Xser@QALD-4: Answering Natural Language Questions via Phrasal Semantic Parsing

Kun Xu, Yansong Feng, and Dongyan Zhao

Institute of Computer Science Technology, Peking University.
Zhongguancun North Road 128#, Beijing, 100871, China.
{xukun, fengyansong, zhaodongyan}@pku.edu.cn

Abstract. We present a question answering system (Xser) over Linked Data(DBpedia), converting users' natural language questions into structured queries. There are two challenges involved: recognizing users' query intention and mapping the involved semantic items against a given knowledge base (KB), which will be in turn assembled into a structured query. In this paper, we propose an efficient pipeline framework to model a user's query intention as a phrase level dependency DAG which is then instantiated according to a given KB to construct the final structured query. We evaluate our approach on the QALD-4 test dataset and achieve an F-measure score of 0.72, an average precision of 0.72 and an average recall of 0.71 over 50 questions.

Keywords: Phrasal Semantic Parsing, Question Answering, Knowledge Base

1 Introduction

As very large structured knowledge bases have become available, e.g.,YAGO [10], DBpedia [16] and Freebase[15], answering natural language questions over structured knowledge facts has attracted increasing research efforts, in both natural language processing and information retrieval communities. Different from keyword based retrieval, the structure of query intentions embedde in a user's question can be represented by a set of predicate-argument structures, e.g., <*subject, predicate, object*> triples, and effectively retrieved by a structured query engine from a given knowledge base. Generally, the main challenge of understanding the query intention in a structural form is to solve two tasks: recognizing the predicate-argument structures and then instantiating these structures regarding a given KB.

Intuitively, the two subtasks would be solved in a joint framework, e.g., [22] proposed a PCFG-based semantic parser to simultaneously learn the combination rules among words or phrases and the mappings to specific KB components. However, given the size of existing KBs (usually thousands of predicates, millions of entities and billions of knowledge facts), it makes difficult to jointly train such a PCFG-based parser (the model of [22] takes several days to train with

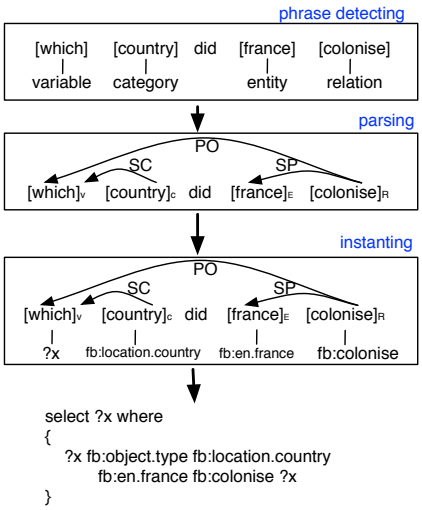

Fig. 1. An example of converting a natural language question into a structured query via phrasal semantic parsing.

3,000 sentences), and even more difficult to adapt to other KBs, let alone retrieving multiple KBs within one query, e.g., some queries in the QALD task[17] are mixed with predicates from both DBpedia and Yago. In contrast, we find that recognizing the query intention structure is usually KB-independent. Take Figure 1 as an example, without grounding to a knowledge base, we can still guess that a location called *france* has some relationship, indicated by the verb *"colonise"*, with some *countries*, (the queried objects), which can be learned directly without reliance on a specified KB. On the other hand, the task of mapping semantic phrases from the intention structures to items in a given KB and producing the final structured queries is KB-dependent, since one has to solve these mappings according to the schema of a specified KB.

Given the observations above, we thus assume that the structure of a question's query intention can be learned independent from a specific knowledge base, while grounding and converting a query intention into a structured query is dependent on a knowledge base. Our assumption will naturally lead to a pipeline paradigm to translating a natural language question into a structured query, which can then be directly retrieved by a structured database query engine, e.g., Virtuoso [1].

