International Journal of Computer Applications (0975 – 8887)
Volume 125 – No.6, September 2015

AR2SPARQL: An Arabic Natural Language Interface for the Semantic Web

Iyad AlAgha
Faculty of Information Technology, The Islamic University of Gaza
Gaza Strip, Palestine

Alaa Abu-Taha
Faculty of Information Technology, The Islamic University of Gaza
Gaza Strip, Palestine

ABSTRACT
With the growing interest in supporting the Arabic language on the Semantic Web (SW), there is an emerging need to enable Arab users to query ontologies and RDF stores without being challenged with the formal logic of the SW. In the domain of English language, several efforts provided Natural Language (NL) interfaces to enable ordinary users to query ontologies using NL queries. However, none of these efforts were designed to support the Arabic language which has different morphological and semantic structures.

As a step towards supporting Arabic Question Answering (QA) on the SW, this work presents AR2SPARQL, a NL interface that takes questions expressed in Arabic and returns answers drawn from an ontology-based knowledge base. The core of AR2SPARQL is the approach we propose to translate Arabic questions into triples which are matched against RDF data to retrieve an answer. The system uses both linguistic and semantic features to resolve ambiguity when matching words to the ontology content. To overcome the limited support for Arabic Natural Language Processing (NLP), the system does not make intensive use of sophisticated linguistic methods. Instead, it relies more on the knowledge defined in the ontology and the grammar rules we define to capture the structures of Arabic questions and to construct an adequate RDF representations. AR2SPARQL has been tested with two different datasets and results have shown that it achieves a good retrieval performance in terms of precision and recall.

Keywords
Semantic Web, SPARQL, Natural Language Interface, Arabic Question Answering

1. INTRODUCTION
Semantic Web (SW) and Linked Data technologies have been widely employed by a considerable number of applications. Consequently, a huge amount of data is constantly being made available on the Web in RDF and OWL format. However, the logic-based infrastructure of the SW makes it difficult for common users to interact with applications by commanding formal logic. In an attempt to bridge the gap between average users and the SW, several approaches have proposed friendly Natural Language (NL) interfaces to enable for querying ontologies and RDF data backends. These approaches aim to hide the complexities of RDF data and query languages, e.g. SPARQL, by getting NL queries as an input, and transforming them into formal queries.

Although NL interfaces to the SW have gained a considerable attention in the past few years, existing approaches are mostly tailored to work with English and Latin-based text. The advancements in NLP of English and Latin based languages has contributed significantly to the development of NL interfaces. However, there has not been a similar progress to support Arabic NL interfaces to the SW. This can be explained by the complexities of linguistic processing of Arabic text: Arabic language has more complex morphological, grammatical and semantic structures that make existing NLP techniques used for the English text inadequate for the Arabic text. The lack of resources, tools and software development environments that process the Arabic script is a major reason for the limited support for Arabic language on the SW [6].

In the past few years, the field of Arabic NLP has gained a considerable attention with the emergence of Arabic NLP tools and free Arabic corpora. This has fostered the development of applications that support Arabic language in a variety of fields including Question Answering (QA), information extraction and search engines. In the past few years, the development of Arabic ontologies and ontology-based representations of Arabic resources has gained a considerable attention. In parallel with these efforts, little attention was given to enable Arab users to query this content through NL interfaces. This will certainly reflect a qualitative shift in the handling and treatment of the Arabic knowledge on the SW. It will also expand the influence of ontologies and the SW among the Arab community.

For the purpose of supporting Arabic QA on the SW, this work presents AR2SPARQL, a NL interface that takes queries expressed in Arabic language and returns answers drawn from an ontology-based knowledge base. In the context of this work, we define a Natural Language (NL) interface as a system that accepts questions formulated in natural language and returns answers on the basis of a given knowledge base. It should be emphasized that a NL interface goes strictly beyond the capabilities of keyword-based retrieval systems, which are not able to retrieve precise answers to questions but only to retrieve a set of relevant documents given a keyword-based query. The major features of AR2SPARQL include:

Firstly, AR2SPARQL translates Arabic NL queries to SPARQL which is the W3C standard query language for the SW. It uses Arabic NLP techniques to effectively maps query terms to ontological entities <classes, properties and instances>. It then utilizes a set of grammar rules as well as the knowledge in the ontology to construct a SPARQL query by linking the ontological entities. AR2SPARQL can handle not only simple queries, but also complex ones such as those consisting of multiple sentences linked by conjunctions, i.e. "و, أو" or interrogative pronouns, i.e. "الذي, التي".

Secondly, AR2SPARQL is designed to be ontology-portable and no assumption is made about any specific domain of knowledge. It can be interfaced to any ontology as long as the ontology terms are represented in Arabic or their Arabic translations are provided within the ontology.

Thirdly, the proposed approach for interpreting NL queries does not make extensive use of NLP techniques such as text

parsing or morphological analysis. It employs only a reduced set of NLP operators, such as stemming and part of speech tagging. Instead, it highly depends on the quality and choice of vocabulary of the ontology as well as the rules we define to interpret the NL query to SPARQL. This decision stems from the fact that linguistic analysis is time-consuming, error-prone and difficult to manipulate [26]. In particular, linguistic analysis of Arabic sentences remains much poorer than English, and the results can very often be misleading [14].

2. A SAMPLE ONTOLOGY

Before explaining the design of AR2SPARQL, it is important to briefly introduce the sample ontology we developed for illustration purposes. The discussion throughout this article is based on this ontology which covers a subset of diseases. Figure 1 depicts an excerpt of the ontology showing some ontology classes (e.g. Treatment, Disease, Symptom, Organ, Diagnosis) as well as the relations between them, i.e. the object properties.

The interpretation from Arabic script into SPARQL requires matching the Arabic query with the ontology in order to extract entities that best describe the query words. Ontology entities refer to classes, properties, instances/individuals or data-type property values such as string literals. To support mapping Arabic queries, ontology entities should have Arabic names. AR2SPARQL assumes that all entities are named using the rdfs:label property, and thus it retrieves the Arabic name of any ontology entity by extracting the value of its rdfs:label property (rdfs:label property is not shown in Figure 1 for simplicity). An ontology entity can have multiples values of the rdfs:label property to indicate synonyms or alternative names. When matching the query terms with the ontology entities, all values of the label property are examined to ensure the best match.

