TPLP **19** (5-6): 636–653, 2019. © Cambridge University Press 2019
doi:10.1017/S1471068419000103

Querying Knowledge via Multi-Hop English Questions

TIANTIAN GAO, PAUL FODOR and MICHAEL KIFER
Stony Brook University, Stony Brook, NY, USA
(*e-mail:* {tiagao,pfodor,kifer}@cs.stonybrook.edu)

submitted 22 July 2019; accepted 31 July 2019

Abstract

The inherent difficulty of knowledge specification and the lack of trained specialists are some of the key obstacles on the way to making intelligent systems based on the knowledge representation and reasoning (KRR) paradigm commonplace. *Knowledge and query authoring* using natural language, especially *controlled* natural language (CNL), is one of the promising approaches that could enable domain experts, who are not trained logicians, to both create formal knowledge and query it. In previous work, we introduced the *KALM* system (Knowledge Authoring Logic Machine) that supports knowledge authoring (and simple querying) with very high accuracy that at present is unachievable via machine learning approaches. The present paper expands on the question answering aspect of KALM and introduces *KALM-QA* (KALM for Question Answering) that is capable of answering much more complex English questions. We show that KALM-QA achieves 100% accuracy on an extensive suite of movie-related questions, called *MetaQA,* which contains almost 29,000 test questions and over 260,000 training questions. We contrast this with a published machine learning approach, which falls far short of this high mark.

KEYWORDS: Controlled Natural Language, Knowledge Representation, Reasoning, Question Answering

1 Introduction

Much of the world knowledge can be described by logical facts and rules, and processed by intelligent *knowledge representation and reasoning* (KRR) systems. Alas, eliciting knowledge from human experts and codifying it in logic is often too big an undertaking, and the learning curve for domain experts is too steep. One would hope that such knowledge could be extracted from text, but this goal is still out of reach. Despite the advances in text understanding and information extraction (e.g., (Mausam et al. 2012; Angeli et al. 2015; Gomez 2008)) this technology is still far from delivering the accuracy needed for KRR, which is very sensitive to errors.

Meanwhile, *controlled natural languages* (CNL) (Kuhn 2014) were proposed as a possible solution to the aforesaid learning curve problem in the belief that a natural language (NL) could be more accessible to domain experts. The idea was that CNL sentences would be usable as knowledge because, as subsets of NLs with restricted grammars and interpretation rules, their meaning can be captured in logic precisely. Some of the better-known CNLs include Attempto Controlled English (ACE) (Fuchs et al. 2008), Processable English (PENG) (Schwitter 2010), and SBVR (Object-Management-Group 2017). CNL's

restrictive grammars could, at times, be problematic, but this is not a show-stopper for knowledge *authoring*, since the difference between knowledge *authoring* and general knowledge *acquisition* is quite significant: whereas knowledge acquisition is about enabling machines to understand the NL that humans write, knowledge authoring is about enabling humans to write in NL that machines could understand.

A more serious obstacle for using CNLs in authoring is that, by design, CNLs do not assume any kind of background knowledge. For example, in ACE, the sentences *Zoe Saldana appears in Avatar* and *Zoe Saldana is an actress of Avatar* would have different logical representations and a query like *Avatar has which actresses* will yield no answers. Background knowledge could be written as CNL sentences in the form of bridge rules, but it is naive to expect that domain experts would know how to do it, and the amount of such background knowledge is prohibitive. One of the main goals of knowledge authoring, which we started to address in our previous work on the *knowledge authoring logic machine* (KALM) (Gao et al. 2018b; Gao et al. 2018a), was to make this process scalable and feasible while still relying on CNLs.

The previous incarnations of KALM supported knowledge authoring and simple question answering with very high accuracy compared to the state-of-the-art machine learning approaches, such as SEMAFOR (Das et al. 2014), SLING (Ringgaard et al. 2017), and Stanford CoreNLP (Manning et al. 2014). KALM achieves this effect with the help of a sophisticated semantic layer on top of ACE, which includes a formal, FrameNet-inspired (Johnson et al. 2002) linguistic ontology, *FrameOnt*, that formalizes FrameNet's frames, plus the linguistic knowledge graph BabelNet (Navigli and Ponzetto 2012). These linguistic resources are relied on by an *incrementally-learnable* semantic parser that maps semantically equivalent CNL sentences into the same *FrameOnt* frames and gives them *unique logical representation* (ULR).

Contributions. The main contribution of this paper is introduction of KALM-QA, an extension of KALM that supports *multi-hop* question answering compared to just 1-hop queries in (Gao et al. 2018a). A *multi-hop question* is one whose answering involves joining (in the database sense) multiple relations, some of which could be self-joins. Thus, answering a 3-hop query requires joining three relations, as in *"Who wrote films that share actors with the film Anastasia"* and *"Who are the screenwriters that the actors in their movies also appear in the movie The Backwoods."* We use the MetaQA dataset (Zhang et al. 2018a) that contains close to 29,000 multi-hop test queries and show that KALM-QA achieves the accuracy of 100% compared to the much lower accuracy of a machine learning approach (Zhang et al. 2018b).

Second, we give a detailed description of the KALM-QA frame-semantic parser, which is generated incrementally by structure learning. These details did not appear in previous publications. Finally, we note that MetaQA sentences use more general syntax than what ACE accepts. (Gao et al. 2019, Appendix A) describes harnessing the power of the Stanford Parser (Manning et al. 2014) for paraphrasing MetaQA sentences (over 260,000 in total) to make them conform to ACE.

For the remainder of this paper, Sections 2 and 3 give the background on KALM and the MetaQA dataset. Section 4 describes KALM's frame-semantic parser in detail and Section 5 explains how KALM-QA translates MetaQA multi-hop questions into logic. Section 6 presents experimental results, Section 7 discusses related works, and Section 8 concludes the paper. (Gao et al. 2019, Appendix A) describes the use of the Stanford

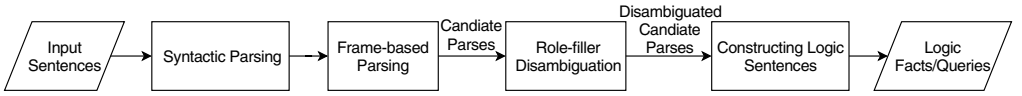

Fig. 1. The KALM Architecture.

Parser for paraphrasing MetaQA sentences to conform to the ACE grammar. (Gao et al. 2019, Appendix B) argues that KALM's approach to mediating between semantically equivalent sentences is much more scalable than the more traditional approach of specifying the equivalences via background bridge rules used in CNLs, as in Attempto or SBVR.

