AskNow: A Framework for Natural Language Query Formalization in SPARQL

Mohnish Dubey[1], and Sourish Dasgupta[2], Ankit Sharma[3], Konrad Höffner[4], Jens Lehmann[1,5]

[1] Computer Science Institute, University of Bonn; dubey@cs.uni-bonn.de, jens.lehmann@cs.uni-bonn.de
[2] DA-IICT, India; sourish@rygbee.com
[3] State University of New York - Buffalo; ankitkai@buffalo.edu
[4] AKSW Group, University of Leipzig; konrad.hoeffner@uni-leipzig.de
[5] Fraunhofer IAIS; jens.lehmann@iais.fraunhofer.de

Abstract. Natural Language Query Formalization involves semantically parsing queries in natural language and translating them into their corresponding formal representations. It is a key component for developing question-answering (QA) systems on RDF data. The chosen formal representation language in this case is often SPARQL. In this paper, we propose a framework, called *AskNow*, where users can pose queries in English to a target RDF knowledge base (e.g. DBpedia), which are first normalized into an intermediary canonical syntactic form, called Normalized Query Structure (*NQS*), and then translated into SPARQL queries. NQS facilitates the identification of the desire (or expected output information) and the user-provided input information, and establishing their mutual semantic relationship. At the same time, it is sufficiently adaptive to query paraphrasing. We have empirically evaluated the framework with respect to the syntactic robustness of NQS and semantic accuracy of the SPARQL translator on standard benchmark datasets.

1 Introduction

With the advent of massive scale knowledge bases (such as DBpedia [1], YAGO [2], Freebase [3], Google Knowledge Vault [4], Microsoft Satori, etc.),the need to have a user-friendly interface for querying them became relevant. However, users usually are not deft in (and in most cases lack the knowledge of) writing formal queries. *Natural language query formalization* (*NLQF*) is a formal and systematic procedure of translating a user query in natural language (NL) into a query expression in a target formal query language. In this paper, we scope the problem of NLQF to RDF/RDF-S knowledge bases only. Within this context, the target formal query language chosen is SPARQL [5] – the W3C recommended and widely adopted query language for RDF data stores.

NLQF into SPARQL for question-answering on RDF data stores is non-trivial. This can be attributed to several reasons: (a) a semantic interpretation of natural language query is intrinsically complex and error-prone, (b) the schema of the target dataset is not fixed, (c) a partial lack of rich schema structures of RDF datasets leading to syntactic mismatches, (d) lexical mismatches of query tokens, and (e) mismatches due to lack

of explicit entailed relations in an RDF store. One of the key linguistic challenges is the accurate identification of the *query desire* (also known as *query intent* or *answer type* in the literature) of a user-query. Another major challenge is that a query can be paraphrased into multiple forms, thereby triggering the potential of lexico-syntactic mismatches. Also, there is no unique way to create schemata in RDF, i.e. the same fact can be written in different triple forms and could also be expressed using multiple triples.

In this paper, we propose an NLQF framework, called *AskNow*, for posing queries in English to target RDF data stores. AskNow uses an intermediate canonical syntactic structure, called *Normalized Query Structure* (*NQS*), into which an *NQS fitting* algorithm normalizes English queries. One of the primary objectives of the algorithm is to normalize paraphrased queries into a common structure. Another objective is to help a SPARQL translator to easily identify the query desire, query input (i.e. additional information provided by the user), and their mutual semantic relation. As an example, given the query: "*What is the capital of India?*", the algorithm will be able to differentiate the query desire (i.e. instances of class *capital*) and relate it to the input *India* via the relation *of*. Here, the input plays an important role in automatically constructing the declarative formal description of the desire. After a query is normalized into an NQS instance, the SPARQL translator then maps the query tokens in the NQS instance so as to entities defined in the RDF data store. This is done to solve the potential problem of lexical and schematic mismatch mentioned earlier. We show empirically, using QALD [6] benchmark datasets, that the devised NQS Fitting algorithm is accurate in correctly characterizing (both in terms of syntax and semantics) most NL queries. Our contributions in this paper is as follows:

- A novel paraphrase resilient query characterization structure (and algorithm), called *NQS*, is proposed. NQS is less sensitive to structural variation. It supports complex queries, hence, serves as a robust intermediary formal query representation.
- An NQS to SPARQL translation algorithm (and tool) is proposed that supports user queries to be agnostic of the target RDF store structure and vocabulary.
- An evaluation of AskNow in terms of: (i) assessing robustness using the Microsoft Encarta data set, and (ii) evaluating accuracy using a community query dataset built on the OWL-S TC v.4.0 dataset and the QALD-4/5 datasets.

The paper is organized into the following sections: (2) *Approach*, where the formal notions of *NQS* query is defined, (3) in which the *Architectural Pipeline* of *AskNow* is elaborated, (4) *Evaluation*, where various evaluation criteria are discussed, and (5) *Related Work* outlining some of the major contributions in NL query processing.

2 Preliminaries

Definition 1 (**Simple query**): A simple query consists of a single and unconstrained query-desire (explicit or implicit) and a single, unconstrained, and explicit query-input. For example, in "*What is the capital of USA?*" the query-desire (i.e. *Capital*) is explicit, single, and not constrained by any clausal phrase. The query-input (i.e. *USA*) is also explicit, single, and unconstrained. The query-desire can be implicit: For example, in

the query "*What is a tomb?*", the implicit query-desire is the *definition* of *tomb*, while the query-input *Tomb* is single and unconstrained.

