Resolving Syntactic Ambiguities in Natural Language Specification of Constraints

Imran Sarwar Bajwa, Mark Lee, and Behzad Bordbar

School of Computer Science, University of Birmingham,
B15 2TT Birmingham, UK
{i.s.bajwa,m.g.lee,b.bordbar}@cs.bham.ac.uk

Abstract. In the NL2OCL project, we aim to translate English specification of software constraints to formal constraints such as OCL (Object Constraint Language). In the used approach, the Stanford POS tagger and the Stanford Parser are employed for syntactic analysis of English specification and the output of syntactic analysis is given to our semantic analyzer for the detailed semantic analysis. However, in few cases, the Stanford POS tagger and parser are not able to handle particular syntactic ambiguities in English specifications of software constraints. In this paper, we highlight the identified cases of syntactic ambiguities and we also present a novel technique to automatically resolve the identified syntactic ambiguities. By addressing the identified cases of syntactic ambiguities, we can generate more accurate and complete formal (OCL) specifications.

Keywords: Syntactic Ambiguities, Attachment Ambiguity, Ambiguity resolution, English constraints.

1 Introduction

In the recent years, a few research contributions have been presented in the area of automatic translation of natural language (NL) specifications to formal specifications such as UML (Unified Modeling Language) models [1] and SQL (Structured Query Language) queries [2]. However, the available tools are limited to 65%-70% levels of accuracy in real time software development. Researchers have shown that the key reason of less accuracy is the inherent ambiguity of the natural languages. For example, Mich [3] showed that 71.8% of a sample of NL software specification is ambiguous. Hence, the ambiguous and incomplete specifications lead to inconsistent and absurd formal specifications such the software models or the software constraints.

In the NL2OCL project [4], we aim to translate the English specification of constraints to the formal constraints such as OCL (Object Constraint Language) [5]. Our contribution is a semantic analyzer that performs semantic role labeling and generates a logical representation in English to OCL translation. Our semantic analyzer relies on the output of syntactic analysis (such as typed dependency [6]) performed by the Stanford parser [7]. However, we have identified a few cases where the Stanford parser is not able to handle particular syntactic ambiguities such as

A. Gelbukh (Ed.): CICLing 2012, Part I, LNCS 7181, pp. 178–187, 2012.

attachment ambiguity and homonymy. In result of a wrong syntax analysis, the semantic analysis goes wrong and finally the wrong OCL is generated. In this paper, we discuss the identified cases of syntactic ambiguities not resolved by the Stanford parser and we also present a novel technique to automatically resolve the identified cases of syntactic ambiguities in English specification. By addressing the identified cases of syntactic ambiguities, we can generate a more accurate and complete OCL specification.

The remaining paper is structured as follows: Section 2 provides the detailed description of the problem and the solution of the problem is given in Section 3. Section 4 presents the evaluation of the presented approach. Section 5 states the related work and the paper is concluded to discuss future work finally.

2 Description of the Problem

The NL2OCL project deals with the automated translation of English specification of constraints to OCL constraints via SBVR [12]. The most important phase in English to OCL translation is the processing of English text and generation of a logical representation such as FOL (First-Order-Logic). Finally, the logical representation is mapped to OCL syntax. In English text processing, the English text is syntactically and semantically analyzed. For syntactic analysis, the Stanford POS (Part-of-Speech) tagger [8] and the Stanford parser are used. The Stanford POS tagger is used to tag the English text and is capable of 97.0% [9] accuracy. Similarly, the Stanford parser is employed for the generation of the parse tree and the Typed Dependencies and is 84.1% [7] accurate. Accuracy of the Stanford parser is low for real-time applications and we need to improve the accuracy for robust and precise machine translation of English text.

In the semantic analysis phase, the output of the syntactic analysis is used for shallow and deep semantic parsing. In shallow semantic parsing, the semantic role labeling heavily relies on the typed dependencies generated by the Stanford parser. In result, the wrong typed decencies lead to wrong semantic role labeling. We have identified many cases where the Stanford parser generates the correct parse tree but the typed dependencies go wrong. There are some other cases where the Stanford POS tagger wrongly tags the tokens and the error propagates in rest of the stages of syntax analysis such as parse tree generation and typed dependency generation. For correct translation of English to OCL, we need to resolve these cases.