We emphasize that AR2SPARQL can be easily configured to use any ontology as long as the Arabic translation of its content is supplied within the ontology through rdfs:label.

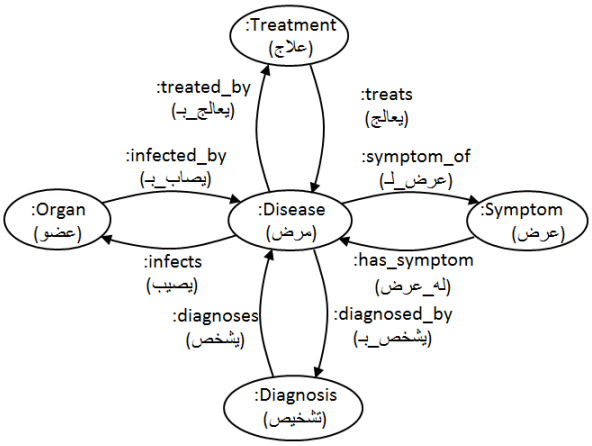

Fig 1: An excerpt of the disease ontology

3. AR2SPARQL ARCHITECTURE

Figure 2 depicts the architecture of the AR2SPARQL system: It takes a NL query as an input and translates it to a SPARQL query, which is then executed over the RDF knowledge base. When an ontology is selected as the underlying knowledge base, the Dictionary Builder automatically extracts ontological entities out of the ontology to build the Ontological Dictionary, which works as a lexicon.

The system process is briefly explained as follows: When the user inputs a query expressed in Arabic, the query is handled by the Interpretation Module, which is the core processing component of AR2SPARQL, and is in charge of interpreting the Arabic query into SPARQL. The query first undergoes a set of NLP techniques. The Ontological Dictionary is then searched for ontology entities that best match with each word in the NL query. Matched ontology entities are used by the SPARQL Generator to construct the SPARQL query. The SPARQL Generator exploits the knowledge in the ontology as well as the grammar rules we define to build meaningful RDF triple patterns by joining ontology entities together. Finally, the SELECT clause and query modifiers, e.g. "UNION" and "FILTER" are generated. The resultant query is executed over the knowledge base to retrieve answers. In the following sections, the components of the system as well as the underlying Arabic-to-SPARQL interpretation process are explained in detail.

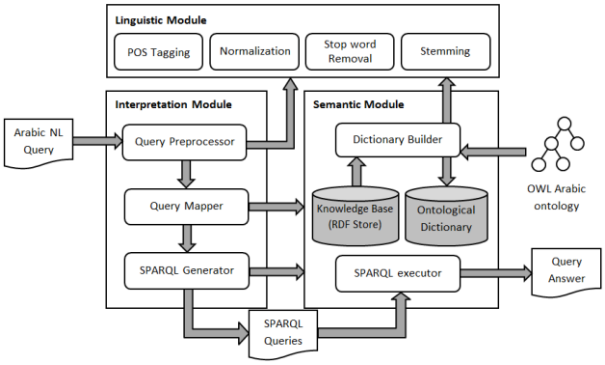

Fig 2: The Architecture of AR2SPARQL System

4. THE SEMANTIC MODULE

The Semantic Module is responsible for maintaining the ontology and the associated data. The ontology is represented in terms of OWL. The data is represented as instances of the corresponding ontology and is stored separately in a RDF database store. This separation between the ontology and the RDF data has many advantages such as better query performance, improved system scalability and ontology re-use[28]. When the system is first configured to use an ontology and its instance data, we operate an inference engine, i.e. reasoner, to infer additional facts and expressive features. This enables the declaration of derived classes or the declaration of further property characteristics (e.g. transitivity and symmetry of properties) which can improve the QA capabilities. The Semantic Module was implemented in Java by using the Jena API[1].

5. MAPPING QUERTY TO ONTOLOGY ENTITIES

A SPARQL query typically consists of a set of RDF triple patterns. A triple pattern is like an RDF triple except that each of the subject, predicate and object may be variables. The first step of transforming a user query to SPARQL is to identify the ontology entities that best match with the user's terms. For example, given the schema in Figure 1, the NL query: " ما الأمراض التي من أعراضها فقر الدم؟" matches with the following ontology entities: ":Disease", ":has_symptom" and ":Anemia". After identifying the ontology entities, a SPARQL query is constructed by combining the discovered ontology entities to formulate RDF triples.

[1]https://jena.apache.org

To map Arabic words to the ontology entities in a proper manner, some challenges should be tackled: These challenges are briefly discussed in what follows:

- The variety of text formats and writing styles: words with similar meanings can be written in different formats which have the same root (e.g. the words "علاج"and "يعالج_بـ"). In addition, Arabic letters can be written in different styles such as "أ" or "ا" or "ه" or "ة".

- Matching phrases in the query: Some entities in the ontology consist of a phrase rather than a single word. Some of the words in the phrase have different corresponding entities if they appear separately (e.g. "الدم"and " ارتفاع ضغط الدم"). It is necessary to map words/phrases in the query with the correct entities in the ontology as possible.

- Entity ambiguity: a single word can match with more than one ontology entity. For example, the word "علاج" can map to the ontology class :Treatment (علاج), the object property :treats (يعالج) and the inverse property :treated_by (يعالج_بـ) since all words share the same stem. The mapping process should decide the correct matching.

- The gap between the user's terminology and the ontological terminology: a user query may contain synonyms of but not the exact terms used in the ontology. For example, the user term "داء" does not match with the word "مرض" even though they share the same meaning. Ontology mapping should capture synonyms of the same word.

To address the above challenges, the following components were designed:

- The Ontological Dictionary: To enable fast access and matching of query words, all ontology entities including classes, properties and instances are extracted, linguistically-processed and stored in the Ontological Dictionary. Given a word from the user query, the Ontological Dictionary should output a set of ontology entities to act as descriptors for the query words. The preprocessing of entities in the ontology aims to apply some standard NLP processes on the Arabic labels to enable for better matching with the user's vocabulary. These processes include: 1) Orthographic normalization (e.g. replacing "أ"with "ا"and "ه"with "ة"). Stanford Arabic Word Segmenter[2] is used to apply normalization to the Arabic words. 2) Removal of stopwords and special characters such as "_" which often occurs in ontology text. 3) Part of speech tagging: Stanford Arabic POS[3] is used for this purpose. Part of Speech tagging is necessary to identify verbs, which often represent predicates in RDF triples, and nouns, which often map to ontology classes and instances. 4) Word Stemming by using the Arabic stemmer proposed by Khoja [25]. Stemming aims to make the Arabic words comparable regardless of the different formats.