Definition 2 (**Complex query**): A complex query consists of a single query-desire (explicit or implicit, constrained or unconstrained) and multiple, explicit query-inputs (constrained or unconstrained). For example, the query "*What is the capital of USA during World War II?*" is a complex query where the implicit, unconstrained query-desire is single (*Capital*) while the query-input is multiple (*USA*, *World War II*).

Definition 3 (**Compound query**): A compound query consists of conjunction/disjunction operator connectives between one or more simple or complex queries. An example of a compound query is "*What are the capitals of USA and Germany?*".

3 Approach

3.1 Motivation

Our chosen query representation language is SPARQL, in which basic graph patterns consist of subject, predicate and object. So, the primary objective of query parsing should be to identify the query desire/s and describe it in terms of the query predicate/s and query input/s. The identification of such a constraint relation is called *query desire-input dependency*. One of the key tasks for solving the problem of NLQF is to do *syntactic normalization* of NL queries. Syntactic normalization is the process that re-structures queries having different syntactic structural variations into a common structure so that subsequent formalization can be executed using a standard translation algorithm on this structure. Such normalization is difficult to achieve through a *query desire-input dependency* identification process alone. It is in this direction that we propose a chunker-styled *pseudo-grammar*, called *Normalized Query Structure* (*NQS*).

3.2 Normalized Query Structure (NQS)

NQS is the proposed basis structure of natural language queries. It acts like a surface level syntactic template for queries, defining the universal linguistic dependencies (i.e. query desire-input dependency) between the various generic sub-structures (i.e. chunks) of any query. The primitive sub-structures of any query is the query token (which determines the query type), query desire, the query input, and the dependency relation that connects them. Each of the primitive sub-structures can be assigned a linguistic characterization (i.e. type). An example characterization are POS (part-of-speech) tag based chunks (such as noun phrase, verb phrase, etc.). For instance, desire and input can be hypothesized to assume a noun form, while the dependency relation can be assumed to be a verb form. An example of such characterization can be seen in the query: "*In which country is New York located?*". Here, the query desire is the noun phrase *country*, the query input is the noun phrase *New York*, and the dependency relation is the verb phrase *located in*. Since the number of query tokens is finite and there are only three query forms (simple, complex, compound), natural language queries can be categorized into a (finite) set of generic NQS templates.

We now introduce the NQS syntax definitions as follows:

Simple Query NQS: A *simple query* can be characterized according to the following

NQS structure:
$[Wh]$ $[R_1]$ $[D]$ $[R_2]$ $[I]$
here, $[D] = Q_D^? M_D^* D$ and $[I] = Q_I^? M_D^* I$
where the notation is defined as follows[6]:
$[D]$: Query desire class/instance-value is restricted to the following POS tags: *NN, NNP, JJ, RB, VBG*. *When* and *where* queries have $[D = \text{NULL}]$, and NQS automatically annotates D as *TIME* and *LOCATION* respectively.
$[I]$: Query input class/instance-value restricted to the POS tags: *NN, NNP, JJ, RB, VBG*.
$[R_1]$: Auxiliary relation - includes lexical variations of the set: $\{$*is*, *is kind of*, *much*, *might be*, *does*$\}$.
$[R_2]$: Relation that acts as (i) predicate having D as the subject and I as the object or (ii) action role having I as the actor - value restricted to the POS tags: *VB, PP, VB-PP*
$Q_D^?$ or $Q_I^?$: Quantifier of D or I - values restricted to the POS tag: DT. The $?$ indicates that Q can occur zero or one time before D or I.
M_D^* or M_I^*: Modifier of D or I - value restricted to the POS tags: *NN, JJ, RB, VBG*. The $*$indicates that M can occur zero or multiple time before D or I.

Characteristics of Relation Tokens: R_1 serves as a good indicator for resolving several linguistic ambiguities. For example, in a *how*-query, if R_1 is *much* (or its lexical variations) then it is a quantitative query. However, in a *who*-query if R_1 is *does* (or its lexical variations) then the associated verb is an activity (i.e. Gerund; ex: "*Who does the everyday singing in the church?*" - *everyday singing* is an activity in this case). R_2 is a relation that can either be associated with D as the subject or I as the subject but not both. If R_2 is positioned after D in the original NL query then R_2's subject is D. For example, in the simple NL query "*What is the capital of USA?*" the subject of R_2 (*of*) is D (*Capital*) and the object is I (*USA*). However, if R_2 is positioned after I in the original query then its subject is I. For example, in the query "*Which country is California located in?*" the subject of R_2 (*located in*) is I (*California*) and object is D (*Country*).

NQS of Complex & Compound Wh-queries: A complex *Wh*-query can be characterized according to:

$[Wh]$ $[R]$ $[D]$ $[Cl_D^?]$ $[R_2]$ $[I_1^1]$ $[([CC]$ $[I_2^1])^*]^*$... $[Cl_2^?]$ $[R_3]$ $[I_1^2]$ $[([CC]$ $[I_2^2])]^*$...
...$[Cl_N^?]]$ $[R_{N+1}]$ $[I_1^N]$ $[([CC]$ $[I_2^N])]^*$
where:
Cl_D : clausal lexeme (constraining *D*). Example of clausal lexemes: *wh*-tokens, *that*, *as*, *during/while/before/after*, etc. It is to be noted that clausal lexemes generates nested sub-queries which themselves may (or may not) be processed independently to the parent query. An example where the sub-query (in bold) has a dependency is: "*Which artists where born on the same date as **Rachel Stevens**?*"
Cl_2: second clausal lexeme (constraining I_1)
Cl_k: clausal lexeme associated with *k*-th sub-structure
$[CC]$: conjunctive/disjunctive lexeme for *I*
$[D]$: query desire - value restricted to POS tags: $\{$*NN, NNP, JJ, RB, VBG*$\}$
$[I_l^k]$: l-th query input for k-th structure - value restricted to POS tags: {NN, NNP, JJ,

[6] All POS-tag notations follow Penn Treebank.

