Development of an Access Method to Menu Ontology Data through Converting Natural Language into SPARQL Queries

Yuri Murayama[1], Ichiro Kobayashi[1], Takeshi Morita[2], Yukiko Nakano[3], and Takahira Yamaguchi[2]

[1] Ochanomizu University 2-1-1 Otsuka Bunkyo-ku, Tokyo 112-8610 Japan
{murayama.yuri,koba}@is.ocha.ac.jp

[2] Keio University 3-14-1 Hiyoshi Kohoku-ku,Yokohama-shi,223-8522 Japan
t_morita@keio.jp, yamaguti@ae.keio.ac.jp

[3] Seikei University 3-3-1 Kichijoji-kitamachi, Musashino-shi, Tokyo 180-8633 Japan
y.nakano@st.seikei.ac.jp

Abstract. In recent years, vast amount of knowledge in various fields is open to the public on the Web. This knowledge will be helpful for understanding natural language if it is used in a natural language processing system. With this background, we propose a method to generate SPARQL queries by reference to the knowledge iteratively. Besides, as an experiment to evaluate our proposed method, we plan to introduce our work to a service interaction at a robot cafeteria.

Keywords: Natural language · Semantic parsing · SPARQL query generation.

1 Introduction

In recent years, vast amount of knowledge in various fields is open as Linked Open Data to the public on the Web. This knowledge will be helpful for understanding natural language if it is used as the background knowledge in natural language processing systems and thus various approaches to connect natural language with such knowledge have been studied [1, 3–5]. As a common problem for their studies, they aim to convert natural language into query language which is enable to access data repositories in order to access from natural language expression to the knowledge represented in the formal form such as Linked Open Data. Based on the approaches of prior studies, in this study, we propose a method that generates SPARQL queries by iteratively referencing to the knowledge compiled as Linked Open Data. As a concrete experiment for the method, we suppose a situation where a robot answers customers' questions at a cafeteria and attempt to automatically convert customers' questions into SPARQL queries to answer the questions through accessing knowledge.

2 Related Studies

To connect natural language to knowledge bases such as the ones represented with the form of Linked Open Data, e.g., DBpedia, etc., many studies have so far done. In Wang et al. [1], they proposed a method to convert natural language inquiry to an RDF triple for the result of the dependency structure parsing on natural language inquiry by applying the vocabularies, which are used to describe knowledge, defined in machine readable dictionaries such as WordNet [2], and defined by users. Zou et al. [3] developed knowledge graph called 'semantic query graph' by connecting multiple RDF triples, and raised the accuracy of understanding natural language inquiry by converting knowledge fragments into semantic query graph and using it to understand natural language input. Suzuki et al. [4] proposed a method to convert the result of parsing natural language inquiry into logical formula, and then generate a SPARQL query. Cui et al. [11] designed a kind of question representation by learning templates over a billion scale knowledge base and a million scale QA corpora.

There are many cases where vocabulary mismatch happens between the expressions in natural language inquiry and the entities of knowledge bases. As a solution for this problem, Shekarpour et al. [5] proposed a method for automatic rewriting input queries on graph-structured RDF knowledge bases by employing a Hidden Markov Model to determine the most suitable derived words from linguistic resources. As the approaches using deep neural network to transform natural language inquiry into a query to access large-scale knowledge bases, Soru et al. [6] proposed Neural SPARQL Machines, end-to-end deep architectures to translate any natural language expression into sentences encoding SPARQL queries, using sequence-to-sequence architecture [7]. In the method of Ferreira Luz et al. [8], their proposal is based in two learning steps: The first step generates a vector representation for the sentence in natural language and SPARQL query, and the second step uses this vector representation as input to a neural network (LSTM with attention mechanism) to generate a model able to encode natural language and decode SPARQL.