2 The KALM system

KALM is a semantic framework that aims to be a practically scalable solution for knowledge authoring. KALM users author knowledge using CNL sentences (ACE, to be specific) and KALM ensures that semantically equivalent sentences have the same logical representation. This is achieved with the help of linguistic and other information resources, which provide the necessary background knowledge in a scalable fashion. The architecture of KALM is shown in Figure 1.

Not shown in the figure is the new *paraphrase* step, which extends the syntax of Attempto[1] and lets KALM parse more complex MetaQA questions. This aspect is discussed in (Gao et al. 2019, Appendix A).

Syntactic parsing. KALM uses Attempto Parsing Engine (APE)[2] as the top level parser. APE extracts the syntactical information from sentences, including the *part of speech* of each word and the grammatical relations between pairs of words. This extracted information is represented by a list of stylized first-order terms known as a *discourse representation structure*, or DRS.[3]

Example 1. The sentence "*a director directs a film*" has the following DRS

```
object(A,director,countable,na,eq,1)-1/2
object(B,film,countable,na,eq,1)-1/5
predicate(C,direct,A,B)-1/3
```

Logically, this should be understood as a list of terms and the capitalized symbols as variables. The notation like `-1/2` says that the term refers to the second word in the first sentence. The `object`-term denotes an entity corresponding to a noun word and the `predicate`-term denotes an event/action corresponding to a verb. The variables `A` and `B` denote the *director*- and *film*-entities, and the variable `C` denotes the *directs*-event. When the same variable appears more than once, it serves as a cross-reference, as usual in logic. In a `predicate`-term, the 3rd and 4th arguments represent the *subject* and *object* of the corresponding event, respectively. Arguments 4-6 in an `object`-term can be ignored for this paper. Thus, the above example shows that the *directs*-event has the *director*-entity as a subject and the *film*-entity as an object. □

[1] http://attempto.ifi.uzh.ch/site/docs/syntax_report.html
[2] https://github.com/Attempto/APE
[3] http://attempto.ifi.uzh.ch/site/pubs/papers/drs_report_66.pdf

Frame-based parsing. Frame-based parsing takes DRS structures and performs linguistic analysis trying to identify possible meanings of the corresponding sentences—an idea inspired by FrameNet (Johnson et al. 2002). A *frame* (Fillmore and Baker 2001) represents one or more related semantic relationships among entities, where each entity plays a particular *role*. While FrameNet is only an informal methodology where each frame is described textually, KALM has a formal ontology, *FrameOnt*, which captures FrameNet's frames formally, as logic formulas, and adds more frames as needed.

Example 2. The following `Movie` frame describes films and their attributes, including actors, directors, release years, and so on. It is encoded by the following Prolog fact:

```
fp('Movie',[role('FilmNm',['bn:00034471n'],[]),
    role('Id',['bn:00045822n'],['Integer']),
    role('Actor',['bn:00001176n'],[]),
    role('Release Year',['bn:00078738n'],['Year']),
    role('Director',['bn:00027368n '],[]),
    role('Writer',['bn:00034485n'],[]),
    role('Genre',['bn:00037744n'],[])]).
```

Much of this is self-explanatory except, possibly, the symbols of the form `bn:xxxxxxxxy`. These are BabelNet *synsets* (Navigli and Ponzetto 2012), which are similar to and derived from WordNet's synsets. Synsets are used because words typically have many meanings and the same meaning may be shared by many words. Thus, relying on words instead of synsets is likely to make information ambiguous and lead to wrong query answers. KALM is unique in the world of knowledge acquisition in its use of synsets. In addition, type constraints may be imposed on some roles. For instance, the `Release Year` role has the type constraint `Year`, which insists that the role-filler word for this role must be a valid year value. □

A frame may represent several related relations, which KALM captures via *logical valence patterns* (*lvps*). An lvp consists of a *lexical unit* and a set of *grammatical patterns*, each associated with a frame role. A lexical unit is a word that could possibly "trigger" a frame relation and thus indicate that the frame is possibly applicable to the sentence. A grammatical pattern represents a grammatical relation between the lexical unit and a frame role. An lvp is applicable to a sentence if (1) the sentence contains the lexical unit and (2) the role-fillers for each frame role can be extracted based on the respective grammatical patterns.

Example 3. The following lvp says that the verb *direct* is a lexical unit of the frame `Movie`.

```
lvp(direct,v,'Movie', [pattern('Director','verb->subject',required),
    pattern('FilmNm','verb->object',required)]).
```

The fillers for the roles `Director` and `FilmNm` can be extracted by following the grammatical patterns `'verb->subject'` and `'verb->object'`, respectively. Each grammatical pattern acts as an independent extraction rule that works on the DRS structure. For instance, the extraction rule `'verb->subject'` (resp., `'verb->object'`) extracts the term that represents the subject (resp., object) of the term that describes the lexical unit `direct`. In the context of the DRS structure from Example 1, the term corresponding

to the lexical unit `direct` is `predicate(C,direct,A,B)`. Applying the above lvp to that DRS extracts a "movie director" relation where the word *director* fills the role `Director` and *film* fills the role `FilmNm`. □

We call the extracted relation a *candidate parse*. Details on the construction of grammatical patterns and lvps will be discussed in Section 4.

Role-filler Disambiguation. Frame parsing produces *candidate* parses by identifying the lvps (i.e., semantic relationships) that a CNL sentence represents. Alas, most of the triggered lvps might not make sense in a particular situation. For instance, the sentence "*a director directs a film*" triggers at least these two frames: making a film and directing a corporation. In the first frame, "*film*" is a filler for the role `FilmNm` and in the second it fills the role `Corporation`. The reason we know that this sentence is related to making a film and not to directing a corporation is because a film is not a corporation. Making such decisions is the job of role-filler disambiguation, which is described in detail in (Gao et al. 2018b) and relies on sophisticated algorithms over the BabelNet knowledge graph (Navigli and Ponzetto 2012). Role-filler disambiguation also determines the right synset for each role-filler. For instance, it will determine that the meaning of "*film*" in our sentence is movie (`bn:00034471n`) and not photographic film (`bn:00034472n`). The result of disambiguation consists of *disambiguated parses*.

