Resolving Quantifier and Number Restriction to Question OWL Ontologies

Shamima Mithun, Leila Kosseim, and Volker Haarslev
Concordia University
Department of Computer Science and Software Engineering
E-mail: {s_mithun, kosseim, haarslev}@cse.concordia.ca

Abstract

This paper describes an approach to resolve quantifiers and number restrictions in natural language questions to query ontologies. Incorporating this feature enables natural language query interfaces to capture a wider range of user queries. To deal with quantifiers and number restrictions, we analyzed a corpus of such questions and derived constraints at the syntactic level to recognize and parse them. The approach was implemented and evaluated through a system called ONLI$^+$. Our method has been evaluated by conducting different experiments using the Mean Reciprocal Rank (MRR) measure. Experimental results show that this feature has been incorporated into ONLI$^+$ without degrading its performance in terms of transforming natural language queries into the nRQL queries, but definitely increases the expressivity of the user. To the best of our knowledge no other natural language interface to query ontologies can deal with quantifiers and number restrictions.

1. Introduction

With the growth of the Semantic Web, knowledge representation repositories such as ontologies are becoming more common. As a result, it becomes increasingly more relevant to support simple query access to these complex knowledge repositories. Existing tools allowing users to query and reason over ontologies [1, 2, 3] require users to write queries using specific query languages. Irrespective of languages it is difficult for common users to write queries especially conjunctive queries. A conjunctive query is defined as a formula built from atomic formulae, conjunctions and existential quantifiers. The use of conjunctions makes the creation of the query very difficult using any query languages. On the other hand, it is very straight forward for a user to express queries in natural language. Many domain experts who contribute to the ontology engineering process are limited by time constraints and have difficulty in writing queries using query languages. To help these domain experts who may have little or no knowledge of the structure of the ontology, we developed ONLI (Ontology Natural Language Interaction)[8] and its successor, ONLI$^+$. ONLI$^+$ is a natural language question answering system used as front-end to the RACER reasoner [3] and RACER's query language, nRQL [4]. ONLI$^+$ supports queries in unrestricted natural language. It assumes that the user is familiar with the ontology domain but is not required to know how to write queries using the nRQL query language. The system will transform the user natural language queries into nRQL query formats. This natural language ontology-based query is equivalent to searching the whole domain quickly and efficiently for an interactive search.

ONLI$^+$ can handle three types of queries with quantifiers and number restrictions:

1. unary concept queries with quantifier – e.g. "Find *5* fungi"
2. binary role queries with quantifier – e.g. "Find *5* fungi that have been reported to have Pectinase"
3. binary role queries with number restriction – e.g. "Find all fungi that have been described to have more than 3 enzymes"

Incorporating quantifiers into unary and binary query atoms and number restrictions into binary query atoms enables the system to capture commonly used natural language queries, and thus supports a wider range of user queries. To our knowledge, no existing natural language interfaces to query ontologies deals with quantifiers and number restrictions. Experimental results show that incorporating quantifiers and number restrictions did not degrade ONLI$^+$'s performance with regard to transforming natural language queries into the nRQL queries.

2 Background: nRQL Query Atoms

The nRQL is a highly expressive description logic-based query language for the RACER reasoner. The nRQL uses a Lisp based syntax and the general structure of a query is composed of a query head e.g. *retrieve (?x)* upon which variables used in the body are projected, e.g., *(?x Fungi)*, where *(retrieve (?x) (?x Fungi))*, queries for instances of the concept Fungi. Unary concept query atoms and binary role query atoms are described as:

Unary Concept Query Atoms Using a unary concept query atom, it is possible to find all instances of a particular concept, for example:

Find all fungi
nRQL atom: *(retrieve (?x) (?x fungi))*

or whether a specific instance belongs to a particular class or concept, for example:

Is Protease an Enzyme?
nRQL atom: *(retrieve ()(Protease Enzyme))*

Binary Role Query Atoms Binary Role query atoms search for the relation between pairs of instances, for example:

What can be used in what?
nRQL atom: *(retrieve (?x ?y) (?x ?y can_be_used_in))*

In this type of query, it is possible to refer to a particular concept or instance name instead of using variables, for example:

What is used in baking?
nRQL atom: *(retrieve (?x) (?x baking can_be_used_in))*

where *baking* is a concept name. By using instance and concept names, various forms of binary role query atoms can be generated.

In addition, complex query atoms can be created by combining unary concept query atoms and binary role query atoms.

