Semantic Web 1 (2016) 1–5
IOS Press

Question Answering on RDF KBs using Controlled Natural Language and Semantic Autocompletion

Editor(s): Stamatia Dasiopoulou, Pompeu Fabra University, Spain; Georgios Meditskos, Centre for Research and Technology Hellas, Greece; Leo Wanner, ICREA and Pompeu Fabra University, Spain; Stefanos Vrochidis, Centre for Research and Technology Hellas, Greece; Philipp Cimiano, Bielefeld University, Germany

Giuseppe M. Mazzeo and Carlo Zaniolo
University of California, Los Angeles,
United States
E-mail: {mazzeo|zaniolo}@cs.ucla.edu

Abstract. The fast growth in number, size and availability of RDF knowledge bases (KBs) is creating a pressing need for research advances that will let people consult them without having to learn structured query languages, such as SPARQL, and the internal organization of the KBs. In this paper, we present our Question Answering (QA) system, that accepts questions posed in a Controlled Natural Language. The questions entered by the user are annotated on the fly, and a KB-driven autocompletion system displays suggestions computed in real time from the partially completed sentence the person is typing. By following these patterns, users can enter only semantically correct questions which are unambiguously interpreted by the system. This approach assures high levels of usability and generality. Experiments conducted on well-known QA benchmarks, including questions on the encyclopedic DBpedia and specialized domains, such as music and medicine, show a better accuracy and precision than previous systems.

Keywords: Natural language interfaces, question answering, RDF, semantic autocompletion

1. Introduction

The last few years have seen major efforts toward building RDF knowledge bases (KBs) for both general and specialized knowledge. In the first group we find DBpedia [1,33], that encodes the Wikipedia encyclopedic knowledge, and in the second group we have the thousands of projects that cover more specialized domains [2]. While these KBs can be effectively queried through their SPARQL [3] endpoints, the great majority of web users are neither familiar with SPARQL nor with the internals of the KBs. Thus, the design of user-friendly interfaces that will grant access to the riches of RDF KBs to a broad spectrum of web users has emerged as a challenging research objective of great significance and interest.

The importance of this topic has inspired a significant body of previous research work and the launching of annual competitions on Question Answering over Linked Data (QALD). In this paper, we describe our `CANaLI` system that has recently won the the 2016 competition (QALD-6). This success is even more remarkable, since it was obtained by using only the NL functionality of our system,without the query completion function that assists users formulating their questions. The important problem of QA has seen much previous work, which we briefly describe next, and in more details in Section 5.

Exploratory browsing Among the many approaches that can be classified under this label, we find the one in [28] where users specify a graphical query "skele-

ton" and annotate it with phrases and entity names, narrowing the search space and generating SPARQL queries through heuristics based on statistical associations and semantic similarities to classes and properties of the KB. This approach is a compromise between SPARQL and natural language interfaces which avoids the problem of relation extraction from NL sentences. However, complex queries, e.g., involving advanced joins and aggregates, are not discussed in [28].

Faceted Search This approach relies on a multi-step formulation of queries on DBpedia [26,27]. For instance, a user searching for “cities in California" will start by supplying an item-type (e.g., “City"). This reduces the search space to cities, whereby the user can specify a number of other type-depending filters, such as population, latitude, and so on.

The Query by Example Approach The SWiPE system proposes a WYSIWYG approach [15], where *by-example structured queries* are entered by (i) letting the user select an example page and activate its InfoBox, on which (ii) she can now click and insert conditions into the relevant fields, that (iii) are translated into a SPARQL query that is executed on the DBpedia KB.

NL Interfafces While system such as [15] allow users to enter complex queries on web browsers, Natural Language (NL) still provides the simplest form of communication for casual users, via voice-recognition systems or other interfaces that do not require the use of web browsers.

Thus, it is hardly surprising that the NL QA problem has been the focus of much research. As surveyed in [24,25,36], this important and challenging problem combines several non-trivial sub-problems, including parsing the syntactic structure of the question, mapping the phrases of the question KB resources, and resolving the ambiguities that are always lurking in NL communications. Resolving ambiguities is indeed the hardest problem, since, e.g., NL systems are limited in their ability to perform anaphora resolution that often depends on context, perceived query intention, and domain knowledge that is available to the interlocutors but not to the system.

Therefore, the paramount objective in the design of our NL system was to avoid ambiguities, and we achieved this objective by the two-pronged approach of using:

(i) *controlled natural language* (CNL) interface, and

(ii) *Question Autocompletion* integrated with the CNL.

At the best of our knowledge, our CANaLI[1] system is the very first among NL and CNL systems to support item (ii) above with its significant benefits. But even without (ii), CANaLI is a CNL system of great effectiveness: tested on various QA testbeds [4], CANaLI delivered superior precision and recall, and came first in the 2016 QALD competition QALD-6 [4]. Indeed CANaLI demonstrates that a reasonable middle ground exists in the design quandary faced by all CNL systems, which need to restrict the grammar allowed for questions to make the language ‘formal’ enough to be accurately interpreted by machines, but still ‘natural’ enough to be readily acquired by people as an idiomatic version of their NL. All these systems, including CANaLI, are based on the idea that it is worth giving up some of the great flexibility and eloquence of the NL in order to make the questions unambiguous to the machines, that will thus produce answers of better accuracy and completeness [32].

CANaLI and its CNL were designed to (i) avoid ambiguities, (ii) achieve enough power and generality to express the example questions from the various NL testbeds, and (iii) support in real time the very desirable function of question autocompletion, which represents novelty for NL QA systems, although it is very popular in Web browsers[2]. The autocompletion function guides users by allowing them to only enter questions that are semantically correct w.r.t. the underlying KB. Moreover, as soon as the user makes a mistake or hesitates when typing, the system suggests possible correct completions. This allows people to self-learn CANaLI easily and quickly. Autocompletions are produced in real time whereby users are never slowed down and an on-line interaction with the system is achieved. The design and implementation techniques that makes this very fast completion possible are described in the paper.

This paper is organized as follows. Section 2 provides an overview of CANaLI, describing its operation, by means of some examples. Section 3 describes how CANaLI suggests semantically valid tokens. An experimental analysis of accuracy and usability is pre-

[1] CANaLI is an acronym for Context-Aware controlled Natural Language Interface.

[2] The autocompletion of web browsers is built from previous popular searches. The CANaLI autocompletion is instead derived automatically from the underlying KB.

sented in Section 4 and related work is presented in Section 5. Finally, we present the conclusions and possible future extensions of CANaLI in Section 6.

