KQA Pro: A Dataset with Explicit Compositional Programs for Complex Question Answering over Knowledge Base

Shulin Cao[1,2*], **Jiaxin Shi**[1,3*], **Liangming Pan**[4], **Lunyiu Nie**[1], **Yutong Xiang**[5], **Lei Hou**[1,2†], **Juanzi Li**[1,2], **Bin He**[6], **Hanwang Zhang**[7]
[1]Department of Computer Science and Technology, BNRist
[2]KIRC, Institute for Artificial Intelligence, Tsinghua University, Beijing 100084, China
[3]Cloud BU, Huawei Technologies, [4]National University of Singapore, [5]ETH Zürich
[6]Noah's Ark Lab, Huawei Technologies, [7]Nanyang Technological University
{caosl19@mails., nlx20@mails., houlei@,lijuanzi@}tsinghua.edu.cn
shijx12@gmail.com, liangmingpan@u.nus.edu, hanwangzhang@ntu.edu.sg

Abstract

Complex question answering over knowledge base (Complex KBQA) is challenging because it requires various compositional reasoning capabilities, such as multi-hop inference, attribute comparison, set operation. Existing benchmarks have some shortcomings that limit the development of Complex KBQA: 1) they only provide QA pairs without explicit reasoning processes; 2) questions are poor in diversity or scale. To this end, we introduce KQA Pro, a dataset for Complex KBQA including ~120K diverse natural language questions. We introduce a compositional and interpretable programming language KoPL to represent the reasoning process of complex questions. For each question, we provide the corresponding KoPL program and SPARQL query, so that KQA Pro serves for both KBQA and semantic parsing tasks. Experimental results show that SOTA KBQA methods cannot achieve promising results on KQA Pro as on current datasets, which suggests that KQA Pro is challenging and Complex KBQA requires further research efforts. We also treat KQA Pro as a diagnostic dataset for testing multiple reasoning skills, conduct a thorough evaluation of existing models and discuss further directions for Complex KBQA. Our codes and datasets can be obtained from https://github.com/shijx12/KQAPro_Baselines.

1 Introduction

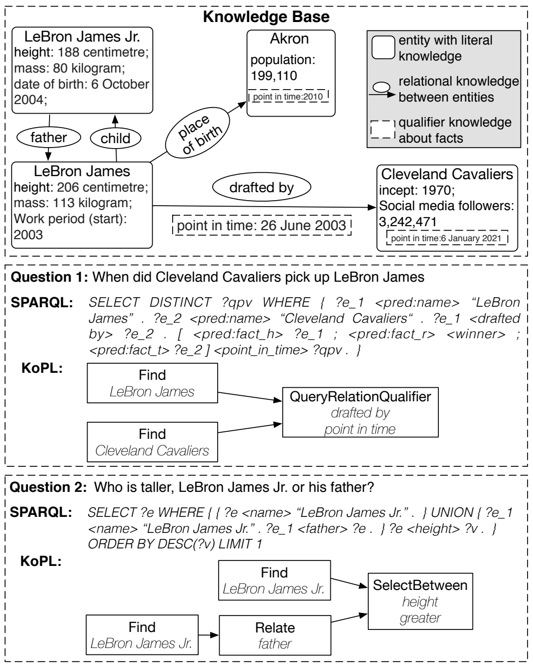
Thanks to the recent advances in deep models, especially large-scale unsupervised representation learning (Devlin et al., 2019), question answering of simple questions over knowledge base (Simple KBQA), i.e., single-relation factoid questions (Bordes et al., 2015), begins to saturate (Petrochuk and Zettlemoyer, 2018; Wu et al., 2019; Huang et al., 2019). However, tackling complex questions (Complex KBQA) is still an ongoing challenge, due to the unsatisfied capability of compositional reasoning. As shown in Table 1, to promote the community development, several benchmarks are proposed for Complex KBQA, including LC-QuAD2.0 (Dubey et al., 2019), ComplexWebQuestions (Talmor and Berant, 2018), MetaQA (Zhang et al., 2018), CSQA (Saha et al., 2018), CFQ (Keysers et al., 2020), and so on. However, they suffer from the following problems:

1) Most of them only provide QA pairs without explicit reasoning processes, making it challenging for models to learn the compositional reasoning. Some researchers try to learn the reasoning processes with reinforcement learning (Liang et al., 2017; Saha et al., 2019; Ansari et al., 2019) and searching (Guo et al., 2018). However, the prohibitively huge search space hinders both the performance and speed, especially when the question complexity increases. For example, Saha et al. (2019) achieved a 96.52% F1 score on simple questions in CSQA, whereas only 0.33% on complex questions that require comparative count. We think that an intermediate supervision is needed for learning the compositional reasoning, mimicking the learning process of human beings (Holt, 2017).

2) Questions are not satisfactory in diversity and scale. For example, MetaQA (Zhang et al., 2018) questions are generated using just 36 templates, and they only consider relations between entities, ignoring literal attributes; LC-QuAD2.0 (Dubey et al., 2019) and ComplexWebQuestions (Talmor and Berant, 2018) have fluent and diverse human-written questions, but their scale is less than 40K.

To address these problems, we create **KQA Pro**, a large-scale benchmark for Complex KBQA. In KQA Pro, we define a Knowledge-oriented Pro-

[*] indicates equal contribution
[†] Corresponding Author


Knowledge Base
LeBron James Jr.
height: 188 centimetre; mass: 80 kilogram; date of birth: 6 October 2004;
Akron
population: 199,110
point in time:2010
entity with literal knowledge
relational knowledge between entities
qualifier knowledge about facts
father
child
place of birth
LeBron James
height: 206 centimetre; mass: 113 kilogram; Work period (start): 2003
drafted by
point in time: 26 June 2003
Cleveland Cavaliers
incept: 1970; Social media followers: 3,242,471
point in time:6 January 2021
KoPL:
Find
LeBron James
Find
Cleveland Cavaliers
QueryRelationQualifier
drafted by
point in time
KoPL:
Find
LeBron James Jr.
SelectBetween
height
greater
Find
LeBron James Jr.
Relate
father

Question 1: When did Cleveland Cavaliers pick up LeBron James

SPARQL:
```
SELECT DISTINCT ?qpv WHERE { ?e_1 <pred:name> “LeBron James” . ?e_2 <pred:name> “Cleveland Cavaliers“ . ?e_1 <drafted by> ?e_2 . [ <pred:fact_h> ?e_1 ; <pred:fact_r> <winner> ; <pred:fact_t> ?e_2 ] <point_in_time> ?qpv . }
```

Question 2: Who is taller, LeBron James Jr. or his father?

SPARQL:
```
SELECT ?e WHERE { { ?e <name> “LeBron James Jr.” . } UNION { ?e_1 <name> “LeBron James Jr.” . ?e_1 <father> ?e . } ?e <height> ?v . } ORDER BY DESC(?v) LIMIT 1
```

Figure 1: Example of our KB and questions. Our KB is a dense subset of Wikidata (Vrandečić and Krötzsch, 2014), including multiple types of knowledge. Our questions are paired with executable KoPL programs and SPARQL queries.

gramming Language (**KoPL**) to explicitly describe the reasoning process for solving complex questions (see Fig. 1). A program is composed of symbolic **functions**, which define the basic, atomic operations on KBs. The composition of functions well captures the language compositionality (Baroni, 2019). Besides KoPL, following previous works (Yih et al., 2016; Su et al., 2016), we also provide the corresponding SPARQL for each question, which solves a complex question by parsing it into a query graph. Compared with SPARQL, KoPL 1) provides more explicit reasoning process. It divides the question into multiple steps, which makes human understanding easier and the intermediate results more transparent; 2) allows human to control the model behavior better, potentially supporting human-in-the-loop. When the system gives a wrong answer, users can easily locate the error by checking the outputs of intermediate functions. We believe the compositionality of KoPL and the graph structure of SPARQL are two complementary directions for Complex KBQA.

To ensure the diversity and scale of KQA Pro, we follow the synthesizing and paraphrasing pipeline in the literature (Wang et al., 2015a; Cao et al., 2020), first synthesize large-scale (canonical question, KoPL, SPARQL) triples, and then paraphrase the canonical questions to natural language questions (NLQs) via crowdsourcing. We combine the following two factors to achieve diversity in questions: (1) To increase structural variety, we leverage a varied set of templates to cover all the possible queries through random sampling and recursive composing; (2) To increase linguistic variety, we filter the paraphases based on their edit distance with the canonical utterance. Finally, KQA Pro consists of 117,970 diverse questions that involve varied reasoning skills (*e.g.*, multi-hop reasoning, value comparisons, set operations, *etc.*). Besides as a QA dataset, it also serves as a semantic parsing dataset. To the best of our knowledge, KQA Pro is currently the largest corpus for NLQ-to-SPARQL and NLQ-to-Program tasks.

We reproduce the state-of-the-art KBQA models and conduct a thorough evaluation of these models on KQA Pro. From the experimental results, we observe significant performance drops of these models as compared with on existing KBQA benchmarks. It indicates that Complex KBQA is still challenging and KQA Pro could support further explorations. We also treat KQA Pro as a diagnostic dataset for analyzing a model's capability of multiple reasoning skills, and discover weaknesses that are not widely known, *e.g.*, current models struggle on comparisonal reasoning for lacking of literal knowledge (*i.e.*, *(LeBron James, height, 206 centimetre)*), or perform poorly on questions whose answers are not obverseved in the training set. We hope all contents of KQA Pro could encourage the community to make further breakthroughs.