To mitigate the gap between different terminologies, the ontology was manually populated with the synonyms of entities' names as possible. The rdfs:label property was used to assign synonyms to each entity in the ontology. Existing efforts working on English text often try to expand the system's terminology by using lexical databases such as the WordNet. Regarding Arabic, few efforts explored the construction of controlled vocabularies for Arabic language such as the Arabic WordNet [10]. However, as of the time of writing this article, we are not aware of any lexical database for the Arabic language which can be programmatically used through an open source API.

- The Query Mapper: The Query Mapper handles the process of mapping the user query to the ontology: When a user query is entered, it is tokenized, normalized and stemmed using the same procedure applied on the ontology content. To help map phrases in the query, all possible n-grams (4-grams, trigrams, bigrams and unigrams) are generated from the user query (n is initially set to 4, but can be reconfigured easily depending on the max length of the ontology entities). Then, n-grams are matched with the content of the Ontological Dictionary starting from the highest n-grams. The assumption here is that longer phrases will represent more specific descriptors than shorter ones.

6. THE SPARQL GENERATOR

The SPARQL Generator is the backbone of the AR2SPARQL system, and is responsible for generating the SPARQL query by linking the ontology entities recognized by the Query Mapper. It links the ontology entities captured from the mapping process in order to create RDF triples. These triples are then aggregated to generate a complete SPARQL query that, when executed, can retrieve the intended answers from the knowledge base. In what follows, we begin by giving a brief overview of the most important concepts underlying SPARQL queries. Thereafter, we present our approach to translate Arabic natural query into SPARQL.

6.1 Translating Arabic NL Queries into SPARQL

A SPARQL query, in its basic format, consists of two parts: the SELECT clause identifies the variables to appear in the query results, and the WHERE clause provides the basic graph pattern to match against the data graph. The WHERE clause consists of one or more triple patterns $< s\ p\ o >$ where s, p and o denote the subject, predicate and object respectively. In SPARQL queries, the subject, predicate and object can be variables, resources (written as URIs) or literal values. Given the SPARQL query: SELECT ?person where ?person <foaf:name> "Ahmed": the subject is the variable denoted by ?person, the predicate is the resource denoted by the URI: foaf:name, and the object is the literal value "Ahmed".

Let C be the set of all classes, P the set of all properties, I the set of all instances and L the set of all literals contained in the target knowledge base of the SPARQL queries at hand. We define the translation function $\rho: E^* \rightarrow < s\ p\ o >^*$as a function that maps an ontology entity E, or a sequence of entities, to one or more RDF triple pattern(s). For the translation function ρ, the input is the sequence of ontology entities recognized by the Query Mapper, and the output is a set of RDF triple patterns. Formally, the goal is to devise the extension of ρ to any ontology entity, or combination of entities, expressed in Arabic. We adopt a rule-based approach to achieve this goal as follows:

Rule 1: $if\ x \in (P \cup I \cup L)\ then\ \rho(x)\ \Rightarrow\ x$

Rule 2: i$f\ x \in C\ then\ \rho(x)\ \Rightarrow ?var \wedge (?var\ rdf{:}type\ x)$

The above two rules define how the atomic types (i.e. classes, instance, properties and literal values) are represented in the SPARQL query. Rule 1 indicates that the properties, instances and literal values remain unchanged in the generated SPARQL body. Rule 2 indicates that an ontology class entity is represented as a variable, ?var, that is of type x. The variable name ?var is randomly-generated.

[2]http://nlp.stanford.edu/software/segmenter.shtml
[3]http://nlp.stanford.edu/software/tagger.shtml

The procedure of constructing a SPARQL query from an Arabic NL query is explained in what follows. This procedure is illustrated with a running example the shows the translation of the query " ما الأمراض التي تصيب الكبد؟" based on the schema shown in Figure 1:

Step 1: The query text is mapped to the ontology content. The output of this step is a sequence of ontology entities that correspond to the query words. Entities are ordered according to the occurrence of their corresponding words in the query. The output of mapping the above query is the sequence: <:Disease (class), :infects (object_property), :Liver (instance)>.

Step 2: The sequence of ontology entities are scanned for a complete triple pattern. A complete triple pattern <s p o> should fulfill the following conditions:

- It is a sequence of ontology terms that map to a subject, a predicate and an object in sequence.
- A subject can be either a class or an instance.
- An object can be a class, an instance or a literal value.

A predicate can by either an object property or a data type property. The subject and the object should belong to the domain and the range of the predicate respectively. If a complete triple pattern is captured based on the above conditions, the interpretation of the NL query is straightforward: Rules 1 and 2 are applied according to the type of each ontology entity, and results are linked together to form one or more triple patterns. The generated triple patterns will formulate the WHERE clause of the SPARQL query. The translation function ρ of a complete triple pattern <s p o> can be expressed as follows:

Rule 3: $\rho(s,p,o) \Rightarrow \ \rho(s) \wedge \rho(p) \wedge \rho(o)$

$where\ s \in (C \cup I), p \in P, o \in (C \cup I \cup L), s \in Domain\ of\ P, o \in Range\ of\ P$

Referring to our running example, A subject (:Disease), a predicate (:infects) and an object (:Liver) appear in sequence. The class :Disease and the instance :Liver both fulfill the condition that they belong to the domain and range of the property :infects respectively. Thus, a complete triple pattern is captured. Rule 1,2 and 3 are applied as follows:

$\rho(:Disease, :infects, :Liver)$
$=>\rho(:Disease) \wedge \rho(:infects) \wedge \rho(:Liver)$

=> ?var ∧ :infects ∧ :Liver ∧ ?var rdf:type :Disease

=> ?var :infects :Liver . ?var rdf:type :Disease

Note that the above output, which will constitute the WHERE clause of the generated SPARQL query, is the composite of two triple patterns: the first indicates that the variable ?var relates to the instance :Liver through the predicate :infects, while the latter defines the type of the variable ?var.