RB, VBG}
$[R_{k+1}]$: relation associated with the k-th clause that acts as (i) predicate of *D* as the subject and *I* as the object or (ii) action role of *I* as the actor - value restricted to to POS tags: {VB, PP, VB-PP}.
Notation with ? may occur zero or one time.
Notation with $*$ may occur zero or multiple time.

In the given complex NQS, we see the possible repetition of the structure: $[I_1^k][([CC]$ $[I_2^k])]$. Within this structure, there is an optional substructure $[([CC][I_2^k])]$ that may add to the number of inputs within each of such structures. A clausal lexeme in a complex clausal *wh*-query is always associated with such a structure. The number of clausal lexemes is the same as the number of such structures in a given query. It should be noted that there must be at least two such structures for a query to qualify as complex. Clausal lexemes are optional and hence, the NQS also works for complex non-clausal wh-queries. We name the following structure as *clausal structure* (*CS*):
$[Cl_D^?]$ $[R_2^?]$ $[I_1^1]$ $[([CC][I_2^1])^*]^?$...$[Cl_2^?]$ $[R_3^?]$ $[I_1^2]$ $([CC]$ $[I_2^2])^?$...
...$([Cl_N^?][R_{N+1}^?][I_1^N]([CC][I_2^N])^*[?]$.

A compound *Wh*-query can then be characterized according to:
$[Wh^1]$ $[R_1^{1?}]$ $[D_1^{1?}]$ $[Cl_D^?]$ $[R_2^?]$ $[I_1^1]$ $([CC]$ $[I_2^1])^*$ $[Cl_2^?][R_3^?][I_1^2]([CC][I_2^2])^*$....
$([Cl_N^?][R_{N+1}^?][I_1^N]([CC][I_2^N])^*)^*[?]$.

4 AskNow Architectural Pipeline

We have outlined the architectural pipeline of AskNow, in Fig. 1 with two basic components: the NQS Instance Generator and the NQS to SPARQL converter.

4.1 NQS Instance Generation

As mentioned in the previous section, the objective of NQS is not to propose yet another grammar but rather to provide a modular format to the internal sub-structures of a query. Therefore, an efficient template-fitting algorithm that can parse the natural language query, identify the sub-structures (using a standard POS tagger), and then fit them into their corresponding *cells* within the larger generic NQS template is required. Our proposed template-fitting algorithm is called *NQS Instance Generator*. Through the fitting process the query-desire, query-input, and other relevant information can be extracted. A fitted NQS is called an *NQS instance*. The fitting process automatically leads to normalization. Also, it is resilient to paraphrasing of queries since the sub-structures in a paraphrase typically remain unaffected[7]. The change is only in the inter sub-structure positioning (eg: "*New York is located in which country?*" vs. ''*Which is the country where New York is located?*")[8]. Note that the original NL query may lose

[7] In certain cases minor splitting has to be handled in the dependency relation, where the query starts with a preposition, e.g.: "***In** which country is New York **located**?*"

[8] Paraphrasing may include lexical substitution of synonymous query tokens and morphological changes of the tokens.

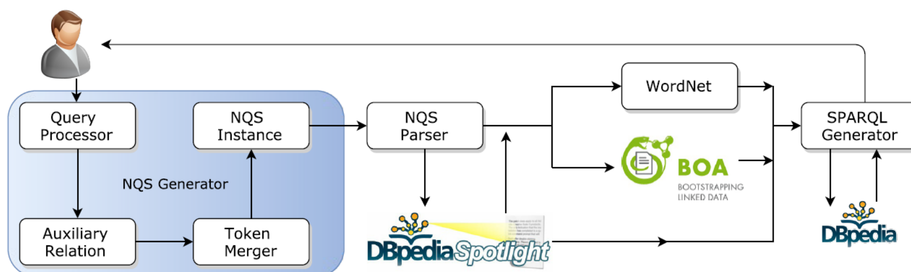

Fig. 1. Architectural Pipeline of AskNow

its syntactic structure during the NQS instance generation process and also, it does not guarantee the grammatical correctness of the normalized query.

In summary, the flexibility of NQS modeling is to be attributed to the NQS Instance Generator algorithm. All internal components of the NQS Instance Generator are described as follows:

Query Processor: This module initiates the NQS query processing system by initializing other modules. It calls the POS-tagger (in our case we used Stanford coreNLP[9]) so as to tag every query token. Then it breaks the query text into individual POS-tagged query tokens. Subsequently, the *Syntactic Normalizer* transforms original queries to have a common syntactic structure. For example, it normalizes each query to start with *wh*-token, handling apostrophe, etc.

Auxiliary Relation Handler: The module to extract $R1$. More details regarding the utility of the auxiliary relation is given in previous section.

