Theorem Proving of First-order Logic for Solving Pronoun Disambiguation Problems

No Author Given

No Institute Given

Abstract. In this paper, we propose theorem proving of first-order logic for solving pronoun disambiguation problems(PDP). The PDP task introduced in this paper contains the extended version of the publicly available questions in the Winograd Schema Challenge(WSC). To solve publicly available questions in the WSC, a question answering system must have the ability of finding the commonsense knowledde and reasoning out the answer based on the commonsense knowledge. To focus on the ability of reasoning out the answer, we build the extended version of publicly available questions in the WSC by manually adding related commonsense knowledge to each question. And then we build a question answering system using the algorithm of theorem proving of first-order logic. In result, our question answering system achieves 63.75% accuracy, outperforming R-NET in the test set.

Keywords: Winograd Schema Challenge(WSC), First-order Logic, Pronoun Disambiguation Problems, Logical Reasoning

1 Introduction

In recent years, many systems which do not have the human-like intelligence have passed the Turing Test. Therefore, many possible Turing Test alternatives[1][2][3] have been proposed. Among these possible alternatives, the Winograd Schema Challenge(WSC)[4]is proposed by Hector Levesque, a computer scientist at the university of Toronto. It is a multiple-choice test that employs questions of a very specific structure. Every Winograd Schema(WS) question is a pronoun disambiguation problem, which consists of three parts, description, question and answers pair. For example:

Description: The trophy doesnt fit into the brown suitcase because its too small.
Question: What is too small?
Answers pair: The trophy / the brown suitcase.

To solve the example problem showed above is to find the word to replace what in the question sentence. As we know, if thing A is big and thing B is small, then thing A cant fit into thing B. So, by substituting "thing A" with "the trophy" and substituting "thing B" with "the brown suitcase", we can reason that if the trophy is big and the brown suitcase is big, then the trophy dose not fit into the brown suitcase. And then we know that the correct answer to this question is the brown suitcase. But for a machine, it might not have commonsense knowledge about the relationship between the word small and the phrase fit into, and it dose not know how to use commonsense knowledge to reason out the correct answer. Therefore, the key to solve WS question is to find the commonsense knowledge related to the question and reason out the answer by using the commonsense knowledge. To solve WS question, Arpit Sharma et al. [5] used Google search to find the related commonsense knowledge and then used semantic parser to build the semantic representation graphs of the commonsense knowledge and the question. Finally, they reasoned on theses semantic representation graphs to find the answer. Among 282 WS questions, they select 71 pairs of the questions to solve and their system was able to answer 53 pairs. Quan Liu et al. [6] proposed a commonsense knowledge enhanced embeddings model, which uses three knowledge base, WordNet, ConceptNet, CauseCom to search for commonsense knowledge related to the question and then uses the commonsense knowledge to learn knowledge enhanced embeddings. It achieves the best system with the highest accuracy 58.3% in 2016 Winograd Schema Challenge. Both of these two methods use its own way to find commonsense knowledge and then use commonsense knowledge to reason out the answer, so theses two methods may get different number of correct commonsense knowledge related to the WS question and we can't explicitly tell which method has the better ability of reasoning. Therefore, we can manually add commonsense knowledge in the form of natural language for each WS question, which we call extended Winograd Schema(ExtWS) question.

The ExtWS questions can be used to test a question answering system's ability of reading understanding and reasoning on natural language. To solve ExtWS questions, we proposed theorem proving of first-order logic algorithm and we got 63.75% on 298 ExtWS questions and we used R-NET[7] to solve ExtWS questions, and make a comparision between these two experiment results. R-NET ranked third on SQuAD and its source code is available on https://github.com/YerevaNN/R-NET-in-Keras.

The remainder of the paper will start with introducing preparation of building our question answering system, which consists of three parts. First is to introduce the ExtWSC dataset and the next two part is to introduce two necessary tools which are used to build our question answering system. One of the tools is SEMPRE[9], which can be used as a semantic parser. The other is Z3-prover[8], a theorem prover, which we use to reason out the answer. Then we will introduce our approach, theorem proving of first-order logic. After that, we present the experiment result and analysis on solving ExtWSC. At last, we make conclusion and talk about our future work.

2 Preparations

In this section, we will introduce the ExtWSC dataset, semantic parser Sempre and theorem prover Z3.

2.1 The ExtWSC Dataset

The ExtWSC dataset includes 298 questions, each of which consist of description sentences, knowledge sentences, question sentences and answers pair. Compared with the WSC dataset, the ExtWSC dataset extends almost every question in the WSC dataset with one or more related commonsense knowledge sentences in the form of natural language. There are still some complicate questions which we dont know how to add commonsense knowledge sentences for. Each commonsense knowledge sentence has the following form.