Step 3: The SELECT clause is constructed by choosing variables that should appear in the query results from those included in the WHERE clause. This process is done as follows: 1) find the question words such as "ما", "من","أين" or command words such as "أذكر", 2) take the nouns that directly follow the question words as targets. 3) From the triples generated in Step 3, take variables that correspond to the target nouns. These variables will be part of the SELECT clause. Detailed rules vary for different question/command words: for example, quantity questions starting with "كم عدد" is interpreted into something like "SELECT COUNT(DISTINCT ?x)". For Yes/No questions, e.g. questions starting with the question word "هل", the ASK form is used to test whether or not a query pattern has a solution. For example, the NL query "هل يصيب السكر الكبد؟" is interpreted into something like "ASK WHERE {:Diabetes :infects :Liver}" which will return either Yes or No depending on whether or not a solution exists.

Referring to the query " ما الأمراض التي تصيب الكبد؟", the noun "الأمراض" is the target that should appear in the query result. The variable ?var from the above WHERE clause corresponds to the target noun. Thereby, the SPARQL query after generating the SELECT clause becomes:

SELECT DISTINCT ?var WHERE {?var :infects :Liver . ?var rdf:type :Disease}

6.2 Generating SPARQL from Incomplete Patterns

The above procedure addresses the optimal case in which a complete triple pattern is captured by combining the ontology entities. However, there are circumstances in which one or more of the triple components can be missing, for different reasons, resulting in an incomplete triple pattern. Consider the following example: "ما أعراض الإنفلونزا؟", the words "أعراض" و "الإنفلونزا"correspond to an ontology class and an instance respectively, but no ontology property is explicitly determined, resulting in an incomplete triple. In another example: "كيف يشخص سرطان القولون ؟": the verb "يشخص" corresponds to a property, the noun " سرطان القولون" corresponds to an instance, but no word maps to a term that represents a valid subject.

In such cases, the procedure explained in section 6.1 fails to generate a valid SPARQL query directly. It is necessary first to determine and replace the missing components. Only then, a complete RDF triple can be captured and, hence, the above procedure can be applied.

We used an approach that leverages knowledge in the ontology to capture missing components of RDF triples. Knowing any two triple components, the third component can be retrieved by querying the knowledge base using the appropriate queries. To illustrate how a missing RDF component can be identified by knowing the other two components, consider the query: "ما أعراض الانفلونزا ؟": Mapping the query words to the ontology will produce the following sequence of entities: <:Symptom (Class), :Flue(instance)>. This sequence does not make a triple because it lacks a predicate. The implicit predicate can be determined by looking in the ontology for properties being used to link the class :Symptom with the class of the instance :Flue. Given a class C and an instance I, the following SPARQL query is executed to obtain candidate properties:

SELECT DISTINCT ?predicate WHERE {?predicate rdfs:domain C . ?predicate rdfs:range ?range_class . I rdf:type ?range_class }

The above query retrieves properties whose domain includes the class C, and whose range includes the type of the instance I. If multiple properties exist, the user is prompted to choose the desired property. Referring to the query " ما أعراض الانفلونزا؟" and to the schema shown in Figure 1, we execute the following query to identify implicit properties:

SELECT DISTINCT ?predicate WHERE {?predicate rdfs:domain :Symptom . ?predicate rdfs:range ?range_class . :Flue rdf:type ?range_class }

Executing the above query will return the property :symptom_of (عرض_ل), and the triple pattern after identifying the property becomes :<:Symptom(Class), :symptom_of

(Property), :Flue (instance)>. Afterwards, the procedure in section 8.1 becomes applicable, and the following SPARQL query will be generated: SELECT DISTINCT ?var WHERE {?var :symptom_of :Flue . ?var rdf:type :Symptom}.

The rules used to determine the missing RDF components vary depending on the type of the missing component as well as the types of other components. Besides the cases where the predicate can be implicit, the object or the subject of the triple pattern can also be implicit. An example of the latter case is the query: "كيف يشخص سرطان القولون؟": The verb succeeding the question word, i.e. "يشخص", maps to the object property ":diagnoses". The phrase "سرطان القولون" maps to the instance ":Colon_Cancer". To have a complete RDF triple, we need to identify the type of the ontology entity that corresponds to the implicit subject. Knowing the property P and the instance I, candidate subject types can be retrieved from the knowledge base by using the following SPARQL query:

SELECT DISTINCT ?subject_class WHERE {P rdfs:domain ?subject_class . P rdfs:range ?object_class . I rdf:type ?object_class }

The above query retrieves ontology classes that fall in the domain of the property P whose range includes the class of the instance I. On executing the query with P equals to :diagnoses and I equals to :Colon_Cancer, we will obtain the subject type :Diagnosis. This will generate the following complete triple: <:Diagnosis (Class), :diagnoses (Property), :Diabetes (Instance)>, which is interpreted into the SPARQL query: SELECT DISTINCT ?var WHERE {?var :diagnoses :Colon_Cancer . ?var rdf:type :Diagnosis}.

6.3 Interpreting Queries with Conjunctive Sub-queries

AR2SPARQL is capable of interpreting queries that are linked with relative pronouns, e.g. "الذي, التي"or conjunctions, e.g. "و, أو". Queries that are connected with conjunctions cannot be processed separately because they often depend on each other, e.g. one query corresponds to entities in the preceding query. Consider the following question: ما المرض الذي يصيب الكبد ويسبب عسر الهضم؟" "عسر الهضم": The sentence after the conjunction refers to the subject of the first sentence, i.e. "المرض". Therefore, missing components of RDF triples cannot be determined without identifying the dependency between sentences around the conjunction.

To illustrate how NL queries consisting of multiple sentences are processed, consider the following query: ما المرض الذي يصيب الكبد ويسبب عسر الهضم ؟" ". The process of generating RDF triples from this query is depicted in Figure 3.A and explained as the following:

Mapping this query to the ontology content will result in the following sequence of ontology entities:

<:Disease (Class), :infects (Property), :Liver (Instance), :causes (Property), :Indigestion (Instance)>

Given the above sequence, the SPARQL Generator tries to generate RDF triples by combining consecutive entities. This will result in the following triples:

- Triple 1: <:Disease :infects :Liver>,which is a complete triple pattern.
- Triple 2: <? :causes :Indigestion>. This combination does not correspond to a complete triple pattern because it lacks a subject.

