Knowledge Authoring and Question Answering via Controlled Natural Language

Tiantian Gao
Department of Computer Science, Stony Brook University
Stony Brook, NY, USA
tiagao@cs.stonybrook.edu

Abstract

Knowledge acquisition from text is the process of automatically acquiring, organizing and structuring knowledge from text which can be used to perform question answering or complex reasoning. However, current state-of-the-art systems are limited by the fact that they are not able to construct the knowledge base with high quality as knowledge representation and reasoning (KRR) has a keen requirement for the accuracy of data. Controlled Natural Languages (CNLs) emerged as a technology to author knowledge using a restricted subset of English. However, they still fail to do so as sentences that express the same information may be represented by different forms. Current CNL systems have limited power to standardize sentences that express the same meaning into the same logical form. We solved this problem by building the Knowledge Authoring Logic Machine (KALM), which is a technology for domain experts who are not familiar with logic to author knowledge using CNL. The system performs semantic analysis of English sentences and achieves superior accuracy of standardizing sentences that express the same meaning to the same logical representation. Besides, we developed the query part of KALM to perform question answering, which also achieves very high accuracy in query understanding.

2012 ACM Subject Classification Computing methodologies → Knowledge representation and reasoning, Computing methodologies → Natural language processing

Keywords and phrases Knowledge Authoring, Question Answering, Controlled Natural Language

Digital Object Identifier 10.4230/OASIcs.ICLP.2018.21

1 Introduction

Knowledge acquisition is the process of extracting, organizing, and structuring knowledge from data sources such that the constructed knowledge base can be used for question answering or performing complex reasoning. Traditional ways of knowledge acquisition largely reply on domain experts to encode the knowledge base in rule-based systems such as XSB [12] and Clingo [4]. However, this requires too much domain specific knowledge and eligible engineers are in very short supply. Information extraction systems emerged as the tools to extract knowledge frame text (i.e., OpenIE [1], SEMAFOR [2], Stanford CoreNLP/KBP [8], SLING [10]). They achieved admirable results in processing free text, however, their accuracy is far from meeting the requirement of knowledge representation. In addition, they are only designed to extract the knowledge from text, but not intended to represent it in a way suitable for reasoning. Controlled Natural Languages (CNLs) [7] emerged as a technology that bridges this gap. Representative systems include Attempto Controlled English (ACE) [3] and Processable English (PENG) [11]. They are designed to process English sentences with restricted grammar but unambiguous interpretations and translate the sentences into logic for reasoning. The main issue with CNLs is that they have

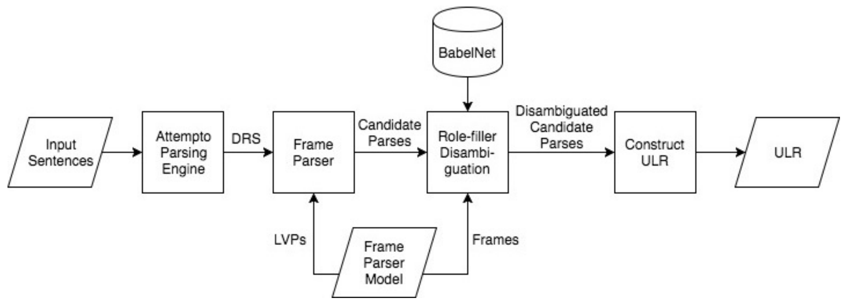

Figure 1 Pipeline for translating a sentence into ULR.

limited power of standardizing sentences that express the same information via different syntactic forms into the same logical representation. For instance, the sentences *a customer buys a phone*, *a customer makes a purchase of a phone*, *a customer is a buyer of a phone* are mapped to different logical representations. Therefore, they are not suffice for question answering or complex logical reasoning.

In this work, we build Knowledge Authoring Logic Machine (KALM), which conducts semantic analysis of CNL sentences and achieves superior accuracy of standardizing English sentences that express the same information via different forms to the same logical form. The system is built based on utilizing linguistic knowledge bases (BabelNet [9] and FrameNet [5]) and our frame-based parsing and disambiguation algorithms. Besides, we developed the query part of KALM which supports high accuracy query parsing and answer retrieval. The following is organized as follows: Section 2 describes the KALM system for knowledge authoring, Section 3 describes the query part of KALM, Section 4 shows the evaluation results of KALM, Section 5 discusses the next steps of work, Section 6 concludes the paper.