2. Short Overview of CANaLI

The inspiring idea of CANaLI consists in viewing questions as navigations through the entities, classes, and properties of the underlying RDF network, and then recasting such navigations into roughly equivalent navigations through the states of a carefully designed finite automaton. By identifying typical connectives used to link semantics concepts in NL, we were thus able to identify with simple tokens that CANaLI can use to transition quasi-deterministically between states as the interpretation of the controlled NL sentences progresses. The simplicity and efficiency of the recognizer so constructed entails a real-time response, whereby the system can assist the user by suggesting alternative question completions derived on-the-fly from the underlying KB. This provides great help to users who are typing their questions. and allows them to enter non-trivial questions with complete confidence that, even when more than one interpretation is possible for the question, the system will answer according to the interpretation intended by the user.

CANaLI enables users to enter questions in a controlled and guided way, as sequence of *tokens* representing:

- KB resources: entities, properties, and classes,
- operators (e.g., equal to, greater than, etc.),
- literals: numbers, strings, and dates,
- auxiliary NL phrases, such as "having", that play a syntactic sugaring role.

Every token used in CANaLI is represented at user-level as an NL phrase, consisting of one or more words from the application domain. No operator, variable, URI or other SPARQL syntax are required for entering questions in CANaLI. The recognizer operates on tokens in the style of a finite state machine having 12 states, including the initial state and a final state. Despite its simplicity, CANaLI is very general, since it can be used with arbitrary RDF KBs, and basically supports nearly all the typical questions asked by users, including those proposed in published papers and testbeds, as discussed in details in Section 4.

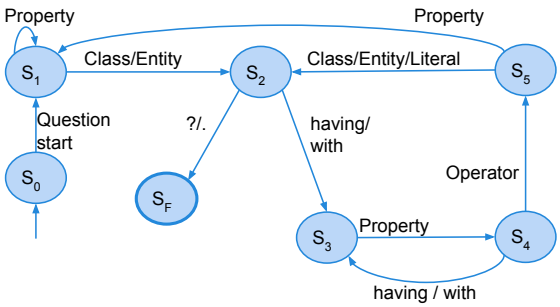

Fig. 1. The main states and transitions of the automaton used by CANaLI

2.1. Answering Simple Questions

The operation of CANaLI can be explained with the help of the transition diagram in Figure 1, and a simple example[3]. Say that the user wants to enter the question: "What is the capital of United States?". When the user starts typing a new question, CANaLI's automaton is in the initial state (S_0), and it is ready to accept the question-start tokens, such as "What is the", that moves our recognizer to state S_1. At S_1, the system can accept a token representing an entity, a property, or a class. In our example, the user enters "capital", that represents a property recognized by CANaLI. Thus, the system loops back to state S_1, ready to accept as next token another property, entity, or class. In our simple example the user enters "United States", that denotes an entity, and the system moves to S_2, after recognizing "United States" as an entity with "capital" as its valid property. Thus, in order to be consistent with the semantics of the knowledge base, our user must enter entities that have the property "capital", and the system will stop her from progressing any further if that is not the case. Of course, to reach this 'no progress' point the user must have ignored the suggestions that the system had previously generated as valid completions of the typed input. CANaLI shows these completions in a drop-down menu appearing under the input window (see Fig. 2). The user has the option of clicking on any such completion, whereby its text is added to the input window. In S_2 a range of new input tokens can be accepted—including the question mark "?" used in our example. Obviously this ends the question, whereby CANaLI moves to the final

[3]Here, the system response is based on the context provided by the question typed so far and the underlying KB, rather than just the current state and last token as a finite state automaton would.

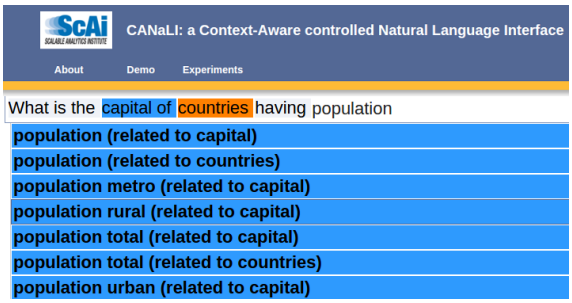

Fig. 2. The autocompleter of CANaLI suggesting properties that (i) can be related to capitals/countries and (ii) contain in their label the last word typed by the user, i.e., "population"

state S_F and launches the actual SPARQL query execution. Alternatively, the user can enter more conditions, e.g., using tokens such as "having", that will be discussed later. Let us now consider an example involving a chain of properties such as: "What is the population of the capital of United States?". In this case, at S_1, the user would input the property "population", whereby the system loops back to S_1, where it accepts the second property: "capital". Now, CANaLI accepts "capital" because the capitals have a population, and loops back to S_1, where "of United States" takes us to state S_2 where the question mark completes the processing of the input and launches the query.

Remarkably, the four basic states S_0, S_1, S_2, and S_F support a large set of very simple questions asked by everyday users[4]. More complicated but nevertheless common questions are those adding constraints, i.e., query conditions. For instance, assume that the user wants to ask: "What is the capital of countries having population greater than 100 million?". After the input "What is the capital" has moved us to S_1, CANaLI accepts "countries", as a class having "capital" as a valid property, and moves to S_2. In S_2, CANaLI accepts "having" (an uninterpreted connective used as syntactic sugar) to move to S_3, where it can only accept a valid property. In this case, "population" can be accepted since countries have this property. However, this example illustrates the ambiguity that beset all NL interfaces, no matter how sophisticated their parser is. In fact, a constraint "population greater than" is also applicable to "capital", since capitals have population too. Clearly every NL system would suffer from the same problem, and only a person who knows that currently no city has more than 100 million people, might be able to suggest that the condition is probably about countries, rather than capitals. However, CANaLI finesses this inherently ambiguous situation by displaying all alternative interpretations whereby the user can make an explicit choice (see Fig. 2). Once the property "population" is accepted, and its context clarified, the automaton moves to the state S_4, where an operator is expected. Thus, the user can input "greater than" and the automaton moves to state S_5, that accepts the right-hand side of the comparison operator. In general, the right-hand side of a constraint can be an element of the KB or a literal. In our example, only a number can be accepted[5], since the right-hand side must be of the same type as the left-hand side, "population", which is numerical. Thus, the user enters "100 million" and the automaton moves to S_2, where she can specify more constraints or enter a question mark, ending the question[6].

Examples of constraints using resources of the KB as right-hand side are the following: "Give me the country having capital equal to Washington D.C."[7], "Give me the movies having director equal to a politician.", "Give me the cities having population greater than the population of Los Angeles.". In all these cases, the token accepted in S_5 must be consistent with the property previously accepted in S_3. Observe that, while accepting an entity or a class moves the automaton to S_2 state, accepting a property (e.g., *population*) yields a transition to S_1, where the element possessing the property must be specified (e.g., *Los Angeles*).