Unger et al. [9] presented an approach that relies on a parse of the question to produce a SPARQL template that directly mirrors the internal structure of the question. Abujabal et al. [10] presented a system that automatically learns utterance-query templates solely from user questions paired with their answers. Arenas et al. [12] considered the relation between well-designed queries and the semantic notion of weak monotonicity. Ferre et al. [13] introduced SQUALL, a controlled natural language for querying and updating RDF graphs. Ebisu et al. [14] proposed a method to extract knowledge from texts by extending the analogy task to use more positive example. Zhang et al. [15] presented a novel joint model uniting two procedures, alignment construction and query construction into a uniform framework.

Compared to the above prior studies, our study particularly focuses on finding a relation between entities in generating a SPARQL query by iteratively referencing to the knowledge compiled as Linked Open Data.

3 Proposed Method

The overview of our proposed method is illustrated in Fig. 1.

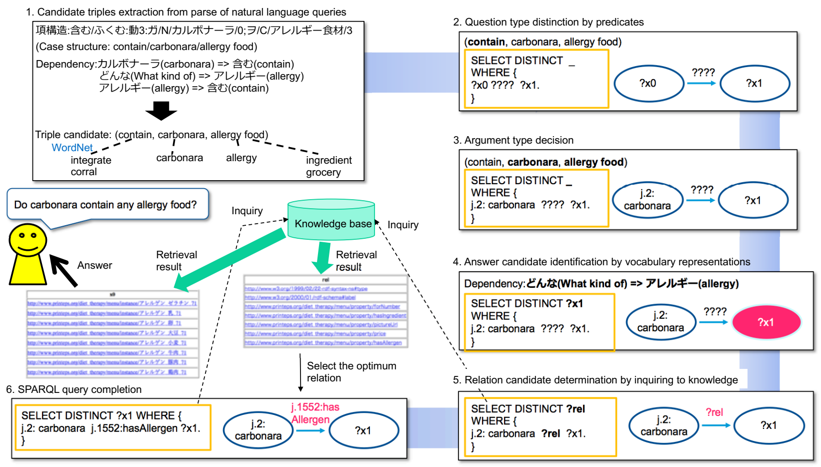

1. Candidate triples extraction from parse of natural language queries
項構造:含む/ふくむ:動3:ガ/N/カルボナーラ/0;ヲ/C/アレルギー食材/3
(Case structure: contain/carbonara/allergy food)
Dependency:カルボナーラ(carbonara) => 含む(contain)
どんな(What kind of) => アレルギー(allergy)
アレルギー(allergy) => 含む(contain)
Triple candidate: (contain, carbonara, allergy food)
WordNet
integrate
corral
carbonara
allergy
ingredient
grocery
Do carbonara contain any allergy food?
Answer
Retrieval result
Inquiry
Knowledge base
Inquiry
Retrieval result
Select the optimum relation
6. SPARQL query completion
SELECT DISTINCT ?x1 WHERE {
j.2: carbonara j.1552:hasAllergen ?x1.
}
j.2: carbonara
j.1552:has Allergen
?x1
2. Question type distinction by predicates
(contain, carbonara, allergy food)
SELECT DISTINCT _
WHERE {
?x0 ???? ?x1.
}
?x0
????
?x1
3. Argument type decision
(contain, carbonara, allergy food)
SELECT DISTINCT _
WHERE {
j.2: carbonara ???? ?x1.
}
j.2: carbonara
????
?x1
4. Answer candidate identification by vocabulary representations
Dependency:どんな(What kind of) => アレルギー(allergy)
SELECT DISTINCT ?x1
WHERE {
j.2: carbonara ???? ?x1.
}
j.2: carbonara
????
?x1
5. Relation candidate determination by inquiring to knowledge
SELECT DISTINCT ?rel
WHERE {
j.2: carbonara ?rel ?x1.
}
j.2: carbonara
?rel
?x1

Fig. 1. Overview of our method to convert a natural language utterance into a SPARQL query

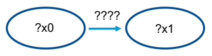
The procedure of automatically generating a SPARQL query from a natural language utterance is as follows:

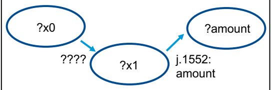
1. Triple candidates extraction from parse of natural language queries
We analyze natural language queries and extract their dependency and predicate-argument structures using a Japanese dependency and case structure analyzer, KNP[4] developed by Kurohashi-Kawahara Laboratory in Kyoto University. In the analysis of predicate-argument structures, we extract three pieces of information, i.e., predicate and its two arguments, from natural language as a candidate for representing the natural language expression in the form of RDF triples. Furthermore, for each word in the triples, we get its synsets by WordNet. This idea is based on the idea in Wang et al. [1] – a surface expressions and synsets in natural language queries may correspond to the vocabularies of the data repository, hence using taxonomy allows us access flexibly to such knowledge.
2. Question type definition for predicates
We classify the question type for the predicates in RDF triple candidates. In this work, we considered some user's utterances prospected in a cafeteria and classified them by their predicate. Table 1 shows the question type classification of each kind of predicate.
For predicates like "contain", we give a template {`?x0 ???? ?x1.`} as Basic type. For predicates inquiring degree like "low", we give a template {`?x0`

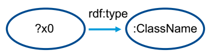
[4] http://nlp.ist.i.kyoto-u.ac.jp/index.php?KNP

Table 1. Question type classification

Type	Predicate	Example and triple candidate	SPARQL query template	RDF graph template

Basic type	contain
	with
		Do carbonara contain any allergy food?
		(contain, carbonara, allergy food)
		
		Do you have pasta with tomato?
		(with, pasta, tomato)
			SELECT DISTINCT _ WHERE {
			?x0 ???? ?x1.
			}
				

Degree type	low high	What are the low/high-salt pasta? (low/high, pasta, salt)	SELECT DISTINCT _ WHERE { ?x0 ???? ?x1. ?x1 j.1552:amount ?amount. } ORDER BY ASC/DSEC(?amount) LIMIT 3	

Count type	----	How many kinds of hamburg steak do you have?
		(have, hamburg steak, how many kinds of)
			SELECT DISTINCT (COUNT(?x0)
			as ?count) WHERE {
			?x0 rdf:type :ClassName.
			}
				

List type	----	What kind of hamburg steak do you have?
		(have, hamburg steak, ----)
		※No triple
			SELECT DISTINCT ?x0 WHERE {
			?x0 rdf:type :ClassName.
			}	

`???? ?x1. ?x1 j.1552:amount ?amount.`} as Degree type. In our work, we regard an inquiring to degree as that to amount because we confine domain of knowledge to food menu. By `ORDER BY ASC(?amount) LIMIT 3`, we set to return top 3 answers arranged in ascending order by its amount. In the case that the triple candidate has "何種類 (How many kinds of)", we give a template {`?x0 rdf:type :ClassName.`} and COUNT function as Count type. In the case of "What kind of hamburg steak do you have?", we cannot get any triple using KNP, so generate SPARQL query except for step 2, 3, and 5 of our method. When a word is sandwiched between two vocabulary representations "どんな (What kind of)" and "がありますか (Do you have)", we classify it as List type that answers all corresponding items.

At this stage, we regard arg1 and arg2 as `?x0` and `?x1` respectively because they are unsettled. For example, if predicate is "含む (contain)", it is identified as Basic type and then {`?x0 ???? ?x1.`} is given to it as template.

3. Argument type identification
By arguments of triple candidates, we decide the type is either class or instance or literal. We identify whether it is class or instance by using a dictionary which has been previously created based on menu ontology, and identify it as literal when the word does not exist in the dictionary. If it is a class entity, we add a triple pattern {`?x rdf:type :ClassName`} in a generated SPARQL query. If it is an instance entity, we interpret '`?x`' as '`:InstanceName`'. If it is a literal type entity, we leave it as a variable.
4. Answer candidate identification for vocabulary expressions
We identify the answer candidates based on vocabulary expressions in the query. We take "どんな (What kind of)", "はありますか (Do you have)", and "は何ですか (What)" as vocabulary expressions to identify the candidates. In the case that the query has "どんな", we decide the word just behind that as the answer candidate. In the case of "はありますか" or "は何ですか", we determine the word just before that as the candidate. Then we add the corresponding variable to a variable list next to `SELECT DISTINCT`.