Token Merger: This module merges (or *chunks*) tokens that together form a single meaningful lexeme. Based on the POS tags of the tokens in the original NL query, the token merging module can guess the possible tokens to be combined so that they can fit the NQS. For example, when the query, "*Who is the Prime Minister of India?*" is passed to the POS-Tagger we get the resulting answer: "Who_{WP} is_{VBZ} the_{DT} $Prime_{NNP}$ $Minister_{NNP}$ of_{IN} $India_{NNP}$?". Then two *consecutive tokens* are taken at a time and checked, using a token-merging map, whether they can be combined or not. We have manually bootstrapped the token-merging map on different types of token-pair lexico-syntactic patterns, using the M.S. Encarta 98 query dataset. The map keeps getting updated as and when other valid token-pairs are identified in future.

NQS Instance Generator: After the individual chunks have been identified, the query then goes through the NQS Instance Generator (see Fig. 1). It uses the following two hypothesis about the generic structure of a query (which was observed to be empirically true when tested on Microsoft Encarta 98, which is a large-scale query dataset).

Hypothesis 1: The query desire is always a noun phrase.

Hypothesis 2: The query desire always precedes the query input in the normalized NL query.

[9] http://nlp.stanford.edu/software/tagger.shtml

The algorithm also utilizes the characteristics of desire-input dependency relations, as discussed in the previous section. Every time it encounters a noun phrase chunk it treats it as a candidate desire. Depending upon the availability of conjunctive connectives, it then does a conflict resolution among all candidate desires by verifying the positioning of the verb phrase. As an example, the query "*Desserts from which country contain fish?*" has three candidate desires: *dessert*, *country*, *fish* (based on Hypothesis 1). The main relation *contain* is positioned after *country*. Therefore, the *potential* subject of *contain* is identified to be *country*[10]. Now according to Hypothesis 2, the desire must precede the input in the NQS instance. So *fish* is resolved not to be a candidate desire any more, but rather an input. Now, the query has another main relation *from*, the subject being *dessert* and the object being a query token *which*. Thus, the algorithm resolves that *country* is the desire while *dessert* is another input. Finally, the algorithm analysizes that *country* being the desire, and also having the inverse relation *from* to the input *dessert*, cannot have the relation *contain* to the second input *fish*. Therefore, it is the input *dessert* which is the true subject of the relation *contain* to the object (i.e. the second input) *fish*. The final NQS will be: $[wh = which][R_1 = null][D = country][R_2 = from][I_1 = dessert][R_3 = contain][I_2 = fish]$. It is to be noted that there is an implicit nested dependent sub-query: "*Which desserts contain fish?*" because of the clausal connective *whose* (*Which country* ***whose*** *desserts* ...) that is an inverse of the relation *from*. This example illustrates that the previously outlined NQS syntax definitions are not static templates, but rather dynamically fitted.

4.2 NQS to SPARQL Conversion

Given an NQS instance, the NQS2SPARQL module translates it to a SPARQL query and returns the result from the SPARQL endpoint. There are four main steps in this module as shown in Algorithm 1.

NQS analysis: Once we have an NQS instance for a query, the system treats it as per its category. The categories are the expected query types, specifically: i) Boolean ii) Ranking iii) Count iv) Set (List) and v) Property Value. In a Boolean query a user asks whether a specific statement is True of False. For instance "Is Barack Obama a democrat?" A Ranking query requires ranking the answers based on some entity dimensions, e.g. "Which is the highest mountain in Asia?". In a Count query the user intent is to get the number of times a certain condition is repeated. A Set query will generate a list of items which satisfy a required condition. In a Property value query the user intent is to ask for the value of a property of the given input. As an example, in the query "What is the capital of India?" the user intents to extract the value of the property "capital" given the input "India". Query-types are chosen based on desire (D) and wh-type (wh) of the NQS instance. Each category is processed by a different SPARQL query syntax converter.

Entity Mapping: The basic operation here is to retrieve the knowledge base entity matching the spotted query desire, query input and their relation. For the QALD experiments described later, we annotated the query using DBpedia Spotlight [7]. As a

[10] A standard dependency parser could also be used to understand the subject of the relation *contain*.

Table 1. QALD-5 example on AskNow

NL Query	List down all the Swedish holidays
NQS values	[WH = What], [R1 = is], [D = list], [R2 = of], [M = Swedish], [I = holiday]
Type	List
SPARQL	SELECT DISTINCT ?uri WHERE { ?uri rdf:type dbo:Holiday. ?uri dbo:country res:Sweden }
NL Query	In which country is Mecca located?
NQS values	[WH = which], [R1 = is], [D = country], [R2 = located In], [I = Mecca]
Type	Property Value
SPARQL	SELECT ?num WHERE { res:Mecca dbo:country ?num . }
NL Query	How many ethnic groups live in Slovenia
NQS values	[WH = How many], [R1 = *null*], [D = count(ethnic group)], [R2 = live in], [I = Slovenia]
Type	Count
SPARQL	SELECT COUNT(DISTINCT ?uri) WHERE { res:Slovenia dbo:ethnicGroup ?uri . }
NL Query	Who is the heaviest player of the Chicago Bulls?
NQS values	[WH = Who], [R1 = is], [M = heaviest], [D = player], [R2 = of], [I = the Chicago Bulls]
Type	Ranking
SPARQL	SELECT DISTINCT ?uri WHERE { ?uri rdf:type dbo:Person . ?uri dbo:weight ?num . ?uri dbp:team res:ChicagoBulls} ORDER BY DESC(?num) OFFSET 0 LIMIT 1

result of the mapping, we get the knowledge base entity equivalent of the query input I which has been identified in the NQS instance. We denote this entity as i. The mapping approach then collects properties related to i (where i is a resource) and their values in set (denoted S).

