Program Transfer for Answering Complex Questions over Knowledge Bases

Anonymous ACL submission

Abstract

Program induction for answering complex questions over knowledge bases (KBs) aims to decompose a question into a multi-step program, whose execution against the KB produces the final answer. Learning to induce programs relies on a large number of parallel question-program pairs for the given KB. However, for most KBs, the gold program annotations are usually lacking, making learning difficult. In this paper, we propose the approach of **program transfer**, which aims to leverage the valuable program annotations on the rich-resourced KBs as external supervision signals to aid program induction for the low-resourced KBs that lack program annotations. For program transfer, we design a novel two-stage parsing framework with an efficient ontology-guided pruning strategy. First, a sketch parser translates the question into a high-level program sketch, which is the composition of functions. Second, given the question and sketch, an argument parser searches the detailed arguments from the KB for functions. During the searching, we incorporate the KB ontology to prune the search space. The experiments on ComplexWebQuestions and WebQuestionSP show that our method outperforms SOTA methods significantly, demonstrating the effectiveness of program transfer and our framework.

1 Introduction

Answering complex questions over knowledge bases (Complex KBQA) is a challenging task requiring logical, quantitative, and comparative reasoning over KBs (Hu et al., 2018; Lan et al., 2021). Recently, the program induction (PI) paradigm, which gains increasing study in various areas (Lake et al., 2015; Neelakantan et al., 2017; Wong et al., 2021), emerges as a promising technique for Complex KBQA (Liang et al., 2017; Saha et al., 2019a; Ansari et al., 2019). Given a KB, PI for Complex KBQA aims to decompose a complex question into a multi-step program, whose execution on the KB produces the answer. Fig. 1 presents a complex question and its corresponding program whose functions take KB elements (*i.e.*, entities, relations and concepts) as arguments. *E.g.*, the relation `tourist attractions` is the argument of function *Relate*.

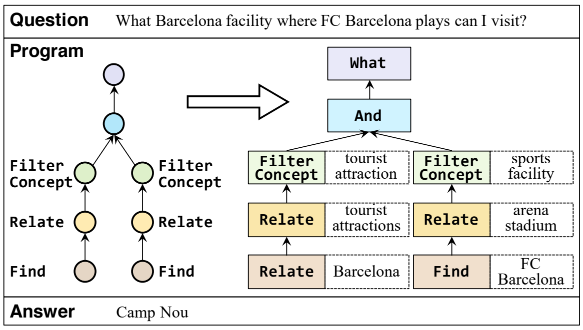

Figure 1: An example question, the corresponding program, and the answer. The left side is the sketch, and the right side is the complete program, with dotted boxes denoting arguments for functions.

For most KBs, the parallel question-program pairs are lacking because such annotation is both expensive and labor-intensive. Thus, the PI models have to learn only from question-answer pairs. Typically, they take the answers as weak supervision and search for gold programs with reinforcement learning (RL) (Saha et al., 2019b; Liang et al., 2017; Ansari et al., 2019). The combinatorial explosion in program space, along with extremely sparse rewards, makes the learning challenging. Abundant attempts have been made to improve the stability of RL algorithms with pseudo-gold programs (Liang et al., 2017), noise-stabilizing wrapper (Ansari et al., 2019), or auxiliary rewards (Saha et al., 2019b). Despite promising results, they require significant human efforts to develop carefully-designed heuristics or are constrained to relatively simple questions.

Recently, for several KBs, there emerge question-program annotation resources (Johnson et al., 2017; Shi et al., 2020). Thanks to the supervision signals (*i.e.*, program annotation for each question), the PI models on these rich-resourced KBs achieve impressive performance for even extremely complex questions, and are free from expert engineering. Intuitively, leveraging these supervision signals to aid program induction for low-resourced KBs with only weak-supervision signals (*i.e.*, question-answer pairs) is a promising direction. In this paper, we formalize it as **Program Transfer**.

In practice, program transfer is challenging due to the following reasons: (a) **Domain Heterogeneity**. The questions and KBs across domains are both heterogeneous due to language and knowledge diversity (Lan et al., 2021). It is hard to decide what to transfer for program induction. (b) **Unseen KB Elements.** The coverage of source KB is limited, *e.g.*, KQA Pro in (Shi et al., 2020) covers only 3.9% relations and 0.24% concepts of Wikidata. Thus, most elements in the massive scale target KB are not covered in the source. (c) **Huge Search Space.** The search space of function arguments depends on the scale of target KB. For realistic KBs containing millions of entities, concepts and relations, the huge search space is unmanageable.

To address the above problems, we propose a novel two-stage parsing framework with an efficient ontology-guided pruning strategy. First, we design a **sketch parser** to parse the question into a program sketch (the left side in Fig. 1), which is composed of functions without arguments. As Baroni (2019) points out, the composition of functions well captures the language compositionality. Translation from questions to sketches is thus relevant to language compositional structure and independent of KB structure. Therefore, our sketch parser can transfer across KBs. Second, we design an **argument parser** to fill in the detailed arguments (typically KB elements) for functions in the sketch. It retrieves relevant KB elements from the target KB and ranks them according to the question. Specifically, it identifies KB elements with their label descriptions and relies on language understanding to resolve unseen ones. We further propose an **ontology-guided pruning** strategy, which introduces high-level KB ontology to prune the candidate space for the argument parser, thus alleviating the problem of huge search space.

Specifically, the sketch parser is implemented with a Seq2Seq model with the attention mechanism. The argument parser identifies elements through semantic matching and utilizes pre-trained language models (Devlin et al., 2019) for language understanding. The high-level ontology includes the domain and range of relations and entity types.

In evaluation, we take the Wikidata-based KQA Pro as the source, Freebase-based ComplexWebQuestions and WebQuestionSP as the target domain datasets. Experimental results show that our method improves the F1 score by 14.7% and 2.5% respectively, compared with SOTA methods that learn from question-answer pairs.

Our contributions include: (a) proposing the approach of program transfer for Complex KBQA for the first time; (b) proposing a novel two-stage parsing framework with an efficient ontology-guided pruning strategy for program transfer; (c) demonstrating the effectiveness of program transfer through extensive experiments and careful ablation studies on two benchmark datasets.