Finally, a question using the edge connecting S_4 to S_3 is "What are the countries having capital with population greater than 10 million?", that uses a chain of properties on the left hand side of a constraint. Like in the case explained above, "capital" can be accepted here since countries have a capital. Then, after accepting the connective "with", "population" can be accepted since capitals have this property.

2.2. More Complex Questions

For the sake of presentation, we have shown in Fig. 1 only the states that are most commonly used in

[4]Indeed, the most frequent web questions are definition questions (e.g., What is Ebola?), that are even simpler.

[5]Or another property with numerical range, as discussed in the following.

[6]Questions can also start with "Give me". In this case, questions are ended using a period.

[7]Indeed, the complete automaton of CANaLI has also a transition from S_4 to S_2 that allows to implicitly assume the equality operator. This allows to accept questions such as "Give me the country having capital Washington D.C."

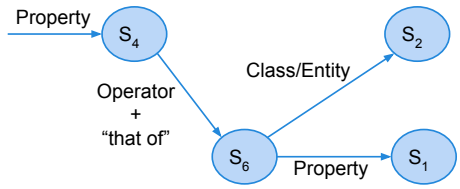

Fig. 3. Fragment of the automaton for accepting chunks of questions like "population greater than that of Los Angeles"

Fig. 4. Fragment of the automaton for accepting chunks of questions like "equal to their death place"

Fig. 5. Fragment of the automaton for accepting chunks of questions like "greater than that of their spouse"

queries. In reality CANaLI has five more states which are used to support the additional patterns that are illustrated by the following examples:

– "Give me the cities having population greater than that of Los Angeles." The use of the pronoun *that* in place of the already used property *population*, makes the question more natural than repeating "population". However, the new state S_6, as shown in Fig. 3, is needed for handling this kind of questions. After accepting the first part of the question "Give me the cities having population", the automaton is in state S_4. From this state, it accepts the token "greater than that of", thus moving to S_6. From this state, it is possible to accept an entity (e.g., *Los Angeles*) or a class (e.g., *country*), thus moving to S_2, or a property (e.g., *capital of*), moving to S_1, where the element possessing the property will be subsequently accepted.
– "Give me the actors having birth place equal to their death place." The possessive determiner implies that the properties *birth place* and *death place* are related to the same variable. A new state S_7 is needed, as shown in Fig. 4. In this case, after accepting "Give me the actors having birth place equal to", the automaton from state S_5 accepts the possessive determiner "their", and moves to S_7. From S_7 it accepts the property "death place", thus moving to S_2.

Fig. 6. Fragment of the automaton for accepting chunks of questions like "with one of the 20 greatest population" or "having the 2nd largest population"

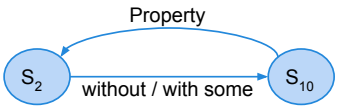

Fig. 7. Fragment of the automaton for accepting chunks of questions like "without side effects"

– "Give me the actors with birth date greater than that of their spouse.". This question combines the two situations described above, and requires to introduce the state S_8, as shown in Fig. 5. After accepting "Give me the actors having birth date greater than that of" the automaton is in state S_6, from which it accepts the possessive determiner "their", thus moving to S_8. From S_8 the property "spouse" is accepted, and the automaton moves to S_2.
– "Give me the countries having one of the 20 greatest population" and "Give me the country having the 2nd largest population". Questions like these require (i) to sort the results by the value of the property recognized in a specific state, and (ii) to set the offset and number of returned results according to the token accepted in a different state, i.e., a token such as *the nth greatest* or *one of the nth greatest*. To this end, we introduced state S_9, as shown in Fig. 6. After accepting "Give me the country having", from S_3 the automaton can accept a ranking token, whereby it moves to S_9, where the property "population" is recognize, thus producing the transition to S_2.
– "Give me the drugs without side effects". This question requires negation. We remark that a token such as *without* or *with some* can not be handled as the tokens like *having*, which defines a comparison between two operands. Therefore, a new state S_{10} was introduced. After accepting "Give me the drugs", it is possible to accept "without" from state S_2, thus moving to state S_{10}. In S_{10} the property *side effect* is accepted, moving the automaton to S_2.

3. Token Selection

A key to effectiveness and usability CANaLI is its ability to traverse the underlying KB to select in real

time the correct transition in the recognizer automaton and generate the correct completions for the partial user input. CANaLI uses Apache Lucene [5], to index our tokens, which are associated with one or more NL phrases. Thus, when the user enters a string S, a query is performed on the index to select among the tokens matching the phrase S those which satisfy the following criteria: (i) they are of acceptable type, according to the current automaton state, and (ii) they are semantically correct with respect to the underlying KB. Now, criterion (ii) requires satisfaction of several conditions for which we index each element of the KB by its label and kind (i.e., entity, property, class, etc.), and the values of the two additional fields: *domain_of*, and *range_of*. The first field is needed for cases such as "What is the population of" that prescribes that the following token can only be accepted if it belongs to domain of "population" (e.g., "capital", "cities", "Washington D.C.", etc.). The second field is used in cases such as "...having capital equal to": that prescribes that the token that follows must belong to the range of "capital" (e.g., "birth place", "city", "Washington D.C.", etc.). A similar domain requirement is enforced when dealing with input such as "What is the capital of countries having", where we have a property having as domain the property "capital" or the class "country".

The domain and range of a property are defined as follows:

Definition The **domain of a property** p is the set of elements (entities, classes, or properties) $\{t\}$ such that

- if t is an entity, there exists a triple $\langle t, p, v\rangle$;
- if t is a class, there exists a triple $\langle e, p, v\rangle$, where e belongs to class t or to a descendant class of t;
- if t is a property, there exists a triple $\langle e, t, v\rangle$ and a triple $\langle v, p, w\rangle$.