Example 4. For another example, consider the sentences (1) *who makes a film* and (2) *who makes a cake*. Both sentences share the same grammatical structure: *make* is the main verb of the sentence, *who* is the subject of the *make*-event, *film* and *cake* are the objects of the *make*-event for the first and second sentences, respectively. The first sentence implies the `Movie` frame while the second implies the `Cooking` frame. However, since both sentences share the same grammatical structure, the lvp used to trigger the `Cooking` frame for the second sentence also applies to the first. This yields a wrong candidate parse for sentence (1) as an instance of the `Cooking` frame, with *who* as `Cook` and *film* as `Food`. Since *film* and `Food` are semantically unrelated, role-filler disambiguation will give a very low score to the second parse and it will get pruned away. □

In the present paper, role-filler disambiguation is much simpler than in (Gao et al. 2018b). First, the main terms in the MetaQA data set are used unambiguously. Second, it includes an ontology that disambiguates role-filler words with respect to the roles uniquely. As a result, we do not need to use BabelNet here and role-filler disambiguation here reduces to simply checking class membership for the different entities.

Constructing logical forms. This step maps the actual instances of semantic relations extracted via lvps to their corresponding representations, called *unique logical representations (ULR)* for assertions of facts and *unique logical representation for queries (ULRQ)* for questions (Gao et al. 2018b; Gao et al. 2018a). In this work, construction of ULRQ is much more complicated because multi-hop questions result in disambiguated parses consisting of multiple lvps, which may even come from different frames. In (Gao et al. 2018a), on the other hand, we considered only 1-hop queries, which correspond to single lvp disambiguated parses. Details of multi-hop translation to ULRQ appear in Section 5.

3 The MetaQA Dataset and Multi-Hop Questions

The MetaQA dataset (Zhang et al. 2018a) consists of a movie ontology derived from the WikiMovies Dataset (Miller et al. 2016) and three sets of question-answer pairs written in natural language: 1-hop, 2-hop, and 3-hop queries. In this paper, we focus on the last two as a more interesting challenge. The movie ontology contains information on films and their attributes including actors, directors, writers, genres, release years, and so on. MetaQA provides enough information to differentiate between films that happen to have the same name, but not namesake actors, writers, or directors. Each film instance is represented by a contiguous chunk of triples with the first argument in a triple being the film name, the second one of the film's attributes (e.g., *directed_by*, *has_genre*, *has_imdb_rating*, *in_language*, *release_year*, *starred_actors*, *written_by*), and the third argument being the attribute value. We assign a unique Id to each film instance and represent each triple as a fact of the form `movie(AttrName1=AttrVal1,...)`. For example, the MetaQA triple `Kismet|directed_by|William Dieterle` is represented as `movie('FilmNm'='Kismet','Id'=72, 'Director'='William Dieterle')`, where `72` is the film Id.

Example 5. The sentence "*Who appears in a Steven Spielberg directed film*" gets parsed into a conjunction of these two frame relations: (1) `movie('FilmNm'=Title,'Id'=ID,'Director'='Steven Spielberg')` and (2) `movie('FilmNm'=Title,'Id'=ID,'Actor'=Y)`. To construct the query to answer the above question we form a conjunction (a database join) of the two expressions. The result will be returned as a set of bindings for the variable `Y`. □

Structure of MetaQA. The *training set* of MetaQA has 118,980 2-hop and 114,196 3-hop questions labeled with the expected query answers. The *testing set* has 14,872 2-hop and 14,274 3-hop questions—also labeled with expected answers. The *movie ontology* itself contains information about 17,341 distinct movies including their directors, writers, actors, and other properties, as described earlier.

Problems with MetaQA. As it turned out, a large fraction of both training and testing queries in MetaQA are mislabeled, i.e., the answers provided are wrong. In the training set, 13,522 out of 118,980 2-hop queries, or 11.36%, are definitely mislabeled, but we suspect this number is as high as 25,487 (21.42%). For 3-hop queries, the situation is even worse: 36,645 questions out of 114,196 (32.09%) are labeled with wrong answers, but this number is likely as high as 64,089 (56.12%). In the testing set, the situation is similar: 3,178 2-hop queries out of 14,872 (21.37%) are mislabeled and for 3-hop queries this number is 8,062 out of 14,274 (56.48%).

Most of the labeling errors are due to the fact that MetaQA's designers (Zhang et al. 2018b) failed to take into account that the movie ontology has many movies with the same title. As a result, queries like "*Who co-acted (or co-directed) with X in a movie*" are likely to have wrong answers if X acted in movie M1, Y acted in M2, and M1 and M2 happen to have exactly the same title. There are, for example, three movies named "Jane Eyre," which trigger this kind of errors. The existence of such errors is easily checked automatically with KALM and this is where the aforesaid counts of certain errors in the training set (13,522 and 36,645) come from.

In the testing set, the same problem accounts for the bulk of the wrong expected answers: 1,707 and 4,705. The remaining errors are due to more subtle reasons such as confusion between a certain number being a movie title vs. it being a movie release year (e.g., the movies "*1941*", "*2010*"), failure to recognize that, say, *Thomas Mann* is a writer and not a director of "*Death in Venice*" (which causes the question "*What are the release years of the movies directed by Thomas Mann?*" to mistakenly return *1971*), or the failure to realize that someone could be both a writer and an actor in the same movie (thus causing the answer "*Bob Peterson*" to be missing in MetaQA's answer for the query "*Who are the actors in the films written by Bob Peterson?*"). For these latter, more subtle, issues we checked manually about 50% of the suspected errors (736 of 1,471 2-hop and 1628 of 3,357 3-hop) for the testing set and verified that KALM provides correct answers while MetaQA does not. The number of suspected errors in the training set is too large to be checked manually, so we checked only a few dozen to confirm our hypothesis.

The large number of errors in the original MetaQA puts in question the accuracy of the results reported in (Zhang et al. 2018b) to which we return in Section 6. A corrected version of MetaQA data set can be found in reference (Gao 2019).

4 Constructing a KALM Semantic Parser via Structural Learning

This section shows how KALM's frame-semantic parser is constructed by incremental structure-learning. First, we show how grammatical patterns for role-filler extraction are learned from training sentences and then how lvps are generated based on annotated training sentences.