2 Related Work

Complex KBQA aims at answering complex questions over KBs, which requires multiple reasoning capabilities such as multi-hop inference, quantitative comparison, and set operation (Lan et al., 2021). Current methods for Complex KBQA can be grouped into two categories: 1) semantic parsing based methods (Liang et al., 2017; Guo et al., 2018; Saha et al., 2019; Ansari et al., 2019), which parses a question to a symbolic logic form (*e.g.*, λ-calculus (Artzi et al., 2013), λ-DCS (Liang, 2013; Pasupat and Liang, 2015; Wang et al., 2015b; Pasupat and Liang, 2016), SQL (Zhong et al., 2017), AMR (Banarescu et al., 2013), SPARQL (Sun et al., 2020), and *etc.*) and then executes it against the KB and obtains the final answers; 2) information

Dataset	multiple kinds of knowledge	number of questions	natural language	query graphs	multi-step programs
WebQuestions (Berant et al., 2013)	✓	5,810	✓	×	×
WebQuestionSP (Yih et al., 2016)	✓	4,737	✓	✓	×
GraphQuestions (Su et al., 2016)	✓	5,166	✓	✓	×
LC-QuAD2.0 (Dubey et al., 2019)	✓	30,000	✓	✓	×
ComplexWebQuestions (Talmor and Berant, 2018)	✓	34,689	✓	✓	×
MetaQA (Zhang et al., 2018)	×	400,000	×	×	×
CSQA (Saha et al., 2018)	×	1.6M	×	×	×
CFQ (Keysers et al., 2020)	×	239,357	×	✓	✓
GrailQA (Gu et al., 2021)	✓	64.331	✓	✓	×
KQA Pro (ours)	✓	117,970	✓	✓	✓

Table 1: Comparison with existing datasets of Complex KBQA. The column *multiple kinds of knowledge* means whether the dataset considers multiple types of knowledge or just relational knowledge (introduced in Sec.3.1). The column *natural language* means whether the questions are in natural language or written by templates.

retrieval based methods (Miller et al., 2016; Saxena et al., 2020; Schlichtkrull et al., 2018; Zhang et al., 2018; Zhou et al., 2018; Qiu et al., 2020; Shi et al., 2021), which constructs a question-specific graph extracted from the KB and ranks all the entities in the extracted graph based on their relevance to the question.

Compared with information retrieval based methods, semantic parsing based methods provides better interpretability by generating expressive logic forms, which represents the intermediate reasoning process. However, manually annotating logic forms is expensive and labor-intensive, and it is challenging to train a semantic parsing model with weak supervision signals (*i.e.*, question-answer pairs). Lacking logic form annotations turns out to be one of the main bottlenecks of semantic parsing.

Table 1 lists the widely-used datasets in Complex KBQA community and their features. MetaQA and CSQA have a large number of questions, but they ignore literal knowledge, lack logic form annotations, and their questions are written by templates. Query graphs (*e.g.*, SPARQLs) are provided in some datasets to help solve complex questions. However, SPARQL is weak in describing the intermediate procedure of the solution, and the scale of existing question-to-SPARQL datasets is small.

In this paper, we introduce a novel logic form KoPL, which models the procedure of Complex KBQA as a multi-step program, and provides more explicit reasoning process compared with query graphs. Furthermore, we propose KQA Pro, a large-scale semantic parsing dataset for Complex KBQA, which contains ~120k diverse natural language questions with both KoPL and SPARQL annotations. It is the largest NLQ-to-SPARQL dataset as far as we know. Compared with these existing datasets, KQA Pro serves as a more well-rounded benchmark.

3 Background

3.1 KB Definition

Typically, a **KB** (*e.g.*, Wikidata (Vrandečić and Krötzsch, 2014)) consists of:
Entity, the most basic item in KB.
Concept, the abstraction of a set of entities, *e.g.*, *basketball player*.
Relation, the link between entities or concepts. Entities are linked to concepts via the relation *instance of*. Concepts are organized into a tree structure via relation *subclass of*.
Attribute, the literal information of an entity. An attribute has a key and a value, which is one of four types[1]: string, number, date, and year. The number value has an extra unit, *e.g.*, 206 centimetre.
Relational knowledge, the triple with form (entity, relation, entity), *e.g.*, *(LeBron James Jr., father, LeBron James)*.
Literal knowledge, the triple with form (entity, attribute key, attribute value), *e.g.*, *(LeBron James, height, 206 centimetre)*.
Qualifier knowledge, the triple whose head is a relational or literal triple, *e.g.*, *((LeBron James, drafted by, Cleveland Cavaliers), point in time, 2003)*. A qualifier also has a key and a value.

3.2 KoPL Design

We design KoPL, a compositional and interpretable programming language to represent the reasoning process of complex questions. It models the complex procedure of question answering with a program of intermediate steps. Each step involves a function with a fixed number of arguments. Every program can be denoted as a binary tree. As shown in Fig. 1, a directed edge between two nodes represents the dependency relationship between two

[1]Wikidata also has other types like geographical and time. We omit them for simplicity and leave them for future work.

functions. That is, the destination function takes the output of the source function as its argument. The tree-structured program can also be serialized by post-order traversal, and formalized as a sequence with n functions. The general form is shown below. Each function f_i takes in a list of textual arguments $\mathbf{a}_i$, which need to be inferred according to the question, and a list of functional arguments $\mathbf{b}_i$, which come from the output of previous functions.

$$f_1(\mathbf{a}_1, \mathbf{b}_1) f_2(\mathbf{a}_2, \mathbf{b}_2) ... f_n(\mathbf{a}_n, \mathbf{b}_n) \qquad (1)$$

Take function *Relate* as an example, it has two textual inputs: relation and direction (i.e., *forward* or *backward*, meaning the output is object or subject). It has one functional input: a unique entity. Its output is a set of entities that hold the specific relation with the input entity. For example, in Question 2 of Fig. 1, the function *Relate*([*father, forward*], [*LeBron James Jr.*]) returns LeBron James, the father of LeBron James Jr. (the direction is omitted in the figure for simplicity).

We analyze the generic, basic operations for Complex KBQA, and design 27 functions[2] in KoPL. They support KB item manipulation (*e.g.*, *Find*, *Relate*, *FilterConcept*, *QueryRelationQualifier*, *etc.*), various reasoning skills (*e.g.*, *And*, *Or*, *etc.*), and multiple question types (*e.g.*, *QueryName*, *SelectBetween*, *etc.*). By composing the finite functions into a KoPL program, we can model the reasoning process of infinite complex questions.

Note that qualifiers play an important role in disambiguating, or restricting the validity of a fact (Galkin et al., 2020; Liu et al., 2021). However, they have not been properly modeled in current KBQA models or datasets. As far as we know, we are the first to explicitly model qualifiers in Complex KBQA.

4 KQA Pro Construction

To build KQA Pro dataset, first, we extract a knowledge base with multiple kinds of knowledge (Section 4.1). Then, we generate canonical questions, corresponding KoPL programs and SPARQL queries with novel compositional strategies (Section 4.2). In this stage, we aim to cover all the possible queries through random sampling and recursive composing. Finally, we rewrite canonical questions into natural language via crowdsourcing (Section 4.3). To further increase linguistic variety, we reject the paraphases whose edit distance with the canonical question is small.

4.1 Knowledge Base Extraction

We took the entities of FB15k-237 (Toutanova et al., 2015) as seeds, and aligned them with Wikidata via Freebase IDs[3]. The reasons are as follows: 1) The huge amount of knowledge in the full knowledge base (*e.g.*, full Freebase (Bollacker et al., 2008) or Wikidata contains millions of entities) may cause both time and space issues, while most of the entities may be never used in questions. 2) FB15k-237 is a high-quality, dense subset of Freebase, whose alignment to Wikidata produces a knowledge base with rich literal and qualifier knowledge. We added 3,000 other entities with the same name as one of FB15k-237 entities to increase the disambiguation difficulty. The statistics of our final knowledge base is listed in Table 2.

# Con.	# Ent.	# Name	# Pred.	# Attr.
794	16,960	14,471	363	846

# Relational facts	# Literal facts	# High-level facts
415,334	174,539	309,407

Table 2: Statistics of our knowledge base. Top section are the numbers of concepts, entities, unique entity names, predicates, and attributes. Bottom section are the numbers of different types of knowledge.

4.2 Question Generation Strategies

To generate diverse complex questions in a scalable manner, we propose to divide the generation into two stages: **locating** and **asking**. In locating stage we describe a single entity or an entity set with various restrictions, while in asking stage we query specific information about the target entity or entity set. We define several strategies for each stage. By sampling from them and composing the two stages, we can generate large-scale and diverse questions with a small number of templates. Fig. 2 gives an example of our generation process.

For locating stage, we propose 7 strategies and show part of them in the top section of Table 3. We can fill the placeholders of templates by sampling from KB to describe a target entity. We support quantitative comparisons of 4 operations: equal, not equal, less than, and greater than, indicated by "`<OP>`" of the template. We support optional qualifier restrictions, indicated by "(`<QK>` is `<QV>`)",

[2]The complete function instructions are in Appendix.

[3]The complete extracting process is in Appendix.

Strategy	Template	Example
Locating Stage		
Entity Name	-	LeBron James
Concept + Literal	the <C> whose <K> is <OP> <V> (<QK> is <QV>)	the basketball team whose social media followers is greater than 3,000,000 (point in time is 2021)
Concept + Relational	the <C> that <P> <E> (<QK> is <QV>)	the basketball player that was drafted by Cleveland Cavaliers
Recursive Multi-Hop	unfold <E> in a *Concept + Relational* description	the basketball player that was drafted by the basketball team whose social media followers is greater than 3,000,000 (point in time is 2021)
Intersection	Condition 1 *and* Condition 2	the basketball players whose height is greater than 190 centimetres and less than 220 centimetres
Asking Stage		
QueryName	What/Who is <E>	Who is the basketball player whose height is equal to 206 centimetres?
Count	How many <E>	How many basketball players that were drafted by Cleveland Cavaliers?
SelectAmong	Among <E>, which one has the largest/smallest <K>	Among basketball players, which one has the largest mass?
Verify	For <E>, is his/her/its <K> <OP> <V> (<QK> is <QV>)	For the human that is the father of LeBron James Jr., is his/her height greater than 180 centimetres?
QualifierRelational	<E> <P> <E>, what is the <QK>	LeBron James was drafted by Cleveland Cavaliers, what is the point in time?