3. The ONLI$^+$ Question Answering System

ONLI$^+$ takes user input in natural language, translates the input into nRQL query format, submits the query to RACER, and presents the RACER output to the user after transforming it into natural language. The architecture of ONLI$^+$ consists of: Syntactic Analysis, Ontology Mapping, and Query Interface to RACER. A full description can be found in [8].

3.1. Syntactic Analysis

Using the Minipar parser [9], the user question is parsed to recognize syntactic constituents; mainly predicates and noun phrases. From the parse tree, predicates and their arguments are extracted and represented into a predicate-argument structure made of the triplet: <*argument, predicate, argument*>.

For example, for "Which vendor sells enzyme products?", the predicate-argument structure produced will be: <*arg1:vendor pred:sell arg2:Enzyme Product*> where, *arg1* and *arg2* represent the arguments and *pred* represents the predicate.

3.2. Ontology Mapping

This module tries to match each constituent of the structure to roles, concepts and instances in the ontology. Predicates are mapped to roles, arguments are mapped to concepts and instances, and empty arguments are mapped to variables. As predicates do not contain much domain terminology, WordNet [12] is used to match the predicates with the ontology role by unifying its stem with the role of the ontology. For mapping arguments, lexical similarity measures are used based on n-gram overlap. Since arguments (noun phases) tend to be much more domain specific (e.g., enzyme names, fungi names), a general purpose lexical resource such as WordNet cannot be used to match arguments.

3.3. Query Interface to RACER

The next module generates nRQL queries using the mapped roles, concepts, and instances and submits this nRQL query to RACER. The nRQL supports the <argument, predicate, argument> triple format query where an argument could be a concept, an instance or a variable and a predicate could be a role or be empty. For a particular predicate structure, different nRQL queries can be generated with their associated confidence level. The best scoring nRQL queries are sent to RACER. At the end, this module transforms the answers retrieved from RACER into natural language, using templates, and presents these to the user.

4. Quantifiers and Number Restrictions

The ONLI system supports queries using unary concept query atoms and binary role query atoms e.g., “Find all fungi”, “Find all fungi that have been reported to have Pectinase”, where the first query is a unary concept query and the second query is a binary role query. To make ONLI more flexible, we incorporated quantifiers and number restrictions into ONLI$^+$. To understand quantifiers and number restrictions let us first illustrate what we are addressing: quantifiers are linguistic expressions like *some*, *few* and *cardinal values* (e.g., 3, 5). Quantifiers are applicable to both types of queries: unary concept queries and binary role queries. For example, “Find *5* fungi” is a unary concept query with a quantifier and “Find *5* fungi that have been reported to have Pectinase” is a binary role query with a quantifier. Number restrictions are linguistic expressions such as *more than*, *less than*, which are only applicable to binary role queries. For example, “Find all fungi that have been reported to have *more than 2* enzymes” is a binary role query with number restriction. This section discusses how ONLI$^+$ deals with quantifiers and number restrictions.

4.1. Quantifiers in Unary and Binary Queries

The addition of quantifier handling to unary and binary queries enables ONLI$^+$ to handle queries like: “Find *5* fungi” or “Find *5* fungi that have been reported to have Pectinase” or “Find *some* vendors who are selling enzyme products that can be used in baking bread”, where the first query is a unary concept query with a quantifier, the second query is a binary role query with a quantifier, and the third query is a complex query with a quantifier.

To deal with quantifiers in unary and binary queries, ONLI$^+$ extracts the quantifier value from the user query and post-processes query results based on it. The system needs to recognize quantifiers from the parsed tree during the Syntactic Analysis phase (Section 3.1). To do this, we analyzed a corpus of 36 questions and identified that all quantifiers are attached to an argument (noun phrase) and this argument is linked to a predicate (verb). For example, for the binary query with a quantifier “Find 2 vendors who sell enzyme products” the generated parse tree is shown in Figure 1. In the parse tree, the cardinality value *2* (noun) is syntactically related to the noun *vendor* and this noun is itself related to the verb *find* (same relations hold for unary queries with quantifiers). As a result, the system will accept this value as a quantifier.

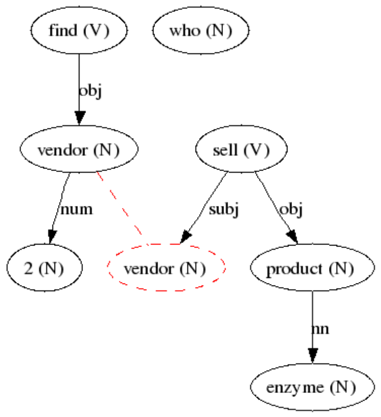

Figure 1. Minipar parse tree for the question *Find 2 vendors who sell enzyme products.*

To identify quantifiers, several syntactic links are used, notably numerals and determiners. Then ONLI$^+$ uses the quantifier value *2* to prune the list of query results. If the ontology does not have enough instance names that match the query then the system will show all available instances. For example, if the ontology has only one vendor name that matches with the query requirement, then the system will show that instance only. In ONLI$^+$, we mapped the values for quantifiers other than cardinal values as: *some* to at least 1, *few* and *a few* to at least 1 and *at most* to 5, but this can be changed by the user.