4.1 Learning Grammatical Patterns for Role-filler Extraction

Example 3 of Section 2 illustrates how a grammatical pattern like `verb->subject` is used to extract the subject of the *direct*-event. Technically, it is done by these Prolog extraction rules:

```
extraction_rule('verb->subject',DRS,LexicUnitTerm,RoleFillerTerm):-
    get_subject_from_verb(DRS,LexicUnitTerm,RoleFillerTerm).
get_subject_from_verb(DRS,VerbTerm,SubjectTerm):-
    get_arg3_from_verb_term(VerbTerm,Arg3),
    SubjectTerm = object(Arg3,_,_,_,_,_),
    member(SubjectTerm,DRS).
```

The first argument in `extraction_rule/4` is a string representing the grammatical pattern. Given the DRS for a sentence and the lexical unit term as the input, the extraction rule outputs the extracted role-filler term by calling a *utility predicate* `get_subject_from_verb/3` that: (1) extracts argument 3, `Arg3`, from of the verb term `VerbTerm` (the verb term here describes the lexical unit and `Arg3` is the subject of that verb); and (2) finds an `object`-term in DRS whose first argument, `Arg1` (that represents the `object`-term), matches `Arg3` (the subject of the `predicate`-term). If the `DRS` argument is as in Example 1 and the lvp is as in Example 3, the lexical unit term in question

would be `predicate(C,direct,A,B)`, the `subject`-term would be `A` and the matching `object`-term would be `object(A,director,countable,na,eq,1)`.

Grammatical patterns and the corresponding extraction rules are learned through annotated training sentences. An annotated sentence is a CNL sentence with lexical information attached to it, which includes the frame name, the lexical unit, and frame roles. Conceptually an annotated sentence looks like this: [Movie]*a director*$_{\text{Director}}$ *directs*$_{\text{lexical\_unit}}$ *a film*$_{\text{Film}}$. Later, in Example 8, we will see the actual structure used to represent annotated sentences.

The structure-learning process starts with ACE parsing the raw CNL sentence found in a provided annotated training sentence, which yields a DRS. Then, for each pair of terms (*lexical-unit,role-filler*) in that DRS structure, KALM produces and saves the corresponding extraction rule, like the one at the start of this subsection. (Recall that the aforesaid lexical unit and the role fillers are marked in the training sentence explicitly.) Creation of extraction rules has three parts: (1) construction of the library of utility rules, which is done in advance; (2) reachability reasoning in DRS; and (3) construction of the actual extraction rules.

Utility predicates and rules. These rules form the core library of the KALM parser; they are used for commonly occurring manipulations with DRS structures and do not depend on the application domain. Each utility predicate is defined by a single utility rule, which handles a concrete navigation scenario among terms that share variables within the same DRS.

Example 6. A prepositional word is represented by the term `modifier_pp(A,Lexem,B)` where the first argument, `A`, denotes the verb being modified, the second argument represents the propositional word and its lexical information, and the third argument denotes the object in the prepositional phrase. Due to the linguistic purpose of a `modifier_pp` term, it can be reached via two DRS terms: via the shared variable `A` in a term like `predicate(A,Verb,SubjRef,ObjRef)`, and via the shared variable `B` in a term like `object(B,Noun,Class,Unit,Op,Count)`. These two cases lead to two utility rules. The first takes the first argument `Arg1` from a `predicate`-term and then finds a `PrepositionalTerm` (of the form `modifier_pp(A,Lexem,B)`) in DRS whose first argument matches `Arg1`.

```
get_preposition_from_verb(DRS,VerbTerm,PrepositionalTerm):-
    get_arg1_from_verb_term(VerbTerm,Arg1),
    PrepositionalTerm = modifier_pp(Arg1,_,_),
    member(PrepositionalTerm,DRS).
```

The second rule takes the third argument `Arg3` from `PrepositionalTerm` and finds an `object`-term in DRS whose first argument matches `Arg3`.

```
get_dependent_from_preposition(DRS,PrepositionalTerm,DependentTerm):-
    get_arg3_from_prepositional_term(PrepositionalTerm,Arg3),
    DependentTerm = object(Arg3,_,_,_,_,_),
    member(DependentTerm,DRS).
```

□

Each utility rule is associated with a unique grammatical pattern (written as a string) via the predicate `utility_grammatical_pattern(UtilityPredicate, GrammaticalPattern)`. For instance, the above utility predicates are associated with

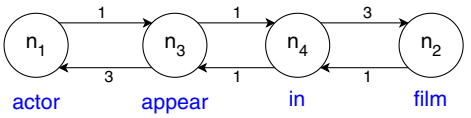

Fig. 2. The DRS embedding for the sentence *an actor appears in a film*.

the grammatical patterns `verb->pp` and `pp->dep`, respectively. This is the source of the grammatical patterns that we saw earlier in `extraction_rule/4`. We will come back to this at the end of this subsection.

Reachability reasoning in DRS. To see how one term in a DRS is connected to another via a sequence of intermediate variables and terms, we reduce this problem to a graph reachability by embedding the DRS in a graph structure:

1. For each term in DRS, create a node in the graph.
2. For any pair of nodes n_1, n_2 that represent the DRS subterms p_1, p_2 *that share a variable*, create a directed labeled edge $n_1 \longrightarrow n_2$ and back. The label of the edge $n_1 \longrightarrow n_2$ is the position of the shared variable in p_1 and the edge $n_2 \longrightarrow n_1$ is labeled with the position of that variable in p_2.

Let n_i be the node for the term that represents a lexical unit and n_j be the node for a role-filler term. Then, the problem of constructing the extraction rule for the role-filler reduces to finding the shortest path from n_i to n_j. The above graph is well-defined, *if* no term in a DRS has repeated variables, which is always the case for sentences that have no co-occurrences of the same entities or anaphoras (e.g., "*Fido eats Fido*" or "*Fido eats himself*" would violate this restriction). This restriction ensures that any pair of terms in DRS shares *at most one* variable. It is needed *only* for training sentences and in practice is easily achievable via a trivial paraphrase, since the training set is under the control of the knowledge engineer. Once learning is done, testing sentences *can* have anaphoras and entity co-occurrences with no restrictions.