Figure 8 depicts a small portion of DBpedia represented as a graph. Entities, classes, and literals are the nodes of the graph, represented in the figure as green ellipses, orange rectangles, and white rounded rectangles, respectively. A triple is represented as a directed edge connecting the subject node to the value node. The label of the edge is the property of the triple, represented as a blue rectangle. The edges with labels `rdf:type` and `rdfs:subClassOf` represent the class of entities and the class hierarchy, respectively. According to our definition, the domain of `dbo:populationTotal` includes the entity `dbr:Washington_D.C.`, the classes `dbo:City`, `dbo:Populated_Place`, and

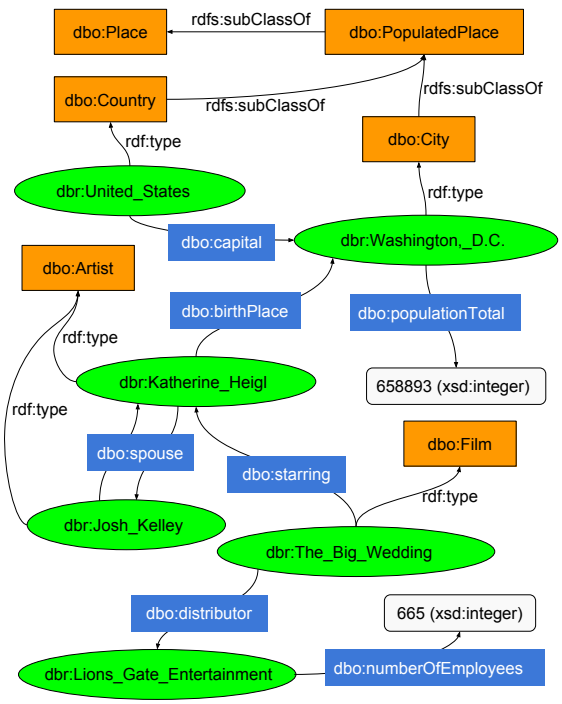

Fig. 8. Fragment of the DBpedia graph. Orange rectangles represent classes, green ellipses are entities, rounded rectangles are literal values, blue rectangles are properties, `rdf:type` and `rdfs:subClassOf` are special properties used to define the class of entities and the class hierarchy, respectively.

`dbo:Place`, and the property `dbo:birthPlace`. We include the ancestor classes of `dbo:City` in the domain of `dbo:populationTotal` to improve the flexibility of the questions that can entered by the user. In fact, even if only the populated places will have this property, by adding also `dbo:Place` to the domain of `dbo:populationTotal` we allow the user to ask for the population total of "places" without being specific about the exact class (i.e., populated places). Clearly, only the populated places are returned as answer, but a user has the flexibility to ask for 'specific' properties of 'generic' classes.

Now we define the range of properties. While the domain of an attribute can not contain literals (the subject of the triples can not be a literal), the range can. As said before, in CANaLI the types of literals are numbers, strings, and dates. We call these types *basic types*.

Definition The **range of a property** p is the set of elements (entities, classes, properties, or basic types) $\{t\}$ such that

- if t is an entity, there exists a triple $\langle s, p, t\rangle$;
- if t is a class, there exists a triple $\langle s, p, e\rangle$, where e belongs to class t or to a descendant class of t;
- if t is a property, there exists a triple $\langle e, t, v\rangle$ and a triple $\langle f, p, v\rangle$;
- if t is a basic type, there exists a triple $\langle s, p, l\rangle$, where l is a literal with basic type t.

According to this definition, the range of the property `dbo:populationTotal` includes the numerical basic type, while the property `dbo:birthPlace` includes the entity `dbr:Washington_D.C.`, the classes `dbo:City`, `dbo:Populated_Place`, and `dbo:Place`, and the property `dbo:capital`.

We can now define the rules that allow `CANaLI` to accept a token and transition from the current state to the next state. Given a sequence of previously accepted tokens,

the ***last accepted property*** denotes the rightmost accepted token that is a property, and

the ***open variables*** are the tokens that are either a class or a property. These tokens are pushed onto a stack as they are accepted.

S_1: In this state, a new input token x is accepted if $x \in domain(p)$, where p denote last accepted property[8]. For instance, if `dbo:populationTotal` is the last accepted property, then only elements that are in its domain can be accepted. Thus, presented with the input "*What is the population total of Washington D.C.?*", `CANaLI` accepts the entity `dbr:Washington_D.C.`. Also, the class `dbo:City` is accepted when the input is "*What is the population total of the cities ...?*". Now, the property `dbo:birthPlace` is also in the `dbo:populationTotal` domain, and thus it is accepted as next token in "*What is the population total of the birth place of ...?*". Observe that this rule prevents some apparently simple questions to be entered. For instance, consider the question "Who is the president of United States". After typing "Who is the president of", the user will not be able to complete the question, since in DBpedia the entity `dbr:United_States` has no property `dbo:president`. In fact, the leaders of countries (not just the presidents) are represented in DBpedia by pairs of properties representing the leader name and title[9]. In general, when `CANaLI` recognizes that the user is searching for a missing property of an existing entity, it displays a message stating that the information is missing, and tells the users to look at the list of properties available for the searched entity.

S_3, **S_9**, and **S_{10}**: being O the stack of open variables, a property p can be accepted if $O \cap domain(p) \neq \emptyset$. For instance, if the user has already typed *What is the birth place of artists having*, from S_3 the next acceptable properties are those having `dbo:birthPlace` or `dbo:Artist` in their domain, whereby both `dbo:spouse` and `dbo:populationTotal` are acceptable here. Here, when the property p has more than one elements of O in its domain, `CANaLI` will propose more possibilities for accepting p, and the user can select that corresponding to her intention (see Fig. 2). The same holds for states S_9 and S_{10}. For instance, if the user already typed *Give me the city having the 2nd greatest*, she can complete the question with "population total" since the class *city* is in the domain of the property *population*. Similarly, if she already typed *Give me the artists without*, she can complete the question with "spouse" since the class *artist* is in the domain of the property *spouse*. We remark that our recognizer accepts properly nested constraints, in the style of *Visibly Pushdown Languages* [14]. However, as we have seen, in sentences like *What is the birth place of artists having* our recognizer can accept a property related to any of the two variables in the stack, not necessarily to that on topof the stack (i.e., "artist" in our case). However, when the user types a property which is related to a variable v which is not on top of the stack (e.g., *What is the birth place of artists having population total*), the variables on top of v a removed from the stack, and can not be longer used to define constraints. This avoid the possibility to input non-natural and 'confusing' questions, such as *What is the birth place of artists having population total greater than* $1,000,000$ *and starring in Star Wars*.

S_5: with p the last accepted property, an element x can be accepted if $x \in range(p)$. The element accepted is the right-hand side of the constraint, according to property p. Assuming that p is `dbo:populationTotal`, `CANaLI` can accept a number (*...having population total greater than* ***50 000****...*) or another property that has a numeric range. For instance, the use of property `dbo:numberOfEmployees` could be used to input a fragment such as *...having population total less than the* ***number of employees*** *of...*, which is semantically correct. If p is `dbo:capital`, then the entity `dbr:Washington_D.C.` (*...having capital*

[8] When the question starts, p does not exist, thus any element of the KB can be accepted

[9] See `http://dbpedia.org/page/United_States`

equal to Washington D.C....), or the class `dbo:City` (...*having capital equal to* ***city***...), or the property `dbo:birthPlace` (...*having capital equal to* ***birth place*** *of*...) can all be accepted.

