A Semantic Approach for Question Analysis

Dai Quoc Nguyen[1], Dat Quoc Nguyen[1], and Son Bao Pham[1,2]

[1] Faculty of Information Technology
University of Engineering and Technology
Vietnam National University, Hanoi
{dainq, datnq, sonpb}@vnu.edu.vn
[2] Information Technology Institute
Vietnam National University, Hanoi

Abstract The first step that a question answering system must perform is to transform an input question into an intermediate representation. All published works so far use rule-based approaches to realize this transformation in question answering systems. Nevertheless, in existing rule-based approaches, manually creating the rules is error-prone and expensive in time and effort. In this paper, we focus on introducing a rule-based approach that offers an intuitive way to create compact rules for extracting intermediate representation of input questions. Experimental results are promising where our system achieves reasonable performance and demonstrate that it is straightforward to adapt to new domains and languages.

1 Introduction

The goal of question answering systems is to give answers to the user's questions instead of ranked lists of related documents as used by most current search engines [3]. Natural language question analysis component is the first component in any question answering systems. This component creates an intermediate representation of the input question, which is expressed in natural language, to be utilized in the rest of the system.

For the task of translating a natural language question into an explicit intermediate representation of the complexity in question answering systems, all published works so far use rule-based approach to the best of our knowledge. Some question answering systems such as Aqualog [5] and VnQAS [7] manually defined a list of sequence rule structures to analyze questions. However, in these rule-based approaches, manually creating the rules is error-prone and expensive in time and effort.

In this paper, we present an approach to return an intermediate representation of question via FrameScript scripting language [6]. Natural language questions will be transformed into intermediate representation elements which include the construction type of question, question class, keywords in question and semantic constraints between them. Framescript allows users to intuitively write rules to directly extract the output tuple.

In section 2, we provide some related works and describe our overall system architecture in section 3. We present our approach for question analysis in section 4 and describe our experiments in section 5. Discussion and conclusion will be presented in section 6.

2 Related works

2.1 Question analysis in question answering systems

Early NLIDB systems used pattern-matching technique to process user's question and generate corresponding answer [1]. A common technique for parsing input questions in NLIDB approaches is syntax analysis where a natural language question is directly mapped to a database query (such as SQL) through grammar rules.

The PRECISE system [10] maps the natural language question to a unique semantic interpretation by analyzing some lexicons and semantic constraints. In [12], the authors described a template-based system to translate English question into SQL query by matching the syntactic parse of the question to a set of fixed semantic templates. Some other systems based on semantic grammar rules such as Planes [14], Eufid [13]. Semantic grammar-based approaches were considered as an engineering methodology, which allows semantic knowledge to be easily included in the system.

Recently, some question answering systems that used semantic annotations generated high results in natural language question analysis. Aqualog [5] and the first Ontology-based question answering system for Vietnamese [7] perform semantic and syntactic analysis of the input question based on semantic annotations in the use of JAPE grammar rules in GATE framework [2]. Nguyen et al. [9] proposed a language independent approach utilizing JAPE grammars to systematically construct a knowledge base for processing natural language questions. The difference between these approach and our approach is that we use FrameScript scripting language [6] to analyze input questions.

2.2 FrameScript Scripting Language

FrameScript [6] is a language for creating a multi-modal user interfaces. It evolves from Sammut's Probot [11] to enable rule-based programming, frame representations and simple function evaluation.

Each script in FrameScript [6] includes a list of rules which will be matched against user inputs to give the corresponding responses. A scripting rule in the FrameScript language consists of a pattern and responses with the form:
pattern ==> responses.

A pattern expression allows the use of non-terminals to reuse other pattern expressions. Response expressions contain two different types namely sequences and alternatives. A sequence of responses has the form surrounded by brackets: [response 1 | response 2 | ... | another response] and it is also possible to specify additional conditions to decide which responses will be selected. Furthermore, once the pattern is matched, its components are numbered in order starting from 1. These component are segments of the input that can be referred to in a response using '^'. When '^' is followed by an integer, the corresponding numbered pattern component is used in the output response. In addition, responses utilize the '#' to perform actions. Many examples using *#goto(a_ script, <<trigger>>)* to transform from the current script to another one are described in our companion paper [8].