In this paper, we deal with the task of understanding natural language questions in a pipeline paradigm, involving mainly two steps: recognizing the query intention structure inherent in the natural language questions, and then instantiating the query intention structures by mapping the involved semantic items into existing KBs. In the first phase, we build a phrase detector to detect possible

[1] http://www.virtuoso.com

semantic phrases, e.g., variables, entity phrases, category phrases and relation phrases. We then develop a semantic parser to predict the predicate-argument structures among phrases to represent the structure of query intentions. In the second phase, given the intention structures, we are then able to adopt a structured perceptron model to jointly solve the mappings between semantic phrases and KB items. By taking a two-phase format, our proposed model can benefit from the separation of KB related components and KB independent steps, and recognize the intention structures more efficiently while making the KB-related component flexible, e.g., we can only retrain the second phase when adapting to new KBs, which is similar in sprite with [23], who rely on a CCG parser to produce an ontological-independent logical representation to express users' intention. We evaluate our model on the QALD-4, and achieve an F-measure score of 0.72, an average precision of 0.72 and an average recall of 0.71 over 50 questions.

2 The Task

We define the task of using a KB to answer natural language questions as follows: given a natural language question q_{NL} and a knowledge base `KB`, our goal is to translate q_{NL} into a structured query in certain structured query language, e.g., SPARQL, which consists of multiple triples: a conjunction of <subject, predicate, object> search conditions.

3 Recognizing the Structure of Query Intention

Our framework first employ a pipeline of phrase detection and phrasal semantic parsing to recognize the inherent structure of user's query intention, and then instantiates the query intention regarding specific KBs.

3.1 Phrase Detection

Before recognizing the query intentions of questions, we first detect phrases of interest from the questions that potentially correspond to semantic items. where a detected phrase is also assigned with a semantic label $l \in \{entity, relation, category, variable\}$. For entity phrases, we need to recognize phrases that may correspond to entities of the KB. Similarity, relation phrases may correspond to KB's predicates, and category phrases may be mapped in KB's classes. This problem can be casted as a sequence labeling problem, where our goal is to build a tagger whose input is a sentence, for example:

$$\begin{array}{cccccccc} Who & has & Tom & Cruise & been\ married & to \\ V-B & none & E-B & E-I & none & R-B & R-I \end{array}$$

(here, we use *B-I* scheme for each phrase label: *R-B* represents the beginning of a relation phrase, *R-I* represents the continuation of a relation phrase). We use

structured perceptron[3] to build our phrase tagger. Structured perceptron is an extension to the standard linear perceptron for structured prediction. Given a question instance $x \in X$, which in our case is a sentence, the structured perceptron involves the following *decoding problem* which finds the best configuration $z \in Y$, which in our case is a label sequence, according to the current model $\mathbf{w}$:

$$z = \arg \max_{y' \in Y(x)} w \cdot f(x, y')$$

where $f(x, y')$ represents the feature vector for instance x along with configuration y'. We observe that many phrases of a specific semantic label may share similar POS tag sequence pattern and NER sequence pattern. For example, the POS tag sequences of relation phrases "*the area of*" and "*the population in*" are in the same pattern "*DT NN IN*". Therefore, we use three types of features: lexical features, POS tag features and NER features. Table 1 summarizes the feature templates we used in the phrase detection.

Table 1. Set of feature templates for phrase detection

p = pos tag; n = ner tag; w = word; t = phrase type tag; i = current index

1	unigram of POS tag	p_i
2	bigram of POS tag	$p_i p_{i+1}$; $p_{i-1} p_i$
3	trigram of POS tag	$p_i p_{i+1} p_{i+2}$; $p_{i-1} p_i p_{i+1}$; $p_{i-2} p_{i-1} p_i$
4	unigram of NER tag	n_i
5	bigram of NER tag	$n_i n_{i+1}$; $n_{i-1} n_i$
6	trigram of NER tag	$n_i n_{i+1} n_{i+2}$; $n_{i-1} n_i n_{i+1}$; $n_{i-2} n_{i-1} n_i$
7	unigram of word	w_i
8	bigram of word	$w_i w_{i+1}$; $w_{i-1} w_i$
9	trigram of word	$w_i w_{i+1} w_{i+2}$; $w_{i-1} w_i w_{i+1}$; $w_{i-2} w_{i-1} w_i$
10	previous phrase type	t_{i-1}
11	conjunction of previous phrase type and current word	$t_{i-1} w_i$

3.2 Phrase DAG Parsing with Multiple Heads

As shown in Figure 1, query intention can be represented by dependencies between "*country*", "*france*" and "*colonise*", forming a phrase DAG, we thus introduce a transition-based DAG parsing algorithm to perform a structural prediction process and reveal the inherent structures.