We have identified various cases where the typed dependencies are wrongly identified because of the attachment ambiguity [10]. Similarly, the most of the errors done by the Stanford POS tagger are due to homonymy (ibid). Following are the details of both types of syntactic ambiguity:

2.1 Attachment Ambiguity

Attachment ambiguity is a type of syntactic ambiguity where a prepositional phrase or a relative clause in sentence can be lawfully attached to one of the two parts of that

sentence [10]. We have also identified some cases where the Stanford parser generates wrong dependencies due to attachment ambiguity. An example of such cases is shown in Fig. 1. In this example, it is shown that the typed dependencies generated by the Stanford parser are wrong such as `prep_with(employees-7, bonus-9)`. However, the correct typed dependency for this example should be `prep_with(pay-2, bonus-9)` to represent the actual meanings of the example i.e. the pay with bonus is given to all the employees.

English: The pay is given to all employees with bonus.

Tagging: [The/DT] [pay/NN] [is/VBZ] [given/VBN] [to/TO] [all/DT] [employees/NNS] [with/IN] [bonus/NN] [./.]

Parse Tree:

```
(ROOT
    (S
      (NP (DT The) (NN pay))
      (VP (VBZ is)
        (VP (VBN given)
          (PP (TO to)
            (NP
              (NP (DT all) (NNS employees))
              (PP (IN with)
                (NP (NN bonus)))))))
      (. .)))
```

Typed Dependency (Collapsed):

```
det(pay-2, The-1)
nsubjpass(given-4, pay-2)
auxpass(given-4, is-3)
det(employees-7, all-6)
prep_to(given-4, employees-7)
prep_with(employees-7, bonus-9)
```

Fig. 1. Typed dependencies (collapsed) generated by the Stanford Parser

2.2 Homonymy

In linguistics, homonymy is a type of syntactic ambiguity in which a word in a phrase or a sentence exhibits different syntactic representations in different cases [10]. In this section, we aim to discuss various cases of homonymy that result in wrong POS tagging of English text. This problem becomes more serious, when the Stanford POS tagger wrongly identifies POS tags and due to the wrong POS tagging, the Stanford parser generates wrong parse trees and wrong dependencies. In NL2OCL project, we are totally relying on the performance of the Stanford parser. Hence, for the accurate and robust syntactic analysis, we need to address the possible cases of homonymy so that the Stanford parser may correctly parse the English text. Following are the few

examples, where the Stanford POS tagger failed to identify right POS tags and the error was propagated in the later stages of the syntax analysis performed by the Stanford parser:

English: A customer books two items.

Tagging: [A/DT] [customer /NN] [books/NNS] [two/CD] [items/NNS] [./.]

Parse Tree:
```
(ROOT
  (NP
    (NP (DT A) (NN customer) (NNS books))
    (NP (CD two) (NNS items))
    (. .)))
```

Typed Dependencies, collapsed:
```
det(books-3, A-1)
nn(books-3, customer-2)
num(items-5, two-4)
dep(books-3, items-5)
```

Fig. 2. Parts-of-Speech tagging and parsing of text by the Stanford POS Tagger & Parser

In Fig. 2, token 'books' is wrongly judged as 'NNS' by the Stanford parser. However, the token 'books' is verb and the correct POS tag is 'VBZ'. It is also shown in Fig 1 that parse tree is also wrong as VP is missing. Similarly, the typed dependencies (collapsed) are also wrongly identified `det(books-3, A-1)` should be `det(customer-2, A-1)`, `nn(books-3, customer-2)` should be `nsubj(books-3, customer-2)`, and `dep(books-3, items-5)` should be `nobj(books-3, item-5)`.

Another example of homonymy is "A customer can bank on manager". In this example, word 'bank' is wrongly POS tagged 'NN' but the correct POS tag is 'VB'. A similar example is "The manager made him type on typewriter." In this example word 'type' is wrongly tagged as 'NN', while the correct tag is 'VB'. Due to the wrong POS tagging, the parse trees of both these example is also wrongly generated by the Stanford parser.

Similar to homonymy cases, the cases for attachment ambiguity are also very important to resolve as the output of the Stanford parser is used in semantic analysis of English text and finally the output of the semantic analysis is mapped to OCL. Hence, the wrong syntax analysis results in wrong semantic analysis that ultimately generates wrong OCL.