Example 7. The annotated sentence [Movie]*an actor*$_{Actor}$ *appears*$_{lexical\ unit}$ *in a film*$_{Film}$ has the following DRS:

```
object(A,actor,countable,na,eq,1)-1/2
object(B,film,countable,na,eq,1)-1/6
predicate(C,appear,A)-1/3
modifier_pp(C,in,B)-1/4
```

Let n_1, n_2, n_3, and n_4 denote each of the terms in DRS, respectively. The DRS is represented by the graph shown in Figure 2. Learning the grammatical pattern that connects the lexical unit *appear* to *film* reduces to finding the shortest path from n_3 to n_2, i.e., $n_3 \xrightarrow{1} n_4 \xrightarrow{3} n_2$. □

Construction of extraction rules. Extraction rules are generated based on the shortest path from a lexical unit node to a role-filler node. As is shown in Figure 2, two adjacent nodes plus one of their connected edges uniquely defines a scenario where one term is connected to the other via a common variable in DRS. Thus, the body of the extraction rule is formed as a join of the corresponding utility predicates for every pair of adjacent edges in the path. For instance, for the path $n_3 \xrightarrow{1} n_4 \xrightarrow{3} n_2$ in Example 7, KALM generates the following extraction rule:

```
extraction_rule('verb->pp,pp->dep',DRS,LexicUnitTerm,RoleFillerTerm):-
    get_preposition_from_verb(DRS,LexicUnitTerm,PrepositionTerm),
    get_dependent_from_preposition(DRS,PrepositionTerm,RoleFillerTerm).
```

The grammatical pattern in the first argument is a concatenation of the grammatical patterns associated with the two utility predicates in the body of the extraction rule, which were introduced right after the end of Example 6.

4.2 Learning Lvps for Frame-semantic Parsing

Lvps are learned from annotated training sentences. An annotated sentence is a term that includes the original CNL sentence, the name of a semantic frame that matches the sentence, the word index of the lexical unit, and the word indices of each role-filler word and the respective frame role. An annotated sentence is represented by a Prolog fact. Note: the same English sentence may carry several different bits of information and thus may match several different frames. In this case, there might be several annotated sentences for the same CNL sentence.

Example 8. Consider the annotated training sentence: `annotation('An actor appears in a film','Movie',3,[['Actor',2],['FilmNm',6]])`. It relates to the `Movie` frame with the word *appears* serving as the lexical unit. The words *actor* and *film* fill the roles `Actor` and `Film`, and the numbers are word indices in the sentence. □

KALM feeds annotated sentences to the `learn_lvp` rule, which generates extraction rules for each frame role in the annotation and then forms an lvp. At a high level, the `learn_lvp` rule looks as shown below:

```
learn_lvp(annotation(Sentence,FrameName,LexicUnitIdx,RoleFillerIdxes)) :-
  ace_to_drs(Sentence,DRS),  %% invoke Attempto parser
  grammatic_patterns(DRS,LexicUnitIdx,RoleFillerIdxes,GrammaticPatterns),
  construct_lvp(DRS,LexicUnitIdx,FrameName,GrammaticPatterns).
```

In the rule body, the predicate `ace_to_drs` calls the ACE parser to produce the DRS of the input sentence. Using the DRS and the word indices of the lexical unit and each of the role-filler words, the predicate `grammatic_patterns` generates the grammatical patterns and the corresponding extraction rules for each frame role. The grammatical patterns are passed to the next subgoal in a variable, while the extraction rules are saved in KALM parser's library. Finally, the predicate `construct_lvp` takes the lexical unit, the frame name, and grammatical patterns constructs the requisite lvp and asserts it in the database. For instance, the annotated sentence of Example 8 will yield the following lvp:

```
lvp(appear,v,'Movie', [pattern('Actor','verb->subject',required),
    pattern('Film','verb->pp,pp->dep',required)]).
```

A similarly annotated sentence "*a director directs a film*" yields an lvp that we met in Example 3.

Once the necessary lvps are learned, KALM uses the following query to parse new (annotation-free) sentences; it finds the lvps that match the sentence and outputs a set of candidate parses:

```
?- parse_sentence(Sentence,CandidateParseList).
```

The `parse_sentence` rule can be described, at a high level, as follows:

```
parse_sentence(Sentence,CandidateFrameList) :-
    ace_to_drs(Sentence,DRS),
    get_candidate_lexical_units_from_drs(DRS,CandidateLexicUnitList),
    get_lvps(CandidateLexicUnitList,LvpList),
    apply_lvps_to_drs(DRS,LvpList,CandidateFrameList).
```

The first step yields a DRS, as before. Then, `get_candidate_lexical_units_from_drs/2` finds all *candidate* lexical units in the DRS that can possibly trigger a frame. Each such candidate is represented by a tuple that includes a lexeme and a part-of-speech. Third, `get_lvps/2` finds all lvps whose lexical unit matches a candidate lexical units in `CandidateLexicUnitList`. Finally, `apply_lvps_to_drs/3` applies each lvp in `LvpList` to the DRS to see which ones succeed. Section 4.1 explained how grammatical patterns are associated with rules for extracting role-filler words. The predicate `apply_lvps_to_drs/3` takes each lvp in `LvpList` and applies the extraction rules matching each `pattern`-term in the lvp to find candidate frames that fit the input `Sentence`. An lvp may fail because its grammatical pattern is such that an associated extraction rule fails to extract the expected filler word or the extracted words have the wrong type or fail other constraints. The successful lvps will yield frame relation instances, i.e., candidate parses.

5 Capturing the Meaning of Multi-Hop Questions in Logic

This section shows how candidate parses of multi-hop questions are translated into logical queries.

Definition 1. *A **candidate parse** is a tuple of the form* $(frame\_name, lexical\_unit, \{(frame\_role_i, role\_filler_i, grammatic\_pattern_i)\})$ $(1 \leq i \leq n)$ *where the first element is a frame name, the second is the word index of the lexical unit used to extract the frame relation, and the third element is a set of tuples, each consisting of three elements: a frame role, the word index of the role-filler word and the grammatical pattern used to extract the role-filler.* □

English sentences for multi-hop questions typically result in multiple candidate parses that form complex inter-relationships. These relationships must be taken into account in order to translate these sets of candidate parses into correct logic queries.

Definition 2. *Let* $f = (name, lu, \{(r_1, w_1, p_1), (r_2, w_2, p_2), \ldots, (r_m, w_m, p_m)\})$ *and* $f' = (name', lu', \{(r'_1, w'_1, p'_1), (r'_2, w'_2, p'_2), \ldots, (r'_n, w'_n, p'_n)\})$ *be two candidate parses where* $m \leq n$. *We say that* f' ***subsumes*** f *if* $name = name'$ *and* $\{(r_1, w_1), \ldots, (r_m, w_m)\} \subseteq \{(r'_1, w'_1), \ldots, (r'_n, w'_n)\}$. □

Subsumed parses represent the relationships implied by subsuming parses and can be ignored.