Note however, that from our development corpus, Minipar was unable to generate a correct parse tree for these 2 questions: “Give me 4 commercial enzyme products that are being sold by Biocatalysts Co.” and “What enzyme is being used in exactly 2 pulp and paper manufacturing industry”. Since the heuristics are based on syntactic constraints, ONLI$^+$ was therefore not able to correctly translate these questions into nRQL.

4.2. Number Restriction on Binary Role Query Atoms

ONLI$^+$ extends the binary role query atoms handled by ONLI with number restrictions. For example, “Find all fungi that have been described to have more than 3 enzymes”. In this example, we are putting a constraint or a number restriction *more than 3* on the range of the role *describe_to_have* which is *enzyme*. We considered 9 types of number restrictions: *less than*, *less than equal to*, *more than*, *greater than*, *greater than equal to*, *at least*, *at most*, and *exactly* with a cardinality value. We also consider *some* as number restriction, which does not require any cardinality value (*some* is implicitly greater than 0).

To deal with number restrictions on binary roles ONLI$^+$ first analyzes the parsed question in the Syntactic Analysis phase. Then it extracts the number restriction used in the query and its associated cardinality value and represents it as a triplet structure:

<modifier, cardinality, object>

where *modifier* specifies number restriction type, *cardinality* describes the cardinal value, and *object* describes which argument is modified by the number restriction.

For example, for the query:
"Find all fungi that have been reported to have less than 2 enzymes"
the following modifier-cardinality-object structure will be generated:

<modifier:less cardinality:2 object:enzymes>

To extract these structures, we analyzed a corpus of 42 questions and concluded that the cardinality value (zero in the case of the number restriction *some*) is typically attached to a set of predefined expressions using number restrictions (e.g., *less_than*) with an argument (e.g.,*enzyme*); and this argument is attached to a predicate (e.g., *have*).

These syntactic relations are shown in Figure 2. In the parse tree, the cardinality value 2 is related to the number restriction "less than" and the noun "enzyme" is related to the verb "have".

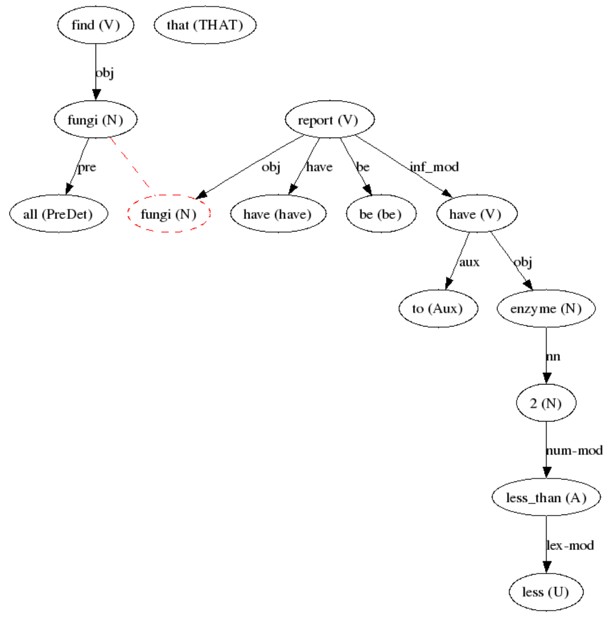

Figure 2. Minipar parse tree for the question *Find all fungi that have been reported to have less than 2 enzymes.*

However, here again, the syntactic constraints cover all cases seen in the development corpus, but when Minipar is not able to parse the question correctly, then ONLI$^+$ is not able to translate the query properly. For example, for the number restricted query "Find all fungi that have been reported to have at most 2 enzymes" the argument *enzymes* will not be attached to the predicate *reported to have*. In this case, Minipar will generate two separate sub-trees, where the argument is in one tree while the predicate is in the other. As a result, the argument will not be attached to the verb which is required for our heuristic.