Phrase DAG In dependency grammar, structures are determined by the relationship between a head and its dependents, e.g., syntactic or morphological dependencies. Here, we propose to use predicate-argument dependencies to capture the query intention, that is, the arguments of a predicate are dependents of that predicate. Each predicate is either a unary predicate (which characterizes

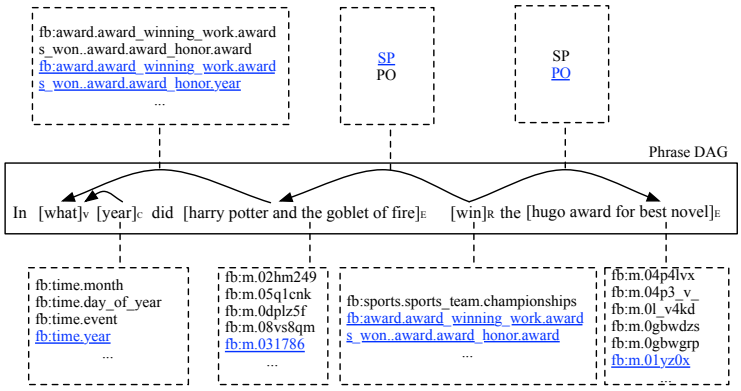

Fig. 2. An example of phrasal semantic DAG, where the dashed boxes list the mapping candidates for all phrases and the underlined are the gold-standard mappings.)

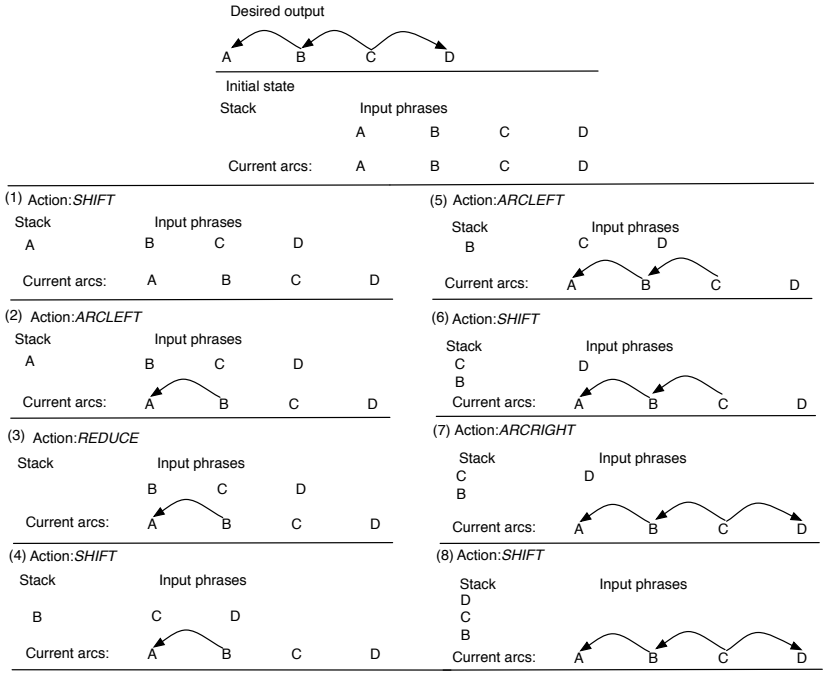

Fig. 3. An example of shift-reduce DAG parsing.

its only argument) or a binary predicate (which represents the semantic relation between its two arguments). Here, category phrases correspond to unary predicates, and relation phrases correspond to binary predicates.

We formulate the problem of recognizing the dependency structures in a phrase level: the detected phrases in a question form the nodes of the phrase dependency DAG, with four types, variable, entity, relation, and category nodes, where the former two are referred as content nodes in the following. We draw a directed edge from the head to its dependent, indicating the predicate-argument or argument-argument relationship.