Definition 3. *Let* $f = (name, lu, \{(r_1, w_1, p_1), (r_2, w_2, p_2), \ldots, (r_m, w_m, p_m)\})$ *and* $f' = (name', lu', \{(r'_1, w'_1, p'_1), (r'_2, w'_2, p'_2), \ldots, (r'_n, w'_n, p'_n)\})$ *be two candidate parses. We say that* f *and* f' *are **alternatives** if:* (1) $lu = lu'$ *and* $\{(w_1, p_1), (w_2, p_2), \ldots, (w_m, p_m)\} \subseteq \{(w_1, p_1), (w_2, p_2), \ldots, (w_n, p_n)\}$ *or vice versa; and* (2) f, f' *do not subsume each other.*

This implies that either name ≠ name′ or some of the extracted words fill different roles in the two parses. Thus, the two parses represent different frame relations among the extracted words. □

Definition 4. *A set of candidate parses* $F = \{f_1, f_2, \ldots, f_n\}$ *is a* ***set of pairwise alternatives*** *if for any* i *and* j*, where* $i \neq j$*,* f_i *and* f_j *are alternatives.* F *is a* ***maximal set of pairwise alternatives*** *if there is no* F' *such that* F' *is pairwise alternative and* $F \subset F'$*.* □

Let $F = \{f_1, f_2, \ldots, f_n\}$ be a set of candidate parses for a CNL sentence C. The *unique logical representation for queries* (*ULRQ*) for C is obtained via the following steps:

1. Prune invalid candidate parses from F via role-filler disambiguation. Let the result be F'.
2. Delete all $f_i \in F'$ that is subsumed by some $f_j \in F'$ $(i \neq j)$, yielding F''.
3. Let $\overline{F}$ be $\{F_1, \ldots, F_k\}$, where each $F_i \subseteq F''$ $(1 \leq i \leq k)$ is a maximal set of pairwise alternatives. Use the mapping ϕ, described below, to map $\overline{F}$ to an ULRQ.

Conceptually, $\overline{F}$ represents a multi-hop question (if $k > 1$), where each F_i is a single hop represented by a union of base relations. This is reflected in the definition of the mapping ϕ below:

$\phi(f) =$ *the relation associated with* f if f is a single candidate parse
$\phi(\{f_1, \ldots, f_k\}) = \phi(f_1) \vee \ldots \vee \phi(f_k)$ if $\{f_1, ..., f_k\}$ is a set of pairwise alternatives
$\phi(\{F_1, \ldots, F_m\}) = \phi(F_1) \wedge \ldots \wedge \phi(F_m)$ if each F_i is a maximal set of pairwise alternatives

If f is a single candidate parse, ϕ maps it to a ULRQ as described in (Gao et al. 2018a)—a relation uniquely associated with f's lvp with arguments filled in by the extracted words (such as `movie('FilmNm'='M','ID'=_,director=D)`). A set of pairwise alternatives is treated by ϕ as a single hop and is mapped to a union of relations. A set of sets of pairwise alternatives is mapped by ϕ into a database join of the queries that correspond to the constituent hops. To illustrate, we give two examples of translation of multi-hop questions to ULRQ.

Example 9. Consider the sentence "*Who wrote a film that shares a director with Titanic?*". This sentence has four candidate parses:

f_1: `(Movie,2,[(FilmNm,4,'verb->subject'), (Writer,2,'verb->object')])`
f_2: `(Movie,6 [(FilmNm,4,'verb->subject'),(Director,8,'verb->object')])`
f_3: `(Movie,6,[(FilmNm,10,'verb->pp,pp->dep'),(Director,8,'verb->object')])`
f_4: `(Distinct,6,[(Item1,4,'verb->subject'),(Item2,10,'verb->pp,pp->dep')])`

with the maximum pairwise alternative sets $F_1 = \{f_1\}$, $F_2 = \{f_2\}$, $F_3 = \{f_3\}$, and $F_4 = \{f_4\}$, yielding the following query:

```
?- movie('FilmNm'=Title1,'Id'=ID1,'Writer'=V2),
   movie('FilmNm'=Title1,'Id'=ID1,'Director'=V3),
   movie('FilmNm'='Titanic','Id'=ID2,'Director'=V3),
   distinct('Item1'=ID1,'Item2'=ID2).
```

where capitalized symbols are variables. Here `distinct` is a domain-independent built-in frame that indicates that the entities involved are distinct. It is triggered by the lexical units such as *…shared…with…*, *…different…from…*, and the like. □

Example 10. The question "*Who is an actor of Pascal Laugier?*" has these two candidate parses:

```
f1: (Coop,4,[(Actor,1,'object->verb,verb->subject'),
             (Director,7,'lobject->rel,rel->robject')])
f2: (Coop,4,[(Actor,1,'object->verb,verb->subject'),
             (Writer,7,'lobject->rel,rel->robject')])
```

The frame `Coop` here represents the concept of cooperation such as writers co-writing a movie, actors co-acting or playing in a movie directed by somebody, etc. Since this is a composite concept, it must be defined via a background rule, which is done using the *same CNL*. For example, the background rule for the concept expressed by the first candidate parse is "*If an actor plays in a film that is directed by a director and the actor is different from the director then the actor is an actor of the director*" and KALM would translate this into the following Prolog rule:

```
coop('Actor'=V1,'Director'=V2):-
     movie('FilmNm'=Title1,'Id'=ID1,'Actor'=V1),
     movie('FilmNm'=Title1,'Id'=ID1,'Director'=V2),
     distinct('Item1'=V1,'Item2'=V2).
```

Since *Pascal Langier* is both a writer and a director in the MetaQA ontology, the two parses are alternatives to each other, which correctly captures the intent that the query should return those actors who play in movies that are either directed or written by *Pascal Laugier*. This sentence has just one maximum pairwise alternative set, $F_1 = \{f_1, f_2\}$ and the corresponding query is

```
?- coop('Actor'=V1,'Writer'='Pascal Laugier');
   coop('Actor'=V1,'Director'='Pascal Laugier').
```

□

6 Experiments

In this section, we first describe the evaluation of KALM-QA on the MetaQA dataset (Zhang et al. 2018a) and then show how we verified that queries generated by KALM-QA are 100% correct.

6.1 Evaluation of KALM-QA on the MetaQA Dataset

We now discuss our experiments using the extensive MetaQA Vanilla dataset (Zhang et al. 2018a) of 2- and 3-hop questions about movies, described in Section 3, and compare our results with the machine learning approach reported in (Zhang et al. 2018b).