Subsequently, each element (a pair of property and value) of S is observed. The next goal is to identify the entity which matches the desire (itself denoted as D) and denote it as d. This is done using three mapping functions as follows: The first test is made by a simple label matching function(lm). If this fails, then the second test for mapping($D \rightarrow d$) is through the WordNet synonym (wns)function. It finds the synonym of user desire using WordNet [8] within set S. If this test fails we move to next test. Here we use BOA pattern library [9] for the same purpose. When this is unsuccessful, then we declare that the query is unprocurable by the system.

SPARQL Generation: This component creates the final SPARQL query using information provided by above two steps. NQS analysis basically gives the SPARQL pattern possible. Where as i, d provide the key DBpedia information (vocabulary) required for SPARQL. Examples are given in Table 1. Currently NQS2SPARQL is functional for DBpedia only. However, we can plugin any other RDF store using suitable corresponding entity mapping module.

```
input  : NQS instance ℵ, knowledge base KB
output: SPARQL query results
   // Step 1: NQS analysis
1  D ⟵ queryDesire(ℵ);
2  wh ⟵ getWhQuestionType(ℵ);
3  t ⟵ determineQueryType(D, wh);
4  I ⟵ queryInput(ℵ);
   // Step 2: SPARQL preparation
5  i ⟵ mapInput(I);
6  S = {(p, v)|(i, p, v) ∈ KB};        // construct predicate object map
7  init p_match;
8  foreach (p, v) ∈ S do
       // label matching
9      if lm(p)==D then p_match = p; break;
       // WordNet synonyms of desire
10     if wns(D) ==p then p_match = p; break;
       // BOA library
11     if BOA(D) ==p then p_match = p; break;
   // Step 3: SPARQL generation and retrieval
12 q ⟵ generateQuery(i, p_match, KB);
13 R ⟵ executeQuery(q, KB);
14 return R
```

Algorithm 1: NQS to SPARQL Algorithm.

5 Evaluation

5.1 Evaluation Goal and Metric

Goal I. Syntactic Robustness: *Syntactic robustness* of *NQS* measures its structuring capacity after normalization. Ideally, the *NQS* algorithm should be correct By *correct structuring* we mean that there should not be any mismatch between the POS tag of a linguistic constituent and its corresponding *NQS* cell. At the same time, the algorithm should be complete (i.e. there should not be any valid English query that is not accepted by the algorithm, either fully or partially). To evaluate robustness we decided on a simple measure called *Structuring Coverage* (*SC*). We measure SC in the following three different perspectives:
(i) SC-Precision: Given a test set of NL queries, *SC-Precision* is calculated as the ratio of the number of correct *NQS-structured* queries (N_{CI}) and the total number of *NQS-structured* queries in the test set (N_I). It largely depends upon the accuracy of the POS tagger used.
(ii) SC-Recall: Given a test set of NL queries, *SC-Recall* is calculated as the ratio of the number of correct *NQS-structured* queries (N_{CI}) and the total number of queries in the test set (N).
(iii) SC-F1: The Simple Harmonic Mean of *SC-Precision* and *SC-Recall*.

Goal II. Sensitivity to Structural Variation: *Sensitivity to structural variation* of *NQS* measures the degree to which *NQS* can *correctly fit* queries having same *desire* (and

its relationship with *input*) yet different syntactic structures. To evaluate *sensitivity to structural variation* we introduce following two measures:
(i) Variational-Precision (*VP*): Given a test set of NL queries, the *VP* is calculated as the ratio of the number of correct *NQS-structured* queries (i.e. without any of their variations getting incorrectly fitted) (N_{VI}) and the total number of identified queries in the test set (N_I).
(i) Variational-Recall (*VR*): Given a test set of NL queries, the *VR* is calculated as the ratio of N_{VI} and total number of queries in the test set (N).

Goal III. Semantic Accuracy: *Semantic accuracy* of *NQS* measures the degree to which the query *desire* and its relation with query *inputs* has been properly identified. To evaluate this we use the following measures:
(i) Semantic-Precision (*SP*): Given a test set of NL queries, the *SP* is calculated as the ratio of the number of correctly identified queries (i.e. in terms of *desire-identification*, *input-identification*, and *desire-input relation identification*) (N_{SI}) with respect to a human-judgement benchmark, and the total number of identified queries in the test dataset (N_I).
(ii) Semantic-Recall (*SR*): Given a test set of NL queries, the *SR* is calculated as the ratio of N_{SI} with respect to a human-judgement benchmark, and the total number of queries in the test dataset (N).
Here are examples to give a better understanding of purpose of each measure:
Failed NQS (i.e. no instance):[Wh= NULL] [R1= is] [D= Berlin] [R2= NULL] [I= country][?]
Incorrectly structured NQS instance: [Wh= In which country] [R1= is] [D= Berlin] [R2= located] [I= NULL]. This will be considered as identified query (i.e. one in N_I).
Correctly structured NQS instance (i.e. in $N_C I$): [Wh = Which] [R1 = is] [D = Berlin] [R2 = located in] [I = country]. We use SC (and also VP, VR) to test $N_C I$ with respect to N_I and total queries (N).
Correctly "identified" NQS instance (i.e. in $N_S I$): [Wh = Which] [R1 = is] [D = country] [R2 = located in] [I = Berlin][?]. We use SP and SR to test this.