Once the system has extracted the required information from the user query, it generates an nRQL query corresponding to that number restriction type. For example,
"Find all fungi that have been reported to have less than 2 enzymes"
will be translated into the nRQL query:
(retrieve(?x) (and(and(?x fungi)(?y enzyme))(?x (atmost 2 reported_to_have ?y))))
where, *at-most* is the nRQL equivalent keyword for *less than*, and 2 is the cardinality value. A direct mapping scheme is used to convert number restriction types to nRQL keywords.

Finally, this newly created query is executed by RACER and its results are returned. If a quantifier is attached with this number restricted query, ONLI$^+$ will post-process results retrieved from RACER and output the corresponding number of matches.

5. Evaluation

To evaluate the quantifier and number restriction approach, we tested our system with 30 pairs of questions and their associated nRQL queries for two genomic related ontologies, the FungalWeb [13] and the MutationMiner ontology [15]. For evaluation, the nRQL queries generated by ONLI$^+$ were compared for query equivalence with the gold standard queries created manually. For comparison, the system-generated queries are ranked according to their confidence score. Then, for each question the reciprocal rank of the first correct answer is computed. If none of the generated query from the system generated query list is equivalent to the gold standard, a score of 0 was given. Otherwise the score is equal to the reciprocal of the rank of the first correct answer. The final result is the average reciprocal score of all questions. This measure is known as mean-reciprocal rank (MRR) score which is a standard measure for question answering [14].

5.1. The FungalWeb and the MutationMiner Ontologies

The two ontologies used in the evaluation are FungalWeb and MutationMiner. The FungalWeb ontology, specified in OWL-DL [5], models conceptualization of multiple domains. It represents knowledge about the fungal taxonomy, enzymology, and industrial enzyme application [13]. This conceptualization enables one to model fungal species, enzyme names, enzyme product name, vendor name as instances of this integrated domain.

The MutationMiner ontology [15], also specified using OWL-DL, represents text mining result from the biology domain, especially the mutational impact on protein properties. This ontology contains information on three main class of the biology domain: protein, mutation, and organism. It also contains information to represent changes in protein properties. Table 1 shows the size of the two ontologies. MutationMiner is a small ontology whereas the FungalWeb is much larger in size (contains many more concepts, instances, and roles), and concept definitions in the FungalWeb are also more complex.

Table 1. Ontologies Used in the Evaluation

	FungalWeb	Mutation Miner
#concepts	3616	51
#instances	11,163	675
#roles	142	41

5.2. Results and Analysis

To evaluate our method for quantifiers and number restrictions, we prepared two sets of questions namely Set1 and Set2. Both question sets are related to the FungalWeb ontology and the MutationMiner ontology, but Set1 contains no questions with quantifiers and number restrictions while Set2 contains the same questions from Set1 but extended with quantifiers and number restrictions. In each set, we had 30 questions. We evaluated ONLI$^+$ with both sets of questions separately and obtained an MRR value of 0.35 for Set1 and an MRR value of 0.33 for Set2 (see Table 2). Since the MRR value for both sets of questions were almost the same, the use of quantifiers and number restrictions into unary and binary query atoms does not seem to lower the system's performance, yet enables the system to handle many new types of queries.

Table 2. Experimental Results (MRR)

	Unary only	Binary only	Unary & Binary
#Questions	9	21	30
Set 1: No Quantifiers and Number Restrictions	0.83	0.17	0.35
Set 2: Quantifiers and Number Restrictions	0.78	0.11	0.33

When we analyzed the results further, we noticed that binary query atoms are more difficult to answer than unary queries. Our testing set was not balanced: in each set of questions, out of 30 questions, 21 were in the form of binary query atoms and only 9 were unary queries. The MRR for unary queries is 0.78-0.83 whereas for binary query types the MRR is 0.11-0.17 (see Table 2). We suspect that the reason behind this is the simpler structure of unary query atoms which allows Minipar to generate more correct syntactic parses and makes the predicates and arguments mapping to ontology roles and concepts easier and nRQL query generating task simpler for the system.

With both sets of questions, 11 questions were actually answered, while 19 were not. Given the MRR of 0.35, this means that when a question is answered, then the first answer is the correct one. Table 3 shows the sources of error of the system for the questions that were not answered. As the table shows, a wrong syntactic parse taken as input accounts for about half the errors; but ONLI$^+$ itself stills needs to be improved.