2 Knowledge Authoring Logic Machine (KALM)

Figure 1 shows the pipeline of KALM that translates a CNL sentence into *unique logical representation* (*ULR*), the semantic form of CNL sentences. The KALM framework consists of five components:

Syntactic Parsing. We use Attempto Parsing Engine (APE)[1] to parse CNL sentences and translate them into Discourse Representation Structure (DRS) [6], which represents the syntactic and dependency information of the sentences. DRS relies on 7 predicates: `object/6`, `predicate/4`, `property/3`, `modifier_adv/3`, `modifier_pp/3`, `relation/3`, and `has_part/2`. For example, the `object`-predicate represents an entity which corresponds to a noun word in the sentences. A `predicate`-predicate represents an event and the subject and object of the events. `predicate`-predicate corresponds to a verb word in a sentence. For example, given the sentence *A customer buys a phone*, it is parsed into DRS as

```
object(A,customer,countable,na,eq,1)
object(B,phone,countable,na,eq,1)
predicate(C,buy,A,B)
```

[1] `https://github.com/Attempto/APE`

Frame-based Parsing. Based on the DRS, the frame-based parser generates a list of *candidate parses*, which represent the frame-semantic relations the sentences entail. For instance, given the sentence *A customer buys a phone*, the frame-based parser generates the following parse result: `Frame(Commerce_Buy, Roles: Buyer =` *customer*, `Goods =` *phone*`)`. The parse says there are two entities: *customer* and *phone*, which are involved in the `Commerce_Buy` relation. The *customer* serves as the `Buyer` role of this frame relation and *phone* serves as the `Goods` role of the frame relation. The parser is constructed based on two components: *logical frames* and *logical valence patterns* (*lvps*). The logical frames represent the definition of the frame relations via Prolog facts. For instance, the `Commerce_Buy`[2] frame is represented as

```
fp(Commerce_Buy,[
    role(Buyer, [bn:00014332n], []),
    role(Seller, [bn:00053479n], []),
    role(Goods, [bn:00006126n,bn:00021045n], []),
    role(Recipient, [bn:00066495n],[]),
    role(Money, [bn:00017803n], [currency])]).
```

where for each `role`-term, the first argument represents the name of the *frame role*, the second argument represents the BabelNet synsets associated which capture the meaning of the role, and the third argument specifies some data type constraints. The lvps represent the grammatical context of a sentence that could potentially entail a frame. Consider the following lvp for extracting an instance of the `Commerce_Buy` frame:

```
lvp(buy, v, Commerce_Buy, [
    pattern(Buyer, verb->subject, required),
    pattern(Goods, verb->object, required),
    pattern(Recipient, verb->pp(for)->dep, optnl),
    pattern(Money, verb->pp(for)->dep, optnl),
    pattern(Seller, verb->pp(from)->dep, optnl)]).
```

The first and second arguments represents a *lexical unit* (a word + *part-of-speech*) that could trigger an instance of the `Commerce_Buy` frame. Next, it comes with a list of `pattern`-terms, each represents the syntactical context between the lexical unit, frame role, and the actual *role-filler* word. The lvps are generated automatically by KALM based on the annotated training sentences, which contains the frame name, lexical unit, and frame elements information. When a new sentence comes, we check every word in the sentence and find whether there exists any lvp whose lexical unit matches the chosen word. If so, we apply the lvp to the sentence and extract an instance of the frame from the sentence.

Role-Filler Disambiguation. Doing frame-based parsing is not enough because the aforementioned frame-based parsing only replies the grammatical information of the sentence. This way of parsing may generate candidate parses that misidentify the frames, role-filler words, or assign the wrong roles to the role-filler words. To rule out the wrong parses, we perform role-filler disambiguation which checks whether the extracted role-filler words are semantically compatible with the frame roles. For each role-filler and role pair, we compute a semantic score. Based on the scores for the role-filler and role pairs, we score the

[2] `https://framenet2.icsi.berkeley.edu/fnReports/data/frame/Commerce_buy.xml`

entire candidate parse and removes the ones that falls below a threshold. To compute the semantic score, we first query the role-filler word against BabelNet and get a list of associated BabelNet synsets (called *candidate role-filler synsets*). Then, we traverse BabelNet semantic network and measure the semantic similarity between each candidate role-filler synset and the corresponding role-synset. Basically, we consider all semantic paths that connect the synset pair, and then use a heuristic scoring function to score the path. The candidate role-filler synset which achieves the highest semantic score is chosen and assigned as the disambiguated role-filler synset for the respective role-filler word.