3 Our Question Answering System Architecture

The architecture of our question answering system is shown in Figure 1. It includes two components: a Natural language question analysis engine and an Answer retrieval. The question analysis component takes the user question as

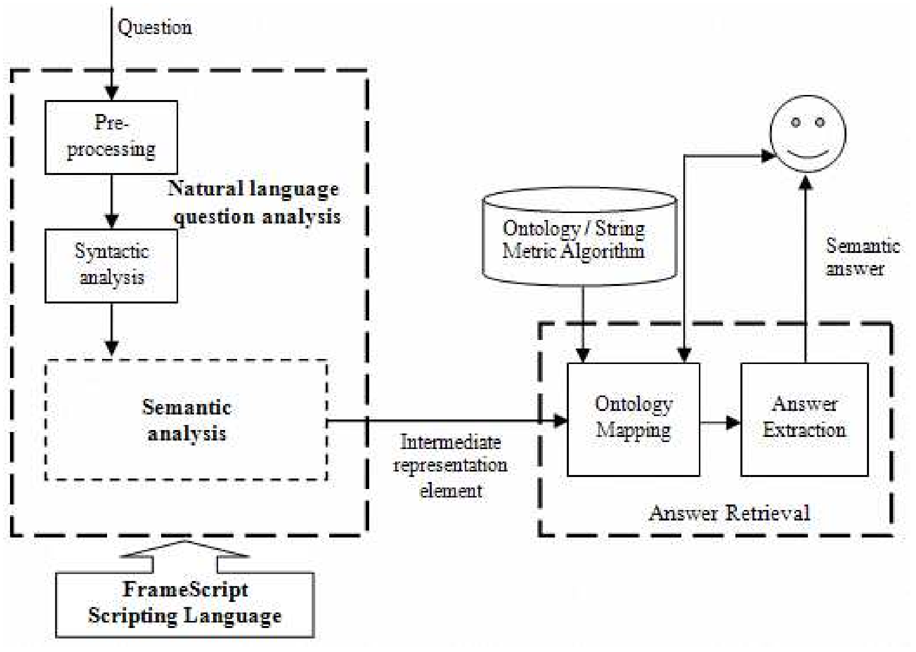
Question
Pre-processing
Natural language question analysis
Syntactic analysis
Semantic analysis
Ontology / String Metric Algorithm
Semantic answer
Intermediate representation element
Ontology Mapping
Answer Extraction
Answer Retrieval
FrameScript Scripting Language

Figure 1. Architecture of our question answering system.

an input and returns an intermediate query-tuple representing the question in a compact form. The role of this intermediate representation is to provide structured information of the input question for later processing such as retrieving answers. Similar to VnQAS [7], the answer retrieval component includes two main modules: Ontology Mapping and Answer Extraction. It takes an intermediate representation produced by the question analysis component and an Ontology as its input to generate semantic answers.

Unlike existing approaches for English [5] and Vietnamese [7] where the intermediate representation couldn't be extracted directly in rules, we will describe an approach to directly extract the representation of user's question using FrameScript scripting language [6].

4 Using FrameScript language for question analysis

The natural language analysis component presented in Figure 1 consists of three modules: preprocessing, syntactic analysis and semantic analysis.

To set the context for the discussion of question analysis, we will first describe the intermediate representation used in our approach, and then focus on proposing our approach to obtain this intermediate representation for a given question.

4.1 Intermediate Representation of an input question

Similar to VnQAS [7], the intermediate representation used in our approach aims to cover a wider variety of question types. It consists of a *question-structure* and one or more *query-tuple* in the following format:

(question-structure, question-class, $Term_1$, Relation, $Term_2$, $Term_3$)

where $Term_1$ represents a concept (object class), $Term_2$ and $Term_3$, if exist, represent entities (objects), *Relation* (property) is a semantic constraint between terms in the question. This representation is meant to capture the semantics of the question.