Goal IV Accuracy of the AskNow System: The final goal of the evaluation is to test the system on the QALD-5 [10] benchmark (Multilingual question answering over DBpedia). Here, we have queries in English language which are answered with NQS translated SPARQL.

5.2 Datasets

In order to evaluate *syntactic robustness* (for goal-I), we have used the Microsoft Encarta 98 [11] query test set. . The test set contains 1365 usable English *wh*-queries. There are total 522 queries of procedural *how* and *why* that have been excluded. We also created an extensive query set based on OWLS-TC v4[12] for evaluation of both *sensitivity*

[11] http://research.microsoft.com/en-us/downloads/88c0021c-328a-4148-a158-a42d7331c6cf/
[12] http://projects.semwebcentral.org/projects/owls-tc/

Table 2. *SC* Evaluation on Different Datasets

	QALD 5			**M.S. Encarta**			**OWL-S TC**			**Total**			**Result**		
	N	N_I	N_{CI}	N	N_I	N_{CI}	N	N_I	N_{CI}	N	N_I	N_{CI}	SC_R	SC_P	SC_{F1}
How	31	31	31	165	158	158	4	4	2	200	193	191	95.50	98.96	97.20
What	37	37	37	406	392	392	1711	1709	1608	2154	2138	2037	94.57	95.28	94.92
When	12	12	12	39	35	35	0	0	0	51	47	47	92.16	100	95.92
Where	5	5	5	85	82	82	20	20	19	110	107	106	96.36	99.07	97.70
Which	81	81	81	5	5	5	316	316	308	402	402	394	98.01	98.01	98.01
Who	48	48	48	143	143	143	166	166	166	357	357	357	100	100	100
Total	214	214	214	843	815	815	2217	2215	2215	3274	3244	3226	98.53	99.45	98.99

to structural variation (goal-II) and *semantic accuracy* (goal-III). Three research assistants independently formulated wh-queries for every web service of OWLS-TC v4 dataset, such that the query desire matches the given service output, and the query input matches the required service input. We had 1083 services to make three different query versions for each service. Similar syntactic structure queries were excluded resulting in a total of 2217 queries It is to be noted that the goal of the experiment (cf.: Goal II) was to test the robustness of an NQS Instance Generator, in terms of POS-tag pattern fitting (i.e. syntactic accuracy), over different syntactic variations of the same query. 90% of the queries were complex or compound queries. Ideally, the extracted query desire by NQS should be semantically equivalent to the output parameter of the corresponding web service specification. Based on this notion, we have calculated *SC-accuracy*, *VP/VR*, and *SP/SR* for each of the three versions of query dataset. We also used the QALD-5 [10] datasets for Goal-IV and QALD-4 [11] for evaluating Goal-II.

5.3 Results

Result I. Syntactic Robustness: We first performed the evaluation of *structural robustness* in terms of SC-Accuracy over different *query*-types on *Microsoft Encarta 98* dataset. We observe 100 % SC-Precision for all types of wh-queries, which shows that the NQS is theoretically sound. The *SC-Recall* came out to be 96.68 %. We then performed the same experiment over different *wh*-types on 2 more datasets: Training set of QALD-5's Multilingual tract (only *english* queries) and *OWLS-TC*. We observed a high overall SC-F1 of 98.99 %. The evaluation results are given in table 2.

Result II. Sensitivity to Structural Variation: We performed evaluation of *sensitivity to structural variation* of *NQS* over the OWL-S TC query dataset (three versions) and the QALD-4 dataset (three versions). *NQS* was able to correctly fit 919 out of the 1083 OWLS-TC queries (along with all their syntactic variation), giving high *VP* of 96.43 %. All 24 out of 24 QALD-4 queries, with all there syntactic variations, were correctly fitted in *NQS*, giving a high sensitivity to structural variation.

Result III. Semantic Accuracy: We observed an *SP* of 91.92 % for the OWL-S TC query dataset. For QALD-4 dataset, it was observed that 21 out of 24 queries (with their variations) were correctly fitted in *NQS*. Analysis of the fail case clearly indicates that NQS failure is dependent upon syntactic and POS Tag failures.

Table 3. Evaluation of Sensitivity to Structural Variation and Semantic Accuracy

Dataset	N_{Wh}	N_I	N_{VI}	N_{SI}	$VR\%$	$VP\%$	$SR\%$	$SP\%$
OWL S TC	1083	953	919	876	84.85	96.43	80.88	91.92
QALD-4	24	24	24	21	100	100	87.50	87.50
Total	1107	977	943	897	85.18	96.51	81.03	91.81

Table 4. Results on the QALD 5 benchmark.