Table 2. The transitions for our arc-eager parsing algorithm

Transitions	
Left-Arc	$(\sigma\|i, j\|\beta, A) \Rightarrow (\sigma\|i, j\|\beta, A \cup \{(j, l, i)\})$
Right-Arc	$(\sigma\|i, j\|\beta, A) \Rightarrow (\sigma\|i, j\|\beta, A \cup \{(i, l, j)\})$
Reduce	$(\sigma\|i, \beta, A) \Rightarrow (\sigma, \beta, A)$
Shift	$(\sigma, i\|\beta, A) \Rightarrow (\sigma\|i, \beta, A)$
Preconditions	
Left-Arc	$\neg[i = 0]$
	$\neg\exists j \exists i[(i, l, j) \in A]$
Right-Arc	$\neg\exists j \exists i[(j, l, i) \in A]$

The Phrase DAG Parsing Note that, in our setup, one phrase can have more than one head, as in Figure 2, variable node *what* has two heads in the resulting dependency DAG. We thus use the framework proposed by [2], i.e., extending traditional arc-eager shift-reduce parsing with multiple heads to find a DAG directly. Specifically, given a question with sequence of phrases, our parser uses a stack of partial DAGs, a queue of incoming phrases, and a series of actions to build a dependency DAG. We assume that each input phrase has been assigned a POS-tag (conjunction of each token's POS-tag which composes of the input phrase) and a semantic label.

Our semantic parser uses four actions: SHIFT, REDUCE, ARCRIGHT and ARCLEFT (Table 2).

The SHIFT action follow the standard definitions that just pushes the next incoming phrase onto the stack, as shown in Figure 3 action(1).

The REDUCE action pops the stack top, as shown in figure 3 action(3). Note that, the standard REDUCE action which is taken on the condition that the stack top has at least one head. This precondition ensures the dependency graph is a connected graph. However, our phrase dependency parser only concerns the predicate-argument structures, and we add a dependency only between the predicate and argument of our interest. In our case, the dependency graph can be an unconnected directed graph.

The ARCRIGHT action adds a dependency edge from the stack top to the first phrase of the incoming queue, where the phrase on the stack is the head

Algorithm 1 The decoding algorithm for the phrase DAG parsing; K is the beam size

Require: sentence x
  agenda: hold the K-best candidate items
Ensure: *candidate_output*

```
agenda.clear()
agenda.insert(GetStartItem(x))
candidate_output = NONE
while not agenda.empty() do
  list.clear()
  for all item ∈ agenda do
    for all action ∈ getActions(actions, item) do
      item' = item.apply(action)
      if item'.F == TRUE then
        if candidate_output == NONE
        or item'.score > candidate_output.score then
          candidate_output = item'
        end if
      else
        list.append(item')
      end if
    end for
  end for
  agenda.clear()
  agenda.insert(list.best(K))
end while
```

and the phrase in the queue is the dependent (the stack and queue are left untouched), as long as a left arc does not already exist between these two phrases, as shown in Figure 3 action(7).

The ARCLEFT action adds a dependency edge from the first phrase on the queue to the stack top, where the phrase in the queue is the head and the phrase on the stack is the dependent (again, the stack and queue are left untouched), as long as a right arc does not already exist between the two phrases, as shown in Figure 3 action(2).

Figure 3 shows an example of shift-reduce parsing process.

The decoding algorithm for phrase DAG parsing We apply the standard beam-search along with early-update to perform inexact decoding [9] during training. To formulate the decoding algorithm, we define a *candidate item* as a tuple $\langle S, Q, F\rangle$, where S represents the stack with partial derivations that have been built, Q represents the queue of incoming phrases that have not been processed, and F is a boolean value that represents whether the candidate item has been finished. A candidate item is finished if and only if the queue is empty, and no more actions can be applied to a candidate item after it reaches the finished status. Given an input sentence x, we define the start item as the unfinished item

with an empty stack and the whole input sentence as the incoming phrases(line 2). A derivation is built from the start item by repeated applications of actions (SHIFT, REDUCE, ARCLEFT and ARCRIGHT) until the item is finished.