2 Related Work

KBQA. KBQA aims to find answers for questions expressed in natural language from a KB, such as Freebase (Bollacker et al., 2008), DBpedia (Lehmann et al., 2015) and Wikidata (Vrandecic and Krötzsch, 2014). Current methods for KBQA can be categorized into two groups: 1) semantic parsing based methods (Berant et al., 2013; Yih et al., 2015; Liang et al., 2017; Ansari et al., 2019), which learn a semantic parser that converts questions into intermediate logic forms which can be executed against a KB; 2) information retrieval based methods (Bordes et al., 2014; Xu et al., 2016; Miller et al., 2016; Zhang et al., 2018; Sun et al., 2018, 2019), which retrieve candidate answers from the topic-entity-centric subgraph and then rank them according to the questions. Recently, semantic parsing for KBQA has gained increasing research attention because the methods are effective and more interpretable. Multiple kinds of logical forms have been proposed and researched, such as SPARQL (hommeaux, 2011), λ-DCS (Liang, 2013), λ-calculus (Artzi et al., 2013), query graph (Yih et al., 2015), program (Liang et al., 2017). PI aims to convert questions into programs, and is in line with semantic parsing.

Cross-domain Semantic Parsing. Cross-domain semantic parsing trains a semantic parser on some source domains and adapts it to the target domain. Some works (Herzig and Berant, 2017; Su and Yan, 2017; Fan et al., 2017) pool together examples from multiple datasets in different domains and train a single sequence-to-sequence model over all examples, sharing parameters across domains. However, these methods rely on annotated logic forms in the target domain. To facilitate low-resource target domains, (Chen et al., 2020) adapts to target domains with a very limited amount of annotated data. Other works consider a zero-shot semantic parsing task (Givoli and Reichart, 2019), decoupling structures from lexicons for transfer. However, they only learn from the source domain without further learning from the target domain using the transferred prior knowledge. In addition, existing works mainly focus on the domains in OVERNIGHT (Wang et al., 2015), which are much smaller than large scale KBs such as Wikidata and Freebase. Considering the complex schema of large scale KBs, transfer in ours setting is more challenging.

3 Problem Formulation

In this section, we first give some necessary definitions and then formulate our task.

Knowledge Bases. Knowledge base describes concepts, entities, and the relations between them. It can be formalized as $\mathcal{KB} = \{\mathcal{C}, \mathcal{E}, \mathcal{R}, \mathcal{T}\}$. $\mathcal{C}$, $\mathcal{E}$, $\mathcal{R}$ and $\mathcal{T}$ denote the sets of concepts, entities, relations and triples respectively. Relation set $\mathcal{R}$ can be formalized as $\mathcal{R} = \{r_e, r_c\} \cup \mathcal{R}_l$, where r_e is `instanceOf`, r_c is `subClassOf`, and $\mathcal{R}_l$ is the general relation set. $\mathcal{T}$ can be divided into three disjoint subsets: (1) `instanceOf` triple set $\mathcal{T}_e = \{(e, r_e, c)|e \in \mathcal{E}, c \in \mathcal{C}\}$; (2) `subClassOf` triple set $\mathcal{T}_c = \{(c_i, r_c, c_j)|c_i, c_j \in \mathcal{C}\}$; (3) relational triple set $\mathcal{T}_l = \{(e_i, r, e_j)|e_i, e_j \in \mathcal{E}, r \in \mathcal{R}_l\}$.

Program. Program is composed of symbolic functions with arguments, and produces an answer when executed against a KB. Each function defines a basic operation on KB and takes a specific type of argument. For example, the function *Relate* aims to find entities that have a specific *relation* with the given entity. Formally, a program y is denoted as $\langle o_1[arg_1], \cdots, o_t[arg_t], \cdots, o_{|y|}[arg_{|y|}] \rangle, o_t \in \mathcal{O}, arg_t \in \mathcal{E} \cup \mathcal{C} \cup \mathcal{R}$. Here, $\mathcal{O}$ is a pre-defined function set, which covers basic reasoning operations over KBs (Shi et al., 2020). According to the argument type, $\mathcal{O}$ can be devided into four disjoint subsets: $\mathcal{O} = \mathcal{O}^{\mathcal{E}} \cup \mathcal{O}^{\mathcal{C}} \cup \mathcal{O}^{\mathcal{R}} \cup \mathcal{O}^{\emptyset}$, representing the functions whose argument type is entity, concept, relation and empty respectively. Table 1 gives some examples of program functions.

Function	Argument Type	Argument	Description
Find	entity	`FC Barcelona`	Find the specific KB entity
Relate	relation	`arena stadium`	Find the entities that hold a specific relation with the given entity
FilterConcept	concept	`sports facility`	Find the entities that belong to a specific concept
And	-	-	Return the intersection of two entity sets

Table 1: Function examples. - means empty.

Program Induction. Given a $\mathcal{KB}$, and a complex natural language question $x = \langle w_1, w_2, \cdots, w_{|x|} \rangle$, it aims to produce a program y that generates the right answer z when executed against $\mathcal{KB}$.

Program Transfer. In this task, we have access to the source domain data $S = \langle \mathcal{KB}^S, \mathcal{D}^S \rangle$, where $\mathcal{D}^S$ contains pairs of question and program $\{(x_i^S, y_i^S)\}_{i=1}^{n^S}$; and target domain data $T = \langle \mathcal{KB}^T, \mathcal{D}^T \rangle$, where $\mathcal{D}^T$ contains pairs of question and answer $\{(x_i^T, z_i^T)\}_{i=1}^{n^T}$. We aim at learning a PI model to translate a question x for $\mathcal{KB}^T$ into program y, which produces the correct answer when executed on $\mathcal{KB}^T$.

4 Framework

As mentioned in the introduction, to perform program transfer for Complex KBQA, we need to address three crucial problems: (1) What to transfer when both questions and KBs are heterogeneous? (2) How to deal with the KB elements unseen in the external annotations? (3) How to prune the search space of input arguments to alleviate the huge search space problem? In this section, we introduce our two-stage parsing framework with an ontology-guided pruning strategy, which is shown in Fig. 2.

(1) **Sketch Parser**: At the first stage, we design a sketch parser f^s to parse x into a program sketch $y_s = \langle o_1, \cdots, o_t, \cdots o_{|y|} \rangle$, which is a sequence of functions without arguments. The sketch parsing process can be formulated as

$$y_s = f^s(x). \quad (1)$$

Translation from question to sketch is relevant to language compositionality, and irrelevant to KB structure. Therefore, the sketch parser can generalize across KBs.