Commonsense knowledge sentence $->$ A | If A, then B.

$S->$ one sentence with one verb and its related nouns

$A->$ S | S or A | S and A

$B->$ S | S or B | S and B

Each ExtWSC question has the following three forms.

One Sentence: In the ExtWSC dataset, some questions are only related to one commonsense knowledge sentence and we human can reason out the answer by using this only one sentence.For example:

Description: The trophy doesnt fit into the brown suitcase because its too small.

Knowledge: If thing A is big and thing B is small, then thing A can't fit into thing B.

Question: What is too small?

Answers pair: The trophy / the brown suitcase.

More Sentences: There are also some questions related to more commonsense knowledge sentences and we can reason out the answer if and only if we use all these sentences.For example:

Description: The customer walked into the bank and stabbed one of the tellers. He was immediately taken to the emergency room.

Knowledge: If person B stabs person C, then person C might be hurt. If person B is hurt, then person B might be taken to emergency room.

Question: Who was taken to the emergency room?

Answers pair: The teller / the customer.

Complicate Questions: There are still some questions which are too complicate to add commonsense knowledge sentences.For example:

Description: This book introduced Shakespeare to Goethe; it was a fine selection of his writing.

Question: Fine selection of whose writing?

Answers pair: Ovid / Shakespeare

To reason out the answer to this question, we must know who Shakespeare is and who Goethe is and there are no direct relations between the word introduced and the phrase selection of his writing. Therefore, our system can not reason out the answer to such question.

2.2 SEMPRE

SEMPRE is a toolkit for training semantic parsers, which transform natural language texts to intermediate logical forms. Its source code is available online, so we can download the source code on the github and then use script to run SEMPRE. In this paper, we use SEMPRE to parse sentences in ExtWSC questions. Its parsing result consists of three main parts. One is Lemmatized Tokens, which shows all the lemmatized tokens in the sentence. Another one is

POS tags, identifying which word is a verb and which a word is noun and so on. The last one is Dependency children, identifying words relations in the sentence. For example, if we use SEMPRE to parse the sentence The trophy cant fit into the brown suitcase, we will know the subject of the phrase fit into is trophy and the object is suitcase. Based on SEMPREs parsing result, our algorithm can automatically generate a Z3 script which can represent information about natural language sentence in first-order logic language. The Z3 script consists of several commands that can be executed by Z3-prover. We introduce this in next part.

2.3 Z3-prover

Z3-prover is a theorem prover from Microsoft Research. Its source code is available online, so we can download the source code on the github and build it. In our algorithm, we use first-order logic language to represent sentences in the form of natural language. So we have to identify predicates and entities in the sentences. Then we transform proper names into individual constants and transform common nouns, verbs and adjectives into predicates. According to Z3 input format which is an extension of the one defined by the SMT-LIB 2.0 standard[9], we can use the below commands in the Z3 script to represent natural language sentences. The declare-const command declares an individual constant, which represent a proper name in sentences . The declare-rel command declares a predicate, which represent a verb, a common noun or an adjective in sentences. Once we identify all the individual constants and predicates , we can use the assert command to represent a complete sentence in the form of first-order logic language. Then, we use Z3-prover to execute these command and get its resolving result. At last, we can reason out the answer to the ExtWSC question based on the result.

3 Our Approach

Our approach is based on two main tasks, namely, semantic parsing of natural language sentences, and introducing four auxiliary rules to reason out the answer. Next, we will explain how these two tasks help our question answering system to answer ExtWSC questions.

3.1 Semantic Parsing of Natural Language Sentences

In our algorithm, we use first-order logic language to represent natural language sentences, and use Z3 command to represent first-order logic language. Therefore, we firstly need to parse the natural language sentences. Here, we s-elect SEMPRE as our semantic parser. Based on its parsing result, we use the following steps to automatically generate Z3 commands to represent first-order logic language information of natural language sentences.

1. **Identifying Individual Constants:** In first-order logic language, each individual constant refers to the entity in the sentences which bears that name. In natural language sentences, proper names can be translated into individu-al constants. For example, in sentence *Joan made sure to thank Susan for all the help she had given*, the proper name *Joan* is an individual constant. To automatically generate Z3 commands, once we identify a individual constant

based on SEMPREs parsing result, we automatically add a Z3 command, *(declare-const %individualConstant %sort)*, into the Z3 script. %individualConstant is the string that represent the proper name in natural language sentences. %sort is the sort that we use Z3 command *declare-sort* to declare in Z3 script. Each individual constant must have a sort.