It is implicitly understood from the context of the query is that the verb after the conjunction, i.e. "يسبب" refers to the subject of the first sentence,"المرض". By exploiting the ontology semantics and constraints, it is possible to identify this dependency and, accordingly, replace missing RDF components by entities from other complete triples. In the previous example, the complete Triple 1 is searched for an entity that can replace the missing subject of Triple 2. The selected entity should fulfill the condition that it should belong to the domain of the property: causes. The class :Disease of Triple 1 is selected because it is the only entity that fulfills this condition, and Triple 2 becomes <:Disease :causes :Indigestion>. Afterwards, all triples become complete, and thus can be interpreted into SPARQL by applying the procedure discussed in Section 8.1. Since both Triple 1 and 2 share the same subject, they will use the same variable in the generated query to denote the shared subject. The output query will be:

SELECT DISTINCT ?var WHERE {?var :infects :Liver . ?var :causes :indigestion. ?var rdf:type :Disease}

In another example, consider the query " ما الأمراض التي تسبب الإسهال وفقر الدم؟" (see Figure 3.B). The sentence " ما الأمراض التي تسبب الإسهال", which appears before the conjunction, comprises a complete triple pattern which is <:Disease :causes :Diarrhea>. The phrase " فقر الدم", which appears after the conjunction, maps to a single ontology instance, and it is not

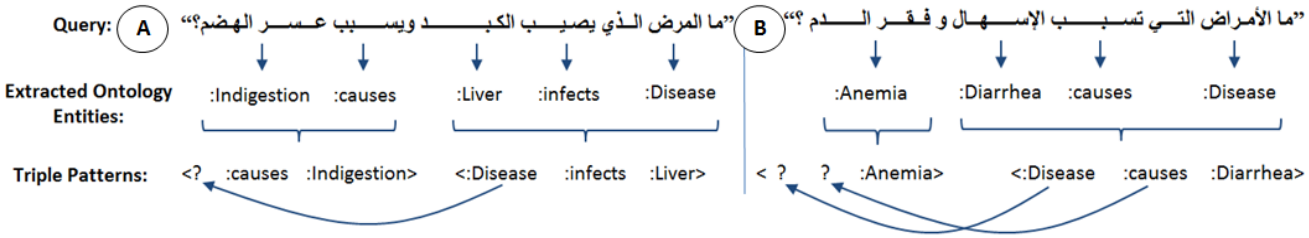

Fig 3: Example of generating RDF triples by capturing dependencies between the query parts: A) The subject ":Disease" of the first triple is used for the second triple. B) The subject ":Disease" and predicate ":causes" of the first triple are used for the second triple.

Related efforts working on the English script often used a statistical parser to identify dependencies between phrases. However, building parse trees from Arabic text is often more complicated, and produces poor results as compared to English counterparts [15]. Therefore, a parser-free approach is proposed and used. The semantics in the ontology are used to identify dependencies between the triple patterns.

query to be: ما الأمراض التي تسبب الإسهال والأمراض التي تسبب فقر الدم؟". Having two complete triple patterns, the SPARQL query will be:

SELECT DISTINCT ?var WHERE {?var :causes :Diarrhea . ?var :causes :Anemia . ?var rdf:type :Disease}

It should be noted that different rules were defined to handle different types of dependencies between triple patterns.

Detailed rules vary for different sequences of entities resulting from the mapping process.

7. AMBIGUITY RESOLUTION

When mapping the user query to the ontology, it is possible that a query word can match with multiple ontology entities. It is necessary to ensure that each word/phrase in the query will only correspond to a single entity in the ontology as possible. AR2SPARQL uses an approach consisting of two levels to resolve ambiguity in the mapping process.

The first level of ambiguity resolution uses the semantic features of the ontology to determine the best match. The point is that only ontological entities that can make a valid and complete RDF triple pattern are chosen. To show how the ontology semantics are used to resolve ambiguity, consider the following query: "ما الأمراض التي من أعراضها ارتفاع ضغط الدم؟" and the schema in Figure 1. The phrase "أعراضها" presents ambiguity because it matches with three ontology entities: the class :Symptom (عرض), the property :has_symptom (له_عرض) and its inverse :symptom_of (عرض_لـ) as they all share the same root. The rule used in this case is that an ambiguous word that occurs between an ontology class and an ontology instance should map to an ontology property because this will result in a complete RDF triple. Therefore, priority in this example is given to the properties: has_symptom and :symptom_of over the class :Symptom.

Note that ambiguity in the above example has not been resolved yet since the word "أعراضها" still corresponds to multiple properties. In this case, the domain and range of candidate properties are examined to determine the correct property that links the corresponding subject and object in the RDF triple. In the previous example, only the property :has_symptom (له_عرض) fulfills this condition because the class :Disease belongs to its domain and the type of the instance :High_Blood_pressure belongs to its range.

Different rules are defined to handle other forms of ambiguity. For example, the word "أعراض" in the query " ما أعراض مرض السكر" is ambiguous as it matches with three ontology entities. The rule used in this case is to prioritize class entity, i.e. :Symptom, over properties if the ambiguous word occurs after the question mark.

If ambiguity cannot be resolved by exploiting the ontology semantics and constraints, the second level which requires the user intervention is used. The system prompts the user with a dialog showing the ambiguous query word/phrase and a list of candidate ontology entities. The user should choose only one entity that will be used to construct the SPARQL query.

8. EXPERIMENTS AND EVALUATION

A full evaluation of the system requires an evaluation of two aspects: 1) Question answering ability: the aim is to asses to what extent the system is able to translate Arabic NL queries to valid SPARQL queries and then retrieve satisfactory answers from a specific knowledge base. 2) Portability across ontologies: AR2SPARQLwas designed with the assumption that it should work with any ontology as long as Arabic translations to all ontology entities are provided.

8.1 Datasets

The assessment of our system was challenged by the lack of Arabic domain ontologies and associated knowledge bases that can be used for question answering. While there are plenty of OWL test data and questions in English [1], we are not aware of any ontology-based test data for Arabic question answering. Therefore, we used two different datasets: The first was obtained from a well-known English-based dataset after adapting it for Arabic use, while the second was constructed from scratch. The details and rationales behind using these datasets are discussed in what follows:

The first dataset is based on the dataset provided by Mooney[32] which has been widely used to assess NL interfaces in English [13, 27, 35]. We used the OWL knowledge base which comprises terminology and data on the geography of the United States. The dataset consists of an OWL ontology and 877 questions expressed in English. To adapt the dataset for Arabic, we populated the ontology with Arabic translations of all ontology entities. Arabic translations were added to the original ontology through the rdfs:label property. Questions were also translated to Arabic, and all translations were validated by a professional translator.