$\mathbf{S_6}$: with p the last accepted property, an element x can be accepted if $x \in domain(p)$. Thus, if the user has already typed *Give me the cities having population total greater than that of*, an element can follow only if it belongs to the domain of `dbo:populationTotal`.

$\mathbf{S_7}$: if p is the last accepted property and e is the element p is related to, a property q can be accepted if $q \in domain(e)$ and $q \in range(p)$. For instance, if the user has typed *"Who are the artists having birth place equal to their"*, she can complete the question with *"death place"* since the class `dbo:Artist` and the property `dbo:birthPlace` are, respectively, in the domain and range of property `dbo:deathPlace`.

$\mathbf{S_8}$: if p is the last accepted property and e is its related element, then property q can be accepted if $e \in domain(q)$ and $q \in domain(p)$. For instance, if the user has typed: *"Who are the artists with birth place equal to that of their,"* she can complete the question with *"spouse"* since the property `dbo:spouse` has the class `dbo:Artist` in its domain and the property `dbo:birthPlace` has property `dbo:spouse` in its domain.

To support the processing of the input just described, we index every element x of the KB using the fields

- *domainOf*: using the computed domains of the properties, we assign a value p to this field if $x \in domain(p)$;
- *rangeOf*: using the computed ranges of the properties, we assign a value p to this field if $x \in range(p)$.

Moreover, we index every property x by using also the field

- *domain*, since we assign to this field the values corresponding to all the classes and properties belonging to $domain(x)$.

According to the rules listed above, in particular those regarding state S_3, the entities belonging to $domain(x)$ do not need to be considered, since an entity is not an open variable to which constraints can be applied. This allows us to reduce significantly the size of the index.

In addition to the properties listed in the KB, `CANaLI` constructs, for each property having non-literal range, the corresponding inverse property. For instance, the property `dbo:team` associates people with sport teams, and thus users can ask "What is the team of Kobe Bryant?", inasmuch as `dbr:Kobe_Bryant` belongs to the domain of `dbo:team`. However the question "Who are the players of Los Angeles Lakers?" requires the use of the inverse of property `dbo:team`. Inverse properties are denoted by adding the suffix '[inverted]' to the label of the original property[10],

We created the Lucene index using the elements of the 2015 DBpedia release. The inverse properties were indexed as well. The time needed to create such an index, that requires to process the $\sim$130 million triples of the English DBpedia ($\sim$160 million, considering also those using inverted properties), is $\sim$55 minutes using a machine with 64GB of RAM. The obtained index is $\sim$1.4 GB large and can be easily stored in the main memory of a server, thus assuring a nearly instantaneous response to our search queries.

4. Experimental evaluation

A popular set of benchmarks has been used to measure the performance of QA systems, i.e., the questions of the sets of the latest three editions of the QALD (Question Answering over Linked Data) annual contest [4]. The benchmarks consist of sets of NL questions, each associated with a gold standard query in SPARQL, representing the translation of the question. The accuracy of the systems participating in the contest is measured by comparing the results obtained by the gold standard queries with the results obtained by the systems. We assessed the performances of `CANaLI` on both the questions over the general DBpedia KB [1], used in the challenges held since 2013, and the questions over the specialized KBs MusicBrainz [6] and 3 biomedical KBs: DrugBank [7], Diseasome [8], and SIDER [9]. The results obtained on each question the benchmarks are reported in [10]. We report here the results obtained on DBpedia questions of QALD-6, biomedical questions of QALD-4, and MusicBrainz questions of QALD-3. In Fig. 9, the column 'Proc.' shows the total number of questions for which the systems provided an answer; 'Rec.', 'Prec.'. and 'F-1' show the average recall, precision, and F-1 score, respectively, ignoring the questions that the

[10] While `CANaLI` already support assigning multiple labels to all the elements of the KB, the use of more descriptive words and synonyms obtained from existing paraphrase dictionaries ([21,35]) is a task left for future work

	Proc.	Rec.	Prec.	F-1	F-1 global
CANaLI	99	**0.89**	**0.89**	**0.89**	**0.88**
UTQA	100	0.69	0.82	0.75	0.75
KWGAnswer	100	0.59	0.85	0.70	0.70
NbFramework	63	0.85	0.87	0.86	0.54
SemGraphQA	100	0.25	0.70	0.37	0.37
UIQA	44	0.63	0.54	0.58	0.25

(a)

	Proc.	Rec.	Prec.	F-1	F-1 global
GFMed	25	0.99	**1.0**	0.99	**0.99**
CANaLI	23	**1.0**	**1.0**	**1.0**	0.92
POMELO	25	0.87	0.82	0.85	0.85
RO_FII	25	0.16	0.16	0.16	0.16

(b)

	Proc.	Rec.	Prec.	F-1	F-1 global
CANaLI	45	**1.0**	**1.0**	**1.0**	**0.9**
SWIP	33	0.77	0.77	0.77	0.51

(c)

Fig. 9. Results on QALD-6 DBpedia (a), QALD-4 biomedical data (b), and QALD-3 MusicBrainz (c) test sets, containing 100, 25, and 50 questions, respectively.

system is unable to understand and process. The final column ('F-1 global') reports the global F-score, computed assuming that the recall, precision, and F-score are 0 on questions the system is unable to process.

Performances of CANaLI. A total of 167 out of the 175 total questions of these three benchmarks (i.e., $\sim$95%) can be expressed in CANaLI. As we will discuss in more details later, the remaining 8 questions could not be expressed because CANaLI still lacks the following two features (i) sorting by an aggregate function (e.g., *Which musician wrote the most books?*), (ii) filtering by an aggregate function (e.g., *Which countries have more than ten volcanoes?*).

While CANaLI is clearly superior to the other systems in terms of precision and recall, we see that it was not able to return all the results produced by the gold-standard SPARQL query when this used union or disjunction to deal with equivalent properties in the KB. For instance, the gold standard query for the question *Which writers studied in Istanbul?* uses the disjunction of properties (dbo:almaMater and dbo:education) for associating the writer with the institution were s/he studied[11].

Answering questions on biomedical KBs, CANaLI came a close second to the GFMed system [34], which is a CNL system that is able able to process almost perfectly all the 25 questions, as shown in Fig. 9(b). However, GFMed is a QA system tailored for biomedical data, whreas CANaLI, that is a general-purpose system, that, as experiments have shown (Fig. 9 (a, c)), can be successfully used to query RDF KBs of broad and varied nature.

Given that CANaLI proved so effective, it is only natural that one should wonder about the extent in which restrictions imposed by the CNL makes it less user friendly than a full natural language interface. To answer this question. we will next contrast the original formulation of the questions in the gold standard with that used in CANaLI.

