for its *dbo:location*, we will get *dbr:City_of_Westminster*, *dbr:Greater_London*, *dbr:United_Kingdom* and *dbr:England*. Since we are only looking for the city, we need to filter the answers to those that are of the right type using the property *rdfs:type*, which in this case is the type *yago:City108524735*, thus eliminating all entities except *dbr:City_of_Westminster*.

A.5.2 Boolean Questions

The first step in the processing of boolean questions is to determine whether the question involves other entities, as in *Is Michelle Obama the wife of Barack Obama?* (note that in this example *Barack Obama* is identified as the concerned entity). If it is the case, the program verifies whether a relation exists between the entities and if it is the right type (in this case, a relation of type *spouse*). When the question does not involve other entities, it is about whether a property exists for a specific entity. We can consider the example *Existe-t-il un jeu vidéo appelé Battle Chess?* (Is there a video game called Battle Chess?). In this case, we simply need to find all entities with a label *Battle Chess* and find out if one of them is a video game. Here again, we rely on the presence of *Existe* and look for a *rdf:type* relation using a mapping available in our lexicon. The last type of question supported by our system is whether an entity is of a specific type. For instance, *Are tree frogs a type of Amphibian?*.

Table A.8 Lexicon of date properties

Keywords	**Possible properties**
mort	death date
année de naissance	birth year
naissance, né	birth date
dissolue	dissolution date
commencé	active years start date, active years start year
terminaison	active years end date, active years end year
terminé	completion date
fondé	founding year
indépendance	founding date

Table A.9 Lexicon of location properties

Keywords	**Possible properties**
mort	death place
naissance	birth place
commence	route start, source country
enterré	resting place, place of burial
vin	wine produced
fondé	founding year
vit	residence

In this case, we extract the entity from the sentence (tree frog) and verify if it has the right type (Amphibian).

A.5.3 Aggregation Questions

Aggregation questions are divided into two different categories. The first category contains questions that are looking for a numeric answer, such as *Combien de langues sont parlées en Colombie?*(How many languages are spoken in Colombia?). This type of question is characterized by the presence of numeric indicators, such as: *How many, All, Number of,* etc. In order to determine the entity subject to enumeration, all possible entities and properties are extracted from the question, with entities/properties immediately after the numeric indicator being considered as more likely to be the main subject. In the case of specific numeric indicators, additional work must be done since the counting is done based on the sentence's verb and not a particular entity. For example, *Combien de fois s'est mariée Jane Fonda?* (How often did Jane Fonda get married?), the enumeration is done on the number of spouses (*dbo:spouse*) and not Jane Fonda.

The second category contains questions that involve ordering result sets in a specific order. Key indicators of this question type are superlative adjectives present in the query. Expressions such as *le plus grand* (the most), *le plus gros* (the biggest), etc., are used to decide the sorting order.The queries are parsed before the *Question Solver* module and if a superlative adjective is found, it is compared to prebuilt custom lists to determine if an ascending or descending order is required. Adjectives such as *le plus grand, le plus gros (biggest, largest)* are marked as *Descending* indicators while *le plus petit, le plus facile (smallest, easiest*) are *Ascending* ones. The only exception to this rule are queries about dates or age (a type of implicit date) given that asking for *le plus jeune (the youngest)* of something is equivalent to looking for the highest date when ordering potential results. As an example for the following two dates : *1970-24-04* and *1969-23-11*, the first date would be considered as *youngest* even if *1970* is bigger than *1969*.

Once the ordering indicator is found and classified, it is removed from the query. This allows for an easier analysis of the question and reduces any possible ambiguity. As an example, in *Quelle est la plus haute montagne d'Australie?* (What is the highest mountain in Australia?), our system identifies *la plus haute* (the highest) as a *Descending* order indicator. After the question indicator *Quelle est (What is)* and the order indicator (the highest) are removed from the query, the two entities *dbo:Mountain* and *dbo:Australia* can be extracted. The system can then try to find a relationship between them, specifically looking for all entities of type *dbo:Mountain* that have a property with a value of *dbo:Australia* and then sorting them by the property *dbo:elevation* in a *Descending* order.

The relation between adjectives and entities can be somewhat ambiguous in most KBs. In fact, the adjective *la plus haute*(the tallest) can be applied to both a *Person* and *Object* (mountain for example). However, in the first case, the KBs property would be *height*, while in the second, we need to be looking for *elevation*. Our system offers a limited context awareness by using manually created *(adjective:entity),property* pairs. In the previously given example, the system is looking for *dbo:elevation*, since the pair *{("la plus haute": dbo:NaturalPlace), dbo:elevation}* and *{("la plus haute": dbo:Person), dbo:height}* exist in the predefined custom pairs and *dbo:mountain* is of type *dbo:NaturalPlace*.

The system does not currently support superlative ordering with multiple answers. Queries asking for the *N most* or *least* Entities, such as *Donnez-moi les 5 premiers présidents des États-Unis*(Give me the five first presidents of the United-States) will currently only return the first result.