2. **Identifying Predicates:** In first-order logic language, each predicate refers to a verb, a common noun or an adjectives in the sentences. Predicates can have one or more arguments. Each argument is an individual constant or an individual variable. A predicate with n arguments is called n-place predicate. For example, in sentence *Joan made sure to thank Susan for all the help she had given*, the verb phrase *thank for* is a 3-place predicate with *Joan, Susan, help* as arguments and *(thankFor Joan Susan help)* is a proposition.
3. **Identifying Universal Quantifiers or Existential Quantifiers:** In ExtWSC questions, variables with universal quantifiers or existential quantifiers only exist in commonsense knowledge sentences. In our manually adding commonsense knowledge sentences, we add two constraints:
1.If a noun phrase consists of a word and an uppercase letter, it is a variable with universal quantifier which we use Z3 command *forall* to declare.
2.If the word is *something, somebody, he, him or it*, the word is a variable with existential quantifier which we use Z3 command *exists* to declare. Each variable with quantifiers also have a sort.
4. **Translating Description** After identifying all constants, predicates, variables in sentences, we add Z3 command, *(assert (and (%predicate1 %constants1) (%predicate2 %constants2) ...))*, into Z3 script, where %predicate1 refer to the verb, common noun or adjective in the sentences and %constants1 refer to all the proper names related to this predicate in the sentences. We use and to represent conjunction to make sure that each proposition is true.
5. **Translating Commonsense Knowledge** As we show above, there are two forms of commonsense knowledge sentences. For the sentence in the form of *If A, then B.* ,we regard A as antecedent and B as consequent. And then add Z3 command to represent A entails B. For the sentence in the form of *A* , we use the method similar to translating description sentences into Z3 commands.
6. **Translating Question** After translating description and commonsense knowledge sentences into Z3 commands, we need to translating question sentence to reason out the answer. Firstly identifying predicates and their arguments from question sentence. Secondly extracting question particle which has a tag *WP* in SEMPREs parsing result. Finally replacing question particle with constants in answers pairs one by one and get its negative form. For example, in sentence *Who had given help*, we add Z3 command *(assert (not (give Joan help)))* and *(assert (not (give Susan help)))* into different Z3 scripts.

3.2 Introducing Four Auxiliary Rules to Reason Out The Answer

After automatically generating Z3 commands of sentences in description, commonsense knowledge and question, we still cant reason out the correct answer to the ExtWSC question. Therefore, we introduce four auxiliary rules to reason out the correct answer.

- **Entity Inequality:** In 4.1, we use Z3 command declare-const to declare proper name in the sentences. From the view of script file, each proper name is represented by a string. But from the view of model, each proper name is just a value. Therefore, two different proper names represented by two different string may have the same value, which may make our system to reason out a wrong answer. So we add the rule of entity inequality for the individual constants that have the same sort. For example, in the sentence Joan made sure to thank Susan for all the help she had given, we firstly get two individual constants Joan and Susan, and then we add Z3 command (assert (not (= Joan Susan))).
- **Closed-world Assumption:** According to closed-world assumption, everything we dont know is false. Therefore, for variables with existential quantifier or universal quantifier, we must limit the range of them so that these variables can only take the value of the individual constants which have been identified in the ExtWSC question. For example, in the sentence Joan made sure to thank Susan for all the help she had given, we firstly declare two individual constants with Z3 commands (declare-const Joan person) and (declare-const Susan person), and then we add Z3 command (assert (forall ((x person)) (or (= x Joan) (= x Susan)))), to limit the range of x when x has the sort person.
- **Closed-reason Assumption:** In our approach, we reason out the correct answer when the propositions in description sentences entail the proposition in question sentence.According to our manually adding commonsense knowledge sentences, sometimes the propositions in description sentences are antecedent in an entailment and sometimes the propositions in question sentence are antecedent. Therefore, we introduce closed-reason assumption. Assuming the question sentence has the predicates (Q1) or (Q1 and Q2) and the description sentences have the predicates (D1) or (D1 and D2). In an entailment of commonsense knowledge sentences, Q1 and Q2 have the following relations with D1 and D2.

  1. Q1 and Q2 in the antecedent:
     if Q1 entails (D1 and D2), we add sentences, D1 entails Q1, D2 entails Q1;
     if Q1 entails (D1 or D2), we add sentences, D1 entails Q1, D2 entails Q2;
     if (Q1 or Q2) entails D1, we add sentences, D1 entails Q1, D1 entails Q2;
     if (Q1 and Q2) entails D1, we add a sentence, D1 entails (Q1 and Q2);
     if Q1 entails D1, we add a sentence, D1 entails Q1.
  2. Q1 and Q2 in the consequent:
     if D1 entails (Q1 or Q2), we add two sentences, D1 entails Q1, D1 entails Q2;
     for other relations, we add one sentence, consequent entails antecedent.