4.1. Working with a CNL Interface

So far we have focused on comparing CANaLI with other NL systems, but there has also been recent work that compares NL systems system that operated in different user-friendly modalities such visual WYSIWYG. For instance, SWiPE extends the QBE (query-by-example) approach to Wikedia pages, by activating its InfoBoxes so that users can easily enter conditions that define powerful queries to search DBpedia [15]. The SWIPE system is of interest here because in [16] the authors claim that it is at least if not more effective than other visual query interfaces to DBpedia proposed so far, and use the QALD-4 testbed to compare the effectiveness of SWiPE [15], against Xser [38], and the Wikipedia keyword search tool [11]. The results of that comparison Figure 10, along with the results obtained using CANaLI. Thus, CANaLI's CNL interface appears to be nearly as effective as the very powerful WYSIWYG interface of SWiPE. The study presented in [16] also addresses the more general question, on which additional benefits these DBpedia query interfaces provide to users over the basic ability of browsing the InfoBoxes of Wikipedia pages and searching for them using the current keyword search tool of

[11] One could also argue that the contest rules here penalized CANaLI unfairly, since its users can achieve complete recall by using two queries instead of one.

Wikipedia. Questions that can be easily answered by browsing and performing keyword search were called "trivial" in [16], and we will keep that nomenclature (although, unlike SWiPE, CANaLI can play a critical role in those "trivial" questions as well, since it does not require a web broswer). Thus, following [16], we divide the questions into 8 trivial (I–VIII) and 12 non-trivial questions (1–12). For each question, we show the original formulation and how it had to be rephrased in order to be accepted by CANaLI.

I. How often did Jane Fonda marry? Since CANaLI allows counting the number of values of a specified property, the supported question becomes: "What is the count of spouse of Jane Fonda?". (The plural form for "spouse" should be used but the plural form is currently only supported for class names.)

II. What is the official website of Tom Cruise? CANaLI does not accept this question. Indeed, while the property "official website" exists in DBpedia, the entity dbr:Tom_Cruise does not have it. However, users can obtain the correct result using the (arguably more natural) question: "What is the website of Tom Cruise?".

III. Who created Wikipedia? This must be rephrased as:"Who is the author of Wikipedia?". In fact, CANaLI requires questions to start with "What/Who is the," followed by a noun representing the property/class the user is looking for.

IV. What is the founding year of the brewery that produces Pilsner Urquell? In CANaLI the question becomes "What is the founding year of brewery of Pilsner Urquell?". In fact, *brewery* is a property for beers.

V. Which river does the Brooklyn Bridge cross? In CANaLI the user must type "What is the river crossed by Brooklyn Bridge?". In fact, some bridges have the property dbo:crosses, with label *crosses*. In this case the inverse property is used, for which we defined the label *crossed by*.

VI. How tall is Claudia Schiffer? Here too, as in III, the user must specify a property using a noun, i.e., "What is the height of Claudia Schiffer?".

VII. In which U.S. state is Mount McKinley located? The question in CANaLI becomes "What is the US state equal to the location of Mount McKinley?". The user could also ask "What is the location of Mount McKinley", but that will also return the park and the country of Mount McKinley.

VIII. When was the Statue of Liberty built? In DBpedia, the desired information is stored under the property *beginning date*. Thus the user must type "What is the beginning date of Statue of Liberty?", which is not very natural.

Let us now consider the 12 more complex questions.

1. Which books by Kerouac were published by Viking Press? In CANaLI the questions becomes "What are the books with author Jack Kerouac published by Viking Press?". In passing, observe *by* cannot be used as synonym for *author* since it is also used for many other properties–e.g., producer, and director.

2. Which U.S. state has the highest population density? In CANaLI the user has to type "What is the u.s. state having the greatest population density?".

3. How many films did Hal Roach produce? This is entered as: "What is the count of films produced by Hal Roach?".

4. Give me all federal chancellors of Germany. DBpedia contains the entity Chancellor_of_Germany which is the value of the property dbo:office for specific people. In CANaLI, it is thus possible to type "Give me the people with office Chancellor of Germany.".

5. Which states of Germany are governed by the Social Democratic Party? In CANaLI the question has to be input as "What are the German states governed by party Social Democratic Party of Germany?". However, this question does not achieve perfect recall, because some DBpedia entries use the abbreviation"SPD" for that party.

6. Which television shows were created by Walt Disney? The question has to be rephrased as "What are the television shows created by Walt Disney?".

7. Give me the websites of companies with more than 500000 employees. The question in CANaLI becomes "Give me the website of companies with number of employees greater than 500000?".

8. Give me all cities in New Jersey with more than 100000 inhabitants. The state of a city is represented in DBpedia through the property dbo:isPartOf. We assigned *in* as additional label to this property, which makes the question look quite natural: "Give me the cities in New Jersey with population total greater than 100000.".

9. Which actors were born in Germany? CANaLI uses an additional label, *born in*, to the property dbo:birthPlace, thus letting the user type "Who are the actors born in Germany?". Without the additional label, a less natural form for this question would have been "Who are the actors having birth place Germany?".

10. Give me all people that were born in Vienna and died in Berlin. This question can be written as 'Give

	Proc.	Right	R.	P.	F	F-g
CANaLI	**19**	16	0.92	**1.0**	0.95	**0.90**
SWiPE	18	**17**	**0.97**	**1.0**	**0.98**	0.88
Xser	16	14	0.9	0.89	0.9	0.72
Wikipedia	11	2	0.24	0.55	0.33	0.18

Fig. 10. Results on 20 questions extracted from the QALD-4 DBpedia training sets

me the people born in Vienna who died in Berlin.", which uses the additional phrase *"who died in"* as label for the property `dbo:deathPlace`.

11. In which country does the Nile start? Writing this question requires to find the property representing the countries in which rivers start. The user will probably try to type *start country*, which is not the correct label, according to DBpedia. However, the autocompletion system will suggest the correct label, i.e., *source country*. Thus, the question can be input as "What is the source country of the Nile?".

12. Which countries have more than two official languages? This question can not be expressed in CANaLI, since constraints using aggregate functions are not supported yet.

Therefore, the questions here discussed illustrate that most free-text questions require some reformulation in order to be accepted by CANaLI. However our experience suggests that users quickly gain that skill with the help of the completions suggested by the system–and even more so when they focus of a specific domain of interest rather than random topics. Also we expect significant improvements once we add the improvements discussed in Section 6.

5. Related Work

In this section we briefly review other systems for QA over RDF data, focusing our attention on systems that attended the QALD challenges [12].