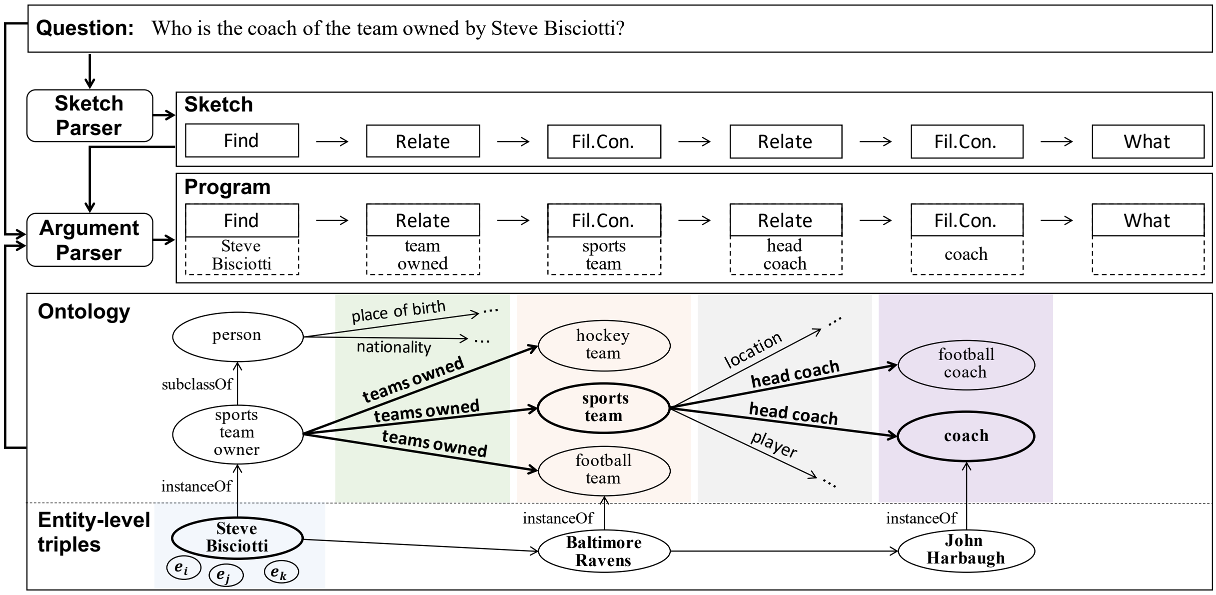

Figure 2: We design a high-level sketch parser to generate the sketch, and a low-level argument parser to predict arguments for the sketch. The arguments are retrieved from candidate pools which are illustrated by the color blocks. The arguments for functions are mutually constrained by the ontology structure. For example, when the second function *Relate* finds the argument `teams owned`, the candidate pool for the third function *Fil.Con.* (short for *FilterConcept*) is reduced to the range of relation `teams owned`.

(2) **Argument Parser**: At the second stage, we design an argument parser f^a to retrieve the argument arg_t from a candidate pool $\mathcal{P}$ for each function o_t, which can be formulated as

$$arg_t = f^a(x, o_t, \mathcal{P}). \tag{2}$$

Here, the candidate pool $\mathcal{P}$ contains the relevant elements in $\mathcal{KB}^T$, including concepts, entities, and relations. In a real KB, the candidate pool is usually huge, which makes searching and learning from answers very hard. Therefore, we propose an ontology-guided pruning strategy, which dynamically updates the candidate pool and progressively reduces its search space.

In the following we will introduce the implementation details of our sketch parser (Section 4.1), argument parser (Section 4.2) and training strategies (Section 4.3).

4.1 Sketch Parser

The sketch parser is based on encoder-decoder model (Sutskever et al., 2014) with attention mechanism (Dong and Lapata, 2016). We aim to estimate $p(y_s|x)$, the conditional probability of sketch y_s given input x. It can be decomposed as:

$$p(y_s|x) = \prod_{t=1}^{|y_s|} p(o_t|o_{<t}, x), \tag{3}$$

where $o_{<t} = o_1, ..., o_{t-1}$.

Specifically, our sketch parser comprises a question encoder that encodes the question into vectors and a sketch decoder that autoregressively outputs the sketch step-by-step. The details are as follows:

Question Encoder. We utilize BERT (Devlin et al., 2019) as the encoder. Formally,

$$\bar{\mathbf{x}}, (\mathbf{x}_1, \cdots, \mathbf{x}_i, \cdots, \mathbf{x}_{|x|}) = \text{BERT}(x), \tag{4}$$

where $\bar{\mathbf{x}} \in \mathbb{R}^{\hat{d}}$ is the question embedding, and $\mathbf{x}_i \in \mathbb{R}^{\hat{d}}$ is the hidden vector of word x_i. $\hat{d}$ is the hidden dimension.

Sketch Decoder. We use Gated Recurrent Unit (GRU) (Cho et al., 2014), a well-known variant of RNNs, as our decoder of program sketch. The decoding is conducted step by step. After we have predicted o_{t-1}, the hidden state of step t is computed as:

$$\mathbf{h}_t = \text{GRU}(\mathbf{h}_{t-1}, \mathbf{o}_{t-1}), \tag{5}$$

where $\mathbf{h}_{t-1}$ is the hidden state from last time step, $\mathbf{o}_{t-1} = [\mathbf{W}]_{o_{t-1}}$ denotes the embedding corresponding to o_{t-1} in the embedding matrix $\mathbf{W} \in \mathbb{R}^{|\mathcal{O}| \times d}$. We use $\mathbf{h}_t$ as the attention key to compute scores for each word in the question based on the hidden vector $\mathbf{x}_i$, and compute the attention vector $\mathbf{c}_t$ as:

$$\alpha_i = \frac{\exp(\mathbf{x}_i^{\mathrm{T}}\mathbf{h}_t)}{\sum_{j=1}^{|x|}\exp(\mathbf{x}_j^{\mathrm{T}}\mathbf{h}_t)}, \qquad \mathbf{c}_t = \sum_{i=1}^{|x|}\alpha_i\mathbf{x}_i. \tag{6}$$

The information of $\mathbf{h}_t$ and $\mathbf{c}_t$ are fused to predict the final probability of the next sketch token:

$$\mathbf{g}_t = \mathbf{h}_t + \mathbf{c}_t, \qquad p(o_t|o_{<t}, x) = [\mathrm{Softmax}(\mathrm{MLP}(\mathbf{g}_t))]_{o_t}, \tag{7}$$

where MLP (short for multi-layer perceptron) projects $\hat{d}$-dimensional feature to $|\mathcal{O}|$-dimension, which consists of two linear layers with ReLU activation.