Settings. Prior to the start of our experiments, KALM had only 50 frames with 217 lvps having nothing to do with movies. Therefore, we expanded the semantic background knowledge with two frames, `Movie` and `Coop`(eration) (the latter to capture aspects like

co-acting and co-directing). In addition, we trained the KALM-QA semantic parser to recognize all of the 234,176 2- and 3-hop training questions in MetaQA. This was done by selecting a small number of such questions, training the parser on them (as in Section 4), checking if there are still sentences that KALM-QA cannot parse, selecting a few of those, train again, and so on. Overall, it took a few rounds and required 88 sentences to generate 88 lvps that enabled KALM-QA to parse all of the training and testing sentences in MetaQA (which is almost 300,000).

Results. KALM-QA achieved 100% accuracy for both 2-hop and 3-hop questions—see Section 6.2. This compares very favorably with the advanced machine learning approach that introduced MetaQA, VRN (Zhang et al. 2018b), which reports the accuracy of 89.9% for the 2-hop dataset and 62.5% for the 3-hop dataset. However, as detailed in Section 3, these results are based on mislabeled data with high rate of incorrect query answers, so it is possible that the real accuracy of this approach is lower (it should be clear that KALM-QA only uses the ontology and the questions from MetaQA, but not the mislabeled answers).

Discussion. The accuracy of 100% achieved by KALM-QA may seem surprising not only compared to machine learning approaches but also in light of our previous work, which reported the accuracy of only about 95% (Gao et al. 2018b; Gao et al. 2018a). The explanation lies in the fact that the 5% error rate in out prior work was exclusively due to the noise and imprecision present in BabelNet (Navigli and Ponzetto 2012)—the background ontology used there. In contrast, the present work uses the provided WikiMovies ontology (Miller et al. 2016), which is precise and has no known errors (not to be confused with the errors in MetaQA query answers).

6.2 Correctness of KALM-QA Query Results

This subsection shows how we verified the correctness of the results produced by KALM-QA for such a large number of MetaQA queries—queries that required joining two or three relations. This was done by verifying that the actual Prolog queries produced by KALM-QA are semantically correct and exactly match their corresponding English sentences.

Since the number of such queries is large (close to 30,000), we had to devise a method that exploits the symmetries present in MetaQA queries. The method consists of three distinct phases:

1. Identifying all the query templates that exist in KALM-QA translations (which are Prolog rules) of MetaQA queries (which are English sentences).
   It turned out that there are only 44 unique 2-hop query patterns and 59 unique 3-hop patterns.
2. Verifying that all MetaQA English questions that correspond to the same KALM-QA template differ only in the constant values used (like the named entities '*Steven Spielberg*', '*Bright Star*' or years like 1980).
   This means that these questions must be translated to logical queries that are instances of the same KALM-QA template.

3. For each KALM-QA query template T, let $MQA(T)$ be the set of all MetaQA questions that get translated into an instance of T. All that is needed now is to manually select one representative $q(T) \in MQA(T)$ for each T and verify that its KALM-QA translation in Prolog is semantically correct. The number of such checks is a few hundreds and was accomplished manually within a few hours.
   Why hundreds and not just 44+59=103? Truth to be told, some templates had two to four different, semantically equivalent English forms in $MQA(T)$, so each of these forms had to be checked separately. For instance, the sentences *which films share the same actor of* [*Bright Star*] and *what are the films that have the same actor of* [*Friday the 13th*] are semantically equivalent and thus have the same corresponding logical template.

Additional details of the above steps are as follows.
Step 1: A query like

```
q(W2):-movie('FilmNm'=W2,'Id'=I2,'Actor'=W6),
       movie('FilmNm'='Bright Star','Id'=I8,'Actor'=W6),
       I2 \= I8.
```

gets standardized into a template like

```
q(A):-movie('FilmNm'=A,'Id'=B,'Actor'=C),
      movie('FilmNm'=xxxx,'Id'=D,'Actor'=C),
      B \= D.
```

where the variables are standardized and the query-specific constants are replaced with fixed constants. For example, `'Bright Star'` got replaced with xxxx. The resulting templates are then sorted lexicographically, which yielded the aforementioned number of templates.
Step 2: MetaQA English sentences are then textually grouped according to their corresponding KALM-QA templates. Each group is then scanned to verify that all sentences there differ only in query-specific constant values. This check is visual and rather tedious, but because the difference between the different sentences in each group is trivial, the check takes only a fraction of a second per line and took a few hours.
Step 3: As mentioned, this required a manual check of a few hundred MetaQA sentences with respect to their corresponding KALM-QA translations.

While KALM-QA yields 100% correct answers, we found that MetaQA misreports a very large percentage of query answers, as discussed in Section 3. Most of this mislabeling is because MetaQA's designers failed to take into account that the movie ontology has many distinct movies with the same title, but there were also more subtle reasons. An annotated report on all these errors appears in https://github.com/tiantiangao7/kalm-qa. It should be noted that explanations for most of the errors in MetaQA were generated by an automated Prolog script.

7 Related Work

We have already discussed the work on VRN (Zhang et al. 2018b), which used machine learning to answer multi-hop questions provided by MetaQA. Other machine learning approaches to query answering include the QA system of (Bordes et al. 2014) and

KV-MemNN (Miller et al. 2016). In (Zhang et al. 2018b), it is reported than these two systems have much lower accuracy and thus, presumably, than KALM-QA.

Non-machine-learning works include the systems that are designed with a multi-stage pipeline that translates NL questions into queries that use SPARQL, SQL, or Prolog to then query the actual databases. In these systems, translation is typically driven by some kind of ontology. Representative systems include ATHENA (Saha et al. 2016), PowerAqua (López et al. 2012), and NaLIR (Li and Jagadish 2014). Compared to these works, KALM encompasses both knowledge authoring and question answering. For the latter aspect, it would be interesting to compare the performance of KALM with ATHENA and the others, and it is also interesting to know how these systems would perform on the MetaQA dataset.

The most obvious relation to KALM is, of course, Attempto itself (Fuchs et al. 2008) and SBVR (Object-Management-Group 2017), as they can also be used to query the Movie ontology. However, since bare Attempto and SBVR include no background linguistic knowledge, that knowledge would have to be supplied via background rules, e.g., *if an actor appears in a film then the actor is an actor of the film.* It is not hard to estimate (see (Gao et al. 2019, Appendix B)) that to capture the semantics expressed by the 88 lvps that KALM-QA had to learn for MetaQA (Section 6), Attempto and SBVR would require 974 background rules, hand-written in CNL (plus the rules for complex concepts like cooperation, as in KALM-QA). A fruitful way to see the relationship between KALM-QA and Attempto (and, to an extent, SBVR) is to view the former as the necessary semantic layer over the latter, which can make the combination into a viable knowledge authoring and question answering technology.