In the second dataset, we constructed a sample ontology of which an excerpt is shown in Figure 1. The intention of creating the ontology was to examine the system's portability when it is interfaced to different ontologies. The ontology consists of 24 classes, 12 object-type properties and 8 data-type properties. All ontology entities were translated to Arabic, and translations were added to the ontology through the rdfs:label property. We created 124 instances of different types, and linked them with the appropriate relations from the ontology. The ontology data and relations were validated by a domain expert. We then presented the ontology and the knowledge base to five human subjects who were medicine students from the local university, thus had prior knowledge of the ontology domain. Each student was asked to formulate 10 questions. At the end, a total of 45 questions were chosen after excluding duplicated ones. Although the number of queries is less than those used in the Mooney's dataset, it helps to assess the performance of the system by using user-define queries collected from native Arabic speakers.

8.2 Evaluation Metrics

AR2SPARQL was evaluated in terms of precision, recall and F-measure, which are defined as follows:

$$\text{Precision} = \frac{number\ \ of\ correctly\ \ translated\ \ queries}{number\ \ of\ queries\ \ generated\ \ by\ the\ system}$$

$$\text{Recall} = \frac{number\ \ of\ corre\ ctly\ translated\ \ queries}{number\ \ of\ testing\ \ queries}$$

$$\text{F-measure} = \frac{2 \times Precision\ \times Recall}{Precision + Recall}$$

8.3 Results and Discussion

Table 1 illustrates the evaluation results obtained over the two datasets, showing the amount of queries tagged as correct (when the system-generated SPARQL query matches the manually-generated one) and incorrect (the systems returns a wrong or incomplete SPARQL query). The system successfully answered 535 queries of the geography queries, thereby achieving 61% average recall and 88.14% average precision. The system also provided answers for 28 queries of the diseases queries with an average recall of 62.22% and an average precision of 82.35%.

Table 1. Performance of AR2SPARQL using the Arabic Mooney and Diseases test data. Row1 shows the number of testing queries. Row 2 shows the number of correctly generated SPARQL queries. Row 3 shows the number of wrong or incorrect SPARQL queries.

Domain	Geography	Diseases
#. of queries	877	45
# of correct	535	28
# of incorrect	72	6
Precision	88.14%	82.35%
Recall	61%	62.22%
F-measure	0.72	0.71

We also analyzed the failures of the system, and classified them into three main categories:

Out-of-coverage failures: This type of failures occurs when a query requires classes, properties or instance not reflected within the ontology. For instance, in the query: " ما مضاعفات الإصابة بمرض تصلب الشرايين ؟" (What are the complications of atherosclerosis?), the system failed to map the word "مضاعفات" (complications) to any of the ontology entities.

Out-of-coverage failures also occur due to the system's inability to map words in the query to ontology entities: In the query: " ما هي الولايات المجاورة لميتشغن؟" (What are the states neighboring Michigan?), the system could not match the word "المجاورة" (neighboring) to any property in the geography ontology. However, this query can be correctly answered if the word "المجاورة" is mapped to the ontology property ":border" whose Arabic label is "يحد". Out-of-coverage failures can also result from the lack of relations between ontology entities. For example, in the query " أذكر أنواع عمليات جراحة القلب" (Mention the types of heart surgery?), there is no explicit relation in the ontology between the instance "heart" and any instance of the type "Surgery", resulting in an incomplete generation of triple patterns. AR2SPARQL cannot infer an answer if there is no relation defined in the ontology between the two terms implied in the relationship.

In general, the out-of-coverage failures contributed for 35% and 50% of the total number of failed queries for the geography and diseases datasets respectively

We believe that an out-of-coverage failure is not considered as a failure of AR2SPARQL. These failures can be easily overcome by enriching the ontology and the knowledge base with more classes, instances and properties so that the system has a better coverage. It is also possible to bridge between the terminology used by the user and the concepts used in the underlying ontology by using external dictionaries, e.g. WordNet for Arabic language.

Semantic failures: This type of failures occurs when the query requires advanced semantic analysis and reasoning that goes beyond the system's capabilities. Examples of these queries include: " ما مساحة كل الولايات مجتمعةً؟"(What is the area of all the states combined?), " ما متوسط عدد السكان لكل كيلومتر مربع في الولايات المتحدة؟" (What is the average population per square km in the US?). Answers to these queries are not explicitly present in the ontology, and require deep analysis and calculations to be performed over the knowledge base. In addition, some words in queries may have multiple meanings, and their interpretations vary from domain to domain. For example, in the query " ما هي المدن الرئيسية في أكبر ولاية؟" ('What are the major cities in the largest state?), it is unclear whether the comparative and superlative words "أكبر, الرئيسية" refer to the area or the population size.

AR2SPARQL does not currently support the processing of comparative and superlative words such as "أكبر/ المشابه أبرز/أهم/لـ" (main, most, largest) since the interpretation of these words often requires specific mechanisms to understand the comparison in different ontologies. It should be noted that AR2SPARQL is designed to be ontology-portable, hence more focus was paid towards the generalization of the interpretation process rather than relying on domain specific interpretations. In fact, even many of the English QA systems that used the same data set do not manage to answer complex queries that require deep semantic analysis [22, 35].

This type of failures contributed for 41% and 17% of the total number of failed queries for the geography and diseases datasets respectively. It is obvious that the semantic failures were less common in the case of the diseases dataset because the human subjects sought to ask questions whose answers could be directly found in the ontology. This is in contrast to the geography dataset where the questions were much diverse and of different complexities.

Linguistic failures: this type of failures originates from linguistic ambiguity that hinders the ability to identify relations between the query words. AR2SPARQL relies on handcrafted rules to identify dependencies between sentences split with conjunctions or pronouns. However, due to the limited coverage of these rules, some queries will be left unresolved. For example, in the query: " ما الأمراض التي تسبب ارتفاع ضغط الدم وكيف تشخص ؟", the system was not able to determine whether the word "تشخص" (is diagnosed) should be linked to the word "الأمراض" or to the phrase " ارتفاع ضغط الدم" since both ways are possible according to the ontology. In addition, Arabic words have different meanings depending on how they are diacritized. However, AR2SPARQL does not currently handle diacritized text, a thing that may lead to linguistic ambiguities.