4.2 Argument Parser

In the above section, the sketch is obtained with a sketch parser. In this section, we will introduce our argument parser, which aims to retrieve the argument arg_t from the target KB for each function o_t in the sketch. To reduce the search space, it retrieves arguments from a restricted candidate pool $\mathcal{P}$, which is constructed with our ontology-guided pruning strategy. In the following, we will introduce the argument retrieval process and the candidate pool construction process.

Argument Retrieval. Specifically, we take $\mathbf{g}_t$ in Equation 7 as the context representation of o_t, learn vector representation $\mathbf{P}_i \in \mathbb{R}^{\hat{d}}$ for each candidate $\mathcal{P}_i$, and calculate the probability for $\mathcal{P}_i$ based on $\mathbf{g}_t$ and $\mathbf{P}_i$. Candidate $\mathcal{P}_i$ is encoded with the BERT encoder in Equation 4, which can be formulated as:

$$\mathbf{P}_i = \mathrm{BERT}(\mathcal{P}_i). \tag{8}$$

$\mathbf{P}_i$ is the i^{th} row of $\mathbf{P}$. The probability of candidate arg_t is calculated as:

$$p(arg_t|x, o_t, \mathcal{P}) = [\mathrm{Softmax}(\mathbf{P}\mathbf{g}_t)]_{arg_t}. \tag{9}$$

Candidate Pool Construction. In the following, we will introduce the KB ontology first. Then, we will describe the rationale of our ontology-guided pruning strategy and its implementation details.

In KB, The domain and range of relations, and the type of entities form the KB ontology. Specifically, a relation r comes with a domain $dom(r) \subseteq \mathcal{C}$ and a range $ran(r) \subseteq \mathcal{C}$. An entity e comes with a type $type(e) = \{c|(e, \texttt{instanceOf}, c) \in \mathcal{T}\}$. For example, as shown in Fig. 2, `sports team owner` $\in dom($`teams owned`$)$, `sports team` $\in ran($`teams owned`$)$, and `sports team` $\in type($`Baltimore Ravens`$)$.

The rationale of our pruning is that the arguments for program functions are mutually constrained according to the KB ontology. Therefore, when the argument arg_t for o_t is determined, the possible candidates for $\{o_i\}_{i=t+1}^{|y_s|}$ will be adjusted. For example, in Fig. 2, when *Relate* takes `teams owned` as the argument, the candidate pool for the next *FilterConcept* is constrained to the range of relation `teams owned`, thus other concepts (*e.g.*, `time zone`) will be excluded from the candidate pool.

In practice, we propose a set of ontology-oriented operators to adjust the candidate pool $\mathcal{P}$ step-by-step. Specifically, we define three ontology-oriented operators $C(e), R(r), D^{-}(c)$, which aim to find the type of entity e, the range of relation r, and the relations whose domain contains c. Furthermore, we use the operators to maintain an entity pool $\mathcal{P}^{\mathcal{E}}$, a relation pool $\mathcal{P}^{\mathcal{R}}$ and a concept pool $\mathcal{P}^{\mathcal{C}}$. When arg_t of o_t is determined, we will update $\mathcal{P}^{\mathcal{E}}$, $\mathcal{P}^{\mathcal{R}}$, and $\mathcal{P}^{\mathcal{C}}$ using $C(e), R(r), D^{-}(c)$. We take one of the three pools as $\mathcal{P}$ according to the argument type of o_t. The detailed algorithm is shown in Appendix.

4.3 Training

We train our model using the popular pretrain-finetune paradigm. Specifically, we pretrain the parsers on the source domain data $\mathcal{D}^S = \left\{\left(x_i^S, y_i^S\right)\right\}_{i=1}^{n^S}$ in a supervised way. After that, we conduct finetuning on the target domain data $\mathcal{D}^T = \left\{\left(x_i^T, z_i^T\right)\right\}_{i=1}^{n^T}$ in a weakly supervised way.

Pretraining in Source Domain. Since the source domain data provides complete annotations, we can directly maximize the log-likelihood of the golden sketch and golden arguments:

$$\mathcal{L}^{\text{pretrain}} = -\sum_{(x^S, y^S)\in\mathcal{D}^S}\Bigg(\log p(y_s^S|x^S) + \sum_{t=1}^{|y_s|}\log p(arg_t^S|x^S, o_t^S, \mathcal{P})\Bigg). \tag{10}$$

Finetuning in Target Domain. At this training phase, questions are labeled with answers while programs remain unknown. The basic idea is to search for potentially correct programs and optimize their corresponding probabilities. Specifically, we propose two training strategies:

- Hard-EM Approach. At each training step, hard-EM generates a set of possible programs with beam search based on current model parameters, and then executes them to find the one whose answers have the highest F1 score compared with the gold. Let $\hat{y}^T$ denote the best program, we directly maximize $p(\hat{y}^T|x^T)$ like Equation 10.
- Reinforcement learning (RL). It formulates the program generation as a decision making procedure and computes the rewards for sampled programs based on their execution results. We take the F1 score between the executed answers and golden answers as the reward value, and use REINFORCE (Williams, 1992) algorithm to optimize the parsers.

5 Experimental Settings

5.1 Datasets

Source Domain. KQA Pro (Shi et al., 2020) provides 117,970 question-program pairs based on a Wikidata (Vrandecic and Krötzsch, 2014) subset.
Target Domain. We use WebQuestionSP (WebQSP) (Yih et al., 2016) and ComplexWebQuestions (CWQ) (Talmor and Berant, 2018) as the target domain datasets for two reasons: (1) They are two widely used benchmark datasets in Complex KBQA; (2) They are based on a large-scale KB Freebase (Bollacker et al., 2008), which makes program transfer challenging. Specifically, WebQSP contains 4,737 questions and is divided into 2,998 train, 100 dev and 1,639 test cases. CWQ is an extended version of WebQSP which is more challenging, with four types of questions: composition (44.7%), conjunction (43.6%), comparative (6.2%), and superlative (5.4%). CWQ is divided into 27,639 train, 3,519 dev and 3,531 test cases.

We use the Freebase dump on 2015-08-09[1], from which we extract the type of entities, domain and range of relations to construct the ontology. The average domain, range, type size is 1.43 per relation, 1.17 per relation, 8.89 per entity respectively.

Table 2 shows the statistics of the source and target domain KB. The target domain KB contains much more KB elements, and most of them are uncovered by the source domain.