Table 3: Representative templates and examples of our locating and asking stage. Placeholders in template have specific implication: <E>-description of an entity or entity set; <C>-concept; <K>-attribute key; <OP>-operator, selected from {=, !=, $<$, $>$}; <V>-attribute value; <QK>-qualifier key; <QV>-qualifier value; <P>-relation description, *e.g.*, *was drafted by*. The complete instruction is in Appendix.

which can narrow the located entity set. In *Recursive Multi-Hop*, we replace the entity of a relational condition with a more detailed description, so that we can easily increase the hop of questions. For asking stage, we propose 9 strategies and show some of them in the bottom section of Table 3. Our *SelectAmong* is similar to *argmax* and *argmin* operations in λ-DCS. The complete generation strategies are shown in Appendix due to space limit.

Our generated instance consists of five elements: question, SPARQL query, KoPL program, 10 answer choices, and a golden answer. Choices are selected by executing an abridged SPARQL, which randomly drops one clauses from the complete SPARQL. With these choices, KQA Pro supports both multiple-choice setting and open-ended setting. We randomly generate lots of questions, and only preserve those with unique answer. For example, since *Akron* has different populations in different years, we will drop questions like *What is the population of Akron*, unless the time constraint (*e.g.*, *in 2010*) is specified.

4.3 Question Paraphrasing and Evaluation

After large-scale generation, we release the generated questions on Amazon Mechanical Turk (AMT) and ask the workers to paraphrase them without changing the original meaning. For the convenience of paraphrasing, we visualize the KoPL flowcharts like Fig. 1 to help workers understand complex questions. We allow workers to mark a question as confusing if they cannot understand it or find some logical errors in it. These instances will be removed from our dataset.

After paraphrasing, we evaluate the quality by 5 other workers. They are asked to check whether the paraphrase keeps the original meaning and give a fluency rating from 1 to 5. We reject those paraphrases which fall into one of following cases: (1) marked as different from the original canonical question by more than 2 workers; (2) whose average fluency rating is lower than 3; (3) having a very small edit distance with the canonical question.

4.4 Dataset Analysis

Our KQA Pro dataset consists of 117,970 instances with 24,724 unique answers. Fig. 3(a) shows the question type distribution of KQA Pro. Within the 9 types, *SelectAmong* accounts for the least fraction (4.6%), while others account for more or less than 10%. Fig. 3(b) shows that multi-hop questions cover 73.7% of KQA Pro, and 4.7% questions even require at least 5 hops. We compare the question length distribution of different Complex KBQA datasets in Fig. 3(c). We observe that on average, our KQA Pro has longer questions than others. In

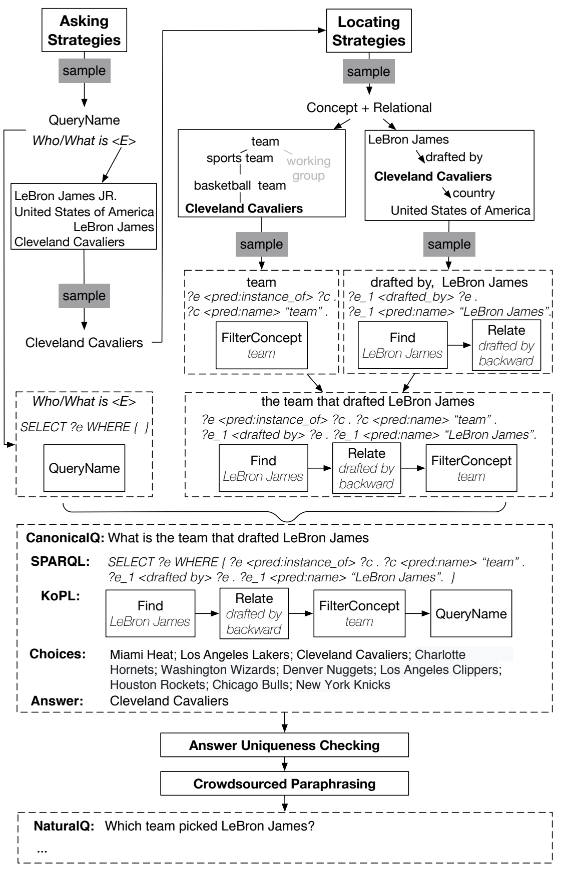
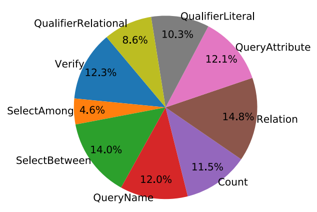
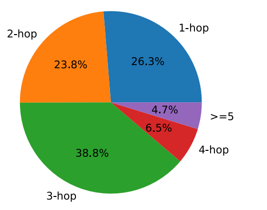

Asking Strategies
sample
QueryName
Who/What is <E>
LeBron James JR.
United States of America
LeBron James
Cleveland Cavaliers
sample
Cleveland Cavaliers
Locating Strategies
sample
Concept + Relational
team
sports team
working group
basketball team
Cleveland Cavaliers
LeBron James
drafted by
Cleveland Cavaliers
country
United States of America
sample
sample
team
?e <pred:instance_of> ?c .
?c <pred:name> "team" .
FilterConcept team
drafted by, LeBron James
?e_1 <drafted_by> ?e .
?e_1 <pred:name> "LeBron James".
Find LeBron James
Relate drafted by backward
Who/What is <E>
SELECT ?e WHERE { }
QueryName
the team that drafted LeBron James
?e <pred:instance_of> ?c . ?c <pred:name> "team" .
?e_1 <drafted by> ?e . ?e_1 <pred:name> "LeBron James".
Find LeBron James
Relate drafted by backward
FilterConcept team
CanonicalQ: What is the team that drafted LeBron James
SPARQL: SELECT ?e WHERE { ?e <pred:instance_of> ?c . ?c <pred:name> "team" . ?e_1 <drafted by> ?e . ?e_1 <pred:name> "LeBron James". }
KoPL:
Find LeBron James
Relate drafted by backward
FilterConcept team
QueryName
Choices: Miami Heat; Los Angeles Lakers; Cleveland Cavaliers; Charlotte Hornets; Washington Wizards; Denver Nuggets; Los Angeles Clippers; Houston Rockets; Chicago Bulls; New York Knicks
Answer: Cleveland Cavaliers
Answer Uniqueness Checking
Crowdsourced Paraphrasing
NaturalQ: Which team picked LeBron James?
...

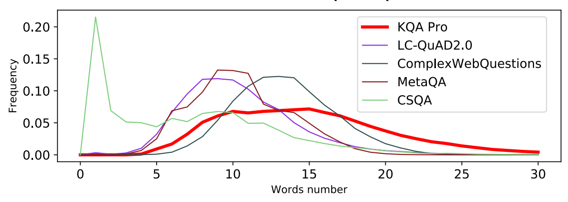
Figure 2: Process of our question generation. First, we sample a question type from asking strategies and sample a target entity from KB. Next, we sample a locating strategy and detailed conditions to describe the target entity. Finally, we combine intermediate snippets into the complete question and check whether the answer is unique. **Note that the snippets of canonical question, SPARQL, and KoPL are operated simultaneously.**



(a) Distribution of 9 different types of questions.



(b) Distribution of question hops. 73.7% of our questions require multiple-hops.



(c) Question length distribution of Complex KBQA datasets. We can see that KQA Pro questions have a wide range of lengths and are longer on average than all others.

Figure 3: Question statistics of KQA Pro.

KQA Pro, the average length of questions/programs/SPARQLs is 14.95/4.79/35.52 respectively. More analysis is included in the Appendix.

5 Experiments

The primary goal of our experiments is to show the challenges of KQA Pro and promising Complex KBQA directions. First, we compare the performance of state-of-the-art KBQA models on current datasets and KQA Pro, to show whether KQA Pro is challenging. Then, we treat KQA Pro as a diagnostic dataset to investigate fine-grained reasoning abilities of models, discuss current weakness and promising directions. We further conduct an experiment to explore the generation ability of our proposed model. Last, we provide a case study to show the interpretablity of KoPL.

5.1 Experimental Settings

Benchmark Settings. We randomly split KQA Pro to train/valid/test set by 8/1/1, resulting in three sets with 94,376/11,797/11,797 instances. About 30% answers of the test set are not seen in training.

Representative Models. KBQA models typically follow a retrieve-and-rank paradigm, by constructing a question-specific graph extracted from the KB and ranks all the entities in the graph based on their relevance to the question (Miller et al., 2016; Saxena et al., 2020; Schlichtkrull et al., 2018; Zhang et al., 2018; Zhou et al., 2018; Qiu et al., 2020); or follow a parse-then-execute paradigm, by parsing a question to a query graph (Berant et al., 2013; Yih et al., 2015) or program (Liang et al., 2017; Guo et al., 2018; Saha et al., 2019; Ansari et al., 2019) through learning from question-answer pairs.