Translating into ULR. Based on the *disambiguated candidate parses* generated from the role-filler disambiguation step, we translate the parses into ULR. ULR uses `frame/2` and `role/2` predicates to represent instances of frames and roles. ULR uses `synset/2` and `text/2` predicates to represent the synset and text information for the role-filler words. For example, the sentence *a customer buys a phone* is translated into ULR as

```
frame(id_1, Commerce_buy).
role(id_1, Buyer, id_2).
role(id_1, Goods, id_3).
synset(id_2, bn:00022095n). % customer synset
text(id_2, customer).
synset(id_3, bn:00062020n). % phone synset
text(id_3, phone).
```

3 Question Answering

3.1 Issues in CNL-based Queries

The ACE query language[3] supports two types of queries: *true/false*- and *wh*-queries where the query words include *who*, *where*, *what*, and so on. A *true/false*-query is translated into DRS the same way as a definite sentence does. For *wh*-queries, APE uses a special predicate `query/2` to represent the *wh*-words. For instance, the query *who buys what?* is represented in DRS as

```
query(A,who)-1/1
query(B,what)-1/3
predicate(C,buy,A,B)-1/2
```

where the variables *who* and *what* are captured by the `query`-predicate.

However, APE only does shallow syntactic analysis of a query. There are a few issues to solve before we can precisely capture the meaning of a query and acquire the intended knowledge. Consider the following query sentences:

1. *Mary buys which car?*
2. *Who buys IBM's stocks?*
3. *Which person buys which car in which place at which price?*
4. *A $person buys a $car in a $place at a $price.*

First of all, as a *wh*-variable is a placeholder for the entities to be shown in the output result, the type of entities the variable represents must be disambiguated and also used for

[3] `http://attempto.ifi.uzh.ch/site/docs/ace/6.7/ace_constructionrules.html`

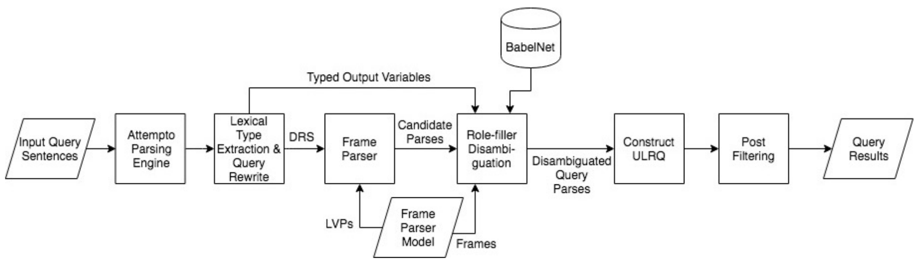

Figure 2 Pipeline for translating a query into ULRQ and answer retrieval and filtering.

acquiring the related information. As shown in Sentence (1) in the above example, we need to identify that the type of the entities associated with the *which*-variable is a *car*. Therefore, if we know *Mary* buys both a *Camry* and a *pen*, only *Camry* should be returned.

Second, ACE's query language has constrained power of denoting types in the query. As shown in Sentence (2) in the above example, emphWho could refer to either a *company* or a *person*. However, it is ambiguous whether the user intends to acquire *company* or *person* entities or both. One solution to that is to use the query word *which* and rewrite the sentence as *Which person buys IBM's stocks* if the user intends to acquire *person* entities. However, the sentence may become cumbersome when there are many such typed variables as shown in Sentence (3). To solve this problem, we introduce *typed output variables* in the query language as the form *$type*. Hence, Sentence (3) will be rewritten to Sentence (4) which is expressed in a more precise and concise way.