5. Relation candidate determination by inquiring to knowledge
   We change `????` into `?rel`, and a variable next to `SELECT DISTINCT` into `?rel` of SPARQL query generated by the above, inquiry to knowledge, and determine relation candidates. For example, when we inquiry relations between `j.2: carbonara` and `?x1`, we get a list `[22-rdf-syntax-ns#type, rdf-schema#label, forNumber, hasIngredient, pictureUrl, price, hasAllergen]` as the relation candidates.
6. SPARQL query completion
   In the retrieval result of relation candidates, by the Jaccard index, we calculate the similarity of both bigrams of each relation candidate and all synsets of each word of the triple obtained by WordNet at step 1 (for example, we obtain bigrams for relation candidates, e.g., {`ha`,`as`,`sA`,`Al`,`ll`,`le`,`er`,`rg`,`ge`,`en`} and {`al`,`ll`,`le`,`er`,`rg`,`gy`}), and then determine a candidate over a threshold previously set as the optimum relation. If there are more than one candidates over the threshold, we select rarer one as the optimum relation according to Zipf's law. In this way, we complete a SPARQL query that reflects the content of the natural language query.

4 Experiments

In the experiment, we focus on a situation where a customer orders food at a cafeteria and confirm our proposed method for the input natural language queries based on four question types set in our work. We evaluate the generated SPARQL query with the validity of its knowledge extraction result.

4.1 Experimental setup

As for the knowledge to be referred to when generating SPARQL query from natural language query, we prepared a small set of menu ontology. Fig. 2 illustrates a part of menu ontology used in our work. As the SPARQL environment, we used Virtuoso (version 07.20.3214), SPARQL 1.1.

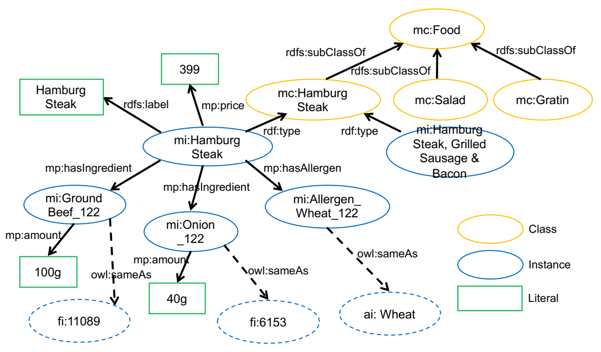
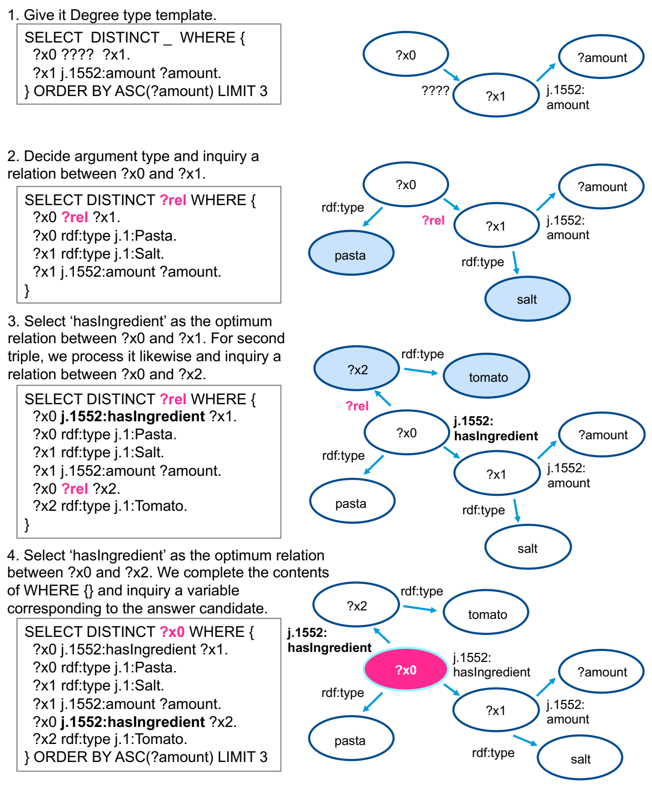