Experimenting with all methods is logistically challenging, so we reproduce a representative subset of mothods: **KVMemNet** (Miller et al., 2016), a well-known model which organizes the knowledge into a memory of key-value pairs, and iteratively reads memory to update its query vector. **EmbedKGQA** (Saxena et al., 2020), a state-of-the art model on MetaQA, which incorporates knowledge embeddings to improve the reasoning performance. **SRN** (Qiu et al., 2020), a typical path search model to start from a topic entity and predict a sequential relation path to find the target entity. **RGCN** (Schlichtkrull et al., 2018), a variant of graph convolutional networks, tackling Complex KBQA through the natural graph structure of knowledge base.

Our models. Since KQA Pro provides the annotations of SPARQL and KoPL, we directly learn our parsers using supervised learning by regarding the semantic parsing as a sequence-to-sequence task. We explore the widely-used sequence-to-sequence model—**RNN** with attention mechanism (Dong and Lapata, 2016), and the pretrained generative language model—**BART** (Lewis et al., 2019), as our SPARQL and KoPL parsers.

To compare machine with **Human**, we sample 200 instances from the test set, and ask experts to answer them by searching our knowledge base.

Implementation Details. For our BART model, we used the bart-base model of HuggingFace[4]. We used the optimizer Adam (Kingma and Ba, 2014) for all models. We searched the learning rate for BART 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 validation set, we finally used learning rate 3e-5 for BART parameters, 1e-3 for other parameters, and weight decay 1e-5.

5.2 Difficulty of KQA Pro

We compare the performance of KBQA models on KQA Pro with MetaQA and WebQSP (short for WebQuestionSP), two commonly used benchmarks in Complex KBQA. The experimental results are in Table 4, from which we observe that:

Although the models perform well on MetaQA and WebQSP, their performances are significantly lower and not satisfying on KQA Pro. It indicates that our KQA Pro is challenging and the Complex KBQA still needs more research efforts. Actually, 1) Both MetaQA and WebQSP mainly focus on relational knowledge, *i.e.*, multi-hop questions. Therefore, previous models on these datasets are designed to handle only entities and relations. In comparison, KQA Pro includes three types of knowledge, *i.e.*, relations, attributes, and qualifiers, thus is much more challenging. 2) Compared with MetaQA which contains template questions, KQA Pro contains diverse natural language questions, and can evaluate models' language understanding abilities. 3) Compared with WebQSP which contains 4,737 fluent and natural questions, KQA Pro covers more question types (*e.g.*, verification, counting) and reasoning operations (*e.g.*, intersect, union).

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

Model	**MetaQA**			**WebQSP**	**KQAPro**
	1-hop	2-hop	3-hop		
KVMemNet	96.2	82.7	48.9	46.7	16.61
SRN	97.0	95.1	75.2	-	12.33
EmbedKGQA	97.5	98.8	94.8	66.6	28.36
RGCN	-	-	-	37.2	35.07
BART	-	-	99.9	67.5	88.55

Table 4: SOTA models of Complex KBQA and their performance on different datasets. SRN's result on KQA Pro, 12.33%, is obtained on questions about only relational knowledge. The RGCN results on WebQSP is from (Sun et al., 2018). The BART results on MetaQA 3-hop WebQSP results are from (Huang et al., 2021).

5.3 Analyses on Reasoning Skills

KQA Pro can serve as a diagnostic dataset for in-depth analyses of reasoning abilities (*e.g.*, counting, comparision, logical reasoning, *etc.*) for Complex KBQA, since KoPL programs underlying the questions provide a tight control over the dataset.

We categorize the test questions to measure fine-grained ability of models. Specifically, *Multi-hop* means multi-hop questions, *Qualifier* means questions containing qualifier knowledge, *Comparison* means quantitative or temporal comparison between two or more entities, *Logical* means logical union or intersection, *Count* means questions that ask the number of target entities, *Verify* means questions that take "yes" or "no" as answer, *Zero-shot* means questions whose answer is not seen in the training set. The results are shown in Table 5, from which we have the following observations:

(1) Benefits of intermediate reasoning supervision. Our RNN and BART models outperform current models significantly on all reasoning skills. This is because KoPL program and SPARQL query provide intermediate supervision which benefits the learning process a lot. As (Dua et al., 2020) suggests, future dataset collection efforts should set aside a fraction of budget for intermediate annotations, particularly as the reasoning required becomes more complex. We hope our dataset KQA Pro with KoPL and SPARQL annotations will help guide futher research in Complex KBQA. (2) More attention to literal and qualifier knowledge. Existing models perform poorly in the situations requiring comparison capability. This is because they only focus the relational knowledge, while ignoring the literal and qualifier knowledge. We hope our dataset will encourage the community to pay more attention to multiple kinds of knowledge in

Model	**Overall**	Multi-hop	Qualifier	Compari-son	Logical	Count	Verify	Zero-shot
KVMemNet	16.61	16.50	18.47	1.17	14.99	27.31	54.70	0.06
SRN	-	12.33	-	-	-	-	-	-
EmbedKGQA	28.36	26.41	25.20	11.93	23.95	32.88	61.05	0.06
RGCN	35.07	34.00	27.61	30.03	35.85	41.91	65.88	0.00
RNN SPARQL	41.98	36.01	19.04	66.98	37.74	50.26	58.84	26.08
RNN KoPL	43.85	37.71	22.19	65.90	47.45	50.04	42.13	34.96
BART SPARQL	89.68	88.49	83.09	96.12	88.67	85.78	92.33	87.88
BART KoPL	88.55	87.12	82.28	93.87	87.92	85.33	90.95	87.25
BART KoPL^{CG}	70.28	70.91	47.92	87.74	70.94	74.89	85.11	57.98
Human	97.50	97.24	95.65	100.00	98.18	83.33	95.24	100.00

Table 5: Accuracy of different models on KQA Pro test set. BART KoPL^{CG} denotes the BART based KoPL parser on the compositional generalization experiment (see Section 5.4)

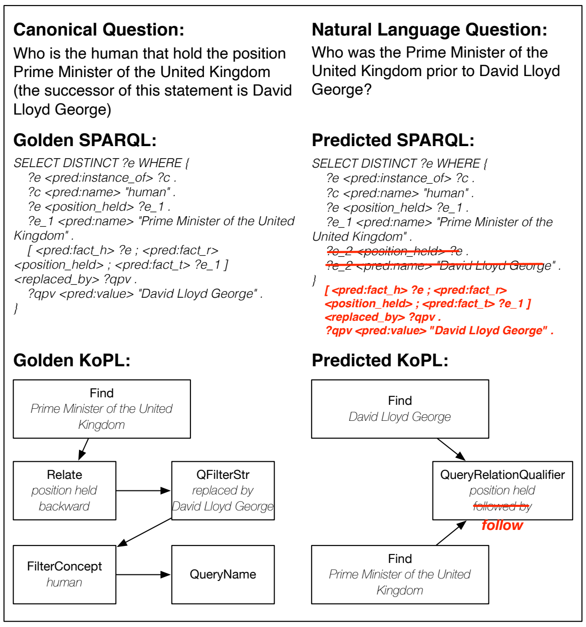
Complex KBQA. (3) Generalization to novel questions and answers. For zero-shot questions, current models all have a close to zero performance. This indicates the models solve the questions by simply memorizing their training data, and perform poorly on generalizing to novel questions and answers.

5.4 Compositional Generalization

We further use KQA Pro to test the ability of KBQA models to generalize to questions that contain novel combinations of the elements observed during training. Following previous works, we conduct the "productivity" experiment (Lake and Baroni, 2018; Shaw et al., 2021), which focuses on generalization to longer sequences or to greater compositional depths than have been seen in training (for example, from a length 4 program to a length 5 program). Specifically, we take the instances with short programs as training examples, and those with long programs as test and valid examples, resulting in three sets including 106,182/5,899/5,899 examples. The performance of BART KoPL drops from 88.55% to 70.28%, which indicates learning to generalize compositionally for pretrained language models requires more research efforts. Our KQA Pro provides an environment for further experimentation on compostional generalization.

5.5 Case Study

To further understand the quality of logical forms predicted by the BART parser, we show a case in Fig. 4, for which the SPARQL and KoPL parsers both give wrong predictions. The SPARQL parser fails to understand *prior to David Lloyd George* and gives a totally wrong prediction for this part. The KoPL parser gives a function prediction which is semantically correct but very different from our


Canonical Question:
Who is the human that hold the position Prime Minister of the United Kingdom (the successor of this statement is David Lloyd George)

Natural Language Question:
Who was the Prime Minister of the United Kingdom prior to David Lloyd George?

Golden SPARQL:

```
SELECT DISTINCT ?e WHERE {
   ?e <pred:instance_of> ?c .
   ?c <pred:name> "human" .
   ?e <position_held> ?e_1 .
   ?e_1 <pred:name> "Prime Minister of the United
Kingdom" .
   [ <pred:fact_h> ?e ; <pred:fact_r>
<position_held> ; <pred:fact_t> ?e_1 ]
<replaced_by> ?qpv .
   ?qpv <pred:value> "David Lloyd George" .
}
```

Predicted SPARQL:

```
SELECT DISTINCT ?e WHERE {
   ?e <pred:instance_of> ?c .
   ?c <pred:name> "human" .
   ?e <position_held> ?e_1 .
   ?e_1 <pred:name> "Prime Minister of the
United Kingdom" .
   ~~?e_2 <position_held> ?e~~ .
   ~~?e_2 <pred:name> "David Lloyd George~~" .
}
   [ <pred:fact_h> ?e ; <pred:fact_r>
   <position_held> ; <pred:fact_t> ?e_1 ]
   <replaced_by> ?qpv .
   ?qpv <pred:value> "David Lloyd George" .
```

Golden KoPL:

Predicted KoPL:

Figure 4: Predicted SPARQL and KoPL by BART. We show the natural language question and canonical question before human rewriting. We mark the error corrections of the wrong predictions in red.

generated golden one. It is a suprising result, revealing that the KoPL parser can understand the semantics and learn multiple solutions for each question, similar to the learning process of humans. We manually correct the errors of predicted SPARQL and KoPL and mark them in red. Compared to SPARQLs, KoPL programs are easier to be understood and more friendly to be modified.