3 Solution for Resolving Syntactic Ambiguity

To address the both types of syntactic ambiguities, discussed in section 2, we present a novel approach. We have identified that the both ambiguities are due to the absence of the context and by suing the context of the English text the correct interpretation of the ambiguous words and phrases is possible. In NL2OCL project, to translate NL

specification of constraints to OCL constraints, two inputs are required: English specification of a constraint and a UML class model. We propose the use of the information (such as classes, methods, attributes, associations, etc) available in the input UML class model for correct syntactic analysis.
The used approach for addressing the both types of syntactic ambiguities is explained in remaining part of the section.

3.1 Addressing Attachment Ambiguity

Similar to homonymy, attachment ambiguity can be resolved using the context. For generating correct dependencies of input English sentences, we again use the information on hand in the input UML class model. As, attachment ambiguity is due to the ambiguous role of noun with a preposition in a sentence. To correctly identify the attachment of the noun with other two nouns, we map the (three) candidate English elements (such as nouns) to the classes in the UML class model.

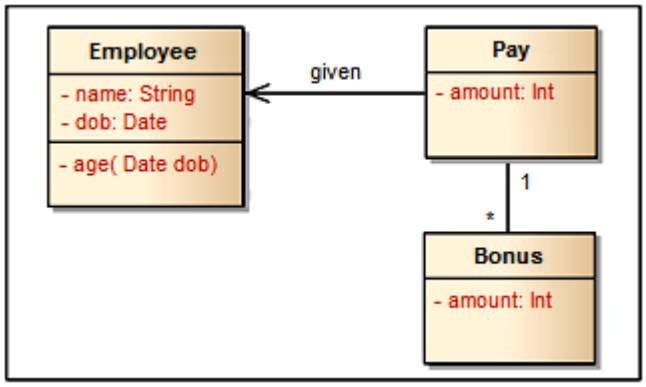

Fig. 3. A UML class model

The used mapping for attachment ambiguity resolution is slightly different from the mapping used for homonymy. First of all, the three (can be four or more) nouns are mapped to the class names in the input UML class model. Once the three classes are identified, the associations in those three classes are analyzed. With the help of the associations in the candidate classes the relationships in nouns are correctly identified. For example, the case of attachment ambiguity discussed in section 2.1 involves three nouns ‘pay’, ‘employees’, and ‘bonus’. All these three nouns are mapped to three classes (such as ‘Employee’, ‘Pay’, and ‘Bonus’) in the UML class model given in Fig. 3. After this mapping, the associations in all three classes are analyzed. The Stanford parser wrongly identifies that noun ‘bonus’ is attached to the noun ‘employees’. However, the UML class model shows that there is no relationship in classes ‘Bonus’ and ‘Employee’. While, there is a relationship in class ‘Pay’ and class ‘Bonus’. By using this information, we can correct the wrong dependencies. The corrected parse tree and dependencies are shown in Figure 4.

English: The pay is given to all employees with bonus.

Tagging: [The/DT] [pay/NN] [is/VBZ] [given/VBN] [to/TO] [all/DT] [employees/NNS] [with/IN] [bonus/NN] [./.]

Parse Tree:

```
(ROOT
   (S
     (NP
       (NP (DT The) (NN pay))
       (PP (IN with)
         (NP (NN bonus))))
     (VP (VBZ is)
       (VP (VBN given)
         (PP (TO to)
           (NP
             (NP (DT all) (NNS employees))))))
     (. .)))
```

Typed Dependency (Collapsed):

```
det(pay-2, The-1)
nsubjpass(given-4, pay-2)
auxpass(given-4, is-3)
det(employees-7, all-6)
prep_to(given-4, employees-7)
prep_with(pay-2, bonus-9)
```

Fig. 4. Corrected typed dependencies (collapsed)

We have generalized the used approach so that all variations of the discussed type of attachment ambiguity can be handled. For this purpose, the analysis of the relationships in classes of a UML class model such as associations (directed and undirected), aggregations and generalizations can play a key role.

3.2 Addressing Homonymy

As, we have explained earlier that the absence of the context is the major reason of ambiguity. For correct POS tagging of all English sentences especially the case of homonymy, we aim to use the available information in the target UML class model such as class names, attribute names, method names, associations, etc. In syntactic analysis, once we get the output of the Stanford POS tagger, we map all the words and their tags with the UML class model and confirm that all POS tags are correctly identified.