	Processed	Right	Partial	Recall	Precision	F_1	F_1 Global
Xser	42	26	7	0.72	0.74	0.73	0.63
AskNow	27	16	1	0.63	0.60	0.61	0.33
QAnswer	37	9	4	0.35	0.46	0.40	0.30
APEQ	26	8	5	0.48	0.40	0.44	0.23
SemGraphQA	31	7	3	0.32	0.31	0.31	0.20
YodaQA	33	8	2	0.25	0.28	0.26	0.18

Results IV. Accuracy of AskNow: We used the benchmark data set of the 5th Workshop on Question Answering over Linked Data (QALD), which defines 50 questions to DBpedia and their answers. Here we compare the our results with the result published by QALD-5 [10]. Out of 50 questions provided by the benchmark we have successfully answered 16 correct and 1 partially correct. There were 5 questions where NQS algorithm fails to correctly identify the Inputs and Desire hence they could not be answered by translating them into SPARQL. The failure analysis of Result IV are as follows:
NQS failure: Queries where NQS failed were not further processed successfully. NQS failed only 5 times, which was due to incorrect dependency analysis.
Entity Mapping: There are 13 questions where AskNow could not map the DBpedia equivalent of correctly identified input and desire. In some cases, the correct mapping was presented but insufficient to answer the query. As an example, the query *"Who killed John Lennon?"* is correctly processed by NQS and forwarded to DBpedia Spotlight for annotation. It maps $JohnLennon$ to `http://dbpedia.org/resource/John_Lennon` which is a correct mapping in general terms. But we can not answer the question based on this resource. For that we would require `http://dbpedia.org/resource/Death_of_John_Lennon`.
Relation Mapping: In some cases, system could not resolve the $R2$ (relation between input and desire) to the correct DBpedia property. Relations such as $study$ and $graduated$ were not mapped to the required DBpedia property $almaMater$.

6 Related Work

Over the last decade, several *NLQF* approaches have been proposed. Several of them attempt to translate NL queries into SPARQL-like formalisms. Early works in this direction includes GiNSENG [12]. It is a guided input NL search engine, that does not understand NL queries, but uses menus to formulate NL queries in small and specific

domains and allow users to query OWL knowledge bases in a controlled language akin to English. Subsequently, Semantic Crystal [13] was proposed, which is also a guided and controlled graphical query language. Systems such as AquaLog [14] and its advancement, PowerAqua [15], are based on mapping linguistic structures to ontology-compliant semantic triples. PowerAqua is the first system to perform QA over structured data, providing a single NL query interface for integrating information from heterogeneous resources. The limitation of PowerAqua is the lack of support for query aggregation functions. Along the same lines, FREyA [16] allows users to enter queries in any form, and uses ontology reasoning to learn more generic rules. It also provides a better handling of ambiguities over heterogeneous domains. But FREyA requires some level of effort in KB structure understanding to efficiently clarify disambiguation. Also, it highly depends on modeling and vocabulary of the data at the user-end, making it inadequate for a naive user. Other works such as NLPReduce [17] allow users to pose questions in full or slightly controlled English. NLPReduce is a domain-independent system, which leverages the lexico-syntactic pattern structures of query input to find better matches in the KB. It maps query tokens with synonym enhanced triple stores in the target corpus, based on which it generates SPARQL statements for those matches. QTL [18] is a feedback mechanism for question answering using supervised machine learning on SPARQL. TBSL [19] uses so called BOA patterns as well as string similarities to fill the missing URIs in query templates and bridge the lexical gap.

Recently, Question Answering over Linked Data (QALD) has become a popular benchmark. In QALD-3 [20], SQUALL2SPARQL [21] achieved the highest precision in the QA track. SQUALL2SPARQL takes an inputs query in SQUALL, which is a special English based language, and translates it to SPARQL. Since no linguistic resource is required, it results in a high performance. But on the other hand, it makes the SQUALL query unnatural to the end-user and requires manual annotations of the URIs. In QALD-4 [11], GFMed [22] achieved the highest precision in the biomedical track. It is based on Grammatical Framework (GF) [23] and a Description Logics based methodology and proposes an algorithm to translate a query in natural language into SPARQL queries using GF resources. It can support complex queries, but only works with controlled languages and biomedical datasets. In POMELO [24], predicates of the RDF triples are mapped to frame predicates while the subjects and objects are mapped to core frame elements. Then after a question abstraction step, the final SPARQL query is generated. POMELO is based on closed environment (biomedical) and fails to relate the disconnected semantic entities. gAnswer [25] proposes a graph mining algorithm to map natural language phrases to top-k possible predicates to form a paraphrase dictionary. It also proposes a novel approach to perform disambiguation in query evaluation phase, which improves the precision and speed up query processing time greatly.

Xser [26], the most successful system in QALD-4 and QALD-5, uses a two-step architecture. It first understands the NL query by extracting phrases and labeling them as *resource, relation, type* or *variable* to produce a Directed Acyclic Graph (DAG). This semantic parser works independent of any Knowledge Base (KB). Then these semantic entities are instantiated with the given KB. However, it requires too much human involvement in manually annotating the questions with phrase dependency DAG to train the system. In comparison to Xser, AskNow requires no training data. The NQS instance

generation step is independent of both the query dataset and the target knowledge base (KB). Xser uses semantic parser with DAG as linguistic analyzer. AskNow use POS-tag and NER to find the main entity of the query. As linguistic analyzer an NQS instance has ability to further distinguish between query desire (D), query input(I)and their relation (R) apart from spotting the main entity only. Xser uses wikipedia miner tool to generate the candidate set of DBpedia entities , AskNow uses DBpedia Spotlight for annotating the query Input to DBpedia entities Xser uses PATTY to map phrases to predicates and categories of DBpedia whereas, AskNow do this by mapping query relation or query desire to DBpedia equivalent using WordNet and BOA pattern library.