Third, to acquire the associated instances of frames from the knowledge base which is constructed in the knowledge authoring phase, we also need to perform frame-semantic parsing based on queries. However, as shown in the previous example, the DRS for query is not exactly the same as the DRS used to represent definite sentences. Therefore, the existing lvps are not applicable for parsing queries. One way to solve this problem is to construct an additional set of training sentences for queries. However, this will requires a lot of work. Besides, since FrameNet doesn't contain any sentences related to queries, it would require a lot of manual work to construct CNL queries. To solve this issue, we perform a DRS rewrite to queries such that we can reuse the existing lvps for definite sentences to parse queries.

3.2 Question Answering

Figure 2 shows the pipeline that translates a CNL query into the logical form, *Unique Logical Representation for Queries* (*ULRQ*), which is used to query the knowledge base to retrieve the answers. The question answering part consists of the following components:

Syntactic Parsing. This is the same as the knowledge authoring part.

Query Parsing. We also perform frame-based parsing to generate several candidate parses which represent the frame relations the query belongs to. However, as mentioned in the previous subsection, the DRS for queries are different from the DRS for definite sentences. Therefore, we perform a DRS adaptation of the DRS corresponding to the query such that the existing lvps for definite sentences can be reused to do frame-based parsing for queries. Besides, we perform a syntactic analysis of the queries and identify the lexical types of the query words (e.g., *which*).

Role-Filler Disambiguation. This is the same as the knowledge authoring part.

Translating Queries into ULRQ. Queries are represented in a similar way as definite sentences except that we use logical variables to denote instances of frames and roles. For instance, the query *Who buys a phone?* is translated into ULRQ as

```
?- frame(FrameV,'Commerce_Buy'),
   role(FrameV,'Buyer',BuyV), synset(BuyV,BuyerRoleFillerOutV),
   role(FrameV,'Goods',GoodV), synset(GoodV,GoodsRoleFillerOutV),
   check_type(BuyerRoleFillerOutV,bn:00046516n),  % person synset
   check_type(GoodsRoleFillerOutV,bn:00062020n).  % phone synset
```

Type Filtering of Query Results. As is shown from the ULRQ above, the clauses from lines 1-3 retrieves all instances of frames and the associated roles from the knowledge base. However, not all role-fillers for `Buyer` and `Goods` may be related to *person* and *phone*. To rule out the unrelated ones, we perform type filtering of the query results, which calls the `check_type` predicate in the above ULRQ.

4 Evaluation

At present, KALM contains 50 logical frames with 213 logical valance patterns. We use the following metrics to measure the performance of the system:

FrSynC	all frames, roles & output variables are identified correctly; all role-filler words & variable types are disambiguated correctly
FrC	all frames, roles and output variables are identified correctly, but some disambiguation mistakes
Wrong	some frames, roles or output variables are misidentified

For knowledge authoring, we achieve an accuracy of 95.6% (FrSynC). This accuracy is far from the state-of-the-art information extraction systems including SEMAFOR, SLING, and Stanford CoreNLP. For understanding of the queries, we achieve an accuracy of 94.49% (FrSynC).

5 Next Steps

The current work focuses on authoring of definite knowledge from CNL sentences and question answering. The next step is to acquire rules from CNL sentences and perform more complex reasoning. This not only requires parsing individual sentences correctly, but also requires multi-sentence parsing and information in different sentences must be related to each other properly. This goes well beyond anaphora resolution, which ACE is already able to handle.

1. *Every bird is an animal.*
2. *Every bird flies.*
3. *Stella is a sea eagle.*
4. *Penguins do not fly.*
5. *A violet is not an animal.*
6. *Sparrow Daffy doesn't fly.*

Consider the above example: Sentences (1) and (2) denote rules which say that if we know there is a *bird*, we can infer the bird is an animal and flies. Therefore, based on Sentences (1), (2) and (3), we can infer *Stella* is an animal and flies. However, this does't hold for *Tweety* because Sentence (4) is an exception to Sentence (2) and therefore refutes any conclusion derived from Sentence (2). Moreover, based on Sentence (1) and (5), since a violet is not an animal, we conclude that a violet is not a bird. But, this way of reasoning is not desired for Sentence (2) and (6) because Daffy may be injured and therefore not being able to fly.