Xser [38] works in two steps. In the first step, phrases are extracted from the question using a structured perceptron that can identify variables, entities, classes and relation phrases. By means of a semantic parser, the predicate-argument structure of phrases is derived, thus obtaining the structure of the query intention. In the second step, the semantic phrases are mapped against the elements of the KB (specifically, DBpedia) by using WikipediaMiner [12] for tentities, and an ad-hoc lexicon that maps classes and relation phrases to elements of DBpedia.

gAnswer [39] uses a data-driven approach that combines the query evaluation with disambiguation. By means of the Stanford parser [18] the question is processed, and from the dependencies so obtained, semantic relations are extracted by exploiting a paraphrase dictionary and by using some linguistic rules. The triples $\langle subject, predicate, value \rangle$ are obtained from a phrase of the question, and are initially mapped to several elements of the KB. The set of extracted semantic relations represents a semantic graph Q^S. Then, with G the RDF graph representing the KB, a sub-graph of G that matches Q^S is extracted, and disambiguation of the phrases is performed during this phase. This approach improves both the accuracy and the processing time.

CASIA [30] is a QA system that performs the joint resolution of mappings of phrases against the elements of the KB by means on an approach based on MLN (Markov Logic Networks). First, the question is processed by using the Stanford parser [18], thus obtaining a dependency tree and a POS (part of speech) annotation for every token. Then, phrases are created as sub-sequences of tokens, with maximum length equal to 3, and each phrase is mapped to a set of candidates, using anchors, redirections and disambiguations information from Wikipedia for entities. Then word2vec [13] is used as the similarity tool for classes, and PATTY and ReVerb are used for properties. Finally, ambiguities are jointly resolved by means of an MLN created on the basis of the dependencies between the phrases of the parse tree. This joint resolution of ambiguities yields the triples that are used to create the final SPARQL query.

Aqqu [29] also performs a joint resolution of ambiguities, such as those due to the same NL term representing several concept in the KB (polysemy). Therefore, Aqqu first identifies the entity candidates, then performs a matching of the NL question againgst some simple templates. This approach works well in the studied context, which involves "structurally simple" queries, i.e., 2 or 3 entities linked via a single relation. The relation is determined by the words used in the questions and by a supervised model built using a training set. The final query is chosen among the candidates by using a sophisticated ranking model.

[12] At the time of writing, we were not able to find papers about the new systems that attended QALD-6 three months ago; therefore, we will focus our attention on systems attending the previous editions of the challenge.

Intui3 [20] works by first splitting the question into chunks, then assigning to each chunk a set of possible candidate elements of the KB, and finally resolving the ambiguities while constructing the query. The natural questions are parsed by means of SENNA [19] and Stanford [18] parsers; i.e., using the POS tags, chunking, and NER obtained from the former and the lemmas obtained from the latter. The tokens obtained from SENNA are merged into chunks, and each chunk is mapped against the KB. Specifically, the DBpedia Lookup service is used to find mappings for entities, and WordNet is used to map the properties using the *Hirst and St-Onge* similarity measure [31]. Each chunk is thus assigned to a set of candidates. The correct candidate is chosen while creating the query. Then, the most probable candidate pair is chosen for each pair of adjacent chunks using an approach that scans the chunks from right to left. At the end of the scanning, all the chunks are assigned to exactly one candidate and the relationships among them are defined, and used to create the final SPARQL query.

RTV [23], uses an HMM (Hidden Markov Model) to find the best mapping of the extracted syntactical elements against the elements of the KB. This is achieved by first processing the question using the Chaos parser [17], whereby a chunk-based dependency graph is constructed (Chaos was enriched with a set of proper nouns extracted from DBpedia). Apache Lucene is then used to retrieve ranked elements of the KB from the chunks of the question. The most likely overall matching is found using an HMM, where the observation set is defined by the chunks of the question, and the state set is defined by the elements of the KB to which the chunks can be mapped. Then the emission and transition matrices are respectively computed on the fly according to (i) the similarity between the set on chunks and elements of the KB, and (ii) the semantic relationships in DBpedia. The sequence of observations is defined by the dependency graph, starting from its root. The best mapping is thus obtained by means of the Viterbi algorithm.

Squall2sparql [22], is a CNL system that translates queries written in SQUALL into SPARQL. The translation is based on about 100 rules of a Montague grammar. The chunks of the SQUALL sentence must be annotated by the user, and written in a form that enables the direct mapping to elements of the KB. SQUALL enables users to both query and update the KB, and uses all the SPARQL features. Therefore, its Squall2sparql interface to RDF KBs seems to push the CNL idea to its extreme, inasmuch as it achieves the greatest expressive power, but the need to manually annotate the chunks of the sentences severely limits the usability of SQUALL— a fact recognized by the author who proposes the use of a meta-level interface to guide the user in writing the annotate questions.

GFMed [34] is a CNL system specialized for the biomedical domain. It is based on the Grammatical Framework [37], which enables to define grammars by means of an abstract syntax and one or more concrete syntaxes. The abstract syntax defines the concepts that can be expressed as non-terminal symbols and the rules for their composition. The concrete grammar defines how the trees specified through the abstract syntax are linearized into sentences of a specific language (e.g., English, SPARQL, etc.). The possibility of defining more concrete syntaxes allows GF to serve as a powerful tool for translating sentences from one language to another. The GFMed system consists of a GF program that defines a grammar allowing to pose questions over the KBs DrugBank, Diseasome and SIDER. The GF program is completed with a post-processing procedure for handling literals, that can not be defined using the concrete syntax. GFMed proved to be very accurate on the biomedical questions of QALD-4. The main limitation of this approach is the need to write the grammar rules for all the concepts of the underlying KBs, which can be a very hard task for large KBs such as DBpedia.

6. Conclusions and future work

This paper presented CANaLI, a natural language QA system that combines effectiveness with simplicity. In fact, while achieving levels of expressive power and accuracy that advance the current state-of-the-art, exemplified by the systems discussed in Section 5, CANaLI achieves greater simplicity both at the logical and the system levels. A first reason for this simplicity is the use of a CNL and a second one is the compactness of the FSA-based recognizer we designed for CANaLI. It is important to remark that CANaLI's CNL interface does not limit its expressive power; this is demonstrated by its performance on the QALD benchmarks [4], where all the queries in the benchmark were expressed (except for one requiring conditions on aggregates, a feature that is now being added to CANaLI). The generality of the approach taken was also confirmed by the fact that, besides DBpedia, CANaLI worked very well on other KBs, including

MusicBrainz [6] and DrugBank [7], Diseasome [8] and SIDER [9].