Despite of these limitations, linguistic failures were the least common type of failures: it contributed only for 24% and 33% of the total number of failed queries for the geography and diseases datasets respectively. This was attributed to the simplicity of the testing query sets which do not often include linguistically-complex structures.

Finally, interfacing the system to two different ontologies confirmed the assumption that it is ontology portable, as we did not notice any mistake or deviation in the behavior when switching the ontologies.

9. RELATED WORK

In this section, we review and discuss the state of the art from three areas related to our work, which are: Arabic QA systems, the support for Arabic language on the SW and NL interfaces to the SW.

9.1 Arabic QA Systems

Despite the Arabic-specific difficulties when compared to English, several efforts have been made to reach an acceptable level in the Arabic QA task. Existing approaches can be divided into two types based on the type of the knowledge domain[12]:1) closed domain systems which deal with

questions under a specific domain. 2) open domain systems which deal with questions of different types and retrieves answers from large databases such as the Internet. In open-domain systems, question analysis and answer extraction tasks are often difficult in comparison with close-domain systems which often rely on application dependent rules and constraints. AQAS [30] is an example of closed-domain systems that was specialized in the restricted domain of radiation and its effects. QARAB[16]system uses an approach that provides short answers to Arabic questions from a collection of Arabic text documents. AQusASys[9]is an open-domain system designed to answer questions related to named entities. It gives attention to question analysis in order to extract informative features.

Most of the above efforts rely on morpho-syntactic approaches in which sophisticated linguistic analysis and NL methods are used. They also provide answers in the form of short passages, extracted from the document collections, rather than giving precise answers. The performance of these systems is limited by the difficulty of Arabic language processing and the considerable lack of effective NLP tools that support Arabic. Few efforts proposed the use of semantic approaches by integrating ontologies or control vocabularies to improve QA. For example, Abuenour et al. [3] used Arabic WordNet to expand the user query by capturing terms that are semantically related to the user terms.

9.2 Support for Arabic language on the SW

To the best of our knowledge, only a few published studies have employed SW technologies in developing Arabic language applications. In general, the studies that addressed the support of Arabic language on the SW can be divided into four categories[7]: 1) the development of Arabic ontologies[18, 20, 21], 2) Employing ontologies to improve Arabic named entities extraction [4, 36], 3) Ontology based representation of Islamic knowledge [2, 19, 24] and 4) supporting cross-language information retrieval and search[17, 33]. Although an increasing number of efforts have started to use ontologies to enhance information retrieval from Arabic data [29, 31], the use of ontologies was almost limited to query expansion, and results were retrieved from unstructured data on the Web. Our work takes a different direction by addressing NL interfaces for querying ontologies and RDF stores.

In the last few years, there has been a growing interest in building Arabic ontologies that can be used in a wide context. For example, the Arabic Ontology project [21] aims to build a formal ontology that resembles an Arabic WordNet but with strict ontological principles. The Quranic Ontology uses knowledge representation to define the key concepts in the Quran, and shows the relationships between these concepts using predicate logic [11]. Other efforts also started to explore ways to enrich the Arabic content over Linked Data such as the creation of Arabic DBpedia [5, 8]. In line with these efforts, NL interfaces will be demanded to enable Arab users to send queries and obtain results from the growing Arabic content on the SW.

9.3 Natural language Interfaces for the SW

In the context of English and Latin based languages, many NL interfaces for querying ontologies and RDF data have been developed in recent years. AquaLog [26] is a QA system over Linked Data that is not tailored towards a particular ontology. It is distinguished by its learning mechanism in a way that it uses ontology reasoning to learn more generic patterns. NLP-Reduce [23] is another domain-independent NL interface to ontologies that avoids using complex linguistic analysis. It tries to identify triple structures in the query words and match them to an OWL knowledge base. The main drawback of these systems is that they rely on handcrafted grammars to identify terms, relations, and to compose triples. Therefore, more expressive queries that do not match any of the predefined patterns cannot be answered.

Some researches proposed the use linguistic parsing to identify and link query terms as an alternative to handcrafted grammars, thus providing the ability to handle linguistically complex questions. For example, PANTO [35] is a portable NL interface that uses a deep parse tree to capture nominal phrases, determine relations and then generate RDF triples. Unger et al. [34] presented an approach that relies on a deep linguistic analysis to produce a SPARQL template that directly mirrors the internal structure of the question and that is instantiated by mapping the occurring natural language expressions to the domain vocabulary. Despite the capabilities offered by the deep linguistic analysis and parsing, it is difficult to generalize these approaches to Arabic language. The rich and complex morphology that Arabic has makes the parsing of Arabic text complicated and error-prone.

10. CONCLUTION AND FUTURE WORK

This paper presents AR2SPARQL, an Arabic natural language interface to ontologies and RDF data. It translates the user query to RDF triple patterns which are then used to build a SPARQL query. Due to the limited available support for Arabic NLP, AR2SPARQL makes less use of linguistic analysis and more of the ontology semantics and constraints in order to translate the Arabic query to SPARQL.

Since this is one of the first works the tackles the notion of Arabic QA on the SW, there are still many directions open for future research: First, researchers can explore ways to handle diacritization, coreference resolution, superlative and comparative nouns and deep reasoning. Second, AR2SPARQL can be interfaced to a single ontology at a time. However, when it comes to the real SW, there is a need to compose information from multiple ontologies. Therefore, a future direction is to upgrade the system from relying on a single ontology to opening up to the rich ontological knowledge available on the Web.

11. REFERENCES

[1] Mooney Natural Language Learning Data. [cited 1-8-2015. Available on: https://files.ifi.uzh.ch/ddis/oldweb/ddis/research/talking-to-the-semantic-web/owl-test-data/.

[2] Abdelnasser, H., et al., 2014. *Al-Bayan: An Arabic Question Answering System for the Holy Quran.* ANLP 2014. p. 57.

[3] Abouenour, L. 2011. *On the improvement of passage retrieval in Arabic question/answering (Q/A) systems*, in *Natural Language Processing and Information Systems* Springer. p. 336-341.

[4] Abouenour, L., K. Bouzoubaa, and P. Rosso. 2010. *Using the Yago ontology as a resource for the enrichment of Named Entities in Arabic WordNet*. in *Editors & Workshop Chairs*.