APEQ [10], from QALD-5 [10], uses a graph traversal based approach, where it first extracts the main entity from the query and then tries to find its relations with the other entities using the given KB. APEQ uses Graph traversal technique to determine the main entity by graph exploration. Finally, the graph with the best scoring entities is returned as the answer.

7 Conclusion

In this paper, we propose *AskNow*, a NLQF framework, based on a novel syntactic structure *Normalized Query Structure* (NQS). We empirically show, using benchmark datasets, that NQS is robust in terms of syntactic variation, and also highly accurate in identifying the query desire (along with its relationship to the query input). Hence, we show that NQS serves as a strong intermediary model for translating NL queries into formal queries. We empirically demonstrated this by converting NQS to SPARQL.

Acknowledgements: This work was supported by a grant from the EU H2020 Framework Programme provided for the project Big Data Europe (GA no. 644564).

References

1. S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives, *Dbpedia: A nucleus for a web of open data*. Springer, 2007.
2. F. M. Suchanek, G. Kasneci, and G. Weikum, “Yago: a core of semantic knowledge,” in *Proceedings of the 16th international conference on World Wide Web*, pp. 697–706, ACM, 2007.
3. K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor, “Freebase: a collaboratively created graph database for structuring human knowledge,” in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pp. 1247–1250, ACM, 2008.
4. X. Dong, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, K. Murphy, T. Strohmann, S. Sun, and W. Zhang, “Knowledge vault: A web-scale approach to probabilistic knowledge fusion,” in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 601–610, ACM, 2014.
5. J. Pérez, M. Arenas, and C. Gutierrez, “Semantics and complexity of sparql,” in *International semantic web conference*, vol. 4273, pp. 30–43, Springer, 2006.
6. 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.

7. P. N. Mendes, M. Jakob, A. García-Silva, and C. Bizer, "Dbpedia spotlight: shedding light on the web of documents," in *Proceedings of the 7th International Conference on Semantic Systems*, pp. 1–8, ACM, 2011.
8. G. A. Miller, "Wordnet: a lexical database for english," *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.
9. D. Gerber and A.-C. Ngonga Ngomo, "Bootstrapping the linked data web," in *1st Workshop on Web Scale Knowledge Extraction @ ISWC 2011*, 2011.
10. C. Unger, C. Forascu, V. Lopez, A.-C. N. Ngomo, E. Cabrio, P. Cimiano, and S. Walter, "Question answering over linked data (qald-5)," in *Working Notes for CLEF 2015 Conference*, 2015.
11. C. Unger, C. Forascu, V. Lopez, A.-C. N. Ngomo, E. Cabrio, P. Cimiano, and S. Walter, "Question answering over linked data (qald-4)," in *Working Notes for CLEF 2014 Conference*, 2014.
12. A. Bernstein, E. Kaufmann, and C. Kaiser, "Querying the semantic web with ginseng: A guided input natural language search engine," in *15th Workshop on Information Technologies and Systems, Las Vegas, NV*, pp. 112–126, Citeseer, 2005.
13. E. Kaufmann and A. Bernstein, *How useful are natural language interfaces to the semantic web for casual end-users?* Springer, 2007.
14. V. Lopez, V. Uren, E. Motta, and M. Pasin, "Aqualog: An ontology-driven question answering system for organizational semantic intranets," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 5, no. 2, pp. 72–105, 2007.
15. 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.
16. D. Damljanovic, M. Agatonovic, and H. Cunningham, "Freya: An interactive way of querying linked data using natural language," in *The Semantic Web: ESWC 2011 Workshops*, pp. 125–138, Springer, 2012.
17. E. Kaufmann, A. Bernstein, and L. Fischer, "Nlp-reduce: A "naıve" but domain-independent natural language interface for querying ontologies," ESWC Zurich, 2007.
18. J. Lehmann and L. Bühmann, "Autosparql: Let users query your knowledge base," in *The Semantic Web: Research and Applications*, pp. 63–79, Springer, 2011.
19. C. Unger, L. Bühmann, J. Lehmann, A.-C. Ngonga Ngomo, D. Gerber, and P. Cimiano, "Template-based question answering over rdf data," in *Proceedings of the 21st international conference on World Wide Web*, pp. 639–648, ACM, 2012.
20. P. Cimiano, V. Lopez, C. Unger, E. Cabrio, A.-C. N. Ngomo, and S. Walter, "Multilingual question answering over linked data (qald-3): Lab overview," in *Information Access Evaluation. Multilinguality, Multimodality, and Visualization*, pp. 321–332, Springer, 2013.
21. S. Ferré, "squall2sparql: a translator from controlled english to full sparql 1.1," in *Work. Multilingual Question Answering over Linked Data (QALD-3)*, 2013.
22. A. Marginean, "Gfmed: Question answering over biomedical linked data with grammatical framework," CLEF, 2014.
23. A. Ranta, *Grammatical framework: Programming with multilingual grammars*. CSLI Publications, Center for the Study of Language and Information, 2011.
24. T. Hamon, N. Grabar, F. Mougin, and F. Thiessard, "Description of the pomelo system for the task 2 of qald-2014," CLEF, 2014.
25. 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*, pp. 313–324, ACM, 2014.
26. K. Xu, S. Zhang, Y. Feng, and D. Zhao, "Answering natural language questions via phrasal semantic parsing," in *Natural Language Processing and Chinese Computing*, pp. 333–344, Springer, 2014.