SUPPLEMENTARY MATERIALS

APPENDIX A PSEUDO CODES

Algorithm 1 Build Relation Mention Dictionary

Input: A relation mention set $T = \{rel_1, ..., rel_n\}$ and each mention rel_i, has a support set $Sup(rel_i) = \{ (v_i^1, v_i^{\prime 1}), ..., (v_i^m, v_i^{\prime m})\}$ and an RDF graph G.
Output: Each relation mention rel_i has the top-k possible predicate paths $\{L_{i_1}, ..., L_{i_k}\}$ with the same semantic equivalence.

```
1: for each relation mention $rel_i$, $i = 1, ..., n$ in $T$ do
2:     for each entity pair $(v_i^j, v_i^{\prime j})$ in $Sup(rel_i)$ do
3:         Find all simple predicate path patterns (with length less than a predefined parameter $\theta$) between $v_i^j$ and $v_i^{\prime j}$, denoted as $Path(v_i^j, v_i^{\prime j})$.
4:     $PS(t_i) = \bigcup_{j=1,...,m} Path(v_i^j, v_i^{\prime j})$
5: for each relation mention $rel_i$ do
6:     for each predicate path pattern $L$ in $PS(t_i)$ do
7:         Compute tf-idf value of $L$ (according to Definition 6)
8:     for relation mention $rel_i$, record the $k$ predicate path patterns with the top-k highest tf-idf values.
```

Algorithm 2 Finding Relation Mentions Occurring in a Natural Language Question N

Input: A dependency tree Y and an inverted index over the relation mention set T.
Output: All embeddings of relation mentions (in T) occurring in Y.

```
1: for each node $w_i$ in $Y$ do
2:     Find a list of relation mentions $PL_i$ occurring in $T$ by the inverted list.
3: for each node $w_i$ in $Y$ do
4:     Set $PL = PL_i$
5:     for each relation mention $rel \in PL$ do
6:         Set $rel[w_i] = 1$ // indicating the appearance of word $w_i$ in $rel$.
7:     Call Probe($w_i$, $PL$)
8:     for each relation mention $rel$ in $PL_i$ do
9:         if all words $w$ of $rel$ have $rel[w] = 1$ then
10:            $rel$ is an occurring relation mention in $Y$
11:            Return $rel$ and a subtree rooted at $w_i$ includes (and only includes) all words in $rel$.
```

Probe(w, PL')

```
1: for each child $w'$ of $w$ do
2:     $PL'' = PL' \cap PL_i$.
3:     if $PL'' == \phi$ then
4:         return
5:     else
6:         for each relation mention $rel \in PL''$ do
7:             Set $rel[w'] = 1$ // indicating the appearance of word $w'$ in $t$.
8:         Call Probe($w'$, $PL''$)
```

Algorithm 3 Generating Top-k SPARQL Queries

Input: A semantic query graph Q^S and a RDF graph G.
Output: Top-k SPARQL Queries, i.e., the top-k matches from Q^S to G.

```
1: Sorting all candidates in a non-ascending order
2: $n = |E(Q^S)| + |V(Q^S)|$
3: Initialize $n$ bit vector $\Gamma$ with zero
4: Initialize maximum heap $H$ with one element ($\Gamma$, score($\Gamma$))
5: while $(\Gamma, s) \leftarrow H.pop()$ do
6:     $Q^*$ = BuildQueryGraph($Q^S$, $\Gamma$)
7:     SubgraphMatching($G$, $Q^*$) // Any subgraph isomorphism algorithm such as VF2
8:     for Each candidate list $L_i$ do
9:         $\mathit{\Gamma} = \Gamma$ plus one at the $i$-th bit
10:        $H.push(\mathit{\Gamma}$, score($\mathit{\Gamma}$))
11:    if already find k matches then
12:        Break
13: Output the top-k matches
```

Algorithm 4 Building Hyper Semantic Query Graph

Input: question sentence N, Nodes set V, dependency tree Y
Output: a super semantic query graph

```
1: for each $u \in V$ do
2:     Initialize $visit$ as an empty set
3:     Expand($u$, $u$)
```

Expand($head$, u)

```
1: $visit \leftarrow u$
2: if $u \in V$ then
3:     connect $head$ and $u$
4:     return
5: for each vetex $v$ connected with $u$ in $Y$ do
6:     if $v \notin visit$ then
7:         Expand($head$,$v$)
```

Algorithm 5 Bottom-up Algorithm

Input: A super semantic query graph Q^U and a RDF graph G.
Output: The top-k approximate matches from Q^U to G.

```
1: $Q \leftarrow$ start node $s$
2: $que$.push($s$)
3: while $x$ = $que$.pop() do
4:     /*Try to expand current query graph*/
5:     for each $\overline{v_i x} \in E(Q^U) \wedge \overline{v_i x} \notin E(Q)$ do
6:         $Q = Q \cup \overline{v_i x}$
7:         if GraphExplore(G, $Q$) find matches then
8:             if $Q$ is a spanning subgraph of $Q^U$ then
9:                 Insert matches of $Q$ into answer set $RS$.
10:                Only keep the matches in $RS$ with the top-k match scores
11:        else
12:            $Q$ = Backtrack($Q$, $\overline{v_i x}$)
13:        if $\overline{v_i x} \in Q$ then
14:            $que$.push($v_i$)
```