[5] Al-Feel, H., 2013. *A Step towards the Arabic DBpedia.* International Journal of Computer Applications. **80**(3): p. 27-33.

[6] Al-Khalifa, H. and A. Al-Wabil. 2007. *The Arabic language and the semantic web: Challenges and*

opportunities. in *The 1st int. symposium on computer and Arabic language*.

[7] Al-Zoghby, A.M., A.S.E. Ahmed, and T.T. Hamza, 2013. *Arabic Semantic Web Applications: A Survey.* Journal of Emerging Technologies in Web Intelligence. **5**(1): p. 52-69.

[8] Bahanshal, A.O. and H.S. Al-Khalifa. 2013. *Toward Recipes for Arabic DBpedia*. in *Proceedings of International Conference on Information Integration and Web-based Applications & Services*: ACM.

[9] BEKHTI, S. and M. AL-HARBI. 2013. *AQuASys: A Question-Answering System For Arabic*. in *WSEAS International Conference. Proceedings. Recent Advances in Computer Engineering Series*: WSEAS.

[10] Black, W., et al. 2006. *Introducing the Arabic wordnet project*. in *Proceedings of the 3rd International WordNet Conference (GWC-06)*.

[11] Dukes, K., E. Atwell, and N. Habash, 2013. *Supervised collaboration for syntactic annotation of Quranic Arabic.* Language resources and evaluation. **47**(1): p. 33-62.

[12] Ezzeldin, A.M. and M. Shaheen. 2012. ***A SURVEY OF ARABIC QUESTION ANSWERING: CHALLENGES, TASKS, APPROACHES, TOOLS, AND FUTURE TRENDS***. in *The 13th International Arab Conference on Information Technology*.

[13] Fader, A., L.S. Zettlemoyer, and O. Etzioni. 2013. *Paraphrase-Driven Learning for Open Question Answering*. in *ACL (1)*.

[14] Farghaly, A. and K. Shaalan, 2009. *Arabic natural language processing: Challenges and solutions.* ACM Transactions on Asian Language Information Processing (TALIP). **8**(4): p. 14.

[15] Green, S. and C.D. Manning. 2010. *Better Arabic parsing: Baselines, evaluations, and analysis*. in *Proceedings of the 23rd International Conference on Computational Linguistics*: Association for Computational Linguistics.

[16] Hammo, B., H. Abu-Salem, and S. Lytinen. 2002. *QARAB: A question answering system to support the Arabic language*. in *Proceedings of the ACL-02 workshop on Computational approaches to semitic languages*: Association for Computational Linguistics.

[17] Hattab, M., et al. 2009. *Addaall Arabic Search Engine: Improving Search based on Combination of Morphological Analysis and Generation Considering Semantic Patterns*. in *The second International Conference on Arabic Language Resources and Tools, Cairo, Egypt*.

[18] Hazman, M., S.R. El-Beltagy, and A. Rafea, 2009. *Ontology learning from domain specific web documents.* International Journal of Metadata, Semantics and Ontologies. **4**(1): p. 24-33.

[19] Iqbal, R., A. Mustapha, and Z.M. Yusoff, 2013. *An experience of developing Quran ontology with contextual information support.* Multicultural Education & Technology Journal. **7**(4): p. 333-343.

[20] Ishkewy, H., H. Harb, and H. Farahat, 2014. *Azhary: An Arabic Lexical Ontology.* International Journal of Web & Semantic Technology. **5**(4).

[21] Jarrar, M. 2011. *Arabic ontology engineering-challenges and opportunities*. in *Proceedings of the 2011 International Conference on Intelligent Semantic Web-Services and Applications*: ACM.

[22] Kaufmann, E. and A. Bernstein. 2007. *How useful are natural language interfaces to the semantic web for casual end-users?*: Springer.

[23] Kaufmann, E., A. Bernstein, and L. Fischer. 2007. *NLP-Reduce: A "naive" but Domain-independent Natural Language Interface for Querying Ontologies*: ESWC Zurich.

[24] Khan, H.U., et al., 2013. *Ontology Based Semantic Search in Holy Quran.* International Journal of Future Computer and Communication. **2**(6): p. 570-575.

[25] Khoja, S. 2001. *APT: Arabic part-of-speech tagger*. in *Proceedings of the Student Workshop at NAACL*.

[26] Lopez, V., M. Pasin, and E. Motta. 2005. *Aqualog: An ontology-portable question answering system for the semantic web*, in *The Semantic Web: Research and Applications* Springer. p. 546-562.

[27] Lopez, V., et al., 2013. *Evaluating question answering over linked data.* Web Semantics: Science, Services and Agents on the World Wide Web. **21**: p. 3-13.

[28] Lu, J., et al. 2007. *SOR: a practical system for ontology storage, reasoning and search*. in *Proceedings of the 33rd international conference on Very large data bases*: VLDB Endowment.

[29] Mahgoub, A.Y., et al., 2014. *Semantic Query Expansion for Arabic Information Retrieval.* ANLP 2014. p. 87.

[30] Mohammed, F., K. Nasser, and H. Harb, 1993. *A knowledge based Arabic question answering system (AQAS).* ACM SIGART Bulletin. **4**(4): p. 21-30.

[31] Soudani, N., et al. 2014. *Toward an Arabic Ontology for Arabic Word Sense Disambiguation Based on Normalized Dictionaries*. in *On the Move to Meaningful Internet Systems: OTM 2014 Workshops*: Springer.

[32] Tang, L.R. and R.J. Mooney. 2001. *Using multiple clause constructors in inductive logic programming for semantic parsing*, in *Machine Learning: ECML 2001* Springer. p. 466-477.

[33] Tazit, N., et al., 2007. *Semantic internet search engine with focus on Arabic language.* The 1st International Sysmposium on Computers and Arabic Language & Exhibition 2007¦© KACST & SCS.

[34] Unger, C., et al. 2012. *Template-based question answering over RDF data*. in *Proceedings of the 21st international conference on World Wide Web*: ACM.

[35] Wang, C., et al. 2007. *Panto: A portable natural language interface to ontologies*, in *The Semantic Web: Research and Applications* Springer. p. 473-487.

[36] Zaidi, S., M. Laskri, and A. Abdelali. 2010. *Arabic collocations extraction using Gate*. in *Machine and Web Intelligence (ICMWI), 2010 International Conference on*: IEEE.