Simple questions corresponding to basic constructions only have one *query-tuple* and its *question-structure* is the query-tuple's question-structure. More complex questions such as composite questions are constructed by several sub-questions, each sub-question is described by a separate *question-structure*, and the *question-structure* capture this composition attribute. This representation is chosen so that it can represent a richer set of question types. Therefore, some terms or relation in the query-tuple can be missed. Composite questions such as:

"list all students in the Faculty of Information Technology whose hometown is Hanoi?"

has question structure of type *And* with two query-tuples where **?** represents a missed element: *(UnknRel , List , students , ? , Faculty of Information Technology, ?)* and *(Normal , List , students, hometown, Hanoi, ?).*

4.2 Preprocessing module

The preprocessing module identifies part-of-speech tags in a user's question. After that, we use part-of-speech tags to create basic scripts for detecting words. The basic scripts *Noun*, *Verb*, *Determiner*, *Adjective*, *Adverb*, *Conjunction* and

Preposition are used to identify corresponding nouns, verbs, determiners, adjectives, adverbs, conjunctions and prepositions. In fact, these scripts will be used in creating rules in the syntactic and semantic analysis modules at later stages.

```
Noun ::
    { NN | NNS | NNP | NNPS | NP | NPS | CD } ;;
Verb ::
    { FVG | VBN | VBZ | VBG | VBD | VBP | VB } ;;
Determiner ::
    { DT | PRP } ;;
Adjective ::
    { JJ | JJR | JJS } ;;
Adverb ::
    { RB | RBR | RBS } ;;
Conjunction ::
    { CC } ;;
Preposition ::
    { PREP | TO | IN } ;;
```

4.3 Syntactic analysis module

This module is responsible for determining noun phrases, question phrases and relation phrases between noun phrases or noun phrases and question phrases.

Concepts and entities are normally expressed in noun phrases. Therefore, it is important that we can reliably identify noun phrases in order to generate the *query-tuple*. Hence we build the script called *NounPhrase* (such as a sample script below) to specify patterns of noun phrases by utilizing scripts generated from the preprocessing module.

```
Composite ::
    { <Noun> | <Conjunction> | <Adjective>
    | <Adverb> <Adjective> } ;;
NounPhrase ::
    { <Noun> | <Determiner> <Noun>
    | <Composite> <Noun>
    | <Determiner> <Composite> <Noun> } ;;
```

For example, given the following question: *"which projects are about ontologies and the semantic web?"*, Three noun phrases *'projects'"*, *"ontologies"* and *"the semantic web"* will be identified.

In addition, we identify the question words, such as $HowWhy_{cause\ |\ method}$, *YesNo* $_{true\ or\ false}$, $What_{something}$, $When_{time\ |\ date}$, $Where_{location}$, $Many_{number}$, Who_{person}, to provide question classes. Accordingly, question phrases are detected by using noun phrases and question words to give information about question categories. Following VnQAS [7], we define the following question categories: *HowWhy, YesNo, What, When, Where, Who, Many, ManyClass, List* and *Entity*. This can be achieved by using the scripts such as the following:

```
Entity ::
    { <Wh_determiner> <Noun> } ;;
ManyClass ::
    { <Wh_adverb> <Many> <Noun> } ;;
```

For example, in the question: *"How many persons work on AKT project?"*, the phrase *"How many persons"* is identified by using the *ManyClass* script shown above.

The next step is to identify the relation phrases or semantic constraints between noun phrases or noun phrases and question phrases. We create a *Relation* script shown in the following sample script which is used to match relation phrases:

```
Noun_Adj ::
    { NN | NNS | NNP | NNPS | NP | NPS
    | CD | PRP | JJ | JJR | JJS } ;;
Relation ::
    { <Verb> | <Verb> <Noun_Adj> <Preposition>
    | <Verb> <Adverb> <Noun_Adj> <Preposition>};;
```

For instance, with the following question: *"who are the researchers in semantic web research area?"*, the phrase *"are the researchers in"* is the relation phrase detected by using the *Relation* script linking the question phrase *"who"* and the noun phrase *"semantic web research area"*.

Based on the scripts described above, we can determine noun phrases, relation phrases and question phrases of a user's question. In the next section, we describe in details the rules used to directly produce the intermediate representation of a question.

4.4 Semantic analysis module

The semantic analysis module identifies the *question-structure* and produces the *query-tuple*s as the intermediate representation *(question-structure, question-class, $Term_1$, Relation, $Term_2$, $Term_3$)* of the input question by using the scripts generated by the previous modules. We define the following question structures: *Normal, UnknTerm, UnknRel, Definition, Compare, ThreeTerm, And, Or, Clause, Combine, Affirm, Affirm_3Term and Affirm_MoreTuples* [7].

Existing scripts of *NounPhrase* and *Relation* are potential candidates for terms and relations respectively, while question phrases are used to detect question classes. We directly specify the rule's response expression to return the output consisting of a question structure and query-tupples.