To apply beam-search, an agenda is used to hold the K-best partial (unfinished) candidate items at each parsing step. A separate *candidate output* is used to record the current best finished item that has been found, since candidate items can be finished at different steps. Initially the agenda contains only the start item, and the candidate output is set to none(line 3). At each step during parsing, each candidate item from the agenda is extended in all possible ways by applying one action according to the current status(line 7), and a number of new candidate items are generated(line 8). If a newly generated candidate is finished, it is compared with the current *candidate output*. If the candidate output is none or the score of the newly generated candidate is higher than the score of the *candidate output*, the *candidate output* is replaced with the newly generated item(line 11); otherwise the newly generated item is discarded (line 14). If the newly generated candidate is unfinished, it is appended to a list of newly generated partial candidates. After all candidate items from the agenda have been processed, the agenda is cleared(line 18) and the K-best items from the list are put on the agenda(line 19). Then the list is cleared and the parser moves on to the next step. This process repeats until the agenda is empty (which means that no new items have been generated in the previous step), and the candidate output is the final derivation. Pseudocode for the decoding algorithm is shown in Algorithm 1.

Table 3. The set of feature templates used in our phrase DAG parser

p = phrase; t = POS-tag; s = phrase type

Category	Description	templates
lexical features	stack top	$STpt$; STp; STt;
	current phrase	N_0pt; N_0p; N_0t
	next phrase	N_1pt; N_1p; N_1t;
	ST and N0	$STptN_0pt$; $STptN_0p$;
	POS bigram	N_0tN_1t
	POS trigrams	$N_0N_1tN_2t$;
	N0 phrase	$N_0pN_1tN_2t$;
semantic features	Conjunction of phrase label and pos tag	N_0s; N_0ts; N_0ps; N_1s;N_1ts;$STtN_0s$; $STsN_0t$; $STpN_0s$; $STtN_0t$; $STsN_0s$;
structural features	Indicates whether exists an arc between the stack top item and next input item, and if so what type of arc	ArcLeft(STs, N_0s); ArcRight(STs, N_0s)

Features Features play an important role in transition-based parsing. Our parser takes three types of features: lexical, semantic and structure-related features. We summarize our feature templates in Table 3, where ST represents the top node in the stack, N_0, N_1, N_2 represent the three incoming phrases from the incoming queue, subscript t indicates POS tags, subscript p indicates lexical surface forms and subscript s represent the semantic label of the phrase(*entity,relation, category* and *variable*).

Lexical features include features used in traditional word level dependency parsing with some modifications: all co-occurrences are built on phrase nodes and the POS tag of a phrase is defined as the concatenation of each token's POS tag in the phrase.

Note that ARCLEFT and ARCRIGHT actions are considered only when the top phrase of the stack and the next phrase are variable, entity or relation phrases.To guide the ARCLEFT and ARCRIGHT actions, we introduce semantic features indicating the semantic label of a phrase.

Recall that, our phrase semantic DAG parser allows one phrase to have multiple heads. Therefore, we modify the ARCLEFT and ARCRIGHT actions so that they can create new dependency arcs without removing the dependent from further consideration for being a dependent of other heads. We thus introduce new structure-related features to indicate whether an arc already exists between the top phrase on the stack and the next phrase on the queue.

4 Instantiating Query Intention Regarding Existing KBs

After recognizing the structure of the user's query intention, our parser can output a KB-independent phrase dependency DAG, as shown in Figure 2. To covert the the query intention into structured queries, semantic items are grounded to a specific KB, e.g., DBpedia. In experiment, we utilize different methods to maintain the candidate sets for different types of phrases.

For the entity phrase, we first employ the wikipedia miner tool[2] to map the phrase to wikipedia entities, and then generate the candidate set of DBpedia entities which share the same wikipedia ids with wikipedia entities.

For the relation and category phrase, we use the PATTY resource to construct a lexicon that maps phrases to predicates and categories of DBpedia.

Note that, a nice paradigm will be jointly resolving the mappings of all semantic items involved: labeling phrase nodes and the dependency relations with corresponding items in the given KB. Therefore, inspired by [21], we formulate this problem as a structured learning task, where we choose a candidate for each phrase and dependency relation, and find a globally maximum configuration among multiple local candidates.