Fig. 2. A part of menu ontology

4.2 Results

We implemented our method, and conducted an experiment for five queries mentioned as examples of Table 1 such as “Does Carbonara contain any allergy ingredient?”, “Do you have pasta with tomato?”, “Which is low-salt pasta?”, “How many kinds of hamburg steak do you have?”, and “What kind of hamburg steak do you have?”. We also did a test for a query which combines Degree type and Basic type and have two predicates such as “Do you have the low-salt pasta with tomato?”. Fig. 3 shows this detailed procedure.



1. Give it Degree type template.

```
SELECT  DISTINCT _  WHERE {
  ?x0 ????  ?x1.
  ?x1 j.1552:amount ?amount.
} ORDER BY ASC(?amount) LIMIT 3
```

2. Decide argument type and inquiry a relation between ?x0 and ?x1.

```
SELECT DISTINCT ?rel WHERE {
  ?x0 ?rel ?x1.
  ?x0 rdf:type j.1:Pasta.
  ?x1 rdf:type j.1:Salt.
  ?x1 j.1552:amount ?amount.
}
```

3. Select ‘hasIngredient’ as the optimum relation between ?x0 and ?x1. For second triple, we process it likewise and inquiry a relation between ?x0 and ?x2.

```
SELECT DISTINCT ?rel WHERE {
  ?x0 j.1552:hasIngredient ?x1.
  ?x0 rdf:type j.1:Pasta.
  ?x1 rdf:type j.1:Salt.
  ?x1 j.1552:amount ?amount.
  ?x0 ?rel ?x2.
  ?x2 rdf:type j.1:Tomato.
}
```

4. Select ‘hasIngredient’ as the optimum relation between ?x0 and ?x2. We complete the contents of WHERE {} and inquiry a variable corresponding to the answer candidate.

```
SELECT DISTINCT ?x0 WHERE {
  ?x0 j.1552:hasIngredient ?x1.
  ?x0 rdf:type j.1:Pasta.
  ?x1 rdf:type j.1:Salt.
  ?x1 j.1552:amount ?amount.
  ?x0 j.1552:hasIngredient ?x2.
  ?x2 rdf:type j.1:Tomato.
} ORDER BY ASC(?amount) LIMIT 3
```

Fig. 3. Example: the procedure of “Do you have the low-salt pasta with tomato?”

In Fig. 3, we firstly extract two triple candidates (low, pasta, salt) and (with, pasta, tomato) by using KNP. For the first triple (low, pasta, salt), we provide it with a Degree type template by step 2 of our proposed method. We decide argument type by step 3 and inquiry a relation between `?x0` and `?x1`, and then select `hasIngredient` as the optimum one. For the second triple (with, pasta, tomato), we process it likewise and complete the contents of `WHERE {}` in the query. Finally, we add a variable corresponding to the answer candidate identified by step 4 to a variable list next to `SELECT DISTINCT`, and generate the SPARQL query we aim to.

In the experimental result as Table 2 shows, for each question type, we got useful knowledge to answer natural language query and therefore confirmed that our method generated the correct SPARQL query.