6 Conclusion and Future Work

In this work, we introduce a large-scale dataset with explicit compositional programs for Complex KBQA. For each question, we provide the corresponding KoPL program and SPARQL query so that KQA Pro can serve for both KBQA and semantic parsing tasks. We conduct a thorough evaluation of various models, discover weaknesses of current models and discuss future directions. Among these models, the KoPL parser shows great interpretability. As shown in Fig. 4, when the model predicts the answer, it will also give a reasoning process and a confidence (which is ommited in the figure for simplicity). When the parser makes mistakes, humans can easily locate the error through reading the human-like reasoning process or checking the outputs of intermediate functions. In addition, using human correction data, the parser can be incrementally trained to continuously improve the performance. We will leave this as our future work.

7 Acknowledgments

This work is supported by the National Key Research and Development Program of China (2020AAA0106501), the NSFC Youth Project (62006136), the Key-Area Research and Development Program of Guangdong Province (2019B010153002), grants from the Institute for Guo Qiang, Tsinghua University (2019GQB0003), Beijing Academy of Artificial Intelligence, and Huawei Noah's Ark Lab.

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*.

Yoav Artzi, Nicholas FitzGerald, and Luke S Zettlemoyer. 2013. Semantic parsing with combinatory categorial grammars. *ACL (Tutorial Abstracts)*, 3.

Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer, and Nathan Schneider. 2013. Abstract meaning representation for sembanking. In *Proceedings of the 7th linguistic annotation workshop and interoperability with discourse*, pages 178–186.

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

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

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

Antoine Bordes, Nicolas Usunier, Sumit Chopra, and Jason Weston. 2015. Large-scale simple question answering with memory networks. *arXiv preprint arXiv:1506.02075*.

Ruisheng Cao, Su Zhu, Chenyu Yang, Chen Liu, Rao Ma, Yanbin Zhao, Lu Chen, and Kai Yu. 2020. Unsupervised dual paraphrasing for two-stage semantic parsing. In *ACL*.

Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

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

Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. In *ACL*.

Dheeru Dua, Sameer Singh, and Matt Gardner. 2020. Benefits of intermediate annotations in reading comprehension. In *ACL*.

Mohnish Dubey, Debayan Banerjee, Abdelrahman Abdelkawi, and Jens Lehmann. 2019. Lc-quad 2.0: A large dataset for complex question answering over wikidata and dbpedia. In *International Semantic Web Conference*, pages 69–78. Springer.

Mikhail Galkin, Priyansh Trivedi, Gaurav Maheshwari, Ricardo Usbeck, and Jens Lehmann. 2020. Message passing for hyper-relational knowledge graphs. In *EMNLP*.

Yu Gu, Sue E. Kase, M. Vanni, Brian M. Sadler, Percy Liang, Xifeng Yan, and Yu Su. 2021. Beyond i.i.d.: Three levels of generalization for question answering on knowledge bases. *Proceedings of the Web Conference 2021*.

Daya Guo, Duyu Tang, Nan Duan, Ming Zhou, and Jian Yin. 2018. Dialog-to-action: Conversational question answering over a large-scale knowledge base. In *Advances in Neural Information Processing Systems*.

John Holt. 2017. *How children learn*. Hachette UK.

Xiao Huang, Jingyuan Zhang, Dingcheng Li, and Ping Li. 2019. Knowledge graph embedding based question answering. In *WSDM*.

Xin Huang, Jung-jae Kim, and Bowei Zou. 2021. Unseen entity handling in complex question answering over knowledge base via language generation. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 547–557.

Robin Jia and Percy Liang. 2016. Data recombination for neural semantic parsing. *ArXiv*, abs/1606.03622.

Daniel Keysers, Nathanael Schärli, Nathan Scales, Hylke Buisman, Daniel Furrer, Sergii Kashubin, Nikola Momchev, Danila Sinopalnikov, Lukasz Stafiniak, Tibor Tihon, et al. 2020. Measuring compositional generalization: A comprehensive method on realistic data.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

B. Lake and Marco Baroni. 2018. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *ICML*.

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*.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*.

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

Percy Liang. 2013. Lambda dependency-based compositional semantics. *arXiv preprint arXiv:1309.4408*.

Yu Liu, Quanming Yao, and Yong Li. 2021. Role-aware modeling for n-ary relational knowledge bases. *Proceedings of the Web Conference 2021*.

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*.

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. *arXiv preprint arXiv:1508.00305*.

Panupong Pasupat and Percy Liang. 2016. Inferring logical forms from denotations. *arXiv preprint arXiv:1606.06900*.

Michael Petrochuk and Luke Zettlemoyer. 2018. Simplequestions nearly solved: A new upperbound and baseline approach. In *EMNLP*.

Yunqi Qiu, Yuanzhuo Wang, Xiaolong Jin, and Kun Zhang. 2020. Stepwise reasoning for multi-relation question answering over knowledge graph with weak supervision. In *WSDM*.

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

Amrita Saha, Vardaan Pahuja, Mitesh M Khapra, Karthik Sankaranarayanan, and Sarath Chandar. 2018. Complex sequential question answering: Towards learning to converse over linked question answer pairs with a knowledge graph. In *AAAI*.

Apoorv Saxena, Aditay Tripathi, and Partha Talukdar. 2020. Improving multi-hop question answering over knowledge graphs using knowledge base embeddings. In *ACL*.

Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks. In *ESWC*.

Peter Shaw, Ming-Wei Chang, Panupong Pasupat, and Kristina Toutanova. 2021. Compositional generalization and natural language variation: Can a semantic parsing approach handle both? In *ACL/IJCNLP*.

Jiaxin Shi, Shulin Cao, Lei Hou, Juan-Zi Li, and Hanwang Zhang. 2021. Transfernet: An effective and transparent framework for multi-hop question answering over relation graph. In *EMNLP*.

Yu Su, Huan Sun, Brian Sadler, Mudhakar Srivatsa, Izzeddin Gür, Zenghui Yan, and Xifeng Yan. 2016. On generating characteristic-rich question sets for qa evaluation. In *EMNLP*.

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*.

Yawei Sun, Lingling Zhang, Gong Cheng, and Yuzhong Qu. 2020. Sparqa: Skeleton-based semantic parsing for complex questions over knowledge bases. In *AAAI*, pages 8952–8959.

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

Kristina Toutanova, Danqi Chen, Patrick Pantel, Hoifung Poon, Pallavi Choudhury, and Michael Gamon. 2015. Representing text for joint embedding of text and knowledge bases. In *EMNLP*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Denny Vrandečić and Markus Krötzsch. 2014. Wikidata: a free collaborative knowledge base.

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

Yushi Wang, Jonathan Berant, and Percy Liang. 2015b. Building a semantic parser overnight. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1332–1342.

Peng Wu, Shujian Huang, Rongxiang Weng, Zaixiang Zheng, Jianbing Zhang, Xiaohui Yan, and Jiajun Chen. 2019. Learning representation mapping for relation detection in knowledge base question answering. In *ACL*.

Shan Wu, Bo Chen, Chunlei Xin, Xianpei Han, Le Sun, Weipeng Zhang, Jiansong Chen, Fan Yang, and Xunliang Cai. 2021. From paraphrasing to semantic parsing: Unsupervised semantic parsing via synchronous semantic decoding. In *ACL/IJCNLP*.

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

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*.

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

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*.

Mantong Zhou, Minlie Huang, and Xiaoyan Zhu. 2018. An interpretable reasoning network for multi-relation question answering. In *COLING*.

A Function Library of KoPL

Table 6 shows our 27 functions and their explanations. Note that we define specific functions for different attribute types (i.e., string, number, date, and year), because the comparison of these types are quite different. Following we explain some necessary items in our functions.

Entities/Entity: *Entities* denotes an entity set, which can be the output or functional input of a function. When the set has a unique element, we get an *Entity*.

Name: A string that denotes the name of an entity or a concept.

Key/Value: The key and value of an attribute.

Op: The comparative operation. It is one of $\{=, \neq, <, >\}$ when comparing two values, one of $\{$*greater*, *less*$\}$ in *SelectBetween*, and one of $\{$*largest*, *smallest*$\}$ in *SelectAmong*.

Pred/Dir: The relation and direction of a relation.

Fact: A literal fact, *e.g.*, (*Yao Ming*, *height*, *229 centimetres*), or a relational fact, *e.g.*, (*Kobe Bryant*, *spouse*, *Vanessa Laine Bryant*).

QKey/QValue: The key and value of a qualifier.

B Grammar Rules of KoPL

As shown in Table 7, the supported program space of KoPL can be defined by a synchrounous context-free grammar (SCFG), which is widely used to generate logical forms paired with canonical questions (Wang et al., 2015a; Jia and Liang, 2016; Wu et al., 2021). The programs are meant to cover the desired set of compositional functions, and the canonical questions are meant to capture the meaning of the programs.

C Knowledge Base Extraction

Specifically, we took the entities of FB15k-237 (Toutanova et al., 2015), a popular subset of Freebase, as seeds, and then aligned them with Wikidata via Freebase IDs[5], so that we could extract their rich literal and qualifier knowledge from Wikidata. Besides, we added 3,000 other entities with the same name as one of FB15k-237 entities, to further increase the difficulty of disambiguation. For the relational knowledge, we manually merged the relations of FB15k-237 (*e.g.*, */people/person-/spouse_s./people/marriage/spouse*) and Wikidata (*e.g.*, *spouse*), obtaining 363 relations totally. Finally, we manually filtered out useless attributes (*e.g.*, about images and Wikidata pages) and entities (*i.e.*, never used in triples).