For the dependency relation candidates especially for that between two content phrases, we develop a function *getDepRel* to retrieve the possible dependency relation candidates which are coherent with the domain of the phrases in the question from the set of all KB predicates.

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

Algorithm 2 The decoding algorithm to instantiate the query intention structures

Require: Instance $x =< (x_1, x_2, \ldots, x_m), \xi, o >$ and the oracle output y if for training.
K:Beam Size.
$\Gamma \cup \{\perp\}$: phrase candidates.
$\Lambda \cup \{\perp, SP, PO, SC\}$: dependency relation candidates.
Ensure: *1-best prediction z for x*

Set $beam \leftarrow [\varepsilon]$ //empty configuration
for all $i \leftarrow 1..m$ **do**
  $buf \leftarrow \{z' \odot l | z' \in B, l \in \Gamma \cup \{\perp\}\}$ //phrase labeling
  $B \leftarrow$ K-best(buf)
  if $y_{1:(i-1)*m+1} \notin B$ **then**
    return $B[0]$ //for early update
  end if
  for all $(x_k, *) \in \xi$ **do**
    /* search for dependent relation*/
    $buf \leftarrow \emptyset$
    for all $z' \in B$ **do**
      $buf \leftarrow buf \cup \{z' \odot \perp\}$
      if $dep(i, k, *) \in o$ **then**
        //this dependency from phrase i to phrase k
        //exists in the DAG structures
        /*dependency relation labeling*/
        $buf \leftarrow buf \cup \{z' \odot r | r \in getDepRel(\Lambda, z') \cup SP \cup PO \cup SC \cup \perp\}$
      end if
    end for
    $B \leftarrow$K-best(buf)
    if $y_{1:i*k} \notin B$ **then**
      return B[0] // for early update
    end if
  end for
end for

4.1 Instantiating Query Intention with Structured Perceptron

As shown in Figure 2, each phrase and dependency relation have multiple candidates and the best configuration is consisted of the blue and underlined candidates. However, it is intractable to perform exact search in our framework because: (1) by simultaneously mapping phrase nodes and dependency relations, the search space becomes much more complex, (2) we propose to make use of arbitrary global features, which makes it infeasible to perform exact inference efficiently.

To address this problem, we apply beam-search along with early-update strategy to perform inexact decoding.

4.2 Label sets

Here we introduce the label sets for phrase node and dependency relation in the model. We use $\Gamma \cup \{\perp\}$ to denote the phrase candidates, and $\perp$ indicates that the phrase node can not be mapped to an entity or a predicate of the KB. Similarly, $\Lambda \cup \{SP, PO, SC, \perp\}$ denotes the dependency relation candidates, where Λ is the set of all KB predicates, SP and PO are the two labels representing the dependency relation between relation phrase and entity phrase, where SP indicates that the entity phrase is the subject of the relation phrase, and PO indicates the other way. SC is the label representing the dependency relation between content phrase and category phrase, where the category phrase characterized the content phrase. and $\perp$ means that this dependency relation can not be instantiated.

4.3 The decoding algorithm for instantiating query intentions

Let $x =< (x_1, x_2, \ldots, x_m), \xi, o >$ denote the sentence instance, where x_i represents the i-th phrase in the sentence, $\xi = \{\{(x_k, c_i)\}_{i=1}^{s}\}_{k=1}^{m}$ is the set of KB item candidates for phrases where s denotes the size of candidates for x_k, and $o = \{\{dep_l(i, j, c_i)\}_{i=1}^{d}\}_{l=1}^{n}$ is the set of KB predicate candidates for dependency relations where i and j denotes the head and dependent index, d denotes the size of candidates for the dependency relation. We use

$$y = (t_1, a_{1,1}, \ldots, a_{1,m}, \ldots, t_m, a_{m,1}, \ldots, a_{m,m})$$

to denote the corresponding gold standard structure, where t_i represents the KB item assignment for the phrase x_i, and $a_{i,k}$ represents KB item assignment for the dependency relation between the head x_i and dependent x_k. Algorithm 2 describes the beam-search procedure with early-update. During each step with phrase i, there are two sub-steps:

Phrase labeling We enumerate all possible KB item candidates for the current phrase(line 3). K-best partial configurations are selected to the beam(line 4), assuming the beam size is K.