Table 2. Knowledge extraction results on each question type

Type	Natural language query	Generated SPARQL query	Answer from knowledge

Basic type	Do Carbonara contain any allergy food?	`SELECT DISTINCT ?x1 WHERE {`
		`j.2:Carbonara j.1552:hasAllergen ?x1.`
		`}`
			Allergen_Gelatin_71,Allergen_Wheat_71
			Allergen_Milk_71,Allergen_Beef_71
			Allergen_Egg_71,Allergen_Pork_71
			Allergen_Soybean_71,Allergen_Chicken_71

	Do you have pasta with tomato?	`SELECT DISTINCT ?x0 WHERE {`
		`?x0 j.1552:hasIngredient ?x1.`
		`?x0 rdf:type j.1:Pasta.`
		`?x1 rdf:type j.1:Tomato.`
		`}`
			Seafood Spaghetti with Tomato & Cream
			Spaghetti with Meat Sauce
			Spaghetti with Meat Sauce & Soft-boiled Egg

Degree type	Which is low-salt pasta?	`SELECT DISTINCT ?x0 WHERE {`
		`?x0 j.1552:hasIngredient ?x1.`
		`?x0 rdf:type j.1:Pasta.`
		`?x1 rdf:type j.1:Salt.`
		`?x1 j.1552:amount ?amount.`
		`} ORDER BY ASC(?amount) LIMIT 3`
			Spaghetti with Tomato & Bacon
			Spaghetti with Meat Sauce
			Spaghetti with Meat Sauce & Soft-boiled Egg

Count type	How many kinds of hamburg steak do you have?	`SELECT DISTINCT (COUNT(?x0) as ?count)`
		`WHERE {`
		`?x0 rdf:type j.1:Hamburg Steak.`
		`}`
			6

List type	What kind of hamburg steak do you have?	`SELECT DISTINCT ?x0 WHERE {`
		`?x0 rdf:type j.1:Hamburg Steak.`
		`}`
			Hamburg Steak with Cheese & Tomato Sauce
			Hamburg Steak
			Hamburg Steak, Grilled Sausage & Bacon
			Hamburg Steak & Teriyaki Pork
			Stewed Hamburg Steak
			Hamburg Steak with Vegetable Salsa

Degree type + Basic type	Do you have the low-salt pasta with tomato?	`SELECT DISTINCT ?x0 WHERE {`
		`?x0 j.1552:hasIngredient ?x1.`
		`?x0 rdf:type j.1:Pasta.`
		`?x1 rdf:type j.1:Salt.`
		`?x1 j.1552:amount ?amount.`
		`?x0 j.1552:hasIngredient ?x2.`
		`?x2 rdf:type j.1:Tomato.`
		`} ORDER BY ASC(?amount) LIMIT 3`
			Spaghetti with Meat Sauce
			Spaghetti with Meat Sauce & Soft-boiled Egg
			Seafood Spaghetti with Tomato & Cream

5 Conclusions and Future Work

We proposed a method to convert natural language into SPARQL queries using an analysis of predicate-argument structures and dependency structures by KNP and a transformation of RDF triples by a data driven approach. Mapping entities to knowledge vocabularies and acquiring a relation between entities from knowledge allow us generate the SPARQL query which is assured to return some sort of answer. We set four question types; i.e., Basic type, Degree type, Count type, and List type. Our method generated the appropriate SPARQL query for each natural language query of question type set in our work and more generalized query that combines them.

In this work, we conducted a preliminary experiment with small knowledge data and limited natural language queries. As future work, we will use larger scale knowledge base like DBpedia and more question types, handle various natural language expressions as input, and introduce quantitative evaluation on results. We will also introduce a method of knowledge graph embedding such as Bordes et al. [16] to generate SPARQL queries more flexibly.

Acknowledgments

This work has been supported by the Core Research for Evolutional Science and Technology (CREST) of the Japan Science and Technology Agency (JST) "A Framework PRINTEPS to Develop Practical Artificial Intelligence"(JPMJCR14E3).