- **Unique Answer:**Each ExtWSC question has only one correct answer, so we add Z3 command to assure that once candidate answer A is the correct answer, candidate answer B can not be the correct answer.

4 Experiments

4.1 Accuracy

In this paper, we aim to test if our question answering system has the better ability of reasoning in ExtWSC questions than R-NET. Among 298 ExtWSC questions, our system can correctly answer 105 questions, while R-NET can only answer 60 questions. Because every ExtWSC question has two candidate answers, both of these two systems can randomly select one of the candidate answers as their output. We add the ability of randomly selecting candidate answer into these two systems when the systems can not reason out the answer. In result, our system achieves 63.75% accuracy while R-NET achieves 55.03% accuracy.

4.2 Error Analysis

Although we manually add commonsense knowledge sentence for each WS question to get ExtWSC questions, we still cant achieve 90% accuracy on ExtWSC questions. We conclude three reasons why our system cant reason out the correct answer. Firstly, our semantic parsing tool SEMPRE can not correctly find the predicate and its related entities in those sentences which have attributive clause or appositive clause, so we can not generate a correct Z3 script for such sentences. Secondly, some words, which should be parsed as verb, are parsed as noun or adjective. Therefore, our system gets wrong words dependency and then gets wrong predicates or wrong entities. Thirdly, some WS questions are so complicate that we cant add related commonsense knowledge sentences for them and our system cant reason out the answer without commonsense knowledge.

5 Conclusion and Future Works

In this paper, by adding commonsense knowledge sentences for every WS question, we build a new dataset, which is called ExtWSC questions. ExtWSC questions can be used to test a question answering systems ability of reading comprehension and reasoning out the answer. Then, we propose theorem proving of first-order logic to build a question answering system. In this algorithm, we firstly propose how to translate natural language sentence into a Z3 script, which can represent first-order logic formula. Secondly, we introduce four auxiliary rules to complete a Z3 script to make it possible to reason out the answer. Finally, we compare our system with R-NET. In result, our question answering system outperforms R-NET in accuracy.

Although our question answering system outperforms R-NET, it still cant achieve 90% accuracy. As error analysis shows, there exist some sentences that cant be correctly parsed by SEMPRE. Therefore, in our future work, we will use coreference resolution tool in NLTK, dependency parser and SEMPRE altogether to parse a sentence and get the predicates and entities more accurately. Besides, our system relies on manually added commonsense knowledge to reason out the answer. So we will aim to solve WS question which doesnt contain the commonsense knowledge sentences by automatically extracting the related commonsense knowledge in ConceptNet, WordNet, CauseCom or ResearchCyc.

References

1. Levesque, H.J., Davis, E., Morgenstern, L.: The winograd schema challenge. In: Principles of Knowledge Representation and Reasoning: Proceedings of the Thirteenth International Conference, KR 2012, Rome, Italy, June 10-14, 2012. (2012)
2. Bringsjord, S., Bello, P., Ferrucci, D.A.: Creativity, the turing test, and the (better) lovelace test. Minds and Machines **11**(1) (2001) 3–27
3. Riedl, M.O.: The lovelace 2.0 test of artificial creativity and intelligence. CoRR **abs/1410.6142** (2014)
4. Levesque, H.J.: The winograd schema challenge. In: Logical Formalizations of Commonsense Reasoning, Papers from the 2011 AAAI Spring Symposium, Technical Report SS-11-06, Stanford, California, USA, March 21-23, 2011. (2011)
5. Sharma, A., Vo, N.H., Aditya, S., Baral, C.: Towards addressing the winograd schema challenge - building and using a semantic parser and a knowledge hunting module. In: Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015. (2015) 1319–1325
6. Liu, Q., Jiang, H., Ling, Z., Zhu, X., Wei, S., Hu, Y.: Combing context and commonsense knowledge through neural networks for solving winograd schema problems. CoRR **abs/1611.04146** (2016)
7. Natural Language Computing Group, M.R.A.: R-net: Machine reading comprehension with self-matching networks. (2017)
8. de Moura, L.M., Bjørner, N.: Z3: an efficient SMT solver. In: Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings. (2008) 337–340
9. Barrett, C.S.C., Stump, A., Tinelli, C.: The smt-lib standard c version 2.0. (2010)