For example, the following two rules:

```
<ManyClass> <Relation> <NounPhrase> ==>
    [ Normal, (Normal, ManyClass, ^1, ^2, ^3, ?) ]
<Entity> <Relation> <NounPhrase> <AND> <NounPhrase> ==>
    [ And, (Normal, Entity, ^1, ^2, ^3, ?), (Normal, Entity, ^1, ^2, ^5, ?) ]
```

are used to process the following two input questions resspectively where **?** represents a missed element in the tuples:

"How many subjects are there in the semester?"

[ManyClass *How many subjects*] [Relation *are there in*] [NounPhrase *the semester*]

and

"Which projects are about ontologies and the semantic web?"

[Entity *Which projects*] [Relation *are about*] [NounPhrase *ontologies*] [AND *and*] [NounPhrase *the semantic web*]

If an input question matches a rule's pattern, the rule's response expression specifies and extracts the corresponding elements in the intermediate representation[3]. For instance, the intermediate representation of the first question has:

question-structure of *Normal*

and *query-tuple (Normal, ManyClass, subjects, there, semester, ?).*

The the above second question has an intermediate representation consisting of:

question-structure of *And*

and two *query-tuple*s, that is, *(Normal, Entity, projects, are, ontologies, ?)* and *(Normal, Entity, projects, are, semantic web, ?).*

A nice feature in FrameScript is that it allows one to specify additional conditions in the response expression, instead of pattern expression, to select the appropriate response. A clear advantage is to group ambiguous cases together as well as conditions to resolve them in a single rule. For example, consider the following rule with conditional response expressions:

```
<What> <Relation> <NounPhrase> ==>
      [ ^(^2 == is or ^2 == are) ->
            Definition, (Definition, What, ?, ?, ^3, ?)
      | UnknTerm, (UnknTerm, What, ?, ^2, ^3, ?) ]
```

Using this rule, the intermediate representation of the question:

"what is the role of the academic regulation?"

has *question-structure* of *UnknTerm* and *query-tuple (UnknTerm, What, ?, role, academic regulation, ?).* However, also using this rule, the question:

"what is the standard program?"

has an intermediate representation containing the *question-structure* of *Definition* and tuple *(Definition, What, ?, ?, standard program, ?).*

Actually, creating the rules manually via three modules represented above is a language independent process as it is straightforward to adapt to a new domain and a new language.

5 Experiments

We take 170 English question examples of AquaLog's corpus[4] to build a set of 52 rules, which consumed about 12 hours of actual time to create all rules. Table 1

[3] Stopwords are removed

[4] http://technologies.kmi.open.ac.uk/aqualog/examples.html

shows the number of rules for each question-structure type. A point worth noting is that in 3 rules for *question-structure* of *UnknTerm*, there are 2 rules having conditional response expressions to resolve ambiguity between *UnknTerm* and *Definition*. These rules can be considered as composite rules and they can reveal ambiguity between different types of question structures. Table 2 presents a list of pairs of ambiguous question structure types together with the number of rules to resolve the ambiguity.

Table 1. Number of rules corresponding with each question-structure type

Question-structure type	Number of rules
Definition	1
UnknTerm	3
UnknRel	4
Normal	8
Affirm	6
ThreeTerm	10
And	13
Or	1
Clause	6

Table 2. Number of rules with conditional responses

Question-structure type	Question-structure type	Number of rules
UnknTerm	Definition	2
ThreeTerm	Normal	1
ThreeTerm	UnknTerm	1
Combine	Normal	1

As the intermediate representation of our system is different to AquaLog and there is no common test set available, it is difficult to directly compare our approach with Aqualog on the English domain.

To demonstrate that our approach could be applied to a new open domain, we use the above 52 rules to test the data of 500 questions[5] from TREC 10. Table 3 shows the number of correctly analyzed questions corresponding with each question-structure type.

Table 4 gives the sources of errors for 259 incorrect cases. It clearly shows that most errors come from unexpected structures. This could be rectified by adding more rules, especially when we construct a larger variety of question structure types from a bigger training data such as 5500 questions [4].