[5]Wikidata provides the Freebase ID for most of its entities, but the relations are not aligned.

D SPARQL Implementation Details

We build a SPARQL engine with Virtuoso [6] to execute generated SPARQLs. To denote qualifiers, we create a virtual node for those literal and relational triples. For example, to denote the point in time of *(LeBron James, drafted by, Cleveland Cavaliers)*, we create a node *_BN* which connects to the subject, the relation, and the object with three special edges, and then add *(_BN, point in time, 2003)* into the graph. Similarly, we use virtual node to represent the attribute value of number type, which has an extra unit. For example, to represent the height of *LeBron James*, we need *(LeBron James, height, _BN), (_BN, value, 206), (_BN, unit, centimetre)*.

E Generation Examples

Consider the example of Fig. 2 in Section 4.2 in the main text, following is a detailed explanation. At the first, the asking stage samples the strategy *QueryName* and samples *Cleveland Cavaliers* from the whole entity set as the target entity. The corresponding textual description, SPARQL, and KoPL of this stage is "*Who is <E>*", "*SELECT ?e WHERE { }*", and "*QueryName*", respectively. Then we switch to the locating stage to describe the target entity *Cleveland Cavaliers*. We sample the strategy, *Concept + Relational*, to locate it. For the concept part, we sample *team* from all concepts of *Cleveland Cavaliers*. The corresponding textual description, SPARQL, and KoPL is "*team*", "*?e <pred:instance_of> ?c . ?c <pred:name> "team" .*", and "*FilterConcept(team)*", respectively. For the relation part, we sample *(LeBron James, drafted by)* from all triples of *Cleveland Cavaliers*. The corresponding textual description, SPARQL, and KoPL is "*drafted LeBron James*", "*?e_1 <drafted by> ?e . ?e_1 <pred:name> 'LeBron James' .*", "*Find(LeBron James) → Relate(drafted by, backward)*", respectively. The locating stage combines the concept and the relation, obtaining the entity description "*the team that drafted LeBron James*" and the corresponding SPARQL and KoPL. Finally, we combine the results of the two stages and output the complete question. Figure 7 and 8 show more examples of KQA Pro.

[6]https://github.com/openlink/virtuoso-opensource

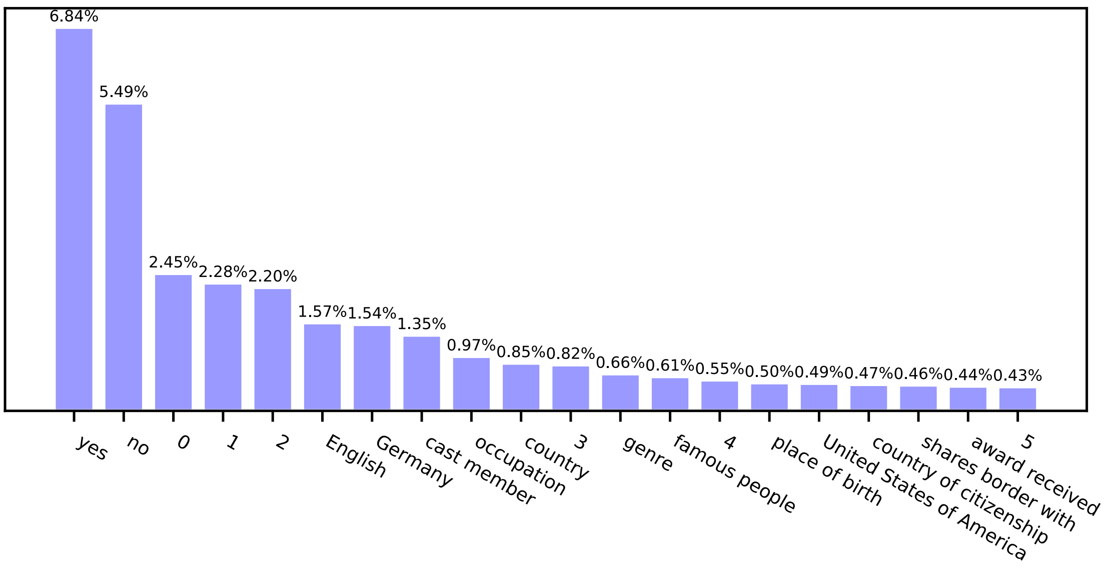


Figure 5: Top 20 most occurring answers in KQA Pro. The most frequent one is ”yes“, which is the answer of about half of type *Verify*.

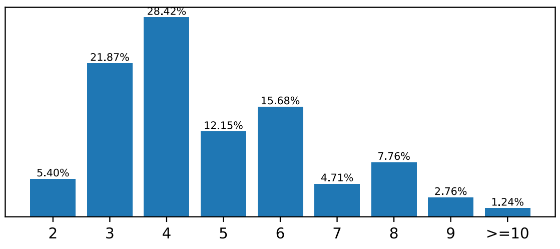


Figure 6: Distribution of program lengths.

F Data Analysis

There are 24,724 unique answers in KQA Pro. We show the top 20 most frequent answers and their fractions in Fig. 5. “yes” and “no” are the most frequent answers, because they cover all questions of type *Verify*. “0”, “1”, “2”, “3”, and other quantity answers are for questions of type *Count*, which accounts for 11.5%.

Fig. 6 shows the Program length distribution. Most of our problems (28.42%) can be solved by 4 functional steps. Some extreme complicated ones (1.24%) need more than 10 steps.

G Model Implementation Details

KVMemNet. For literal and relational knowledge, we concatenated the subject and the attribute/relation as the memory key, *e.g.*, “LeBron James drafted by”, leaving the object as the memory value. For high-level knowledge, we concatenated the fact and the qualifier key as the memory key, *e.g.*, “LeBron James drafted by Cleveland Cavaliers point in time”. For each question, we pre-selected a small subset of the KB as its relavant memory. Following (Miller et al., 2016), we retrieved 1,000 key-value pairs where the key shares at least one word with the question with frequency < 1000 (to ignore stop words). KVMemNet iteratively updates a query vector by reading the memory attentively. In our experiment we set the update steps to 3.

SRN. SRN can only handle relational knowledge. It must start from a topic entity and terminate with a predicted entity. So we filtered out questions that contain literal knowledge or qualifier knowledge, retaining 5,004 and 649 questions as its training set and test set. Specifically, we retained the questions with *Find* as the first function and *QueryName* as the last function. The textual input of the first *Find* was regarded as the topic entity and was fed into the model during both training and testing phase.

EmbedKGQA. EmbedKGQA utilizes knowledge graph embedding to improve multi-hop reasoning. To adapt to existing knowledge embedding techniques, we added virtual nodes to represent the qualifier knowledge of KQA Pro. Different from SRN, we applied EmbedKGQA on the entire KQA Pro dataset, because its classification layer is more flexible than SRN and can predict answers outside the entity set. The topic entity of each question was extracted from the golden program and then fed into the model during both training and testing.

RGCN. To build the graph, we took entities as nodes, connections between them as edges, and re-

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(LeBron James)*
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, 2019)*
QueryRelation	(*Entity*, *Entity*) $\times$ () $\rightarrow$ (*Pred*)	Return the relation between two entities	*QueryRelation(LeBron James, Cleveland Cavaliers)*
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, 199,110, point in time)*
QueryRelationQualifier	(*Entity*, *Entity*) $\times$ (*Pred*, *QKey*) $\rightarrow$ (*QValue*)	Return the qualifier value of the fact (*Entity*, *Pred*, *Entity*)	*QueryRelationQualifier(drafted by, point in time)*

Table 6: Details of our 27 functions. 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.

lations as edge labels. We concatenated the literal attributes of an entity into a sequence as the node description. For simplicity, we ignored the qualifier knowledge. Given a question, we first initialized node vectors by fusing the information of node descriptions and the question, then conducted RGCN to update the node features, and finally aggregated features of nodes and the question to predict the answer via a classification layer. Our RGCN implementation is based on DGL,[7] a high performance Python package for deep learning on graphs. Due to the memory limit, we set the graph layer to 1 and set the hidden dimension of nodes and edges to 32.