While CANaLI does not suffer from expressive power and generality issues because of its reliance on CNL, its restricted CNL syntax can produce an interface that is less user-friendly, particularly for beginners, compared to an unrestricted NL interface. However, the results we obtained with CANaLI are encouraging, as illustrated by the long list of testbed questions we have presented. Out of those many questions, the only one that required a truly stilted formulation was question VIII expressed as follows: “What is the beginning date of Statue of Liberty?". The source of this problem is the fact that, in English and many other languages, people refer to the “beginning date" of a building ot monument with words such as “inauguration date," “built date," or "dedication date," which denote concepts that are similar but not identical. This has so far discouraged us from applying the simple solution of allowing the use of synonyms in lieu of the internal names used in DBpedia. In fact, the current version of CANaLI only uses synonyms in a very conservative way, trying to avoid any risk of ambiguity. While, so far we only applied the synomym solution in very simple cases, in the future, we plan to explore a more aggressive usage of synonyms. We expect that this will improve usability (particularly, in application domains that employ special jargons) but still avoid ambiguities, and we can therefore remain in full compliance with CANaLI's design principles.

In summary, besides bringing QA NL interfaces to new levels of performance in terms of precision, recall, expressive power and generality, CANaLI has introduced the concept of autocompletion for QA on KBs and demonstrated its important in applications. In fact, CANaLI's autocompletion function reveals ambiguities in the NL sentence—see e.g. the “population” question in Figure 2. When presented with this question, CANaLI will show all alternative interpretations that are semantically and syntactically correct and then let the user select the intended one. A even more common and important situation is when, as the user types the question, the autocompletion system halts and tells the user that no mapping is possible between sentence entered so far and the underlying KB. As a result, the user will have to review and revise the names of properties and entities, and the CNL connectives used in the question to revise the question into one that is unambiguously interpreted by CANaLI. The effectiveness of the guidance and assistance so provided are greatly enhanced by the fact that the autocompletion system is extremely fast and thus provides real-time on-line assistance to users. This remarkable speed of CANaLI follows from the simplicity of the FSA-based recognizer it uses, and the compactness of the index and code it employees.

References

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

[2] http://linkeddatacatalog.dws.informatik.uni-mannheim.de/.

[3] http://www.w3.org/TR/sparql11-overview/.

[4] http://www.sc.cit-ec.uni-bielefeld.de/qald.

[5] http://lucene.apache.org/.

[6] http://musicbrainz.org/.

[7] http://www.drugbank.ca/.

[8] http://wifo5-03.informatik.uni-mannheim.de/diseasome/.

[9] http://sideeffects.embl.de/.

[10] http://canali.link.

[11] https://en.wikipedia.org/wiki/Special:Search.

[12] http://wikipedia-miner.cms.waikato.ac.nz/.

[13] https://code.google.com/p/word2vec/.

[14] R. Alur and P. Madhusudan. Visibly pushdown languages. In *Proceedings of the Thirty-sixth Annual ACM Symposium on Theory of Computing*, STOC '04, pages 202–211, 2004.

[15] M. Atzori and C. Zaniolo. Swipe: searching wikipedia by example. In *Proceedings of the 21st World Wide Web Conference*, 2012.

[16] M. Atzori and C. Zaniolo. Expressivity and accuracy of by-example structured queries on wikipedia. In *24th IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE Workshops 2015, Larnaca, Cyprus, June 15-17, 2015*, pages 239–244, 2015.

[17] R. Basili and F. M. Zanzotto. Parsing engineering and empirical robustness. *Nat. Lang. Eng.*, 8(3):97–120, June 2002.

[18] D. Chen and C. D. Manning. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar*, pages 740–750, 2014.

[19] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *J. Mach. Learn. Res.*, 12:2493–2537, Nov. 2011.

[20] C. Dima. Answering natural language questions with intui3. In *Working Notes for CLEF 2014 Conference, Sheffield, UK, September 15-18, 2014.*, pages 1201–1211, 2014.

[21] A. Fader, S. Soderland, and O. Etzioni. Identifying relations for open information extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, pages 1535–1545, 2011.

[22] S. Ferré. SQUALL: the expressiveness of SPARQL 1.1 made available as a controlled natural language. *Data Knowledge Engineering*, 94:163–188, 2014.

[23] C. Giannone, V. Bellomaria, and R. Basili. A hmm-based approach to question answering against linked data. In *Working*

Notes for CLEF 2013 Conference , Valencia, Spain, September 23-26, 2013., 2013.

[24] B. F. Green, Jr., A. K. Wolf, C. Chomsky, and K. Laughery. Baseball: An automatic question-answerer. In *Papers Presented at the May 9-11, 1961, Western Joint IRE-AIEE-ACM Computer Conference*, 1961.

[25] P. Gupta and V. Gupta. A survey of text question answering techniques. *International Journal of Computer Applications*, 53(4):1–8, 2012.

[26] J. Guyonvarch and S. Ferré. Scalewelis: a scalable query-based faceted search system on top of SPARQL endpoints. In *Working Notes for CLEF 2013 Conference , Valencia, Spain, September 23-26, 2013.*, 2013.

[27] R. Hahn, C. Bizer, C. Sahnwaldt, C. Herta, S. Robinson, M. Bürgle, H. Düwiger, and U. Scheel. Faceted wikipedia search. In *Business Information Systems, 13th International Conference*, 2010.

[28] L. Han, T. Finin, and A. Joshi. Schema-free structured querying of dbpedia data. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, 2012.

[29] B. Hannah and H. Elmar. More accurate question answering on freebase. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM '15, pages 1431–1440, New York, NY, USA, 2015. ACM.

[30] S. He, Y. Zhang, K. Liu, and J. Zhao. Casia@v2: A mln-based question answering system over linked data. In *Working Notes for CLEF 2014 Conference, Sheffield, UK, September 15-18, 2014.*, pages 1249–1259, 2014.

[31] G. Hirst and D. St-Onge. Lexical chains as representations of context for the detection and correction of malapropisms. 1998.

[32] T. Kuhn. A survey and classification of controlled natural languages. *CoRR*, abs/1507.01701, 2015.

[33] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer. DBpedia - a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web Journal*, 6(2):167–195, 2015.

[34] A. Marginean. Gfmed: Question answering over biomedical linked data with grammatical framework. In *Working Notes for CLEF 2014 Conference*, pages 1224–1235, 2014.

[35] N. Nakashole, G. Weikum, and F. M. Suchanek. PATTY: A taxonomy of relational patterns with semantic types. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, EMNLP-CoNLL 2012, July 12-14, 2012, Jeju Island, Korea*, pages 1135–1145, 2012.

[36] S. R. Petrick. On natural language based computer systems. *IBM J. Res. Dev.*, 20(4):314–325, July 1976.

[37] A. Ranta. Grammatical framework. *J. Funct. Program.*, 14(2):145–189, 2004.

[38] K. Xu, Y. Feng, and D. Zhao. Answering natural language questions via phrasal semantic parsing. In *Working Notes for CLEF 2014 Conference*, 2014.

[39] 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 *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 313–324, 2014.