References

1. Chong WangMiao XiongQi ZhouYong Yu, PANTO: A Portable Natural Language Interface to Ontologies, European Semantic Web Conference ESWC 2007: The Semantic Web: Research and Applications pp.473-487, 2007.
2. Miller, George A.,WordNet: A Lexical Database for English, Commun. ACM, Vol. 38, Num. 11, pp.39-41, Nov. 1995.
3. Zou, Lei and Huang, Ruizhe and Wang, Haixun and Yu, Jeffrey Xu and He, Wenqiang and Zhao, Dongyan, Natural Language Question Answering over RDF: A Graph Data Driven Approach, Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data, pp.313–324, Snowbird, Utah, USA, 2014.
4. Ryoji Suzuki, Makoto Miwa, Yutaka Sasaki, Converting Questions into SPARQL Queries for Traffic ontology , 2015 The Association for Natural Language Processing, P2-13, pp.171-174, Kyoto, 2015.
5. Saeedeh Shekarpour, Edgard Marx, Soren Auer, Amit Sheth, RQUERY: Rewriting Natural Language Queries on Knowledge Graphs to Alleviate the Vocabulary Mismatch Problem, the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17), San Francisco, CA, February 4-9, 2017.
6. Tommaso Soru, Edgard Marx, Diego Moussallem, Gustavo Publio, Andre Valdestilhas, Diego Esteves, and Ciro Baron Neto, SPARQL as a Foreign Language, http://arxiv.org/abs/1708.07624, 2017.
7. Ilya Sutskever, Oriol Vinyals, and Quoc V. Le, Sequence to Sequence Learning with Neural Networks, http://arxiv.org/abs/1409.3215, 2014.
8. Fabiano Ferreira Luz and Marcelo Finger, Semantic Parsing Natural Language into SPARQL: Improving Target Language Representation with Neural Attention, http://arxiv.org/abs/1803.04329, 2018.
9. Christina Unger, Lorenz Buhmann, Jens Lehmann, Axel-Cyrille Ngonga Ngomo, Daniel Gerber, Philipp Cimiano Template-based Question Answering over RDF Data, WWW 2012, Lyon, France, April 1620, 2012.
10. Abdalghani Abujabal, Mohamed Yahya, Mirek Riedewald, Gerhard Weikum, Automated Template Generation for Question Answering over Knowledge Graphs, WWW 2017, Perth, Australia, April 37, 2017.
11. Wanyun Cui, Yanghua Xiao, Haixun Wang, Yangqiu Song, Seung-won Hwang, Wei Wang, KBQA: Learning Question Answering over QA Corpora and Knowledge Bases, 3rd International Conference on Very Large Data Bases, VLDB 2017, Munich, DEU, 28 August - 01 September 2017.
12. Marcelo Arenas, Martin Ugarte, Designing a Query Language for RDF: Marrying Open and Closed Worlds, PODS '16, San Francisco, CA, USA, June 26July 1, 2016.
13. S ebastien Ferr e, SQUALL: a Controlled Natural Language for Querying and Updating RDF Graphs, T. Kuhn and N.E. Fuchs. Controlled Natural Languages, Zurich, Switzerland, Aug. Springer, pp.11-25, 2012.
14. Takuma Ebisu, Ryutaro Ichise, Triple Extraction from Texts using Distributed Representations of Words, The 31st Annual Conference of the Japanese Society for Artificial Intelligence, 2017.
15. Yuanzhe Zhang, Shizhu He, Kang Liu, Jun Zhao, A Joint Model for Question Answering over Multiple Knowledge Bases, the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16), Phoenix, Arizona — February 12 - 17, 2016.
16. Antoine Bordes, Jason Weston, Ronan Collobert, Yoshua Bengio, Learning Structured Embeddings of Knowledge Bases, the Twenty-Fifth AAAI Conference on Artificial Intelligence, pp. 301-306, San Francisco, California, August 07 - 11, 2011.