SNOWY (Gomez et al. 1994), in development from mid 1980s to 2008, is related to KALM-QA in that it does limited mediation between semantically equivalent sentences with the help of an ontology, but likely still requires many bridge rules to fully capture semantic equivalence. Unfortunately, this system is no longer available and seems to have been abandoned, so a more detailed comparison is hard to do.

8 Conclusion

This paper introduced KALM-QA, an extension of KALM that can answer complex multi-hop CNL questions with very high accuracy. Contrary to the prevailing trend, KALM-QA is based on logic programming rather than machine learning. Using MetaQA, a large collection of 2- and 3-hop questions against a large movie ontology, we demonstrated that KALM-QA achieves superior accuracy compared to machine learning approaches. In fact, the error rate for the most accurate machine learning approach in our comparison turned out to be much higher than what was reported in (Zhang et al. 2018b) because the end-to-end learning used in that work corrupted the expected query answers and the system learned wrong answers for a very large subset of the questions (north of 50% for 3-hop queries). For future work, we plan to extend KALM with support for rule authoring and common sense and temporal reasoning.

Acknowledgments. This work was partially supported by NSF grant 1814457. We are grateful to Norbert Fuchs and Rolf Schwitter for enlightening discussions on Attempto. Many thanks to Katherine Choi for validating the bulk of MetaQA queries.

References

Angeli, G., Premkumar, M. J. J., and Manning, C. D. 2015. Leveraging linguistic structure for open domain information extraction. In *53rd Annual Meeting of the Association for Computational Linguistics*. ACL, Beijing, China, 344–354.

Bordes, A., Chopra, S., and Weston, J. 2014. Question answering with subgraph embeddings. In *2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, A. Moschitti, B. Pang, and W. Daelemans, Eds. ACL, Doha, Qatar, 615–620.

Das, D., Chen, D., Martins, A. F. T., Schneider, N., and Smith, N. A. 2014. Frame-semantic parsing. *Comp, Linguistics 40,* 1, 9–56.

Fillmore, C. J. and Baker, C. F. 2001. Frame semantics for text understanding. In *Proceedings of WordNet and Other Lexical Resources Workshop*. NAACL, Pittsburgh, USA.

Fuchs, N. E., Kaljurand, K., and Kuhn, T. 2008. Attempto controlled english for knowledge representation. In *Reasoning Web*. Springer, Venice, Italy, 104–124.

Gao, T. 2019. *Development of KALM-QA*. Stony Brook University. https://github.com/tiantiangao7/kalm-qa.

Gao, T., Fodor, P., and Kifer, M. 2018a. High accuracy question answering via hybrid controlled natural language. In *2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI2018)*. IEEE, Santiago, Chile, 17–24.

Gao, T., Fodor, P., and Kifer, M. 2018b. Knowledge authoring for rule-based reasoning. In *On the Move to Meaningful Internet Systems. OTM 2018 Conferences - Confederated International Conferences: CoopIS, C&TC, and ODBASE 2018*, H. Panetto, C. Debruyne, H. A. Proper, C. A. Ardagna, D. Roman, and R. Meersman, Eds. Lecture Notes in Computer Science, vol. 11230. Springer, Valletta, Malta, 461–480.

Gao, T., Fodor, P., and Kifer, M. 2019. Querying Knowledge via Multi-Hop English Questions. *ArXiv e-prints abs/1907.08176*, 1–19.

Gomez, F. 2008. The acquisition of common sense knowledge by being told: an application of nlp to itself. In *International Conference on Application of Natural Language to Information Systems*. Springer, London, UK, 40–51.

Gomez, F., Hull, R. D., and Segami, C. 1994. Acquiring knowledge from encyclopedic texts. In *4th Applied Natural Language Processing Conference (ANLP)*. ACL, Stuttgart, Germany, 84–90.

Johnson, C. R., Fillmore, C. J., Petruck, M. R., Baker, C. F., Ellsworth, M. J., Ruppenhofer, J., and Wood, E. J. 2002. FrameNet: Theory and Practice.

Kuhn, T. 2014. A survey and classification of controlled natural languages. *Comp. Linguistics 40,* 1, 121–170.

Li, F. and Jagadish, H. V. 2014. Constructing an interactive natural language interface for relational databases. *PVLDB 8,* 1, 73–84.

López, V., Fernández, M., Motta, E., and Stieler, N. 2012. PowerAqua: Supporting users in querying and exploring the semantic web. *Semantic Web 3,* 3, 249–265.

Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S. J., and McClosky, D. 2014. The Stanford CoreNLP natural language processing toolkit.

Mausam, Schmitz, M., Soderland, S., Bart, R., and Etzioni, O. 2012. Open language learning for information extraction. In *The Joint Conf. on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, EMNLP-CoNLL*. Association for Computational Linguistics, Jeju Island, Korea, 523–534.

Miller, A. H., Fisch, A., Dodge, J., Karimi, A., Bordes, A., and Weston, J. 2016. Key-value memory networks for directly reading documents. In *2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, J. Su, X. Carreras, and K. Duh, Eds. The Association for Computational Linguistics, Austin, TX, 1400–1409.

Navigli, R. and Ponzetto, S. P. 2012. BabelNet: The automatic construction, evaluation and application of a wide-coverage multilingual semantic network. *Artificial Intelligence 193*, 217–250.

Object-Management-Group. 2017. Semantics of business vocabulary and business rules (SBVR), v. 1.4. OMG standards document. http://www.omg.org/spec/SBVR/Current.

Ringgaard, M., Gupta, R., and Pereira, F. C. N. 2017. SLING: A framework for frame semantic parsing. *CoRR 1710.07032*, 1–9.

Saha, D., Floratou, A., Sankaranarayanan, K., Minhas, U. F., Mittal, A. R., and Özcan, F. 2016. ATHENA: an ontology-driven system for natural language querying over relational data stores. *PVLDB 9,* 12, 1209–1220.

Schwitter, R. 2010. Controlled natural languages for knowledge representation. In *COLING 2010, 23rd Intl. Conf. on Computational Linguistics, Posters Volume, 23-27.* ACL, Beijing, China, 1113–1121.

Zhang, Y., Dai, H., Kozareva, Z., Smola, A. J., and Song, L. 2018a. The MetaQA dataset. https://github.com/yuyuz/MetaQA.

Zhang, Y., Dai, H., Kozareva, Z., Smola, A. J., and Song, L. 2018b. Variational reasoning for question answering with knowledge graph. See McIlraith and Weinberger (2018), 6069–6076.