Dependency relation labeling After the phrase labeling step, we traverse all configurations in the beam. Once a phrase candidate for x_i is found in the beam, the decoder searches through the dependency relation candidates o to label each dependency relation where the current phrase as the head and x_i as the dependent(line11-19). After labeling each dependency relation, we again score each partial assignment and select the K-best results to the beam(line 20).

Note that, the score of a configuration is indeed the sum of two scores from each sub-step. Therefore the resulting best configuration will jointly solve the phrase and dependency relation labeling. We use two types of features: local features and global features. Table 4 summarize these features.

Table 4. Features used to instantiate query intention

p = phrase; r = dependency relation c = candidate
s = the surface name of c

Local features		
Category	Type	Feature Description
Phrase	Text similarity	p is equal/prefix/suffix of s overlap of p and s
	Popularity	the popularity of s
	Lexical	p is equal/prefix/suffix of the synonym of s
Dependency Relation	Text similarity	r is equal/prefix/suffix of s overlap of p and s
Global features		
Phrase	KB related	whether p has a head, and if so, if p fulfills the expectation type of its head
		number of phrases in the question that are in the same domain of p
Dependency Relation	KB related	whether r has an argument, and if so, if the argument fulfills the expectation type of r
		number of phrases in the question that are in the same domain of r

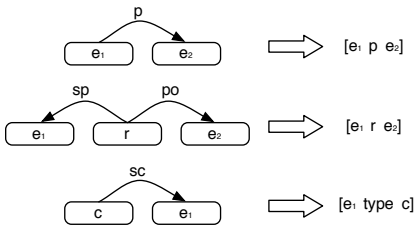

Fig. 4. Rules used to convert query intentions into structured queries.

4.4 Converting Query Intention into Structured Queries

After instantiating the query intention, our system outputs a phrasal semantic DAG regarding KB. We convert the DAG into structured queries, e.g., SPARQL, mainly by the rules shown in Figure 4, where e_1 and e_2 represent entity phrases, r denotes a relation phrase and c denotes a category phrase. The left is the graph pattern which may occur in the semantic DAG, and the right is the query triple converted from the pattern. Note that, multiple graph patterns may occur in a semantic DAG, as shown in Figure 3. The final structured queries are the conjunction of query triples converted from DAG.

5 Experimental

5.1 Data set

We use the Free917 dataset [6] as our training data, which contains 917 questions annotated with logical forms. Recall that, in our model, we require gold-standard phrase dependency DAGs of these questions as our training data, which we prepare as follows. For each question, we find its variable, entity, category and relation phrases, manually annotate the question with phrase dependency DAG and map phrases and dependencies to Freebase semantic items.

The QALD-4 task for question answering over linked data consists of 50 test questions. We use these 50 questions as our test data and directly use the parser trained on Free917 to predict the query intention and instantiate the query intention regarding DBpedia.

5.2 Results on QALD-4 data

The following table gives the evaluation results with average precision, average recall and F-measure. It shows the number of question our system can answer, the number of right and partially right answers among them.

Table 5. Evaluate results on QALD-4 test dataset

Total	Processed	Right	Partially right	Avg.Precision	Avg.Recall	F1
50	40	36	6	0.72	0.71	0.72

5.3 Error Analysis

We notice that in many cases our system can be further improved. For example, in the question from QALD4 "*what is the bridge with the longest span*", our system incorrectly detects "*the longest span*" as a category phrase, while this phrase is certainly a relation phrase. This detection error is propagated to phrasal semantic parsing. A necessary direction for our future step should be jointly solve phrase detection and phrasal semantic DAG parsing to avoid such errors.

6 Conclusion and Future Work

In this paper, we propose a novel framework to translate natural language questions into structural queries, which can be effectively retrieved by structured query engines and return the answers according a KB. The novelty of our framework lies in modeling the task in a KB-independent and KB-related pipeline paradigm, where we use phrasal semantic DAG to represent users' query intention, and develop a KB-independent shift-reduce DAG parser to capture the

structure of the query intentions, which are then grounded to a given KB via joint mappings. This gives the advantages to analyze the questions independent from a KB and easily adapt to new KBs without much human involvement.