This experiment is indicative of the ability in using our system to quickly build rules for a new domain. We believe that our approach could be applied to

[5] http://cogcomp.cs.illinois.edu/Data/QA/QC/TREC_10.label

Table 3. Number of questions corresponding with each question-structure type

Question-structure type	Number of questions
Definition	130
UnknTerm	66
UnknRel	4
Normal	20
ThreeTerm	15
And	6

Table 4. Error results

Reason	Number of questions
Have special characters (such as / – “ ” **’s**) and abbreviations	64
Not have compatible patterns	185
Semantic error in elements of the intermediate representation	10

a new language because creating the rules manually for question analysis is a language independent process.

6 Conclusion

In this paper, we introduced a rule-based approach for converting a natural language question into an intermediate representation in a question answering system. Our system utilizes FrameScript to help users to create intuitive and compact rules for extracting elements of the intermediate representation. We constructed rules including patterns and associated responses, in which pattern is used to match user’ questions and its corresponding response as output is sent to return the intermediate representation of question. Experimental results of our system on a wide range of questions are promising with reasonable performance. We believe our approach can be applied to question answering in open domain against text corpora that requires an analysis to turn an input question to an explicit representation of some sort. Our method could be combined nicely with the processing of annotating corpus and it is straightforward to apply for a new domain and a new language.

In the future, we will extend our system to assist the rule creation process on a wide range of questions in open domain and to improve the generalization capability of existing rules.

Acknowledgements

This work is partially supported by the Research Grant from Vietnam National University, Hanoi No. QG.10.23.

The authors would like to acknowledge Vietnam National Foundation for Science and Technology Development (NAFOSTED) for their financial support to present the work at the conference.

References

1. Androutsopoulos, I., Ritchie, G., Thanisch, P.: Natural Language Interfaces to Databases - An Introduction. Natural Language Engineering Vol. 1, 29–81 (1995)
2. Cunningham, H., Maynard, D., Bontcheva, K., Tablan, V.: GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. In: Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL). pp. 168–175 (2002)
3. Hirschman, L., Gaizauskas, R.: Natural Language Question Answering: The View from here. Natural Language Engineering Vol. 7, 275–300 (2001)
4. Li, X., Roth, D.: Learning Question Classifiers. In: Proceedings of the 19th International Conference on Computational Linguistics. COLING '02, vol. 1, pp. 1–7 (2002)
5. Lopez, V., Uren, V., Motta, E., Pasin, M.: AquaLog: An Ontology-Driven Question Answering System for Organizational Semantic Intranets . Web Semantics: Science, Services and Agents on the World Wide Web Vol. 5, 72–105 (2007)
6. McGill, M., Sammut, C., Westendorp, J., Kadous, W.: Framescript: A Multi-modal Scripting Language. In: The School of Computer Science and Engineering, UNSW (Copyright © 2003-2008)
7. Nguyen, D.Q., Nguyen, D.Q., Pham, S.B.: A vietnamese question answering system. In: Proceedings of the 2009 International Conference on Knowledge and Systems Engineering. pp. 26–32 (2009)
8. Nguyen, D.Q., Nguyen, D.Q., Pham, S.B.: A Vietnamese Text-based Conversational Agent. In: Proc. of 25th International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems (IEA/AIE) (2012)
9. Nguyen, D.Q., Nguyen, D.Q., Pham, S.B.: Systematic Knowledge Acquisition for Question Analysis. In: Proceedings of Recent Advances in Natural Language Processing (RANLP 2011). pp. 406–412 (2011)
10. Popescu, A.M., Etzioni, O., Kautz, H.: Towards A Theory of Natural Language Interfaces to Databases. In: Proceedings of the 8th International Conference on Intelligent User Interfaces. pp. 149–157. IUI '03 (2003)
11. Sammut, C.: Managing Context in A Conversational Agent. In: Electronic Transactions on Artificial Intelligence. vol. 5, pp. 189–202 (2001)
12. Stratica, N., Kosseim, L., Desai, B.C.: NLIDB Templates for Semantic Parsing. In: Proceedings of the 8th International Conference on Applications of Natural Language to Information Systems. pp. 235–241 (2003)
13. Templeton, M., Burger, J.: Problems in Natural-Language Interface to DBMS with Examples from EUFID. In: Proceedings of the First Conference on Applied Natural Language Processing. pp. 3–16 (1983)
14. Waltz, D.L.: An English Language Question Answering System for A Large Relational Database. Communications of the ACM Vol. 21, 526–539 (1978)