The process of mapping of POS tagged text to the UML class model is very simple. The POS tags of all the words are mapped to the elements of the UML class model. A set of mappings were defined for this purpose as shown in Table 1. If the token matches to an operation-name or a relationship name then it is a verb or if the ambiguous token matches to a class-name or attribute-name then it is classified as a common noun or proper noun.

Table 1. Mapping of English elements to UML class model elements

UML class model elements		English language elements
Class names	→	Common Nouns
Object names	→	Proper Nouns
Attribute names	→	Generative Nouns, Adjectives
Method names	→	Action Verbs
A ssociations	→	Action Verbs

By using the information shown in Table 1, we can correctly POS tag the example of homonymy discussed in Section 2.2. In Figure 5, it is shown that 'books' is an association in two classes 'Customer' and 'Item'. By using such information, it is identified that 'books' cannot be a noun in the context of UML class model (see figure 3). However 'books' can be a verb and the correct POS tag of token 'books' should be 'VBZ' as the token 'books' comes after a model verb (MD) 'can'.

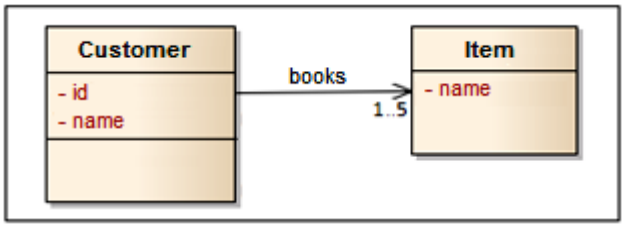

Fig. 5. A UML Class model

Once the POS tag is corrected, the parse tree and dependencies are also corrected as shown in the Figure 6.

English: A customer books two items.

Tagging: [A/DT] [customer /NN] [books/VBZ] [two/CD] [items/NNS] [./.]

Parse:
```
(ROOT
  (S
    (NP (DT A) (NN customer))
    (VP (VBZ books)
      (NP (CD two) (NNS items)))
    (. .)))
```

Typed Dependencies, collapsed:
```
det(customer-2, A-1)
nsubj(books-3, customer-2)
num(items-5, two-4)
dobj(books-3, items-5)
```

Fig. 6. Corrected Parts-of-Speech tag, parse tree and dependencies

4 Evaluation

To evaluate the impact of the presented approach in translation of English constraints to formal (OCL) constraints, we have calculated accuracy of the NL2OCL tool before resolving cases of syntactic ambiguities and after resolving syntactic ambiguities. The cases of attachment ambiguity and homonymy are separately evaluated as below:

4.1 Evaluating NL2OCL Tool for Attachment Ambiguity Cases

We have classified the results into two types: number of correctly parsed sentences ($N_{correct}$) and number of wrongly (inaccurate) parsed sentences ($N_{incorrect}$). The Recall value and Precision value calculated for the results is shown in Table 2.

Table 2. NL2OCL Evaluation results for Attachment Ambiguity

Example	N_{Total}	$N_{correct}$	$N_{incorrect}$	***Rec%***	***Prec%***
Inputs	14	13	1	92.85	92.85

We have also compared the results of NL2OCL with other available tools that can perform automated analysis of the NL requirement specifications. Recall value was not available for all tools, so we have used the precision value for comparison as shown in Table 3:

Table 3. NL2OCL Evaluation results after Ambiguity Resolution

The Stanford Parser accuracy	***Recall***	***Precision***
The Stanford Parser Accuracy before Ambiguity Resolution	84.1%	84.2%
The Stanford Parser Accuracy after Ambiguity Resolution	92.8%	92.8%

Comparison in table 3 shows that by handling attachment ambiguity the accuracy of the Stanford parser was improved from 84.1% to 92.85.

4.2 Evaluating NL2OCL Tool for Homonymy Cases

Similar to the evaluation of the attachment ambiguity, the results of homonym cases are divided into two types: number of correctly parsed sentences ($N_{correct}$) and number of wrongly (inaccurate) parsed sentences ($N_{incorrect}$). The accuracy (%age) calculated for the homonymy cases is 99%.

Table 4. NL2OCL Evaluation results after resolving Homonymy

The Stanford Parser accuracy	***Accuracy***
The Stanford Parser Accuracy before Homonymy Resolution	97.0%
The Stanford Parser Accuracy after Homonymy Resolution	99.0%

Table 4 shows that the accuracy of the Stanford parser is improved from 97% to 99%. The results of this initial performance evaluation are very encouraging and support both the approach adopted in this paper and the potential of this technology in general.