Currently, our model requires question-DAG pairs as the training data, though we do not need to annotate new data for every datasets or KBs, it is still promising to extend our model to train on question-answer pairs only, further saving human involvement. Secondly, in the pipeline of phrase detection and phrasal semantic parsing, error propagated from the upstream to downstream. A joint model with multiple perceptrons may help to eliminate the error propagation.

References

1. A. Fader, S. Soderland, and O. Etzioni. Identifying relations for open information extraction. In EMNLP, pages 1535-1545, 2011.
2. K. Sagae and J. Tsujii. Shift-reduce dependency dag parsing. In COLING, pages 753-760, 2008.
3. M. Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP-2002), 2002.
4. K. D. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In SIGMOD Conference, pages 1247-1250, 2008.
5. Q. Cai and A. Yates. Semantic Parsing Freebase: Towards Open-domain Semantic Parsing. In Proceedings of the Second Joint Conference on Lexical and Computational Semantics (*SEM), 2013.
6. M. Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP-2002), 2002.
7. A. Frank, H.-U. Krieger, F. Xu, H. Uszkoreit, B. Crysmann, B. Jorg, and U. Schafer. Question answering from structured knowledge sources. J. Applied Logic, 5(1):20-48, 2007.
8. T. Lin, Mausam, and O. Etzioni. Entity linking at web scale. In Proceedings of the Joint Workshop on Automatic Knowledge Base Construction and Web-scale Knowledge Extraction, AKBC-WEKEX 12, pages 84-88, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.
9. M. Collins and B. Roark. Incremental parsing with the perceptron algorithm. In ACL, pages 111-118, 2004.
10. F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: a core of semantic knowledge. In WWW, pages 697-706, 2007.
11. J. Krishnamurthy and T. Mitchell. Weakly supervised training of semantic parsers. In EMNLP-CoNLL, pages 754-765, 2012.
12. M. Yahya, K. Berberich, S. Elbassuoni, M. Ramanath, V. Tresp, and G. Weikum. Natural language questions for the web of data. In EMNLP-CoNLL, pages 379-390, 2012.
13. C. Unger and P. Cimiano. Pythia: Compositional meaning construction for ontology-based question answering on the semantic web. In Proceedings of the 16th International Conference on Natural Language Processing and Information Systems, NLDB11, pages 153-160, Berlin, Heidelberg, 2011. Springer-Verlag.

14. C. Unger, L. Buhmann, J. Lehmann, A.-C. N. Ngomo, D. Gerber, and P. Cimiano. Template-based question answering over rdf data. In WWW, pages 639-648, 2012.
15. K. D. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In SIGMOD Conference, pages 1247-1250, 2008.
16. S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. G. Ives. Dbpedia: A nucleus for a web of open data. In ISWC/ASWC, pages 722-735, 2007.
17. P. Cimiano, V. Lopez, C. Unger, E. Cabrio, A.-C. N. Ngomo, and S. Walter. Multilingual question answering over linked data (qald-3): Lab overview. In CLEF, pages 321-332, 2013.
18. T. Kwiatkowski, L. S. Zettlemoyer, S. Goldwater, and M. Steedman. Inducing probabilistic ccg grammars from logical form with higher-order unification. In EMNLP, pages 1223-1233, 2010.
19. J. Berant and P. Liang. Semantic parsing via paraphrasing. In Association for Computational Linguistics (ACL), 2014.
20. L. Zou, R. Huang, H. Wang, J. X. Yu, W. He, and D. Zhao. Natural language question answering over rdf. In Special Interest Group on Management Of Data(SIGMOD), 2014.
21. Q. Li, H. Ji, and L. Huang. Joint event extraction via structured prediction with global features. In ACL, pages 73-82, 2013.
22. J. Berant, A. Chou, R. Frostig, and P. Liang. Semantic parsing on freebase from question-answer pairs. In EMNLP, pages 1533-1544, 2013.
23. T. Kwiatkowski, E. Choi, Y. Artzi, and L. S. Zettlemoyer. Scaling semantic parsers with on-the-fly ontology matching. In EMNLP, pages 1545-1556, 2013.