[7]https://github.com/dmlc/dgl

RNN-based KoPL and SPARQL Parsers. For KoPL prediction, we first parsed the question to the sequence of functions, and then predicted textual inputs for each function. We used Gated Recurrent Unit (GRU) (Cho et al., 2014; Chung et al., 2014), a well-known variant of RNNs, as our encoder of questions and decoder of functions. Attention mechanism (Dong and Lapata, 2016) was applied by focusing on the most relavant question words when predicting each function and each textual input. The SPARQL parser used the same encoder-decoder structure to produce SPARQL token sequences. We tokenized the SPARQL query

Non-terminal	KoPL Program	Canonical Question
<ROOT> →	<ES> *QueryName()*	[What | Who] is <ES>
	<ES> *Count()*	How many <ES>
	<ES> *QueryAttr(*<K>*)*	For <ES>, what is [his/her | its] <K>
	<ES> <ES> *QueryRelation()*	What is the relation from <ES> to <ES>
	<ES> *SelectAmong(*<K>,<SOP>*)*	Among <ES>, which one has the <SOP>
	<ES> <ES> *SelectBetween(*<K>,<COP>*)*	Which one has the <COP> <K>, <ES> or <ES>
	<ES> [*QueryAttr(*<K>*)* | *QueryAttrUnderCondition(*<K>,<QK>,<QV>*)*] <Verify>	For <ES>, is <K> <Verify> [(<QK> is <QV>)]?
	<ES> *QueryAttrQualifier(*<K>,<V>,<QK>*)*	For <ES>, [his/her | its] <K> is <V>, [What | Who] is the <QK>
	<ES> <ES> *QueryRelationQualifier(*<P>,<QK>*)*	<ES> <P> <ES>, [What | Who] is the <QK>
<ES> →	<ES> <ES> <Bool>	<ES> <Bool> <ES>
	<E>	<E>
<E> →	[<ES> | *FindAll()*] <FilterAttr> <FilterQualifier>? *FilterConcept(*<C>*)*?	the [one | <C>] whose <FilterAttr> <FilterQualifier>?
	[<ES> | *FindAll()*] *Relate(*<P>, DIR*)* <FilterQualifier>? *FilterConcept(*<C>*)*?	the [one | <C>] that <P> <ES> <FilterQualifier>?
	FindAll() FilterConcept(<C>*)*	<C>
	Find(Name)	*Name*
<FilterAttr> →	*FilterStr(*<K>,<V>*)*	<K> is <V>
	FilterNum(<K>,<V>,<OP>*)*	<K> is <OP> <V>
	FilterYear(<K>,<V>,<OP>*)*	<K> is <OP> <V>
	FilterDate(<K>,<V>,<OP>*)*	<K> is <OP> <V>
<FilterQualifier> →	*QFilterStr(*<QK>,<QV>*)*	(<K> is <V>)
	QFilterNum(<QK>,<QV>,<OP>*)*	(<K> is <OP> <V>)
	QFilterYear(<QK>,<QV>,<OP>*)*	(<K> is <OP> <V>)
	QFilterDate(<QK>,<QV>,<OP>*)*	(<K> is <OP> <V>)
<Verify> →	*VerifyStr(*<V>*)*	<V>
	VerifyNum(<V>,<OP>*)*	<OP> <V>
	VerifyYear(<V>,<OP>*)*	<OP> <V>
	VerifyDate(<V>,<OP>*)*	<OP> <V>
<Bool> →	*And()*	and
	Or()	or
<SOP> →	largest | smallest	largest | smallest
<COP> →	greater | less	greater | less
<OP> →	= | != | < | >	[equal to | in | on] | [not equal to | not in] | [less than | before] | [greater than | after]
<K>/<QK> →	*Key* | *QKey*	*Key_Text* | *QKey_Text*
<V>/<QV> →	*Value* | *QValue*	*Value* | *QValue*
<C> →	*Concept*	*Concept*
<P>→	*Pred*	*Pred_Text*

Table 7: SCFG rules for producing KoPL program and canonical question pairs. "|" matches either expression in a group. "?" denotes the expression preceding it is optional. *Key_Text*, *QKey_Text*, and *Pred_Text* denote the annotated template for attribute keys, qualifier keys, and relations. For example, for *Pred place of birth*, the *Pred_Text* is "was born in".

by delimiting spaces and some special punctuation symbols.

BART-based KoPL and SPARQL Parsers. BART is a pretrained sequence-to-sequence model based on Transformer (Vaswani et al., 2017), achieving state-of-the-art performance on a range of language comprehension and generation tasks. We used the widely-used online implementation[8] and finetuned the pretrained model on our NLQ-to-SPARQL and NLQ-to-KoPL corpus. For KoPL learning, we concatenated textual inputs with functions to obtain a totally sequential format of program, *e.g.*, Question 2 of Fig. 1 is converted to the sequence "*Find(LeBron Jaems Jr.), Find(LeBron James Jr.), Relate(father, forward), SelectBetween(height, greater)*".

H BART KoPL Accuracy of different #hops

In Table 8, we presents the BART KoPL accuracy of different #hops. Note that KQA Pro not only consider multi-hop relations, but also consider attributes and qualifiers. We count all of them into the hop number. So in KQA Pro, given a question with "4-hops", it does not mean 4 relations, but may be 1 relations + 2 attributes + 1 comparison. *E.g.*, "Who is taller, LeBron James Jr. or his father?".

2-3 Hop	4-5 Hop	6-7 Hop	8-9 Hop
92.4	87.71	86.41	86.51

Table 8: BART KoPL Accuracy of different #hops.

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

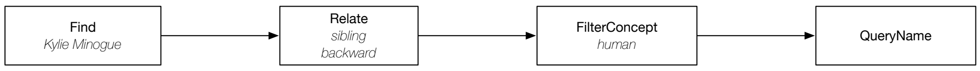
Strategy	**Template**	**Example**
	Locating Stage	
Entity Name	-	LeBron James
Concept Name	`<C>`	basketball players
Concept + Literal	the (<C>) whose `<K>` is `<OP>` `<V>` (`<QK>` is `<QV>`)	the basketball team whose social media followers is greater than 3,000,000 (point in time is 2021)
Concept + Relational	the (<C>) that `<P>` `<E>` (`<QK>` is `<QV>`)	the basketball player that was drafted by Cleveland Cavaliers
Recursive Multi-Hop	unfold `<E>` in a *Concept* + *Relational* description	the basketball player that was drafted by the basketball team whose social media followers is greater than 3,000,000 (point in time is 2021)
Intersection	Condition 1 *and* Condition 2	the basketball players whose height is greater than 190 centimetres and less than 220 centimetres
Union	Condition 1 *or* Condition 2	the basketball players that were born in Akron or Cleveland
	Asking Stage	
QueryName	What/Who is `<E>`	Who is the basketball player whose height is equal to 206 centimetres?
Count	How many `<E>`	How many basketball players that were drafted by Cleveland Cavaliers?
QueryAttribute	For `<E>`, what is his/her/its `<K>` (`<QK>` is `<QV>`)	For Cleveland Cavaliers, what is its social media follower number (point in time is January 2021)?
Relation	What is the relation from `<E>` to `<E>`	What is the relation from LeBron James Jr. to LeBron James?
SelectAmong	Among `<E>`, which one has the largest/smallest `<K>`	Among basketball players, which one has the largest mass?
SelectBetween	Which one has the larger/smaller `<K>`, `<E>` or `<E>`	Which one has the larger mass, LeBron James Jr. or LeBron James?
Verify	For `<E>`, is his/her/its `<K>` `<OP>` `<V>` (`<QK>` is `<QV>`)	For the human that is the father of LeBron James Jr., is his/her height greater than 180 centimetres?
QualifierLiteral	For `<E>`, his/her/its `<K>` is `<V>`, what is the `<QK>`	For Akron, its population is 199,110, what is the point in time?
QualifierRelational	`<E>` `<P>` `<E>`, what is the `<QK>`	LeBron James was drafted by Cleveland Cavaliers, what is the point in time?

Table 9: Templates and examples of our locating and asking stage. Placeholders in template have specific implication: `<E>`-description of an entity or entity set; `<C>`-concept; `<K>`-attribute key; `<OP>`-operator, selected from {=, !=, <, >}; `<V>`-attribute value; `<QK>`-qualifier key; `<QV>`-qualifier value; `<P>`-relation description, *e.g.*, *was drafted by*.

Question: Who is the person that is Kylie Minogue's sibling?

SPARQL: *SELECT DISTINCT ?e WHERE { ?e <pred:instance_of> ?c . ?c <pred:name> "human" . ?e <sibling> ?e_1 . ?e_1 <pred:name> "Kylie Minogue" . }*

Program:



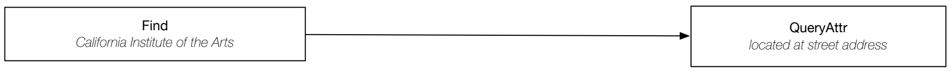
Choices: Rick Baker; John Carpenter; Bobby; Sylvester Stallone; Max Fleischer; Michael Jackson; Richard Gere; William Henry Harrison; Shirley MacLaine; Dannii Minogue

Answer: Dannii Minogue

Question: What is the street address of the California Institute of the Arts?

SPARQL: *SELECT DISTINCT ?pv WHERE { ?e <pred:name> "California Institute of the Arts" . ?e <located_at_street_address> ?pv . }*

Program:



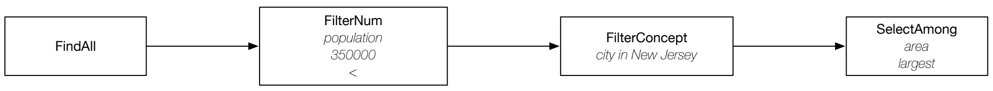
Choices: 1501 W Bradley Ave, Peoria, IL, 61625-0001; 600 Lincoln Avenue, Charleston, IL, 61920; 500 College Ave, Swarthmore, PA, 19081; 24700 W McBean Pky, Valencia, CA, 91355-2397; 403 Main Street, Grambling, LA, 71245; Administration Building, Athens, GA, 30602; 1280 Main Street West; 2 E South St, Galesburg, IL, 61401-9999; 140 West Street; Columbia-Campus, Columbia, SC, 29208

Answer: 24700 W McBean Pky, Valencia, CA, 91355-2397

Question: Of New Jersey cities with under 350000 in population, which is biggest in terms of area?

SPARQL: *SELECT ?e WHERE { ?e <pred:instance_of> ?c . ?c <pred:name> "city in New Jersey" . ?e <population> ?pv_1 . ?pv_1 <pred:unit> "1" . ?pv_1 <pred:value> ?v_1 . FILTER (?v_1 < "350000"^^xsd:double) . ?e <area> ?pv . ?pv <pred:value> ?v . } ORDER BY DESC(?v) LIMIT 1*

Program:



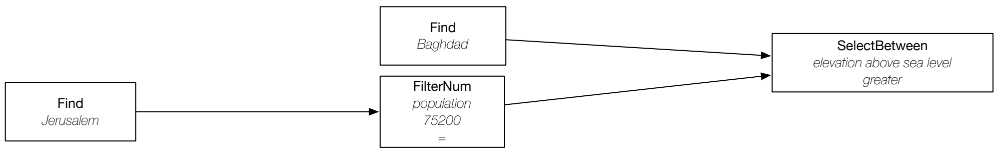
Choices: Hoboken; Bayonne; Paterson; Perth Amboy; New Brunswick; Trenton; Camden; Atlantic City; Newark; East Orange

Answer: Newark

Question: Which area has higher elevation (above sea level), Baghdad or Jerusalem (the one whose population is 75200)?