[1]http://commondatastorage.googleapis.com/freebase-public/rdf/freebase-rdf-latest.gz

Domain	# Entities	# Relations	# Concepts
Source	16,960	363	794
Target	30,943,204	15,015	2,519

Table 2: The statistics for source and target domain KB.

5.2 Baselines

In our experiments, we select representative models that learn from question-answer pairs as our baselines. They can be categorized into three groups: program induction methods, query graph generation methods and information retrieval methods.

Existing program induction methods search for gold programs with RL. They usually require human efforts or are constrained to simple questions. **NSM** (Liang et al., 2017) uses the provided entity, relation and type annotations to ease the search, and can solve relatively simple questions. **NPI** (Ansari et al., 2019) designs heuristic rules such as disallowing repeating or useless actions for efficient search.

Existing query graph generation methods generate query graphs whose execution on KBs produces the answer. They use entity-level triples as search guidance, ignoring the useful ontology. **TEXTRAY** (Bhutani et al., 2019) uses a decompose-execute-join approach. **QGG** (Lan and Jiang, 2020) incorporates constraints into query graphs in the early stage. **TeacherNet** (He et al., 2021) utilizes bidirectional searching.

Existing information retrieval methods directly construct a question-specific sub-KB and then rank the entities in the sub-KB to get the answer. **GraftNet** (Sun et al., 2018) uses heuristics to create the subgraph and uses a variant of graph convolutional networks to rank the entities. **PullNet** (Sun et al., 2019) improves GraftNet by iteratively constructing the subgraph instead of using heuristics.

Besides, we compare our full model **Ours** with Ours$_{\text{-f}}$, Ours$_{\text{-p}}$, Ours$_{\text{-pa}}$, Ours$_{\text{-o}}$, which denotes our model without finetuning, without pretraining, without pretraining of argument parser, and without our ontology-guided pruning strategy respectively.

5.3 Evaluation Metrics

Following prior works (Berant et al., 2013; Sun et al., 2018; He et al., 2021), we use F1 score and Hit@1 as the evaluation metrics. Since questions in the datasets have multiple answers, F1 score reflects the coverage of predicted answers better.

5.4 Implementations

We used the bert-base-cased model of HuggingFace[2] as our BERT encoder with the hidden dimension $\hat{d}$ 768. The hidden dimension of the sketch decoder d was 1024. We used AdamW (Loshchilov and Hutter, 2019) as our optimizer. We searched the learning rate for BERT paramters in {1e-4, 3e-5, 1e-5}, the learning rate for other parameters in {1e-3, 1e-4, 1e-5}, and the weight decay in {1e-4, 1e-5, 1e-6}. According to the performance on dev set, we finally used learning rate 3e-5 for BERT parameters, 1e-3 for other parameters, and weight decay 1e-5.

6 Experimental Results

Models	WebQSP		CWQ	
	F1	Hit@1	F1	Hit@1
NSM	-	69.0	-	-
NPI	-	72.6	-	-
TEXTRAY	60.3	72.2	33.9	40.8
QGG	<u>74.0</u>	-	40.4	44.1
TeacherNet	67.4	<u>74.3</u>	<u>44.0</u>	<u>48.8</u>
GraftNet	62.3	68.7	-	32.8*
PullNet	-	68.1	-	47.2*
Ours$_{\text{-f}}$	53.8	53.0	45.9	45.2
Ours$_{\text{-p}}$	3.2	3.1	2.3	2.1
Ours$_{\text{-pa}}$	70.8	68.9	54.5	54.3
Ours$_{\text{-o}}$	72.0	71.3	55.8	54.7
Ours	**76.5**	**74.6**	**58.7**	**58.1**

Table 3: Performance comparison of different methods (F1 score and Hits@1 in percent). We highlight the best results in bold and second with an underline. *: reported by PullNet on the dev set.

6.1 Overall Results

As shown in Table 3, our model achieves the best performance on both WebQSP and CWQ. Especially on CWQ, we have an absolute gain of 14.7% in F1 and 9.3% in Hit@1, beating previous methods by a large margin. Note that CWQ is much more challenging than WebQSP because it includes more compositional and conjunctional questions. Previous works mainly suffer from the huge search space and sparse training signals. We alleviate these issues by transferring the prior knowledge from external annotations and incorporating the ontology guidance. Both of them reduce the search space substantially. On WebQSP, we achieve an absolute gain of 2.5% and 0.3% in F1 and Hit@1, respectively, demonstrating that our model can also handle simple questions well, and can adapt to different complexities of questions.

Note that our F1 scores are higher than the corresponding Hit@1. This is because we just randomly sampled one answer from the returned answer set as the top 1 without ranking them.

Models	WebQSP	CWQ
Top-1	76.5	58.7
Top-2	81.1	61.2
Top-5	85.4	63.3
Top-10	86.9	65.0

Table 4: The highest F1 score in the top-k programs.

We utilize beam search to generate multiple possible programs and evaluate their performance. Table 4 shows the highest F1 score in the top-k generated programs, where top-1 is the same as Table 3. We can see that the best F1 in the top-10 programs is much higher than the F1 of the top-1 (*e.g.*, with an absolute gain 10.4% for WebQSP and 6.3% for CWQ). This indicates that a good re-ranking method can further improve the overall performance of our model. We leave this as our future work.

6.2 Ablation study

Pretraining: As shown in Table 3, when comparing Ours$_{\text{-pa}}$ with Ours, the F1 and Hit@1 on CWQ drop by 4.2% and 3.8% respectively, which indicates that the pretraining for the argument parser is necessary. Ours$_{\text{-p}}$ denotes the model without pretraining for neither sketch parser nor argument parser. We can see that its results are very poor, achieving just about 3% and 2% on WebQSP and CWQ, indicating that the pretraining is essential, especially for the sketch parser.

Finetuning: Without finetuning on the target data, *i.e.*, in Ours$_{\text{-f}}$, performance drops a lot compared with the complete model. For example, F1 and Hit@1 on CWQ drop by 12.8% and 12.9% respectively. It indicates that finetuning is necessary for the model's performance. As shown in Table 2, most of the relations and concepts in the target domain are uncovered by the source domain. Due to the semantic gap between source and target data, the prior knowledge must be properly transferred to the target domain to bring into full play.