Table 3. Sources of Errors for Unanswered Questions

	Minipar Error	ONLI$^+$ Error
Set 1: No Quantifiers and Number Restrictions	37%	63%
Set 2: Quantifiers and Number Restrictions	47%	53%

6. Related Work

Question answering systems with a user friendly interface and which allows queries in unrestricted language are still very rare. AquaLog [10] and its successor PowerAqua [11] are ontology-driven question answering systems. They support queries in unrestricted natural language and are ontology independent. They are built on the GATE NLP platform [6] and use several measures to compute the similarity between the terms in the questions and roles and concepts of the ontologies. For this purpose they use string-based algorithms and WordNet [12]. PowerAqua provides facilities to query multiple ontologies. Our ONLI$^+$ system is similar to these two systems but ONLI$^+$ can handle quantifier and number restriction queries which are absent in those two systems. Querix [7] is another ontology-based question answering system that translates generic natural language queries into SPARQL. In case of ambiguities, Querix relies on clarification dialogs with users. In this process users need to disambiguate the sense from the system-provided suggestions. In contrast to ONLI$^+$, Querix does not support queries across multiple ontologies. Moreover, in [7] it is not mentioned what types of questions are handled by the system.

7. Conclusions and Future Work

ONLI$^+$ is a portable ontology-driven question answering system which accepts questions in unrestricted natural language. It can handle queries with quantifiers and number restrictions. To deal with these linguistic expressions, we analyzed a corpus of such questions and derived constraints at the syntactic level to recognize and parse them. Preliminary evaluation shows that incorporating this feature into the system does not degrade its overall performance, but definitely increases its expressivity.

Currently, the syntactic heuristics used to parse quantifiers and number restrictions were tested on a very small and unbalanced corpus of 30 questions. As future work, we definitely need to scale up the evaluation and use a distribution of questions that is representative of what users actually ask.

The next interesting challenge for ONLI$^+$ is to handle queries with negations (e.g. "Find all fungi which *do not* have Cellulase") or queries using conjunction or disjunction (e.g., "Find all fungi which have pectinase *and/or* Cellulase"). We need to explore how to add these linguistic phenomena namely conjunction, disjunction, and negation into ONLI$^+$.

References

[1] FaCT++: http://owl.man.ac.uk/factplusplus/. (last accessed 2007-07-25).

[2] Pellet: http://pellet.owldl.com/. (last accessed 2007-01-25).

[3] Racer Systems GmbH & Co. KG : RacerPro User's Guide Version 1.9.

[4] nRQL: The new racer query language. http://www.racer-systems.com/products/racerpro/manual.phtml. (last accessed 2007-07-16).

[5] OWL: http://www.w3.org/2004/OWL/. (last accessed 2007-02-16).

[6] H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. GATE: an architecture for development of robust HLT applications. In *40th Annual Meeting of the ACL*, pages 168–175, 2001.

[7] E. Kaufmann, A. Bernstein, and R. Zumstein. Querix: A natural language interface to query ontologies based on clarification dialogs. In *5th International Semantic Web Conference (ISWC 2006)*, pages 980–981, 2006.

[8] L. Kosseim, R. Siblini, C. Baker, and S. Bergler. Using selectional restrictions to query an OWL ontology. In *The 2nd International Conference on Formal Ontology in Information Systems*, Baltimore, Maryland (USA), November 2006. FOIS-2006.

[9] D. Lin. Dependency based evaluation of MINIPAR. In *Workshop on the Evaluation of Parsing Systems*, Granada, Spain, 1998.

[10] V. Lopez, E. Motta, and M. Pasin. AquaLog: An ontology-portable question answering system for the semantic web. In *European Semantic Web Conference, Crete*, 2005.

[11] V. Lopez, E. Motta, and V. Uren. PowerAqua: Fishing the semantic web. *Lecture Notes in Computer Science : The Semantic Web: Research and Applications*, pages 393–410, 2006.

[12] T. Pedersen, S. Patwardhan, and J. Michelizzi. WordNet::Similarity - measuring the relatedness of concepts. In *Nineteenth National Conference on Artificial Intelligence (AAAI-04)*, San Jose, California(USA), 2004.

[13] Arash Shaban-Nejad, Christopher J.O. Baker, Greg Butler, and Volker Haarslev. The FungalWeb Ontology: Semantic Web Challenges in Bioinformatics and Genomics. In *4th International Semantic Web Conference (ISWC)*, Lecture Notes in Computer Science 3729, pages 1063–1066, Galway, Ireland, 2004.

[14] E.M. Voorhees. Overview of the TREC 2001 Question Answering Track. In *Proceedings of The Tenth Text REtrieval Conference (TREC-X)*, pages 157–165, Gaithersburg, Maryland, 2001.

[15] R. Witte, T. Kappler, and C. J. O. Baker. Ontology design for biomedical text mining. *Semantic Web: Revolutionizing Knowledge Discovery in the Life Sciences, Springer Verlag*, 2006.