TABLE 1
A Sample of Textual Patterns and Predicates/Predicate Paths in DBpedia

Relation Phrases	Predicates \Predicate Paths	Confidence Probability
"was married to"	<spouse>	1.00
"was born in"	<birthplace>	1.00
"mother of"	<parent>	0.95
"are located in"	<locatedInArea>	0.98
"is fed by"	<inflow>	1.00
"open in"	<locationCity>	1.00
"is coauthor of"	<author> <author>	1.00

TABLE 2
Running Time of Offline Processing

	$\theta = 2$	$\theta = 4$
wordnet-wikipedia	17 mins	3.88 hrs
freebase-wikipedia	119 mins	30.33 hrs

Appendix B
Experiments of Offline Performance

Exp 1. Precision of Relation Mention Dictionary. In this experiment, we evaluate the accuracy of our building relation mention dictionary method. For each relation mention, we output a list of predicates/predicate paths. They are ranked in the non-descending order of confidence probabilities. Table 1 shows a sample of outputs in DBpedia. Note that the confidence probabilities in Tables 1 are normalized.

In order to measure the accuracy, we perform the following experiments. We randomly select 1000 relation mentions from wordnet-wikipedia and freebase-wikipedia datasets, respectively. For each relation mention, we output top-3 corresponding predicates/predicate paths. These results are shown to human judges. For each relation mention and its corresponding predicate/predicate path, the judge has to decide a scale from 2 to 0. The result is correct and highly relevant (2), correct but less relevant (1), or irrelevant (0). We find the precision (P@3) is about 50$\%$ when the path length is 1. However, while increasing of path length (from 2 to 4), the precision goes down greatly. To guarantee the precision of the relation mention dictionary for online process, the top-3 predicate paths (for each relation mention) should go through a human verification process.

Exp 2. Running Time of Building Relation Mention Dictionary. In this experiment, we evaluate the efficiency of our approach. Table 2 shows the total time. For example, when the path length threshold $\theta = 2$, the running time is 17 minutes using wordnet-wikipedia relation mention dataset and DBpedia RDF graph. Obviously, with the increasing of path length, the running time is increasing as well. On the other hand, with the increasing of path length, the precision of the results is decreasing. As default, we set $\theta = 4$. The predicate paths with length longer than 4 will not be considered in our method.

Appendix C
Examples

In Section 4.1.2, to find associated nodes of recognized relations, we introduce several heuristic rules as following.

- Rule 1: Extend the embedding t with some light words, such as prepositions, auxiliaries.
- Rule 2: If the root node of t has subject/object-like relations with its parent node in Y, add the parent node to arg1.
- Rule 3: If the parent of the root node of t has subject-like relations with its neighbors, add the child to arg1.
- Rule 4: If one of arg1/arg2 is empty, add the nearest wh-word or the first noun phrase in t to arg1/arg2.

Now we give the examples of the four heuristic rules.

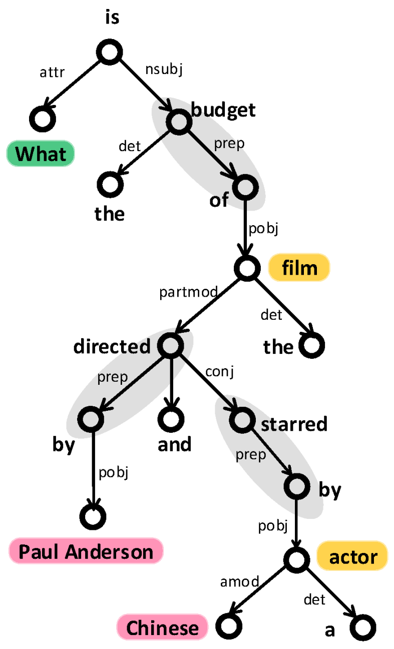

Fig. 1. Finding associated nodes in $Y(N_2)$

Example 1. Let us consider the dependency tree $Y(N_2)$ in Figure 1. For the relation embedding t_1="budget of", we use Rule 1 to extend t_1 with the word "is". Then the Rule 4 can be used for t_1 to get the wh-word "What" as the arg1. For the relation embedding t_2="directed by", as the root node of t_2 ("directed") has subject-like relation with its parent node ("film"), we use Rule 2 to add "film" to arg1. We get "what" as the first node of relation mention "budget of" by applying Rule 4. For the relation embedding t_3="starred by", as the parent of the root node of t_3 ("directed") has subject-like relations with its neighbors ("film"), we use Rule 3 to add "film" to arg1.