Ontology: We implemented Ours$_{\text{-o}}$ by removing ontology from KB and removing *FilterConcept* from the program. Comparing Ours$_{\text{-o}}$ with Ours, the F1 and Hit@1 on CWQ drops by 2.9% and

[2]https://github.com/huggingface/transformers

3.4% respectively, which demonstrates the importance of ontology-guided pruning strategy. We calculated the search space size for each compositional and conjunctive question in the dev set of CWQ, and report the average size in Table 5. The statistics shows that, the average search space size of Ours is only 0.26% and 3.2% of that in Ours$_{\text{-o}}$ for the two kinds of questions. By incorporating the ontology guidance, Ours substantially reduces the search space.

Model	Composition	Conjunction
Ours$_{\text{-o}}$	4,248,824.5	33,152.1
Ours	11,200.7	1,066.5

Table 5: The average search space size for composition and conjunction questions in CWQ set for Ours and Ours$_{\text{-o}}$.

Hard-EM *v.s.* RL: For both WebQSP and CWQ, training with Hard-EM achieves better performance. For RL, we simply employed the REINFORCE algorithm and did not implement any auxiliary reward strategy since this is not the focus of our work. The sparse, delayed reward causes high variance, instability, and local minima issues, making the training hard (Saha et al., 2019b). We leave exploring more complex training strategies as our future work.

Models	WebQSP		CWQ	
	F1	Hit@1	F1	Hit@1
Hard-EM	76.5	74.6	58.7	58.1
RL	71.4	72.0	46.1	45.4

Table 6: Results of different training strategies.

6.3 Case Study

Fig. 3 gives a case, where our model parses an question into multiple programs along with their probablility scores and F1 scores of executed answers. Given the question "*The person whose education institution is Robert G. Cole Junior-Senior High School played for what basketball teams?*", we show the programs with the largest, 2-nd largest and 10-th largest possibility score. Both of the top-2 programs get the correct answer set and are semantically equivelant with the question, while the 10-th best program is wrong.

Error Analysis We randomly sampled 100 error cases whose F1 score is lower than 0.1 for manual inspection. The errors can be summarized into the

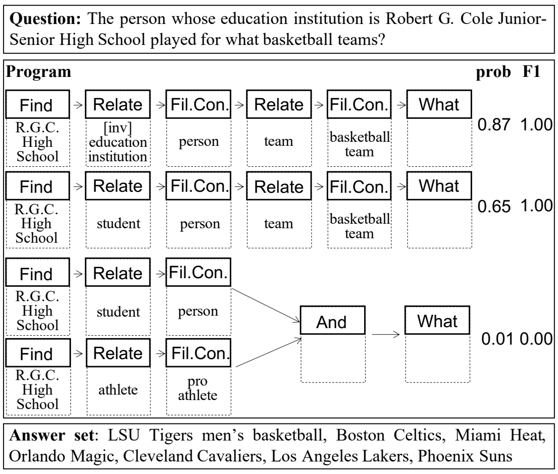

Figure 3: An example from CWQ dev set. Our model translates the question into multiple programs with the corresponding probability and F1 score. We show the best, 2-nd best and 10-th best programs. Both the best and 2-nd best programs are correct.

following categories: (1) Wrong relation (53%): wrongly predicted relation makes the program wrong, *e.g.*, for question " *What language do people in the Central Western Time Zone speak?*", our model predicts the relation `main country`, while the ground truth is `countries spoken in`; (2) Wrong concept (38%): wrongly predicted concept makes the program wrong, *e.g.*, for the question "*What continent does the leader Ovadia Yosel live in?*", our model predicted the concept `location`, whereas the ground truth is *continent*. (3) Model limitation (9%): Handling attribute constraint was not considered in our model, *e.g.*, for the question "*Who held his governmental position from before April 4, 1861 and influenced Whitman's poetry?*", the time constraint *April 4, 1861* cannot be handled.

7 Conclusion

In this parper, we propose program transfer for Complex KBQA for the first time. We propose a novel two-stage parsing framework with an efficient ontology-guided pruning strategy. First, a sketch parser translates a question into the program, and then an argument parser fills in the detailed arguments for functions, whose search space is restricted by an ontology-guided pruning strategy. The experimental results demonstrate that our program transfer approach outperforms the previous methods significantly. The ablation studies show that our two-stage parsing paradigm and ontology-guided pruning are both effective.

References

Ghulam Ahmed Ansari, Amrita Saha, Vishwajeet Kumar, Mohan Bhambhani, Karthik Sankaranarayanan, and Soumen Chakrabarti. 2019. Neural program induction for kbqa without gold programs or query annotations. In *IJCAI'19*.

Yoav Artzi, Nicholas FitzGerald, and Luke Zettlemoyer. 2013. Semantic parsing with combinatory categorial grammars. In *ACL'13*.

Marco Baroni. 2019. Linguistic generalization and compositionality in modern artificial neural networks. *Philosophical Transactions of the Royal Society B*, 375.

Jonathan Berant, Andrew K. Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on freebase from question-answer pairs. In *EMNLP'13*.

Nikita Bhutani, Xinyi Zheng, and H. Jagadish. 2019. Learning to answer complex questions over knowledge bases with query composition. In *CIKM'19*.

K. Bollacker, Colin Evans, Praveen K. Paritosh, Tim Sturge, and J. Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD'08*.

Antoine Bordes, Sumit Chopra, and Jason Weston. 2014. Question answering with subgraph embeddings. In *EMNLP'14*.

Xilun Chen, Asish Ghoshal, Yashar Mehdad, Luke Zettlemoyer, and S. Gupta. 2020. Low-resource domain adaptation for compositional task-oriented semantic parsing. In *EMNLP'20*.

Kyunghyun Cho, B. V. Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. *CoRR*, abs/1409.1259.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT'19*.

Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. *CoRR*, abs/1601.01280.

X. Fan, Emilio Monti, Lambert Mathias, and Markus Dreyer. 2017. Transfer learning for neural semantic parsing. *CoRR*, abs/1706.04326.

Ofer Givoli and Roi Reichart. 2019. Zero-shot semantic parsing for instructions. *CoRR*, abs/1911.08827.

Gaole He, Yunshi Lan, Jing Jiang, Wayne Xin Zhao, and Ji-Rong Wen. 2021. Improving multi-hop knowledge base question answering by learning intermediate supervision signals.

Jonathan Herzig and Jonathan Berant. 2017. Neural semantic parsing over multiple knowledge-bases. *CoRR*, abs/1702.01569.

E. P. hommeaux. 2011. SPARQL query language for RDF.

Sen Hu, Lei Zou, and Xinbo Zhang. 2018. A state-transition framework to answer complex questions over knowledge base. In *EMNLP*.

Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C. Lawrence Zitnick, and Ross B. Girshick. 2017. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1988–1997.

B. Lake, R. Salakhutdinov, and J. Tenenbaum. 2015. Human-level concept learning through probabilistic program induction. *Science*, 350:1332 – 1338.

Yunshi Lan, Gaole He, Jinhao Jiang, Jing Jiang, Wayne Xin Zhao, and Ji-Rong Wen. 2021. A survey on complex knowledge base question answering: Methods, challenges and solutions. In *IJCAI*.

Yunshi Lan and J. Jiang. 2020. Query graph generation for answering multi-hop complex questions from knowledge bases. In *ACL'20*.

Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, D. Kontokostas, Pablo N. Mendes, Sebastian Hellmann, M. Morsey, Patrick van Kleef, S. Auer, and C. Bizer. 2015. Dbpedia - a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*.

Chen Liang, Jonathan Berant, Quoc V. Le, Kenneth D. Forbus, and N. Lao. 2017. Neural symbolic machines: Learning semantic parsers on freebase with weak supervision. In *ACL'17*.

P. Liang. 2013. Lambda dependency-based compositional semantics. *CoRR*, abs/1309.4408.

I. Loshchilov and F. Hutter. 2019. Decoupled weight decay regularization. In *ICLR'19*.

Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. 2016. Key-value memory networks for directly reading documents. In *EMNLP'16*.

Arvind Neelakantan, Quoc V. Le, Martín Abadi, A. McCallum, and Dario Amodei. 2017. Learning a natural language interface with neural programmer. *ArXiv*, abs/1611.08945.

Amrita Saha, Ghulam Ahmed Ansari, Abhishek Laddha, Karthik Sankaranarayanan, and Soumen Chakrabarti. 2019a. Complex program induction for querying knowledge bases in the absence of gold programs. *Transactions of the Association for Computational Linguistics*, 7:185–200.

Amrita Saha, Ghulam Ahmed Ansari, Abhishek Laddha, Karthik Sankaranarayanan, and Soumen Chakrabarti. 2019b. Complex program induction for querying knowledge bases in the absence of gold programs. *Transactions of the Association for Computational Linguistics*, 7:185–200.

Jiaxin Shi, Shulin Cao, Liangming Pan, Yutong Xiang, Lei Hou, Juanzi Li, Hanwang Zhang, and Bin He. 2020. KQA Pro: A large diagnostic dataset for complex question answering over knowledge base. *CoRR*, abs/2007.03875.

Yu Su and X. Yan. 2017. Cross-domain semantic parsing via paraphrasing. In *EMNLP'17*.

Haitian Sun, Tania Bedrax-Weiss, and William Cohen. 2019. Pullnet: Open domain question answering with iterative retrieval on knowledge bases and text. In *EMNLP'19*.

Haitian Sun, Bhuwan Dhingra, Manzil Zaheer, Kathryn Mazaitis, Ruslan Salakhutdinov, and William Cohen. 2018. Open domain question answering using early fusion of knowledge bases and text. In *EMNLP'18*.

I. Sutskever, O. Vinyals, and Q. V Le. 2014. Sequence to sequence learning with neural networks. In *NIPS'14*.

Alon Talmor and Jonathan Berant. 2018. The web as a knowledge-base for answering complex questions. In *NAACL-HLT'18*.

Denny Vrandecic and M. Krötzsch. 2014. Wikidata: a free collaborative knowledgebase. *Communications of the ACM*.

Y. Wang, Jonathan Berant, and Percy Liang. 2015. Building a semantic parser overnight. In *ACL'15*.

Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*.

Catherine Wong, Kevin Ellis, Joshua B. Tenenbaum, and Jacob Andreas. 2021. Leveraging language to learn program abstractions and search heuristics. In *ICML*.

Kun Xu, Siva Reddy, Yansong Feng, Songfang Huang, and Dongyan Zhao. 2016. Question answering on freebase via relation extraction and textual evidence. In *ACL'16*.

Wen-tau Yih, Ming-Wei Chang, X He, and Jianfeng Gao. 2015. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *ACL*.

Wen-tau Yih, Matthew Richardson, Christopher Meek, Ming-Wei Chang, and Jina Suh. 2016. The value of semantic parse labeling for knowledge base question answering. In *ACL'16*.

Yuyu Zhang, Hanjun Dai, Zornitsa Kozareva, Alexander Smola, and Le Song. 2018. Variational reasoning for question answering with knowledge graph. In *AAAI'18*.

A Ontology-guided Pruning

Algorithm 1 Ontology-guided Pruning

Input: natural language question x, program sketch y_s, knowledge base $\mathcal{KB} = \{\mathcal{C}, \mathcal{E}, \mathcal{R}, \mathcal{T}\}$

Output: $\{arg_t\}_{t=1}^{|y_s|}$

$\mathcal{P}^{\mathcal{E}} \leftarrow \mathcal{E}, \mathcal{P}^{\mathcal{R}} \leftarrow \mathcal{R}, \mathcal{P}^{\mathcal{C}} \leftarrow \mathcal{C}, \mathcal{P} \leftarrow \emptyset$

for all o_t in y_s **do**

  if $o_t \in \mathcal{O}^{\mathcal{E}}$ **then**

    $\mathcal{P} \leftarrow \mathcal{P}^{\mathcal{E}}$

    $arg_t = f^a(x, o_t, \mathcal{P})$

    $\mathcal{P}^{\mathcal{C}} \leftarrow C(arg_t)$

    $\mathcal{P}^{\mathcal{R}} \leftarrow \bigcup_{c \in \mathcal{P}^{\mathcal{C}}} D^-(c)$

  else if $o_t \in \mathcal{O}^{\mathcal{R}}$ **then**

    $\mathcal{P} \leftarrow \mathcal{P}^{\mathcal{R}}$

    $arg_t = f^a(x, o_t, \mathcal{P})$

    $\mathcal{P}^{\mathcal{C}} \leftarrow R(arg_t)$

  else if $o_t \in \mathcal{O}^{\mathcal{C}}$ **then**

    $\mathcal{P} \leftarrow \mathcal{P}^{\mathcal{C}}$

    $arg_t = f^a(x, o_t, \mathcal{P})$

    $\mathcal{P}^{\mathcal{R}} \leftarrow D^-(arg_t)$

  end if

end for

B Freebase Details

We extracted a subset of Freebase which contains all facts that are within 4-hops of entities mentioned in the questions of CWQ and WebQSP. We extracted the domain constraint for relations according to “ */type/property/schema*”, range constraint for relations according to “*/type/property/expected_type*”, type constraint for entities according to “*/type/type/instance*”. CVT nodes in the Freebase were dealed with concatenation of neiborhood relations.