To precisely capture the meaning of rules in CNL and perform reasoning for the above cases, the research issues are three-fold: the first issue is the development of CNL extensions that are suitable for representing rules and inter-sentence dependencies/references. For instance, in addition to the form "*every ...*" as shown in Sentence (1) and (2), we can also use an "*if...then*" statement to represent a rule. Besides, we need a mechanism to indicate the inter-sentence dependencies as shown in Sentence (1) and (4) where Sentence (4) is an exception case to Sentence (2). This could be done either by specifying the inter-sentence dependencies explicitly or by an automatic mechanism to recognize these dependencies without explicit mentioning.

The second issue is the actual nature of the logic to be used for capturing rules. As shown in the above example, when there is a fact that a *violet* is not an animal, it is natural to infer that it is not an animal. But, it is not reasonable to infer the Daffy is not a bird if Daffy doesn't fly. To distinguish the differences, we can use a *first-order logic* rule to represent Sentence (1) where *contrapositive inference* is desired and use a *Prolog* rule to represent Sentence (2) where contrapositive inference is not required. As to the inter-sentence dependency between Sentence (2) and (4), we believe *defeasible logic* is a good fit. Basically, rules have priories in defeasible logic where the rule with a higher priority can defeat a default rule which has a lower priority. For the above example, we label Sentence (4) as a rule with a higher priority than the rule corresponding Sentence (2) and also defeat the low priority rule when incompatible conclusions are derived.

The third issue is standardization. Same as knowledge authoring for definite sentences and queries, we will also standardize rules that express the same meaning via different syntactic forms.

6 Conclusion

In this work, we described the KALM system, which achieves superior accuracy in knowledge authoring and question answering. The system is built on our frame-based parsing and disambiguation algorithms and the use of external linguistic knowledge bases including BabelNet and FrameNet. As the next step, we plan to work on extracting rules from sentences and perform common sense reasoning.

References

1 Gabor Angeli, Melvin Jose Johnson Premkumar, and Christopher D. Manning. Leveraging Linguistic Structure For Open Domain Information Extraction. In *53rd Annual Meeting of the Association for Computational Linguistics*, pages 344–354, Beijing, China, 2015.

2 Dipanjan Das, Desai Chen, André F. T. Martins, Nathan Schneider, and Noah A. Smith. Frame-Semantic Parsing. *Comp, Linguistics*, 40(1):9–56, 2014. `doi:10.1162/COLI_a_00163`.

3 Norbert E. Fuchs, Kaarel Kaljurand, and Tobias Kuhn. Attempto Controlled English for Knowledge Representation. In *Reasoning Web, 4th Intl. Summer School, Sept. 7-11*, pages 104–124, Venice, Italy, 2008.

4 Martin Gebser, Benjamin Kaufmann, Roland Kaminski, Max Ostrowski, Torsten Schaub, and Marius Thomas Schneider. Potassco: The Potsdam Answer Set Solving Collection. *AI Commun.*, 24(2):107–124, 2011.

5 Chrstopher R. Johnson, Charles J. Fillmore, Miriam R.L. Petruck, Collin F. Baker, Michael J. Ellsworth, Josef Ruppenhofer, and Esther J. Wood. FrameNet: Theory and Practice, 2002.

6 Hans Kamp and Uwe Reyle. *From discourse to logic: Introduction to modeltheoretic semantics of natural language, formal logic and discourse representation theory*, volume 42. Springer Science & Business Media, 2013.

7 Tobias Kuhn. A Survey and Classification of Controlled Natural Languages. *Comp. Linguistics*, 40(1):121–170, 2014.

8 Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. The Stanford CoreNLP Natural Language Processing Toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60, 2014. URL: `http://www.aclweb.org/anthology/P/P14/P14-5010`.

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

10 Michael Ringgaard, Rahul Gupta, and Fernando C. N. Pereira. SLING: A framework for frame semantic parsing. *CoRR*, 1710.07032:1–9, 2017. `arXiv:1710.07032`.

11 Rolf Schwitter. English as a Formal Specification Language. In *13th Intl. Workshop on Database and Expert Systems Appl. (DEXA 2002)*, pages 228–232, Aix-en-Provence, France, 2002.

12 T. Swift and D.S. Warren. XSB: Extending the power of prolog using tabling. *Theory and Practice of Logic Programming*, 2011.