SPARQL: *SELECT ?e WHERE { { ?e <pred:name> "Baghdad" . } UNION { ?e <pred:name> "Jerusalem" . ?e <population> ?pv_1 . ?pv_1 <pred:unit> "1" . ?pv_1 <pred:value> "75200"^^xsd:double . } ?e <elevation_above_sea_level> ?pv . ?pv <pred:value> ?v . } ORDER BY DESC(?v) LIMIT 1*

Program:



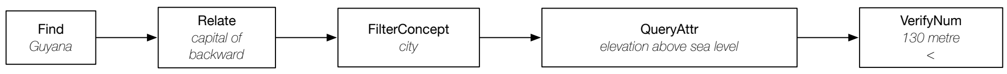
Choices: Santo Domingo; Kingston; Trieste; Jerusalem; Cork; Abidjan; Bergen; Baghdad; Chihuahua; Dundee

Answer: Jerusalem

Question: Is the elevation above sea level for the capital city of Guyana less than 130 meters?

SPARQL: *ASK { ?e <pred:instance_of> ?c . ?c <pred:name> "city" . ?e <capital_of> ?e_1 . ?e_1 <pred:name> "Guyana" . ?e <elevation_above_sea_level> ?pv . ?pv <pred:unit> "metre" . ?pv <pred:value> ?v . FILTER (?v < "130"^^xsd:double) . }*

Program:



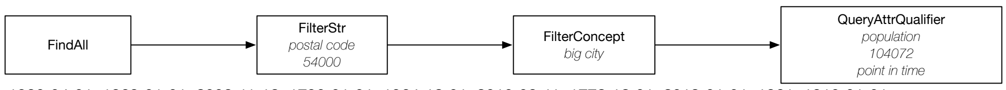
Choices: yes; no; unknown; unknown; unknown; unknown; unknown; unknown; unknown; unknown

Answer: yes

Question: When did the big city whose postal code is 54000 have a population of 104072?

SPARQL: *SELECT DISTINCT ?qpv WHERE { ?e <pred:instance_of> ?c . ?c <pred:name> "big city" . ?e <postal_code> ?pv_1 . ?pv_1 <pred:value> "54000" . ?e <population> ?pv . ?pv <pred:unit> "1" . ?pv <pred:value> "104072"^^xsd:double . [<pred:fact_h> ?e ; <pred:fact_r> <population> ; <pred:fact_t> ?pv] <point_in_time> ?qpv . }*

Program:



Choices: 1980-04-01; 1868-01-01; 2008-11-12; 1790-01-01; 1964-12-01; 2010-08-11; 1772-12-01; 2013-01-01; 1861; 1810-01-01

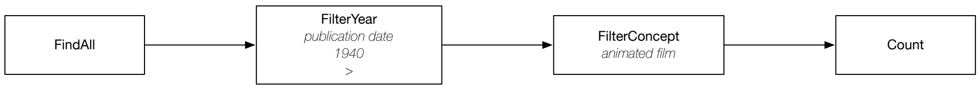
Answer: 2013-01-01

Figure 7: Examples of KQA Pro. In KQA Pro, each instance consists of 5 components: the textual question, the corresponding SPARQL, the corresponding KoPL, 10 candidate choices, and the golden answer. Choices are separated by semicolons in this figure. For questions of *Verify* type, the choices are composed of "yes", "no", and 8 special token "unknown" for padding.

Question: What number of animated movies were published after 1940?

SPARQL: *SELECT (COUNT(DISTINCT ?e) AS ?count) WHERE { ?e <pred:instance_of> ?c . ?c <pred:name> "animated film" . ?e <publication_date> ?pv . ?pv <pred:year> ?v . FILTER (?v > 1940) . }*

Program:



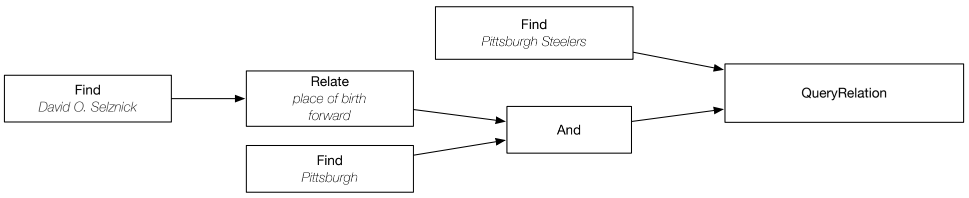
Choices: 35; 36; 37; 38; 39; 40; 41; 42; 43; 44

Answer: 39

Question: How are the Pittsburgh Steelers related to the Pittsburgh where David O. Selznick was born?

SPARQL: *SELECT DISTINCT ?p WHERE { ?e_1 <pred:name> "Pittsburgh Steelers" . ?e_2 <pred:name> "Pittsburgh" . ?e_3 <place_of_birth> ?e_2 . ?e_3 <pred:name> "David O. Selznick" . ?e_1 ?p ?e_2 . }*

Program:



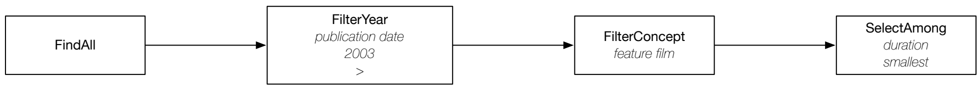
Choices: organisation directed from the office or person; given name; genre; headquarters location; office held by head of state; officeholder; country; operating system; dedicated to; product or material produced

Answer: headquarters location

Question: Among the feature films with a publication date after 2003, which one has the smallest duration?

SPARQL: *SELECT ?e WHERE { ?e <pred:instance_of> ?c . ?c <pred:name> "feature film" . ?e <publication_date> ?pv_1 . ?pv_1 <pred:year> ?v_1 . FILTER (?v_1 > 2003) . ?e <duration> ?pv . ?pv <pred:value> ?v . } ORDER BY ?v LIMIT 1*

Program:



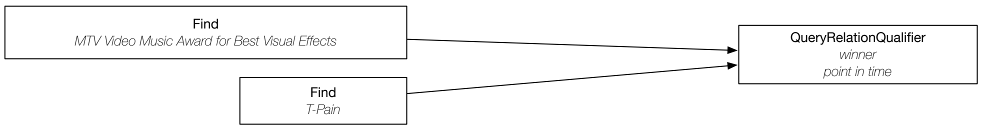
Choices: Alice in Wonderland; Pirates of the Caribbean: Dead Man's Chest; Wallace & Gromit: The Curse of the Were-Rabbit; Bedtime Stories; Secretariat; The Sorcerer's Apprentice; Enchanted; Old Dogs; Harry Potter and the Prisoner of Azkaban; Prince of Persia: The Sands of Time

Answer: Wallace & Gromit: The Curse of the Were-Rabbit

Question: When did T-Pain win the MTV Video Music Award for Best Visual Effects?

SPARQL: *SELECT DISTINCT ?qpv WHERE { ?e_1 <pred:name> "MTV Video Music Award for Best Visual Effects" . ?e_2 <pred:name> "T-Pain" . ?e_1 <winner> ?e_2 . [<pred:fact_h> ?e_1 ; <pred:fact_r> <winner> ; <pred:fact_t> ?e_2] <point_in_time> ?qpv . }*

Program:



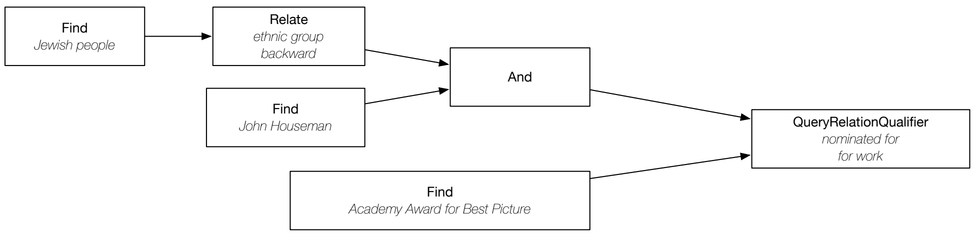
Choices: 1955-12-01; 1966-04-18; 2005-12-31; 1375; 1995-12-19; 1980-10-01; 1944-01-01; 1885-01-01; 1976-12-01; 2008

Answer: 2008

Question: For what was John Houseman (who is in the Jewish ethnic group) nominated for an Academy Award for Best Picture?

SPARQL: *SELECT DISTINCT ?qpv WHERE { ?e_1 <pred:name> "John Houseman" . ?e_1 <ethnic_group> ?e_3 . ?e_3 <pred:name> "Jewish people" . ?e_2 <pred:name> "Academy Award for Best Picture" . ?e_1 <nominated_for> ?e_2 . [<pred:fact_h> ?e_1 ; <pred:fact_r> <nominated_for> ; <pred:fact_t> ?e_2] <for_work> ?qpv . }*

Program:



Choices: My Fair Lady; With a Song in My Heart; The Bicentennial Man; In America; WarGames; Bernie; The Facts of Life; Hotel Rwanda; The Sunshine Boys; Julius Caesar

Answer: Julius Caesar

Figure 8: Examples of KQA Pro.