A.5.4 List Questions

List questions are those that can but do not necessarily require an answer containing multiple entries. They can be seen as a subset of aggregation questions, but without additional sorting or counting over the result set. For example, *Who are the founders of DBpedia?* is a list question that returns multiple resources. Given that the type of results can vary, list questions are first analyzed based on their type.

Some types, such as *Boolean* and *Number*, are guaranteed to have only a single possible answer and all additional answers beyond the first ones that are returned by the SPARQL query are ignored. *Resource* and *Dates* types can return multiple possible answers, so all results returned by the SPARQL queries are added to the final result set. In the current version of the system, there is no imposed limit to the number of elements returned given the inherent difficulty of predicting the expected size of the answer for some questions.

Some questions can ask for a single entity but actually have multiple answers, such as *Qui était le successeur de John F. Kennedy?* (Who **was the successor** of John F. Kennedy?), which has 3 different answers in DBpedia (Lyndon B. Johnson, Benjamin A. Smith III and Tip O'Neill) despite us looking for a single entity.For this reason, our system does not restrict the result set to a single answer. An exception is made for *Aggregation* questions, since the results of a *Aggregation-Counting* question is processed as a *Number* response using a single literal as an answer. For example, *Combien de livres a écrit Isaac Asimov?* (How many books did Isaac Asimov write) is supported. But we do not currently support *Aggregation-Ordering* queries that return more than a single result, such as *Donnez-moi les 5 premiers présidents des États-Unis* (Give me the five first presidents of the United-States).

A.6 Evaluation

To evaluate the efficiency of our system, we used the train and test datasets for Task 1 of QALD-7. Table A.10 shows, for each dataset, the number of questions and precision, recall and F-Score. The train dataset was evaluated using GERBIL-QA [57], while the test set was manually compared to the provided answers provided for the competition. Any results provided by our system that did not exactly match the provided answers (containing more or less elements for multiple element answers) were considered as failed.

A.7 Future work

For further improvement of the AMAL system, we have considered many possible approaches. First, we would like to be able to improve our disambiguation algorithm. In its current version, AMAL looks through all possible disambiguation resources, whenever available, and attempts to find the property it extracted in the description of each resource. However, this can quickly fail when many entities have the property, especially when looking for more generic properties such as a birth date or location.

To improve our entity extraction, we plan to look at whether possible entities contain prop-

Table A.10 Results on the QALD 7 datasets

Dataset	**Number of questions**	**Precision**	**Recall**	**F-Score**
QALD-7 train dataset	214	0.9708	0.6967	0.8112
QALD-7 test dataset	50	0.7046	0.88	0.7825

erties from the question. This would allow us to be more precise in our extraction and limit the amount of disambiguation we have to do, or at least help with it. For example, for the sentences *What is Battleship's budget?* and *Which platforms support Battleship?*, we can extract both the Battleship movie and the Battleship video game. However, in the first case, with the budget property we rank the movie with a higher likeliness of being the right entity. In the case of the video game, we can use the platform property to determine that we are looking for a video game rather than a movie.

There is also the possibility to add more lexicalization of DBpedia properties to our system as it only covers the ones we thought were most important for the challenge. To do so, an automated approach would help improve our results while reducing the amount of time it takes to build such a dataset. The WikiFramework could be used, or the lexicalization dataset from DBpedia-spotlight.

We also plan to process more complex queries involving more than one entity and property, starting by processing queries with multiple properties. This will be achieved by chaining requests until no properties are left to answer in the sentence. For example, *Quelle est la date de naissance du créateur de Dracula?* (What is the birthdate of the creator of Dracula), we can extract *dbo:birthDate* and *dbo:creator* from the sentence. Once this is done, we use our question solvers to link the properties in reverse order. In this case, we query DBpedia for Dracula's creator and then for this answer's birth date. This approach does not cover more complex cases but it is a realistic improvement goal.

Given that our system was built by using separate modules, we hope to be able to extend the custom logic modules that are directly affected by the French languages particularities allowing for an implementation of the system in other languages, mainly English, thus rendering it a truly multilingual Question-Answering system.

The use of multiple KBs to answer questions is another improvement that could be made in the future, which would allow the use of a greater body of knowledge to answer questions. The use of Wikidata would improve our ability to answer general knowledge questions.

A.8 Conclusion

The AMAL system is one of systems in the QALD competition that supports the French language. However, unlike other systems, AMAL focuses specifically on French questions and is thus able to obtain better and more accurate results. In this paper we have outlined the main functionality of the system and its modular, custom rule driven approach to Question Answering over Linked Data. Possible improvements are also outlined as both an analysis of the system current limitations and a source of any future work done on our system.

In its current version, DBpedia is offered in more than 30 languages including English, French, Spanish and many others. While the English version is by far the most complete and rich knowledge base, other versions can offer a substantial amount of information and knowledge. We hope that our work on a system that focuses on a language other than English can benefit other works that focus on more than the single dominating language that is English in the present time.