C Program

We list the functions of KQA Pro in Table 7. The arguments in our paper are the textual inputs. To reduce the burden of the argument parser, for the functions that take multiple textual inputs, we concatenate them to a single input.

Function	**Functional Inputs** $\times$ **Textual Inputs** $\rightarrow$ **Outputs**	**Description**	**Example** (only show textual inputs)
FindAll	() $\times$ () $\rightarrow$ (*Entities*)	Return all entities in KB	-
Find	() $\times$ (*Name*) $\rightarrow$ (*Entities*)	Return all entities with the given name	*Find(Kobe Bryant)*
FilterConcept	(*Entities*) $\times$ (*Name*) $\rightarrow$ (*Entities*)	Find those belonging to the given concept	*FilterConcept(athlete)*
FilterStr	(*Entities*) $\times$ (*Key*, *Value*) $\rightarrow$ (*Entities*, *Facts*)	Filter entities with an attribute condition of string type, return entities and corresponding facts	*FilterStr(gender, male)*
FilterNum	(*Entities*) $\times$ (*Key*, *Value*, *Op*) $\rightarrow$ (*Entities*, *Facts*)	Similar to *FilterStr*, except that the attribute type is number	*FilterNum(height, 200 centimetres, $>$)*
FilterYear	(*Entities*) $\times$ (*Key*, *Value*, *Op*) $\rightarrow$ (*Entities*, *Facts*)	Similar to *FilterStr*, except that the attribute type is year	*FilterYear(birthday, 1980, $=$)*
FilterDate	(*Entities*) $\times$ (*Key*, *Value*, *Op*) $\rightarrow$ (*Entities*, *Facts*)	Similar to *FilterStr*, except that the attribute type is date	*FilterDate(birthday, 1980-06-01, $<$)*
QFilterStr	(*Entities*, *Facts*) $\times$ (*QKey*, *QValue*) $\rightarrow$ (*Entities*, *Facts*)	Filter entities and corresponding facts with a qualifier condition of string type	*QFilterStr(language, English)*
QFilterNum	(*Entities*, *Facts*) $\times$ (*QKey*, *QValue*, *Op*) $\rightarrow$ (*Entities*, *Facts*)	Similar to *QFilterStr*, except that the qualifier type is number	*QFilterNum(bonus, 20000 dollars, $>$)*
QFilterYear	(*Entities*, *Facts*) $\times$ (*QKey*, *QValue*, *Op*) $\rightarrow$ (*Entities*, *Facts*)	Similar to *QFilterStr*, except that the qualifier type is year	*QFilterYear(start time, 1980, $=$)*
QFilterDate	(*Entities*, *Facts*) $\times$ (*QKey*, *QValue*, *Op*) $\rightarrow$ (*Entities*, *Facts*)	Similar to *QFilterStr*, except that the qualifier type is date	*QFilterDate(start time, 1980-06-01, $<$)*
Relate	(*Entity*) $\times$ (*Pred*, *Dir*) $\rightarrow$ (*Entities*, *Facts*)	Find entities that have a specific relation with the given entity	*Relate(capital, forward)*
And	(*Entities*, *Entities*) $\times$ () $\rightarrow$ (*Entities*)	Return the intersection of two entity sets	-
Or	(*Entities*, *Entities*) $\times$ () $\rightarrow$ (*Entities*)	Return the union of two entity sets	-
QueryName	(*Entity*) $\times$ () $\rightarrow$ (string)	Return the entity name	-
Count	(*Entities*) $\times$ () $\rightarrow$ (number)	Return the number of entities	-
QueryAttr	(*Entity*) $\times$ (*Key*) $\rightarrow$ (*Value*)	Return the attribute value of the entity	*QueryAttr(height)*
QueryAttrUnderCondition	(*Entity*) $\times$ (*Key*, *QKey*, *QValue*) $\rightarrow$ (*Value*)	Return the attribute value, whose corresponding fact should satisfy the qualifier condition	*QueryAttrUnderCondition(population, point in time, 2016)*
QueryRelation	(*Entity*, *Entity*) $\times$ () $\rightarrow$ (*Pred*)	Return the predicate between two entities	*QueryRelation(Kobe Bryant, America)*
SelectBetween	(*Entity*, *Entity*) $\times$ (*Key*, *Op*) $\rightarrow$ (string)	From the two entities, find the one whose attribute value is greater or less and return its name	*SelectBetween(height, greater)*
SelectAmong	(*Entities*) $\times$ (*Key*, *Op*) $\rightarrow$ (string)	From the entity set, find the one whose attribute value is the largest or smallest	*SelectAmong(height, largest)*
VerifyStr	(*Value*) $\times$ (*Value*) $\rightarrow$ (boolean)	Return whether the output of *QueryAttr* or *QueryAttrUnderCondition* and the given value are equal as string	*VerifyStr(male)*
VerifyNum	(*Value*) $\times$ (*Value*, *Op*) $\rightarrow$ (boolean)	Return whether the two numbers satisfy the condition	*VerifyNum(20000 dollars, $>$)*
VerifyYear	(*Value*) $\times$ (*Value*, *Op*) $\rightarrow$ (boolean)	Return whether the two years satisfy the condition	*VerifyYear(1980, $>$)*
VerifyDate	(*Value*) $\times$ (*Value*, *Op*) $\rightarrow$ (boolean)	Return whether the two dates satisfy the condition	*VerifyDate(1980-06-01, $>$)*
QueryAttrQualifier	(*Entity*) $\times$ (*Key*, *Value*, *QKey*) $\rightarrow$ (*QValue*)	Return the qualifier value of the fact (*Entity*, *Key*, *Value*)	*QueryAttrQualifier(population, 23,390,000, point in time)*
QueryRelationQualifier	(*Entity*, *Entity*) $\times$ (*Pred*, *QKey*) $\rightarrow$ (*QValue*)	Return the qualifier value of the fact (*Entity*, *Pred*, *Entity*)	*QueryRelationQualifier(spouse, start time)*

Table 7: Details of 27 functions in KQA Pro. Each function has 2 kinds of inputs: the functional inputs come from the output of previous functions, while the textual inputs come from the question.