5 Related Work

Natural languages are inherently ambiguous and resolution of all types of ambiguities such as lexical, syntactic, semantic ambiguities is a long standing challenge. Much work has been done in the field of natural language ambiguity identification and resolution. Some of the researchers [3], [10], [11], [13] have presented approaches to identify the various types of ambiguities in a natural language text especially the natural language software requirements. Mich showed that 90% of the software requirements are captured in a natural language [3] such as English. Hence, the resolution of ambiguities in natural language specifications of software requirements and software constraints become more critical. However, translation of natural language such English to OCL is relatively a new area of research. We aim to contribute this area of research to improve the automated software modeling from natural language software requirements that also contains constraints.

6 Conclusion and Future Work

The primary objective of the paper was to address the challenge of resolving various cases of syntactic ambiguity such as attachment ambiguity and homonymy. By resolving the said cases of syntactic ambiguity the accuracy of machine processing can be improved. To address this challenge we have presented a NL based automated approach that uses a UML class model as a context of the input English (constraints) and by using the available information in the UML class model (such as classes, methods, associations, etc) we can resolve attachment ambiguity and homonymy. The results show a significant improvement in the accuracy of the Stanford POS tagger and the Stanford parser. By improving the accuracy of the Stanford POS tagger and the Stanford parser, the accuracy of English to OCL translation is also improved to 92.85% that was earlier 84.7%.

To further improve the accuracy of English to OCL translation we need to work on semantic ambiguities, and extra semantic ambiguities such as implicatures and presuppositions.

References

1. Harmain, H.M., Gaizauskas, R.: CM-Builder: A Natural Language-Based CASE Tool for Object-Oriented Analysis. Automated Software Engineering 10(2), 157–181 (2003)
2. Giordani, A., Moschitti, A.: Semantic Mapping Between Natural Language Questions and SQL Queries Via Syntactic Pairing. In: Horacek, H., Métais, E., Muñoz, R., Wolska, M. (eds.) NLDB 2009. LNCS, vol. 5723, pp. 207–221. Springer, Heidelberg (2010)
3. Mich, L., Franch, M., Inverardi, P.N.: Market research for requirements analysis using linguistic tools. Requir. Eng., 40–56 (2004)

4. Bajwa, I.S., Bordbar, B., Lee, M.G.: OCL Constraints Generation from NL Text. In: IEEE International EDOC Conference 2010, Vitoria, Brazil, pp. 201–214 (2010)
5. OMG: Object Constraint Language (OCL) Standard v. 2.0, Object Management Group (2006), http://www.omg.org/spec/OCL/2.0/
6. Marneffe, M.C., Bill, M., Manning, C.D.: Generating Typed Dependency Parses from Phrase Structure Parses. In: LREC 2006 (2006)
7. Cer, D., Marneffe, M.C., Jurafsky, D., Manning, C.D.: Parsing to Stanford Dependencies: Trade-offs between speed and accuracy. In: Proceedings of LREC 2010 (2010)
8. Toutanova, K., Klein, D., Manning, C., Singer, Y.: Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network. In: Proceedings of HLT-NAACL 2003, pp. 252–259 (2003)
9. Manning, C.D.: Part-of-Speech Tagging from 97% to 100%: Is It Time for Some Linguistics? In: Gelbukh, A.F. (ed.) CICLing 2011, Part I. LNCS, vol. 6608, pp. 171–189. Springer, Heidelberg (2011)
10. Kiyavitskaya, N., Zeni, N., Mich, L., Berry, D.: Requirements for tools for ambiguity identification and measurement in natural language requirements specifications. Requirements Engineering 13(3), 207–239 (2008)
11. Uejima, H., Miura, T., Shioya, I.: Improving text categorization by resolving semantic ambiguity. Communications, Computers and Signal Processing, 796–799 (2003)
12. Bajwa, I.S., Lee, M.G., Bordbar, B.: SBVR Business Rules Generation from Natural Language Specification. In: AAAI 2011 Spring Symposium -Artificial Intelligence for Business Agility (AI4BA), San Francisco, USA, pp. 2–8 (March 2011)
13. Umber, A., Bajwa, I.S.: Minimizing Ambiguity in Natural Language Software Requirements Specification. In: IEEE 6th International Conference on Digital Information Management (ICDIM 2011), Melbourne, Australia, pp. 102–107 (September 2011)